Introduction to

GEOSPATIAL ARTIFICIAL INTELLIGENCE

Gordana Jakovljević, Miro Govedarica, Maria Antonia Brovelli

Gordana Jakovljević

Miro Govedarica

Maria Antonia Brovelli

Introduction to Geospatial Artificial Intelligence

Banja Luka, 2025

Title of the publication:
Introduction to Geospatial Artificial Intelligence

Author(s):
Gordana Jakovljević, Ph.D. (assistant professor)
University of Banja Luka, Faculty of Architecture, Civil Engineering and Geodesy
Prof. Miro Govedarica, Ph.D. (full professor)
University of Novi Sad, Faculty of Techinical Science
Prof. Maria Antonia Brovelli, Ph.D. (full professor)
Politecnico di Milano, Department of Civil and Environmental Engineering

Reviewers:
Prof. Flor Álvarez Taboada, Ph.D. (full professor)
University of Leon, Department of Mining Engineering, School of Agrarian and Forest Engineering
Prof. Milan Rapaić, Ph.D. (full professor)
University of Novi Sad, Faculty of Technical Science

Publisher:
University of Banja Luka, Faculty of Architecture, Civil Engineering and Geodesy
Campus, 1A Vojvode Petra Bojovića Blvd.

For publisher:
Prof. Saša Cvoro, Ph.D. (full professor), dean

Place and year of printing:
Banja Luka, 2025

Printed by: Skandi, *Tešana Podrugovića 67, Banja Luka*;

ISBN: 978-99976-82-18-5
COBISS.RS-ID 143742721

At its first session on November 13, 2025, the Academic Council of the Faculty of Architecture, Civil Engineering, and Geodesy, University of Banja Luka, adopted Decision No. 14/3.1707-13/25 to publish *Introduction to Geospatial Artificial Intelligence* as a scientific monograph.

Preface

It gives me great pleasure to provide this Preface to the book on *Introduction to Geospatial Artificial Intelligence*, a timely and important contribution to the evolving relationship between geospatial sciences and artificial intelligence. The book arrives at a critical moment in our shared journey toward achieving the 2030 Agenda for Sustainable Development. As recent assessments have shown, progress across many of the Sustainable Development Goals remains uneven, with some targets even regressing. Meeting these challenges requires new ways of thinking, new tools, and deeper collaboration across borders and disciplines.

Geospatial information has long been recognized as indispensable for evidence-based decision-making, for managing natural resources, planning resilient cities, protecting the environment, and responding to crises. Today, with the rapid advances in earth observation, unmanned aerial systems, LiDAR, and other emerging technologies, we are able to collect data at unprecedented scale and detail. Yet data alone is not enough. It is the intelligent integration of geospatial data with innovative methods of processing and analysis that unlocks its true potential. Artificial intelligence offers precisely this: the capacity to transform raw data into decision-ready

knowledge, delivered with the speed and reliability needed to address complex global challenges.

This book makes a valuable contribution by bridging two communities: geospatial professionals and AI researchers. It demonstrates in practical terms how their combined efforts can support governments, institutions, and societies. It aligns with the vision set out in the United Nations Integrated Geospatial Information Framework (UN-IGIF), which emphasises governance, standards, and above all, capacity development and education. Building the skills and human capital to apply geospatial AI responsibly and effectively is as important as building the technical infrastructure itself.

At the General Authority for Survey and Geospatial Information in Saudi Arabia, we are committed to advancing national geospatial capabilities in line with our Vision 2030, while also contributing to the global community through the work of UN-GGIM and related international partnerships. We see in this book both a reflection of those shared ambitions and a practical guide for how they can be realised.

I commend the authors for their efforts in creating a resource that will inform, educate, and inspire. It is my hope that this book will not only serve as an introduction, but also as a catalyst for further innovation, collaboration, and capacity building in the geospatial community.

Dr. Mohammed bin Yahya Al-Sayel
Co-Chair, UNGGIM
President, General Authority for Survey and Geospatial Information (GEOSA)
Kingdom of Saudi Arabia

CONTENTS

List of Figures

List of Tables

1 INTRODUCTION TO GEOSPATIAL AI

Over the years, significant advances have been made in sensing hardware in the area of passive optical images, such as multispectral or hyperspectral imaging, and active sensors, such as Lidar Detection and Ranging (LiDAR) and Synthetic Aperture Radar (SAR). In addition to improvements in sensors, the advancements in platforms have enabled higher frequency and greater flexibility in data acquisition, allowing for more comprehensive and timely analysis of geospatial phenomena. This led to the development of a large heterogeneous multi-scale spatiotemporal dataset that can be used in various applications.

In parallel, over recent years, there has been an enormous advantage in computer vision and machine learning, which have significantly increased the capability to understand and analyze that data. Geospatial Artificial Intelligence (GeoAI) is an interdisciplinary field that applies Artificial Intelligence (AI) to studying and understanding geographic and spatial data. It combines the power of AI and Geospatial Information Systems (GIS) to solve location-based problems. The GeoAI encompasses a range of tasks, including object detection, image and point cloud classification, anomaly detection, semantic segmentation, super-resolution, and multi-resolution data fusion.

AI is the field of computer science that focuses on creating systems capable of performing tasks that typically require some level of human intelligence. AI represents a general field that includes machine learning (ML) and deep learning (DL), but also comprises other approaches that don't involve any form of learning, such as rule-based systems or symbolic AI.

AI was born in the 1950s when Alan Turing proposed the concept of machine intelligence in his paper „Computing Machinery and Intelligence“ [1]. Turing raised the question „Can machines think?” and created the Turing test as a way to evaluate a machine's ability to exhibit human-like intelligence. The term AI was introduced by John McCarthy et al. [2], proposing „that every aspect of learning or any other feature of intelligence can in principle be so

precisely described that a machine can be made to simulate it". Symbolic AI, which is based on programmers manually crafting a comprehensive set of explicit rules for representing knowledge, was the dominant paradigm in artificial intelligence from the 1950s through the 1980s. Therefore, in symbolic AI, humans define and input a set of rules and data for the system to process, and then the system provides results based on those predefined rules.

Although symbolic AI works well for tasks with well-defined rules, such as playing chess, for complex problems such as satellite image classification and object detection, creating and maintaining those rules becomes extremely difficult and time-consuming. These limitations in learning, scalability, and complex data processing led to the development of ML.

Instead of relying on rules written by humans, ML automatically learns those rules by analyzing data.

An ML system includes four main steps: data collection, model training, prediction, and accuracy assessment. The model is fed by a large amount of data. These data consist of:

- Features – input variables or characteristics of data that the model uses to make predictions, such as intensity of reflected electromagnetic radiation in different bands, band ratios, spectral indices, textures, etc, and
- Labels – correct answers or output of models, such as labels for the pixel ("water," "forest," or "urban area").

The data are usually split into two sets: a training dataset (used to train the model) and a testing set (used to evaluate how well the model performs on unseen data). In model training, ML uses algorithms to detect patterns or relationships between features and labels by analyzing the provided data. The comparison between the algorithm prediction and the expected outcome is performed. This information is used as a feedback signal to adjust the way the algorithm works. This iterative adjustment is called *learning*. Once trained, the model makes predictions based on new unseen data. Thus, an ML system is trained instead of being explicitly programmed.

The central problem in ML is learning adequate input data representations. A representation can be defined as a way to encode or transform data to make

it more useful for a specific task. The goal is to transform the input data into a form that makes it easier to predict output, such as classifying land cover types or detecting water bodies. In ML, input data are defined by the operator. This process, in which humans decide which attribute of the data is relevant for a specific task, is called *feature engineering*.

For example, satellite images can be encoded in the RGB format or multispectral or hyperspectral format. The task is to classify pixels in an image into „water" and „vegetation". In the RGB format, the identification of those classes can be challenging due to similar spectral signatures in the visible part of the electromagnetic spectrum. Additional features, such as the Normalized Difference Vegetation Index (NDVI) that combines Red (R) and Near-Infrared (NIR) bands, emphasizing areas with high chlorophyll content; the Normalized Difference Water Index (NDWI) that combines NIR and Short-wave infrared (SWIR), and emphasizes the water, band ratios, or raw bands can be extremely useful. ML algorithms automatically find transformations of input data that turn them into more useful representations and use the percentage of correctly classified pixels as feedback to *learn* an appropriate representation. For example, the ML model might learn that NDVI > 0.5 is a good indicator of vegetation, that pixels with low NIR/R ratio are likely water, or it can combine existing features such as NDVI and Green (G) band into a new representation that better captures patterns in the data.

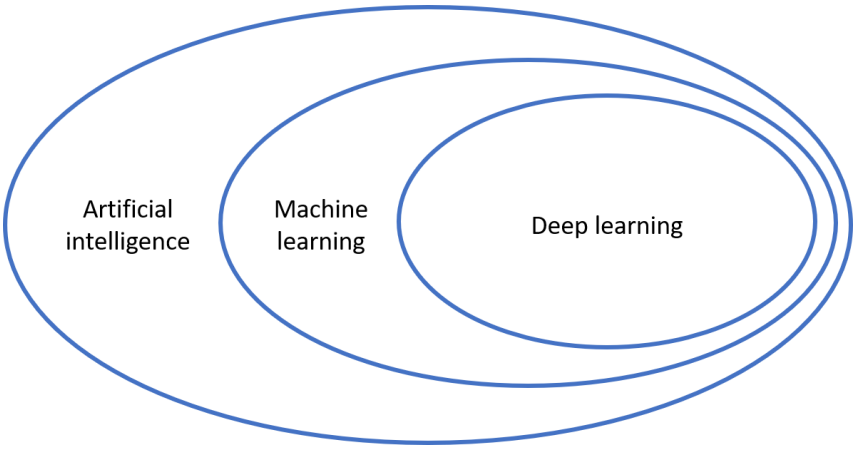


Figure 1 Artificial Intelligence, machine learning, and deep learning

Deep learning (DL) is a subfield of ML that uses many successive *layers* to learn complex representations and patterns from data automatically (**Figure**

1). The „*deep*" in the DL model is defined by the number of layers used in the network. Today, a DL network can consist of hundreds or thousands of hierarchical layers. Unlike traditional ML, DL models use layers to extract and learn the hierarchical representation of the data without explicit feature engineering. This ability enables DL models to uncover complex, hidden patterns that may not be immediately obvious or easily detected by humans, thereby creating new features for classification. In the satellite image classification task, early layers learn simple features such as edges or corners, deeper layers combine the results of previous layers into more complex features like shapes, and the final layer synthesizes all these features to classify the image into predefined classes such as roads or buildings. The comparison between ML and DL is provided in **Table 1**.

Table 1 Comparison between ML and DL

Aspect	**ML**	**DL**
Approach	Statistical techniques	Neural networks with multiple layers
Feature engineering	Manual (hand-crafted feature)	Automatic (learns features from the data)
Data dependency	Small to medium dataset	Large datasets
Task complexity	Low to moderate	High-dimensional and complex
Computational need	Low to moderate	High
Training time	Less time needed	Much more time needed
Accuracy	High accuracy on large datasets	Good results on both small and large datasets

1.1 ALGORITHMS

In recent years, ML algorithms have been used in a broad spectrum of domains. In these applications, each instance in the dataset is represented by a consistent set of features/attributes. For example, in RS, a pixel can be characterized by its reflectance values across different spectral bands such as blue, green, red, and NIR. Additionally, each instance in the dataset can be

associated with a corresponding label, i.e., the correct output for that instance. Each feature can be binary, categorical, or continuous.

ML is categorized into four categories:

1. Unsupervised learning,
2. Supervised learning,
3. Semi-supervised learning, and
4. Reinforcement learning.

Unsupervised learning is based on systems that analyze the patterns in unlabeled data. It provides a better understanding of the correlations, structures, and patterns present in the data. Unsupervised learning is usually used as the initial step before supervised classification.

Supervised learning uses a labeled dataset to train AI algorithms to identify patterns and relationships between input features and outputs. In supervised learning, analytics manually identifies examples of interest, i.e., creates labeled data. Labeled data represents data points with corresponding labels (i.e., correct output). The supervised learning algorithm is trained on labeled data. The goal is to learn to map input data to known labels. During training, algorithms process large datasets to understand potential correlations between input and output variables. Algorithms can apply what they learned on the training set to the unseen data and predict the output values. Generally, it is the most common approach, and almost all deep learning applications belong to this category. Supervised learning is mainly divided into two categories:

- Regression – used when the target variable is continuous and the task is to predict a real value instead of a discrete label. For example, we want to predict Land Surface Temperature using satellite data from Landsat 8 images. The most commonly used algorithms include Linear Regression, Decision Trees for Regression, and Support Vector Regression (SVR), among others.
- Classification – used when the target variable is discrete or categorical. The primary aim is to assign each input to predefined classes.

Traditionally, in geomatics, classification methods are categorized based on the smallest unit of analysis, on pixel-based classification and object-based image analysis (OBIA).

Pixel-based classification observes each pixel in an image as an independent unit. The primary aim is to classify each pixel into one of the predefined classes based only on its spectral characteristics. Spatial context and relationships between neighboring pixels are not considered. Pixel-based classification is straightforward and has been commonly used in tasks such as land use/ land cover classification (where individual pixels are labeled as vegetation, water, or urban areas based on their reflectance values in different spectral bands).

Object-Based Image Analysis (OBIA), on the other hand, groups pixels into meaningful objects or segments before classification. Objects are created through image segmentation before classification using segmentation algorithms such as multiresolution segmentation or watershed analysis. Pixels are grouped based on spatial, spectral, and contextual properties such as shape, texture, and proximity. OBIA is primarily used for the classification of high-resolution imagery, where features like buildings or tree crowns need to be classified as cohesive units rather than isolated pixels.

In GeoAI classification tasks, they can be divided into semantic segmentation, object detection, and instance segmentation, each addressing different aspects of spatial data interpretation. Semantic segmentation (i.e., pixel-based classification) labels each pixel within an image (or point in a 3D point cloud) with a class, providing a complete understanding of the spatial distribution of objects. For instance, a satellite image of an urban area might be segmented into classes such as roads, buildings, vegetation, and water bodies.

Object detection, on the other hand, focuses on both identification and localization of objects of interest within an image using bounding boxes. For example, detecting vehicles in parking lots or identifying individual trees in a forested landscape are everyday tasks in this category. Finally, instance segmentation combines the pixel-level detail of semantic segmentation with the localization of object detection by identifying and delineating each instance of an object separately. A typical use case is segmenting individual buildings in a dense urban environment or distinguishing between

overlapping tree canopies in a forest. These methods have been extensively used in different applications, including but not limited to environmental monitoring, urban planning, agriculture, and disaster management.

Semi-supervised learning represents a branch of ML that combines supervised and unsupervised learning by using both labeled and unlabeled data to train models for regression or classification tasks. It uses a small amount of labeled data alongside a vast amount of unlabeled data to train models. As in supervised learning, the goal is to train algorithms that can accurately predict the output variables based on the input. The difference is that semi-supervised learning uses both labeled and unlabeled data in the training process. This concept is particularly beneficial when a large amount of unlabeled data is available, but labeling it all is too time-consuming and expensive. The semi-supervised learning model uses labeled data to learn a preliminary representation of the problem (such as the number of classes). At the same time, it exploits the structure and distribution of unlabeled data. It is based on several assumptions, including: unlabeled data used in model training must be relative to the task (if a task is to classify land cover classes, the images of cats and dogs will not be helpful), cluster assumption, i.e., the data with similar feature space should belong to the same class. The similarity can be based on different definitions, such as:

- Smoothness assumption – if data points are close in feature space, then they are more likely to share the same labels. For example, if pixels in satellite images have similar reflection values, they likely belong to the same land cover class.
- Low-density assumption - the decision boundary between different classes should lie in low-density regions where few or no data exist.
- Manifold assumption - high-dimensional feature datasets comprise multiple lower-dimensional manifolds on which all points lie, and the data points on the same manifold belong to the same class. For example, a Landsat 8 image of 256 x 256 pixels has 393 216 dimensions. Comparing such a high-dimensional dataset is challenging due to its complexity and computational resources. Moreover, not all features are equally important for the specific task. Following proper dimensionality reduction and feature transformation, similar data points tend to cluster together.

Semi-supervised learning can be classified based on the way in which it incorporates unlabeled data:

- pseudo labeling - model trained on labeled data is used to predict labels for the unlabeled data (pseudo labels). The model can then be retrained iteratively by using real and pseudo labels to improve accuracy. For example, training a Random Forest (RF) model on labeled data, using that model to classify unlabeled pixels and assign pseudo classes where predictions are of high confidence, and then retraining the model using both real and pseudo classes to improve accuracy.
- unsupervised pre-processing - data transformation techniques are used to generate better feature representation or reduce noise in raw data before training an ML model with labeled data. For example, Principal Component Analysis (PCA) can be used to reduce Sentinel-2 spectral bands from 10 to 5 while preserving maximum variance in the cropland classification task.
- Objective function adjustment - modification of the loss function to enforce smoothness. Instead of minimizing only the usual loss function (e.g., cross-entropy for classification), additional terms can be added to promote better generalization (for instance, a consistency loss can be added to cross-entropy to ensure accurate prediction of land cover classes if the brightness condition on the satellite image slightly changes) and reduce overfitting.

Semi-supervised learning has been used for various tasks, including text classification, image classification, and anomaly detection.

Self-supervised learning is an ML technique that uses unsupervised learning for tasks that require supervised learning. Instead of relying on human-annotated labels, self-supervised learning creates output labels directly from input data. In the pre-training phase, the model learns discriminative features by solving pretext tasks before applying them to the actual downstream task (such as classification). Predefined pretext tasks typically involve simple functions that help models understand the structure of key data features, which are used to solve real-world problems, enabling models to learn without human supervision. Self-supervised learning is particularly used in computer vision and natural language processing, but it has also been

successfully implemented for remote sensing tasks. For example, the model is trained on combining different sources of data, such as visible and radar images, improving its understanding of land cover types from diverse perspectives, or the model is given multiple time-step satellite images from the same region, and the task is to predict temporal changes, such as seasonal variation in vegetation.

Reinforcement learning is a type of ML where an agent learns to make decisions by interacting with an environment. Unlike supervised learning, where the model learns from labeled data, in reinforcement learning, an agent learns by trial and error, receiving feedback through rewards and penalties based on its actions. The primary aim is to maximize cumulative reward over time. The agent is any system that can make decisions, such as self-driving cars. The environment encompasses everything an agent interacts with, providing information on its current state. The agent uses that information to determine which action to take. Action changes the state of the environment, and the agent receives a reward. The reward measures how good or bad an action was in achieving a goal. For self-driving vehicles, a reward can be reducing traveling time, remaining on the road, or being in the proper lane, among other objectives. Over time, the agent learns which actions lead to the most rewards. According to that agent, adjust strategy to favor actions that yield higher rewards, altering its decision-making process. The process continues until the agent reaches its goal. In remote sensing, reinforcement learning can be beneficial for tasks that involve real-time data analysis and sequential decision-making.

1.2 Historical development of AI

In recent years, AI and deep learning have made remarkable achievements in a wide range of applications. However, the development of AI started several decades ago, stretching back to the 40s. During that period, AI evolved from theoretical concepts to applications in different fields. There are several milestones in the development of AI:

- 1943 - Walter Pitts and Warren McCulloch [3] presented the first mathematical model of neural networks in their paper “A logical calculus of the ideas immanent in nervous activity”

- 1950 - Alan Turing published a landmark paper, "Computing Machinery and Intelligence" [1]. This paper introduced the Turing test, designed to determine whether a computer can mimic human-like intelligence. The Turing test became a central concept in AI, serving as a way to test a machine's ability to exhibit intelligent behavior equal to that of humans.
- 1952 - Arthur Samuel developed a program for playing checkers at a champion level
- 1955 - the term "artificial intelligence" was introduced in a workshop proposal, "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence," submitted by John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon [2]. During this workshop, a group of prominent scientists has gathered to establish the fields of AI and ML research.
- 1957 - Frank Rosenblatt designed the first two-layer computer neural network that enabled pattern recognition, called the Perceptron [4].
- 1957 - The launch of Sputnik 1, the first artificial Earth satellite, marked the beginning of the space era.
- 1963 - Roger Tomlinson introduced the term Geographic Information System (GIS).
- 1966 - Thomas Cover and Peter Hart introduced the k Nearest Neighbor algorithm in their paper "Nearest Neighbor Pattern classification" [5].
- 1969 - The Harvard Lab for Computer Graphics created the first vector-based GIS called SYMAP, which allowed for the visualization of spatial data.
- 1972 - NASA launched Landsat 1. The Landsat mission provides continuous multispectral imagery of the Earth's surface
- 1978 - First Global Navigation Satellite System satellite, Navstar I, was launched. It achieved full global coverage in 1995.
- 1986 - Rumelhart, Hinton, and Williams published a paper "Learning representations by back-propagating errors" [6], in which they used backpropagation for training neural networks and improving their ability to learn complex patterns. This research is a fundamental breakthrough in modern deep learning. It builds on the 1969 research

of Bryson and Ho [7] that first introduced backpropagation for overcoming the limitation in training multilayer networks.

- 1986 - The SPOT satellite was launched, providing higher resolution images, allowing better mapping and monitoring of environmental changes.
- 1988 - Judea Pearl published "Probabilistic Reasoning in Intelligent Systems," [8] introducing Bayesian networks as a framework for probabilistic reasoning and decision making under uncertainty.
- 1989 - Yann LeCun and his team successfully applied the backpropagation algorithm to Convolution Neural Networks to recognize handwritten zip code images [9]. This is one of the first applications of CNNs demonstrating the potential of AI for image recognition tasks.
- 1995 - Vladimir Vapnik and Corinna Cortes introduced the Support Vector Machine algorithm [10], widely used in ML for linear and nonlinear classification tasks.
- 1995 - one of the first applications of AI for processing geospatial data, such as decision tree-based land cover classification or ANN-based spatial interaction modeling [10].
- 1997 - Sepp Hochreiter and Jürgen Schmidhuber introduced Long Short-Term Memory (LSTM) [11], a type of Recurrent Neural Network (RNN) designed to capture long-term dependencies in data effectively.
- 1997 - IBM chess computer, Deep Blue, beats world chess champion Garry Kasparov in a six-game match. This victory demonstrated that computers can outperform human intelligence in strategic games.
- 1999 - IKONOS, the first high-resolution satellite with a panhromatic spatial resolution of 0.82 m and a multispectral resolution of 3.2 m, was launched.
- 2007 - Fei-Gei Li and her team launch the most massive and comprehensive databases of annotated images. ImageNet [12] provides millions of labeled images across thousands of categories to support the development of visual object recognition software.
- 2009 - Rahat Raina, Anand Madhavan, and Andrew Ng publish "Large-scale Deep Unsupervised Learning using Graphics

Processors," [13] demonstrating the GPU's superior computational power over traditional multi-core CPUs for deep learning tasks.

- 2010 - Google Earth Engine (GEE) [14], a cloud-based service for geospatial processing, was launched. It integrates petabytes of geospatial data, free planetary-scale processing power, and AI technology. GEE has been a game-changer for GeoAI development, impacting both research and practical application.
- 2011 - Apple launches Siri, a virtual assistant integrated into iOS, allowing users to interact with their devices through voice commands.
- 2012 - Geoffrey Hinton and his team designed the AlexNet [15], a CNN network that achieves a 16% error rate in the ImageNet Large Scale Visual Recognition Challenge. This research shows that CNNs can outperform traditional image classification methods.
- 2014 - Ian Goodfellow and his collaborators in their groundbreaking paper titled "Generative Adversarial Nets" [16] introduced GAN frameworks that teach AI how to generate realistic data through adversarial training between two networks (generator and discriminator).
- 2015 - Kaiming He and his team introduced ResNet [17], a deep CNN architecture that introduced the idea of skip connection, allowing training of much deeper networks without the problem of vanishing gradients.
- 2015 - Olaf Ronnenberger, Philipp Fischer, and Thomas Brox introduced U-Net in their paper titled "U-Net: Convolutional Networks for Biomedical Image Segmentation" [18]. U-Net represents a turning point for deep learning-based semantic segmentation, especially for high-resolution images and in tasks requiring spatial accuracy. In addition, UNet's ability to work effectively with a limited labeled dataset is a significant advantage in GeoAI.
- 2017 - Vaswani et al. [19] introduced the Transformer, the first architecture based entirely on self-attention mechanism to enhance feature representation, achieving substantially faster training compared with recurrent or convolution models.
- 2020 - OpenAI introduces GPT-3, a groundbreaking natural language processing algorithm able to generate human-like text, engage in conversations, write code, translate languages, etc.

- 2021 - DeepMind (a subsidiary of Alphabet) uses its neural network for accurate prediction of 3D structures of proteins for amino acid sequences with unprecedented accuracy. DeepMind's breakthrough demonstrates that deep learning has the potential to accelerate drug development and disease research dramatically.
- 2021 - Tesla launched the Full Self-Driving Beta that uses deep learning to navigate complex driving scenarios.
- 2021-2023 - OpenAI launches DALL-E, followed by DALL-E 2 and DALL-E 3, generative AI models capable of generating highly detailed images from textual descriptions.
- 2025 - DeepSeek realizes DeepSeek-V3, a free AI-powered chatbot aiming to achieve Artificial General Intelligence. They adopt Multi-head Latent Attention (MLA) and DeepSeekMoE architectures to achieve efficient inference and cost-effective training, spending only $5.6 million on computing power for development. The development of AI models for a fraction of what other companies have been spending caused shock waves throughout the GPU market.

1.3 Limitations

In recent years, AI has revolutionized multiple industries and transformed how humans interact with technology. Despite its impressive capabilities, AI has significant limitations that need to be considered. Those concerns can be divided into several areas, including data, ethical concerns, and model interpretability.

1.3.1 Data limitations

AI, especially DL, relies on large amounts of data to make accurate decisions and predictions. This is the most obvious limitation. The rapid advancement of DL, especially during the 2010s, has been largely enabled by the availability of vast amounts of data. Despite the progress in storage hardware, the rise of the internet has made it possible to collect (social media, mobile devices, Internet of Things, public datasets), store, process, and distribute (cloud computing, big data technologies) a large dataset, which was a game-changer. Data limitations can manifest as a lack of data and a lack of quality data.

Lack of data. As mentioned earlier, many AI algorithms require vast amounts of labeled data, which can be expensive and time-consuming to create, especially in specific domains. The larger the architecture, the more data is needed to produce viable results. Although techniques such as fine-tuning and data augmentation are practical, having a larger dataset is always a preferred solution.

Data quality and bias. AI models are only as good as the data they are trained on. If the training data is biased or unrepresentative of the real world, the model can inherit those biases and lead to inexact outcomes. For example, in land use classification, if certain geographical areas are underrepresented in the training dataset (for example, underdeveloped regions), AI models might perform poorly in those areas. The most ideal way to mitigate such a risk is by collecting data from broad and diverse geographical areas, as heterogeneous datasets limit exposure to bias and result in higher accuracy.

1.3.2 Ethical limitations

AI has had a profound impact on the world, but its exact capabilities and limitations are not clearly defined. In recent years, AI has been integrated into the critical decision-making process, raising several ethical concerns. One of the key concerns, particularly in information retrieval, is the tendency to define a singular „truth“ rather than presenting different perspectives. Consider, for example, a search for a particular term on the internet. Earlier search engines would provide millions of different pages, allowing you to analyze and synthesize information from different sources to draw your own conclusions. However, AI tools like ChatGPT often provide just one or two definitive answers, streamlining information but also potentially narrowing the range of perspective and reducing the critical engagement with multiple viewpoints. Moreover, there is no guarantee that the provided conclusions are correct.

As AI becomes more autonomous, determining who is responsible for its decisions becomes increasingly complex. One of the most discussed examples is autonomous driving. If an AI driving vehicle causes an accident, who is responsible: the manufacturer, the software developers, the user, or the AI itself?

Also, AI-powered systems such as face recognition or video monitoring have raised concerns regarding security, surveillance, and invasion of privacy. Although these technologies can be helpful for security purposes, they also raise discussion about transparency, consent, and the potential for abuse by governments or corporations. Additionally, in combination with an increase in sensor capabilities, such as high-resolution satellite or UAV imagery, it can expose sensitive information about personal activities or locations.

Moreover, AI's capability to automatically perform tasks that are traditionally done by humans raises concerns about its impact on the economy, income, and job security. The International Monetary Fund reported [20] that almost 40 percent of global jobs are exposed to AI (up to 60 percent in developed economies). The report states that half of those may be negatively impacted, while the rest could have gains in productivity [20].

2 GEOSPATIAL DATA

Geospatial data refers to information related to locations on the Earth's surface. Geodata (another name that can be used instead of geospatial data) describe objects, events, and other real-world phenomena within a specific geographical area, typically identified by coordinates, addresses, zip codes, or names. They combine location information (identifying where something exists or takes place), descriptive attributes (what is present at that location), and often include temporal information (when it occurs). They are essential for more efficient decision-making across multiple applications and have significant economic importance.

Geospatial data can be static, such as the location of an event, or dynamic, like the movement of vehicles or the spread of an infectious disease, where the distinction between static and dynamic data depends strongly on the time scale considered. Geospatial data are collected from many diverse sources in varying formats. They can include various information such as census data, satellite imagery, weather data, smartphone data, as well as data from social networks, IoT sensors, etc. Moreover, integrating geospatial data with traditional business data can be especially valuable, allowing visualization in the form of maps, graphs, cartograms, or virtual globes and providing a deeper understanding of events, monitoring of changes over time, and recognizing the patterns and insights that might be overlooked in a large data spreadsheet.

With advancements in technology, its application will continue to grow, contributing to smarter cities, sustainable environment, economic growth, and improved quality of life globally.

There are two primary data models used to represent geospatial data. These models enable real-world geographic objects to be digitally stored in a database and visualized on maps or computer screens. The vector model is used for objects with clearly defined boundaries, for which it is precisely known where they begin and where they end. It uses points, lines, and polygons to represent features such as buildings, roads, trees, rivers, etc. A raster model is used for continuous phenomena. It represents data as a grid of pixels. Each pixel contains a unique value corresponding to a specific attribute such as temperature or elevation. The raster model is commonly

used for remote sensing, digital terrain models, and land cover classification, among other applications.

All geospatial data have four key characteristics: location, scale, accuracy and resolution. Each geospatial dataset is associated to a location defined within a Coordinate Reference System (CRS), which provides a numerical framework to represent points on the Earth's surface consistently. The scale refers to the ratio between a distance on the map and the corresponding distance on the ground, typically measured along representative lines, making it effectively an average value. Accuracy represents how closely information in the dataset matches the real world. For instance, positional accuracy represents how close geospatial objects (features) are to their real-world locations. Resolution refers to the smallest distinguishable unit in the data (e.g., pixel size in a raster image, or minimum distance between points in a vector dataset). Higher resolution means finer detail. The three concepts: scale, resolution and positional accuracy are closely related.

Other relevant characteristics of the geospatial data are consistency and completeness. Consistency refers to whether the characteristics of geospatial objects in the dataset match those in the real world. For example, does a building in the dataset represent a building in the real world? Completeness refers to the extent to which instances of features are included in the dataset. For example, are all buildings in a city presented in the dataset?

Metadata is often defined as "data about data," and it describes properties, origin, ownership, quality, history, and other valuable properties. The primary purpose of metadata is to enable search and evaluation of geospatial data, to provide information on how to access and use resources effectively.

Accordingly, metadata can be classified into three main types:

- descriptive metadata - provides helpful information for discovering and identifying data resources. It describes a resource's what (name of dataset and description), when (when dataset is created and cycles of update if defined), where (geospatial extent of the dataset based on geographical coordinates, administrative units or geographical names), why (why data are created and), who (data sources, provider and leading target group), and how (how dataset is created and how to access to the,);

- evaluation metadata - includes all necessary data needed to determine if selected data meet all user needs, to assess their characteristic and quality;
- Structure metadata - includes information necessary for access, transfer, loading, implementation, and use of data in the final user application. This type of metadata often contains details about data dictionaries, data organization, i.e., schemas, spatial references, geometric characteristics, and other information relevant for people and machines to use geospatial data properly.

Metadata is particularly important because, despite the increasing availability of geospatial data from diverse sources and the relative ease of locating it, researchers often face challenges in finding and assessing data due to limited metadata. Recognizing the critical role of metadata, several standards (ISO 19115-1, ISO 19115-2, ISO 19115-3) have been established to define the essential elements for describing geospatial datasets and related resources.

3 REMOTE SENSING

Remote sensing is the science of acquiring information about Earth's surface without being in contact with it. It is done by recording the reflected electromagnetic energy and using non-imaging sensors. The elements of remorse sensing (**Figure 2**) are:

- Energy source (A) - the first element of remote sensing that illuminates or provides electromagnetic energy to the target of interest.
- Interaction between radiance and atmosphere (B) - as the electromagnetic energy travels from the source to the target, it passes through the atmosphere and interacts with it. This interaction is repeated a second time as the reflected energy travels back from the target to the sensor. Atmospheric correction is used to minimize this effect and improve image quality.
- Interaction with target (C) - once the energy makes its way to the target, it interacts with it depending on the properties of both the target and the radiance.
- Recording of reflected energy by the sensor (D) - after the target reflects energy, it travels back to the sensor that collects and records the electromagnetic radiation.
- Processing (E) - the energy recorded by the sensor has to be transmitted to a receiving and processing station, where the data are processed into images.
- Interpretation and analysis (F) - the image is interpreted, visually or digitally, to extract useful information about the target.
- Application (G) - the final element of remote sensing is the application of extracted information for better understanding the target, revealing some new aspects, monitoring changes over time, and supporting decision-making processes.

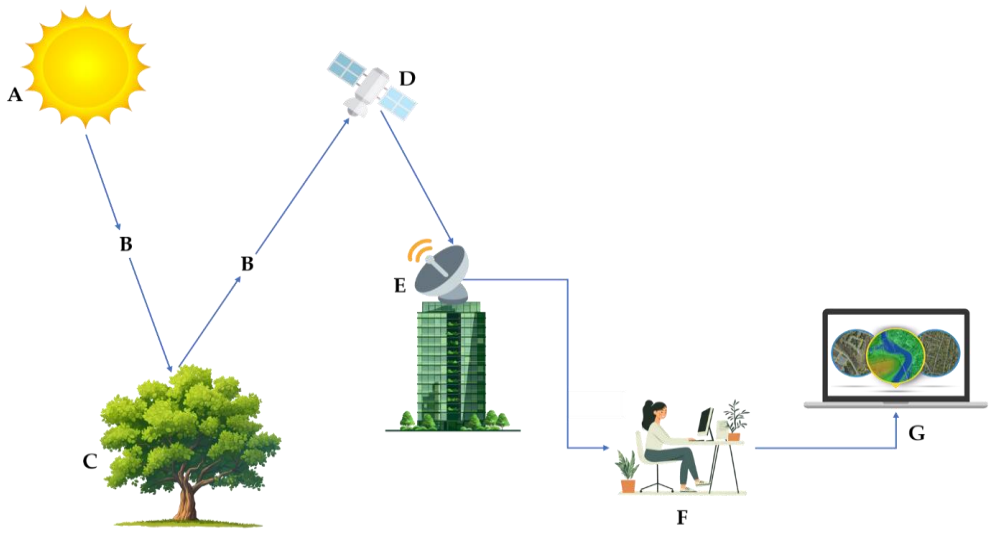


Figure 2 Elements of remote sensing

3.1 Energy source

Every object on the Earth's surface, due to solar radiation, has energy of a specific frequency and wavelength, and it emits energy from the electromagnetic spectrum. Every object is composed of charged particles, such as protons and electrons, which generate electric fields. These electric fields influence other charged particles within their range. When charged particles move, they create an electric current, which in turn produces a magnetic field. The interaction between changing electric and magnetic fields generates electromagnetic (EM) radiation, which can be described as either a wave (electromagnetic waves) or as discrete packets of energy (photons) in the quantum model of radiation.

The electrical field varies in magnitude in a direction perpendicular to the direction in which radiation is traveling. In contrast, the magnetic field is oriented at a right angle to the electrical field. It can be noted that generated waves are in phase, i.e., when the electric field is maximum, the magnetic energy is maximum. Two main characteristics of EM radiation are:

- wavelength - is the length of one wave cycle, and it is defined as the distance between successive wave crests. Wavelength (λ) is usually

measured in nanometers (nm, 10^{-9}meters), micrometers (μm, 10^{-6}meters), or centimeters (cm, 10^{-2}meters).

- frequency - number of cycles of a wave passing a fixed point per unit of time. Frequency is usually represented by a Greek letter ν and it is measured in hertz (Hz), equal to one cycle per second.

The following formula relates wavelength and frequency:

$$c = \lambda\nu$$

where c is the speed of light, λ is wavelength and ν is frequency. It can be concluded that frequency and wavelength are inversely related to each other, i.e. the shorter the wavelength, the higher the frequency and reverse.

The EM spectrum represents the collection of all wavelengths of EM radiation, which can be categorized into regions based on wavelength and frequency. The EM spectrum consists of gamma, X, ultraviolet, visible, infrared, microwaves, and radio waves. The range of gamma and X waves includes wavelengths less than 0.01 μm, and they are unusable in remote sensing due to their low penetration through the atmosphere. The ultraviolet range comprises wavelengths between 0.01 μm and 0.04 μm. This region has the shortest wavelength that can be practically used in remote sensing. Certain materials on Earth's surface, primarily rocks and minerals, fluoresce or emit visible light when illuminated by UV radiation. However, this range is highly absorbed in the upper layer of the atmosphere by ozone, so its application is limited.

The range of the visible part of the EM spectrum is defined by the sensitivity of the human eye and includes wavelengths between *0.4* μm to *0.7* μm. Humans perceive the combination of all visible wavelength radiation as white light because the cones in our retinas respond to the full spectrum of visible colors simultaneously, but according to the wavelengths in the visible spectrum, different colors are distinguished from violet (*0.4* μm - *0.44* μm), through blue (*0.45* μm - *0.5* μm), green (*0.5* μm - *0.57* μm), yellow (*0.57* μm - *0.59* μm), orange (*0.59* μm - *0.62* μm) to the red (*0.62* μm - *0.7* μm) with the largest wavelength. Although its portion is small compared to the rest of the spectrum, it is one of the most used in remote sensing due to its reflection properties.

The infrared (IR) portion of the spectrum includes wavelengths between *0.7* μm and *100* μm. It can be divided into two categories: reflective IR and emitted (thermal part). Radiation in reflective IR (*0.7* μm - 3.0 μm) region is used in remote sensing similarly to radiation in the visible part. The thermal IR covers a range from approximately 3.0 μm to 100.0 μm and represents the radiation that is emitted from the Earth's surface in the form of heat. Both parts of the IR spectrum are essential for applications in remote sensing. Thermal IR is used to monitor temperature variations, making it useful for tracking water temperature, land surface temperature, and heat emission. Reflective IR has been widely used for vegetation monitoring. Also, they are frequently used in geology, geomorphology, agriculture, military surveillance, and environmental monitoring, aiding in tasks such as mineral exploration, terrain analysis, crop health assessment, and target detection in defense applications.

Microwave range includes waves with a wavelength from 1000 μm to 1 m. These are the longest waves that are applied in remote sensing, and their ability to penetrate the atmosphere is higher than that of the visible spectrum. Microwaves can penetrate clouds and haze, making them useful for all-weather and day-night monitoring. They can be emitted from artificial (such as Synthetic Aperture Radar (SAR) or radar altimetry) or natural sources.

Based on the source of energy, remote sensing can be categorized as passive and active. Sun Remote sensing that measures the naturally available energy is called a passive sensor. The sun is often used as a source of energy for remote sensing. Satellite missions such as Landsat and Sentinel-2 are examples of passive sensors. Passive sensors can only detect energy when it is naturally available. As a result, it can only collect data during the day. Additionally, their shorter wavelengths are unable to penetrate clouds, haze, and other atmospheric conditions, which limits data acquisition and causes data gaps.

On the other hand, sensors that provide their own energy source are active. Aster and Sentinel-1 are examples of this sensor type. Active sensors emit the energy toward the target of interest and measure the amount of reflected energy. The main advantage of an active sensor is the possibility to collect data in all weather conditions and at all times of the day or season.

Passive sensors are more cost-effective and offer a wide set of observables, with high resolution. In contrast, active sensors enable observation in all weather conditions but require moderately complex and sophisticated processing techniques. The choice between active and passive sensors depends on specific application requirements, available budget, and environmental conditions.

3.2 Interaction with atmosphere

EM radiation travels from the source through the atmosphere, interacts with the target, and the reflected radiation travels again through the atmosphere before reaching a sensor. Unlike the vacuum of space, the atmosphere contains gases and particles that interact with and modify the radiation passing through it. The influence of the atmosphere on radiation is a function of atmosphere permeability, i.e., the physical characteristics of gases and the number and size of particles, and is known as the atmospheric effect. The permeability coefficient shows how much of the radiation that reaches the upper atmosphere will reach the Earth's surface. The permeability coefficient varies both spatially and with height. The mechanisms of scattering and absorption cause the atmospheric effect.

Scattering occurs when EM waves interact with particles in the atmosphere, causing them to be redirected from their original path. The amount of scattering depends on several factors, including the wavelength of radiation, the abundance of particles and gases, and the distance that radiation travels through the atmosphere. There are three main types of scattering:

- Rayleigh scattering - if the size of particles is smaller than the EM wavelength, they can result in diffuse and elastic scattering (change in wavelength) depending on size and dielectricity. It is inversely proportional to the one-quarter power, meaning that the scattering of blue light is much higher than that of EM with longer wavelengths. Rayleigh scattering is dominant in the upper atmosphere, and because of it, the sky appears blue during the day. During sunrise or sunset, the light has to travel farther through the atmosphere, and scattering of shorter wavelengths is more complete, which increases the percentage of longer wavelengths, i.e., red sky color

- Mie scattering - if particle size is about the same as wavelength. Dust, pollen, smoke, and water vapor most often cause this scattering, and it tends to affect longer wavelengths. Mie scattering occurs in the lower atmosphere where the presence of these particles is higher.
- Non-selective scattering occurs when particles are much larger than the wavelength. It scatters all wavelengths equally, causing the white color of clouds and fog (blue + green + red = white)

Scattering can significantly reduce the amount of information collected by remote sensing.

Absorption is the process by which the energy of EM is transformed into other types of energy. Ozone, carbon dioxide, and water vapor are three leading causes of absorption. Ozone absorbs the ultraviolet range. Carbon dioxide is also called a greenhouse gas. Absorb the long IR region (area associated with thermal heating), trapping this heat in the atmosphere. Water vapor absorbs a large portion of IR and a short microwave. The combined effects of different atmospheric layers on absorption can make certain regions of the atmosphere to become impenetrable in certain parts of the spectrum. Consequently, there will be no registered energy in remote sensing, which reduces available information and limits the application.

3.3 Interaction with target

EM radiation that is not scattered or absorbed by the atmosphere can reach the target and interact with it. There are three forms of interaction: absorption, transmission, and reflection. The form of interaction is the function of the radiation wavelength, the properties (physical and chemical characteristics), and the conditions of the target. Absorption occurs when radiation is absorbed, and transmission occurs when radiation is passed through an object. Once transmitted, energy can be absorbed or reflected. The reflection occurs when energy "bounces" off the target and is redirected. In remote sensing, the amount of reflected energy is measured. There are two main types of reflection: specular and diffuse.

If the target surface is smooth, radiation is primarily reflected in a single direction (specular reflection). On the other hand, if the target surface is rough, the radiation will be reflected almost uniformly in all directions

(diffuse reflection). Most of the targets on Earth's surface are between those two phenomena. The type of reflection is the function of surface roughness in comparison to the wavelength. If the wavelength of incoming radiation is much smaller than the surface variation or size of the particles that it is made from, diffuse reflection will be dominant. For example, fine-grained sand on a beach appears relatively smooth to long-wavelength microwaves, which can penetrate the surface and interact with larger features. However, this same sand may appear rougher in the visible spectrum due to the smaller scale of surface irregularities compared to the wavelength of visible light.

The target of interest can be detected and analyzed based on its spectral characteristics. Reflectance describes the interaction of EM radiation with an object of interest, forming a unique spectral signature which is used for target identification in remote sensing. Spectral signatures represent the average value of reflected energy of an object within a specific part of the EM spectrum. A graph showing a spectral signature is called a spectral curve.

Reflectance can be calculated by using the following expression:

$$\rho\ [\%] = \frac{L_r}{L_i} \cdot\ 100$$

where ρ Represents reflection, usually expressed as a percentage, L_r reflected radiance measured by the sensor, L_i Incident radiance refers to the energy coming from the sun or another source.

Reflectance is an inherent property of an object. This property allows for distinguishing between objects based on their spectral reflectance characteristics, independent of the amount of incoming energy when normalized or calibrated. Different materials have unique spectral signatures, while similar objects exhibit variations in reflectance only if their physical and/or chemical characteristics change (e.g., vegetation stress, soil moisture, etc.).

Spectral characteristics of vegetation are influenced by factors of leaves, including orientation and structure. The amount of reflectance for specific wavelengths is influenced by pigment and leaf thickness, composition (cell structure), and the amount of water in leaf tissue. In the visible part of the spectrum, the reflectance of blue and red wavelengths is relatively small

because these components are absorbed during photosynthesis. In contrast, green wavelengths are reflected due to the presence of chlorophyll. Leaves appear greenest in the summer, when chlorophyll concentration is maximum. During the autumn, the amount of chlorophyll decreased, there is less absorption and proportionally more reflectance in the red part of the spectrum, so leaves appear as yellow or red (yellow is a combination of red and green wavelength). The inner structure of healthy leaves represents a diffuse reflector in the NIR part of the spectrum. This is because under the top surface of the leaf (epidermis) (**Figure 3**), there are primarily two layers of cells; the top one is the palisade parenchyma consisting of elongated cells arranged tightly in a vertical orientation. Those cells absorb blue and red light to create chlorophyll and power photosynthesis. The lower level is the spongy parenchyma (**Figure 4**), consisting of irregularly shaped cells with numerous air spaces between them, allowing for the circulation of gases. The NIR energy is not affected by these pigments and almost completely penetrates the palisade parenchyma. When it reaches the spongy parenchyma, the presence of air spaces causes the refraction of the NIR energy in various directions. This results in approximately half the energy exiting the leaf from the lower epidermis and the other half from the top epidermis, towards the sky. Therefore, the reflection is the highest in the NIR part and depends on the degree of leaf development, while the SWIR range depends on the amount of water in the leaf tissue. This region has been widely used for distinguishing different species, but also for monitoring the health status of vegetation, since vegetation that is stressed shows higher reflectivity in the SWIR portion of the spectrum, and healthy vegetation is detected in the NIR.

Water highly absorbs EM radiation. The most essential characteristics are reflection in the visible and a small portion of the NIR region, while almost all radiation with wavelengths longer than 1.2 μm is absorbed. Due to that, detection and separation of water bodies is easiest in this region. Therefore, water absorbs longer wavelengths more than shorter wavelengths. Consequently, water usually appears as blue or blue-green.

However, changes in the water physicochemical characteristics can significantly change spectral properties. Clear water reflects most of the radiance with wavelengths shorter than $0.6\ \mu m$. However, if the turbidity of

water changes (due to increased concentration of suspended organic and inorganic matter), transmission and, therefore, reflection drastically change.

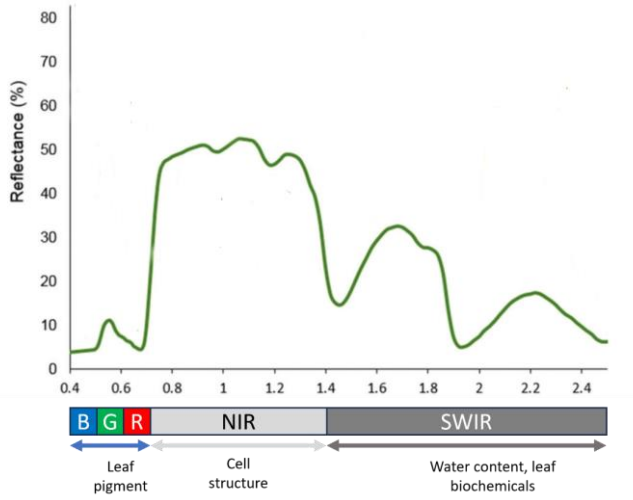


Figure 3 Spectral signature of vegetation

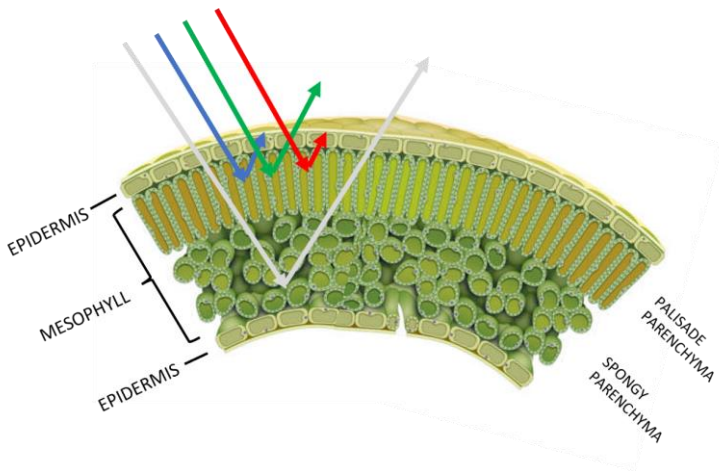


Figure 4 Structure of a healthy leaf

As a result, turbid waters have much higher reflectance in the visible spectrum compared with clean water.

Reflection of the water also changes with a change in chlorophyll concentration. An increase in chlorophyll concentration will increase absorption in the blue part of the spectrum and increase reflectance in the

green part of the spectrum. Based on this, it is obvious that depending on the object complexity, the absorption, transmittance, and reflection mechanisms can completely change.

By comparing the spectral curves of different objects, we can distinguish them in ways that would not be possible using a single wavelength. For example, vegetation and water have similar spectral signatures in the visible spectrum but they are completely different in NIR. Spectral curves can vary significantly even for the same object, and can vary in space and time. Understanding which part of the spectrum needs to be analyzed for a specific target of interest, as well as which factors influence the spectral signature, is crucial for accurate interpretation of remote sensing data.

3.4 Spectral indices

While the interpretation of visible bands is relatively straightforward, data collected from a non-visible part of the spectrum or its combination with visible bands must be processed and analyzed using mathematical transformations, indices, or predefined intervals to be interpretable. Spectral indices have been widely used in remote sensing by enhancing specific features in satellite imagery, making it easier to analyze and interpret different land cover types, vegetation health, water bodies, etc. These indices are derived by mathematically combining reflectance values from multiple spectral bands. The main objective of using the spectral index is to improve the detection and differentiation of specific features that might be difficult to distinguish using raw reflectance. The application of spectral signatures enables large-scale and cost-effective environmental monitoring. By emphasizing particular spectral properties, indices can be applied in:

- Vegetation monitoring - assessing stress conditions, biomass, plant health, phenology, etc. Indices such as Normalized Vegetation Difference Index (NDVI), Soil Adjusted Vegetation Index (SAVI), Enhanced Vegetation Index (EVI), Fraction of absorbed photosynthetically active radiation (fPAR), and Normalized Difference Moisture Index (NDMI) are used.

- Water body analysis - identification of water bodies, monitoring of changes in aquatic environment (Normalized Difference Water Index - NDWI).
- Forestry - Estimating biomass, deforestation monitoring, wildfire impact assessment (NDVI, SAVI, Normalized Burning Ratio -NBR)

The simple form of indices is the ratio between two spectral bands. In contrast, normalized difference indices enhance the contrast between two spectral bands and reduce environmental effects (difference in slope, aspect, shadows, etc.), and are among the most widely used. Vegetation indices are quantitative measures that operate by contrasting chlorophyll pigment absorption in R against the high reflectance of leaf mesophyll in NIR.

The most often used vegetation index is NDVI. It is widely used in vegetation identification, monitoring of phenology, and vegetation stress. NDVI is calculated using the following expression:

$$NDVI = \frac{NIR - RED}{NIR + RED}$$

NDVI always ranges from -1 to 1. Pixels with values close to 1 indicate a high vegetation density, while values below 0 suggest the absence of vegetation. Healthy vegetation has a high NDVI as it strongly absorbs *R* and reflects *NIR*. In contrast, unhealthy and sparse vegetation increases reflection in visible and decreases reflection in *NIR* reflectance, resulting in a lower NDVI value. However, NDVI is sensitive to soil brightness, soil color, atmospheric conditions, cloud cover, cloud shadow, and leaf canopy shadow, necessitating proper remote sensing calibration (radiometric correction, atmospheric correction, etc.) to improve accuracy.

To reduce NDVI's sensitivity to soil background reflectance, Huete [21] established SAVI, which can be expressed as follows:

$$SAVI = \frac{(NIR - R)}{(NIR + R + L)}(1 + L)$$

where L is a soil conditioning factor, varying depending on the amount of green vegetation cover. It can have a value from 0 to 1. In an area with no green vegetation cover, L will be equal to 1, in an area with moderate green vegetation cover, $L=0.5$ (commonly used as a default value), and in an area

with high vegetation cover, L is close to 0, showing that the soil background does not affect the extraction of vegetation information. SAVI improves vegetation detection in mixed land cover types and is more reliable than NDVI in areas with low vegetation density. However, the L parameter must be chosen carefully based on vegetation cover, which requires prior knowledge.

EVI [22] is similar to NDVI but corrects for atmospheric conditions and soil background effects. It is beneficial in high biomass regions, where NDVI may oversaturate or hit the maximum value. The formula for EVI is:

$$EVI = \frac{G \cdot (NIR - R)}{(NIR + C_1 \cdot R - C_2 \cdot B + L)}$$

where G is the gain factor, L is the canopy background adjustment, C_1, C_2 coefficients represents atmospheric resistance terms and uses B i.e. blue band to correct aerosol influences in the R band. It is especially useful for monitoring forest and high biomass environments where NDVI tends to reach maximum, making it difficult to detect and monitor changes.

NDWI [23] is one of the most commonly used indices in water body mapping and management. It enhances water features while minimizing the influence of vegetation and soil. The standard NDWI formula is:

$$NDWI = \frac{G - NIR}{G + NIR}$$

However, it is sensitive to atmospheric conditions, particularly aerosols and cloud presence.

To address this issue, Xu [24] proposed the Modified Normalized Difference Water Index (MNDWI), which replaces *NIR* with *SWIR* to improve water body extraction:

$$MNDWI = \frac{G - SWIR}{G + SWIR}$$

Table 2 Interpretation of NDVI, SAVI, EVI, NDWI and MNDWI value

Index	**> 0.5**	**0.2 to 0.5**	**0 to 0.2**	**0 to -0.2**	**-0.2 >**
NDVI	Dense, healthy vegetation (forests, peak-season crops)	Moderate vegetation (grasslands, shrubs, cultivated crops)	Sparse or stressed vegetation, dry grass	Barren land, degraded soil, some built-up areas	Water, snow, ice, highly developed urban areas (concrete, asphalt)
SAVI	Healthy vegetation, soil-adjusted	Moderate vegetation, some soil impact	Sparse vegetation, exposed soil influence	Degraded soil, barren land, desert regions	Urban areas, highly reflective surfaces, snow, water
EVI	Healthy green vegetation (croplands, pastures, well-irrigated fields)	Moderate vegetation (grasslands, shrubs, semi-arid vegetation)	Sparse or stressed vegetation, bare land	Degraded land, semi-arid areas, some built-up zones	Urban infrastructure, snow, ice, water

NDWI	Deep open water bodies (lakes, reservoirs, oceans)	Wetlands, saturated vegetation, high moisture	Vegetation with moderate water content, inundation, shadows	Dry vegetation, bare soil, or built-up areas, moderate drought	Urban, barren land, or non-vegetated areas, drought
MNDWI	Open water, wetlands (better than NDWI in urban areas)	Shallow water, pools, irrigated fields	Moist soil or vegetation with high water content, inundation, humidity	Dry soil, urban areas, or vegetation with low water content	Highly urbanized zones, industrial areas, deserts, barren landscapes

MNDWI is widely used for urban and flood mapping, where NDWI may misclassify features. The interpretation of the index value is shown in **Table 2**.

NBR is used to detect burned areas and assess fire severity by leveraging the contrast between *NIR* and *SWIR* reflectance. Healthy vegetation has high *NIR* reflectance and low *SWIR* reflectance. On another hand, burned areas have low *NIR* reflectance but high *SWIR* reflectance, making NBR an effective post-fire assessment tool:

$$NBR = \frac{NIR - SWIR}{NIR + SWIR}$$

To measure fire impact and assess burn severity, the difference between pre-fire and post-fire NBR (*Δ*NBR) has been used:

$$\Delta NBR = NBR_{pre-fire} - NBR_{post-fire}$$

NBR has been widely used for post-fire monitoring, wildfire management, and vegetation recovery studies.

*Δ*NBR values can vary from case to case, so interpretation in specific study areas should be done through field assessment. However, the United States Geological Survey (USGS) proposed a threshold to interpret the burn severity [25], which can be seen in **Error! Reference source not found.**.

Table 3 Burn severity labels based on the ΔNBR, proposed by USGS [25]

Severity level	**NBR**
Enhanced regrowth, high (post-fire)	< -0.251
Enhanced regrowth, low (post-fire)	-0.25 to 0.10
Unburned	-0.10 to 0.10
Low severity	0.10 to 0.27
Moderate-low severity	0.27 to 0.44
Moderate-high severity	0.44 to 0.66
High severity	> 0.66

3.5 Characteristics of images

As previously mentioned, geospatial data can be modeled either as raster or vector formats. Remote sensing data, in particular, are typically represented as a raster. The term 'resolution' is commonly used to describe the quality of a digital image, usually referring to the size of the pixel. However, this definition alone is insufficient for remote sensing. In this context, four different types of resolution are used:

- Spatial resolution refers to the size of the smallest object that can be detected in an image. In digital and satellite imagery, resolution is defined by the size of the pixels. The smallest object that can be identified in an image cannot be smaller than the pixel size; thus, spatial resolution represents the dimensions of the pixels on the Earth's surface. For example, consider a satellite image whose spatial resolution is 10 m. This means that each pixel covers an area of 10 x 10 m on the Earth's surface, i.e., covers a total area of 100 square meters. The spatial resolution and pixel size are inversely proportional; that is, the smaller the pixel size, the higher the spatial resolution.
- Spectral resolution is defined as the specific range of EM radiation that a sensor registers, indicating the sensor's ability to distinguish between different wavelengths. Each spectral channel (also called a band) represents a narrow wavelength range in which information is collected. Higher spectral resolution corresponds to narrower spectral channels and a greater number of spectral channels. If the spectral resolution is too coarse, it can lead to a loss of information, hindering the accurate identification of target objects. Conversely, if the spectral resolution is too high, both data acquisition and processing become time-consuming and costly. Based on spectral resolution, satellite images can be categorized into three types: panchromatic (uses a single spectral channel), multispectral (comprises a collection of a few bands of the same area), and hyperspectral (involves the collection of hundreds of spectral bands). Black-and-white (panchromatic) images have low spectral resolution because they integrate radiation across the entire visible spectrum into a single band. In contrast, RGB images provide higher spectral resolution by capturing radiation separately in the blue, green, and red regions of the spectrum, though their

resolution remains lower than that of multispectral or hyperspectral data.

- Radiometric resolution corresponds to a sensor's sensitivity to detect slight differences in radiance reflected from the Earth's Surface. Higher radiometric resolutions enable the detection of subtle changes within the same spectral bands; therefore, it improves the ability to distinguish between different features and materials. Radiometric resolution is typically measured in bits (e.g., 8-bit, 12-bit, 16-bit), indicating the number of gray levels available for each pixel. For example, an 8-bit image can represent 256 different intensity levels (e.g. pixel value ranging from 0 to 255).
- Temporal resolution refers to the frequency at which a sensor can revisit the same area of Earth. It is inversely proportional to the time period between two subsequent observations; as the time period decreases, the temporal resolution increases, enabling accurate monitoring of changes over time. In remote sensing, temporal resolution is influenced by the characteristics of the satellite's orbit and the specifications of the sensor. It is typically measured in days but can also be expressed in hours or weeks. Higher temporal resolution is crucial for effectively monitoring changes over time, such as environmental shifts, seasonal variations, and dynamic events.

4 MICROWAVE REMOTE SENSING

As already mentioned, the microwaves represent the portion of EM radiation between infrared and radio waves. This part of EM is characterized by wavelength from 1 mm to 1 m and frequency from 0.3 to 300 GHz. Microwaves have much longer wavelengths than visible and infrared radiation, which makes them largely unaffected by atmospheric particles such as ozone, carbon dioxide, water vapor, and dust. Moreover, active microwave sensors operate independently of ambient illumination conditions.

Microwave remote sensing can be divided into two main categories: active and passive. These two sensor systems are fundamentally different, sharing only the fact that they both operate in the microwave spectral range. In passive microwave remote sensing, a sensor detects and records the radiation that is naturally emitted by objects or surfaces. Natural emission is influenced by the object's physical properties, such as temperature, moisture, and surface roughness. In addition, atomic composition and crystal structure also influence the amount of emitted radiance (for example, ice, due to its crystal structure, emits more microwave energy than liquid water). Although, due to longer wavelengths, microwave radiation can penetrate clouds, rain, and haze, enabling application in all weather conditions, the amount of available energy is relatively low which requires a wider field of view and, consequently, leads to lower spatial resolution in passive microwave datasets. Passive microwave sensors, such as radiometers, are particularly useful for many climate applications such as monitoring weather conditions, sea ice, or surface temperature.

Active microwave remote sensing, on the other hand, involves the transmission of microwave signals toward a target and detects the backscattering signal i.e. active sensors operated independently of sunlight eliminating problems due to bad illumination. Active radar can be categorized into imaging and non-imaging. The most common imaging active systems are RADAR (RAdio Detection And Ranging) systems, including Synthetic Aperture Radar (SAR). Similar to passive microwave systems, a major advantage compared to optical systems is the ability to observe in almost all-weather conditions and time, day or night.

The microwave part of the EM spectrum is quite large and it is usually divided into several bands (wavelength ranges). The most commonly used bands are: Ku bands, X-band, C-band, S-band and L-band. The description of different bands is shown in **Table 4**.

Table 4 Common microwave bands in remote sensing

Band	**Frequency [GHz]**	**Wavelength [cm]**	**Application**
P band	0.3 to 1	100 to 30	Penetrates vegetation and soil; used in biomass studies, vegetation mapping and soil moisture monitoring.
L band	1 to 2	30 to 15	Vegetation, soil moisture, forest structure, geophysical monitoring; used in missions like NASA's SMAP.
S band	2 to 4	15 to 7.5	Weather radar, wave monitoring, some soil and vegetation studies.
C band	4 to 8	7.5 to 3.8	Global monitoring, change detection, monitoring of ice, ocean; used in Sentinel-1, RADARSAT.
X band	8 to 12	3.8 to 2.5	High-resolution SAR: urban monitoring, ice, snow, sensitive to surface texture; used in TerraSAR-X, COSMO-SkyMed.
Ku band	12 to 18	2.5 to 1.7	Snow measurement, Ocean surface wave measurements.
K band	18 to 27	1.7 to 1.1	Rain radar, cloud profiling.
Ka band	27 to 40	1.1 to 0.75	High-resolution cloud and precipitation studies.

In remote sensing, the polarization of microwaves—that is, the orientation of the plane in which the transmitted wave oscillates—is an important characteristic. Depending on the orientation of the transmitted and received radar wave, the emitted pulse results in a different information. Radar sensors emit radiation in horizontal (H) or vertical (V) polarization. Similarly, an antenna can receive horizontally or vertically polarized backscattered energy, or some radars can receive both. Therefore, the four combinations of transmitted and received polarization are possible:

1. HH - horizontal transmit and horizontal receive,
2. VV - vertical transmit and vertical receive,
3. HV - horizontal transmit and vertical receive, and
4. VH - vertical transmit and horizontal receive.

Rough surfaces, such as bare soil or water, are most sensitive to VV scattering, volumetric scattering is most sensitive to cross-polarized data (HV or VH), while double bounced scattering is most sensitive to an HH polarized signal. Therefore, VV polarization is most suitable for bare surfaces, rough surfaces, vegetation with vertical structures, HV polarization for forest/non forest distinguishing, while HH polarization is recommended for mapping flooded/non flooded vegetation, urban areas.

4.1 Radar basics

Imaging radar systems consists of a transmitter, a receiver, an antenna and an electronic system to process and record the data (**Figure 5**). The transmitter generates and emits microwave pulses at specific frequencies to the antenna. The antenna focuses outgoing microwave energy into a beam, directs toward the target and receives the reflected echo within the illuminated beam allowing determination of the direction of the target echo. The duplexer is used to alternately switch the antenna between transmitters allowing usage of only one antenna. The duplexer sends the weak echo signal to the receiver. The receiver amplifies and processes the recorded backscattered signal returned from the target, while the signal processor converts raw signal data into interpretable forms. To create an image, each transmitted-received echo pulse sequence is sampled, and these samples are stored in a range line. As

the sensor moves, the recorded and processed echo builds up an image of the surveyed region.

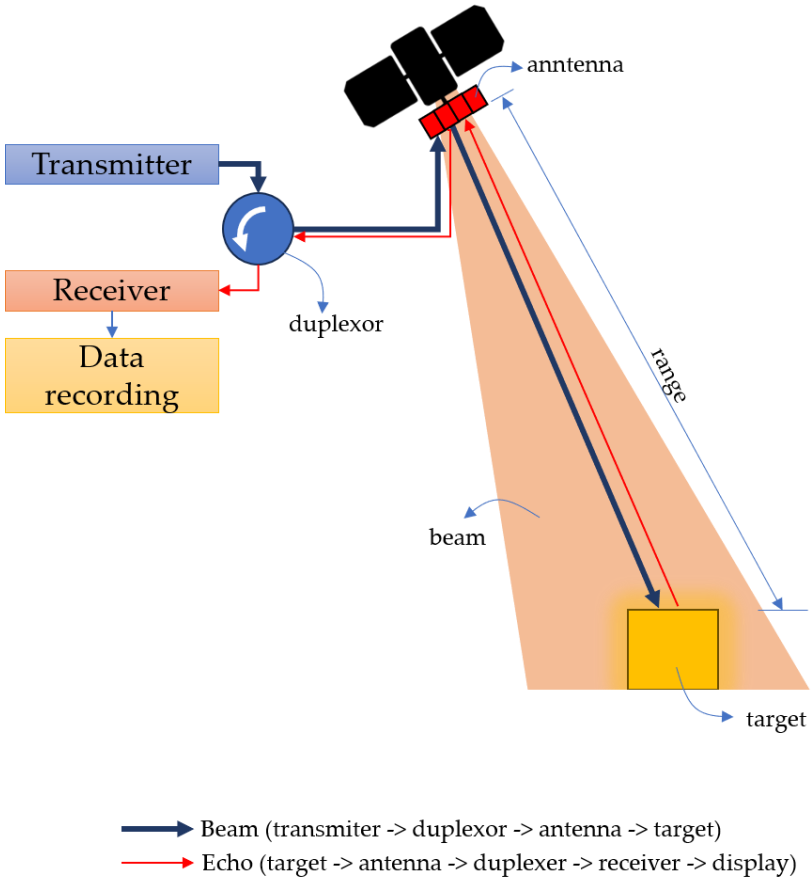


Figure 5 Components of Radar

The operation and data reconstruction principles of radar imagery differ fundamentally from those of optical sensors. Proper interpretation requires an understanding of what the radar actually detects: the intensity of the backscattered signal and the travel time of the returned echo. From this measured intensity, the backscattering coefficient is then obtained through radiometric calibration. It is a function of the radar system parameters and the physical properties of the surface. The backscattering coefficient σ_o is given as:

$$\sigma_o = \frac{average\ backscattered\ energy\ per\ unit\ area}{incident\ power\ per\ unit\ area}$$

The backscattering coefficient expresses the fraction of the radar signal that is scattered back toward the sensor. In linear units it is always non-negative, but when expressed in decibels (dB) it can take negative values. This occurs, for example, over smooth surfaces - like calm water , which scatter most of the energy away from the radar rather than back to it.

There are three main backscattering mechanisms in radar remote sensing: surface, double-bounce and volume scattering. Surface scattering (**Figure 6** (a)) is strongly dependent on surface roughness and sensor wavelength; it is characteristic of water, bare soils, roads, etc. The surface roughness (h) represents the average height variation in the target surface from a perfectly smooth surface. It can be expressed by using Rayleigh definition, i.e.

$$h < \frac{\lambda}{8sin\theta}$$

A smooth surface in radar images is represented as black since it acts as a mirror, reflecting the incoming radar beam at an equal and opposite angle to the incident angle (away from the antenna). The rough surfaces ($h > \frac{\lambda}{2}$) will reflect the incident radar beam equally into all directions (diffuse scattering).

The double bounce mechanism (**Figure 6** (b)) occurs when targets have two (or more) perpendicular surfaces (such as buildings, tree trunks). The perpendicular surfaces cause most of the radar energy to be reflected directly to the antenna due to the double balance. It causes a very strong backscatter that is not wavelength dependent.

The volumetric scattering occurs when signals penetrate inside the medium and scatter from different components within the medium, such as branches in the vegetation (**Figure 6** (c)). It is common for natural surfaces due to their inhomogeneous structure.

The backscattering coefficient and penetration depth of the radar signal are functions of many parameters, such as wavelength and surface characteristics (dielectric characteristics, surface roughness, orientation) of the target. The typical levels of backscattering, depending on target type, are presented in **Table 5**.

The dielectric properties of the target dictate how much incoming radiation will scatter at the surface, penetrate into the target, or get absorbed.

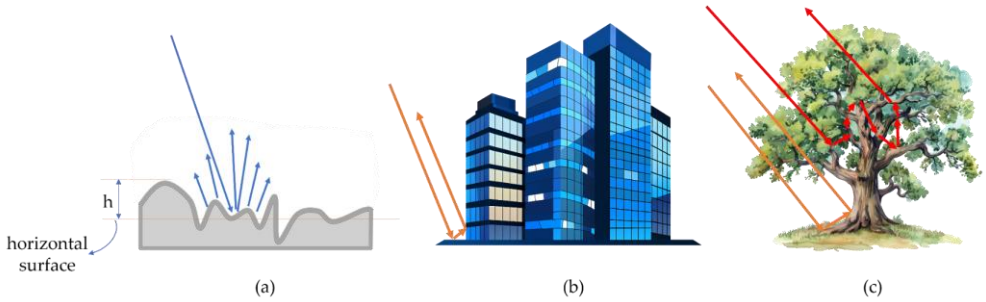


Figure 6 Radar backscattering mechanisms (a) surface reflection (blue), (b) double bouncing mechanism (orange) and (c) volumetric mechanism (red)

The penetration increases with the increase in wavelength. Shorter wavelengths (such as X-band radar) are scattered from the top of the trees, while longer wavelengths (L-band) will return from the ground in vegetated areas. Although the bare surfaces (such as glacier ice or alluvium solid) follow this rule, the penetration is strongly dependent on dielectric properties.

Table 5 Relationship between target type and level of backscattering

Type of the target	**Level of backscattering**
Man-made objects Terrain slops oriented toward radar Very rough surfaces Steep look direction	Very high
Rough surfaces Dense vegetation	High
Medium level of vegetation Agricultural crops Moderate rough surfaces	Moderate
Smooth surfaces Calm waters Impervious surfaces Very dry terrain	Low

The dielectric constant describes a material's ability to store and transmit electrical energy when exposed to electromagnetic radiation. It is strongly influenced by moisture content: dry soil has a low dielectric constant, whereas water surfaces exhibit very high values. The radar signal's penetration depth is inversely related to the dielectric constant—meaning that higher dielectric values lead to shallower penetration.

When surface scattering dominates, the return signal is strongly affected by surface roughness. If the roughness height is much smaller than the radar wavelength, most of the energy is reflected specularly away from the sensor, producing a weak return (dark pixels). When the roughness height is comparable to the wavelength, the signal is scattered diffusely in many directions, resulting in a stronger return. This explains why a surface with the same height variations can appear rough in X-band but smooth in C-band. The incidence angle is also important: moving from near range to far range, the return decreases because less energy is directed back to the sensor.

Most conventional radar uses pulsed radar systems, which transmit short radar pulses and listen for the return echoes. The distance (range) between the sensor and the individual target within the range line is calculated based on the traveling time T (time interval the signal needs to pass twice the distance between object and antenna) and the known speed of light c:

$$r = \frac{cT}{2}$$

While the range to the target can be determined, the radar measurement does not inherently contain information about the precise direction of the scattered signal. This ambiguity can be addressed by using two approaches, i.e., real-aperture radar (RAR) or synthetic aperture radar (SAR).

The RAR (**Figure 7** (a)) handles direction ambiguity by reducing the physical angular size (θ_{ant}) of the radar beam. The radar beam width is proportional to the wavelength and inversely proportional to the antenna length, i.e., aperture. This means the longer the antenna, the narrower the beam. The angular width of the antenna is approximately given by

$$\theta_{ant} = \frac{\lambda}{L_{ant}}$$

where λ is the wavelength of the signal and L_{ant} is the physical size of the antenna.

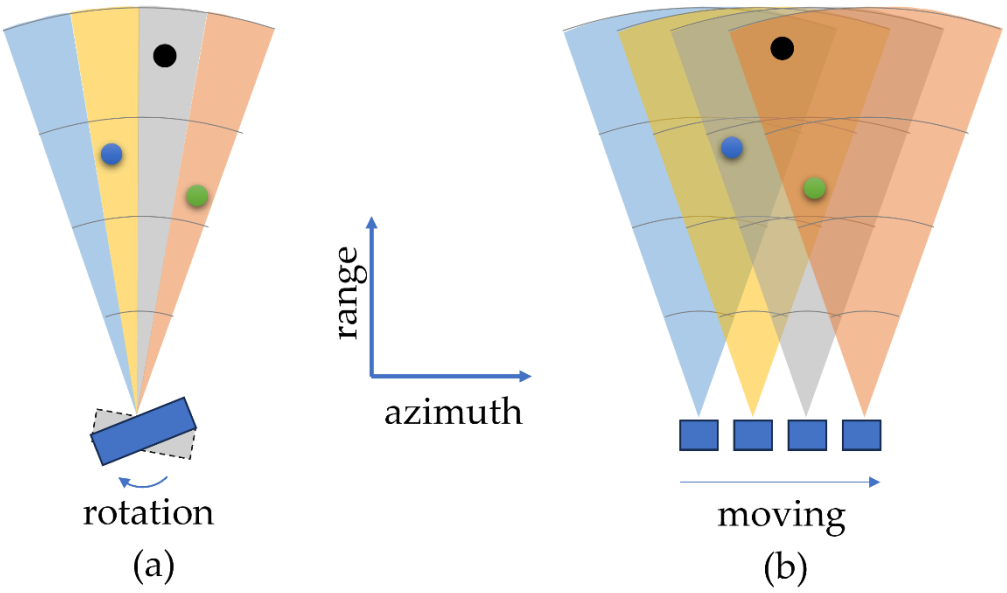


Figure 7 Comparison between RAR and SAR systems

Therefore, the usage of larger antennas will reduce the width of the beam along the directions, since the range of possible directions from which the signal could be reflected becomes smaller. Moreover, it increases the spatial resolution in the azimuth direction.

The radar remote sensing is based on coherent EM waves, i.e., the waves that are in constant phase with each other over space (spatial coherence) and time (temporal coherence), allowing the system to measure not just amplitude but also the phase of the returned signal. Temporal coherence represents the correlation coefficient between the radar signal phase at different times with the same observation geometry, and it quantifies the quality of pixel values between two time periods. Temporal coherence is the main factor limiting the accuracy in interferometric and tomographic SAR applications. Several factors can cause decoherence, such as spatial and temporal baselines, thermal noise, transpiration processes, physical changes, atmospheric conditions, etc. The temporal coherence usually decreases with increasing temporal baseline, while the spatial coherence usually decreases with increasing spatial baseline.

4.2 Radar viewing geometry

The platform carrying the radar sensor moves along the orbit flight direction with the nadir directly beneath the platform. The sequence of short microwave pulses of pulse length (τ_p) is transmitted perpendicularly to flight direction and it illuminates an area, i.e., a swath, with offset from the nadir. The direction along-track is called azimuth, and the direction across track is called range.

Radar sensors are side-looking instruments. The portion of the image swath closest to the nadir track of the radar platform is called the near range, while the portion farthest from the nadir is called the far range. The incidence angle of the system is the angle between the radar beam and the local vertical. Moving from near to far range, the incidence angle increases. The look angle is the angle at which the radar observes the surface. Together with the incidence angle of the sensor and the local incidence angle, which varies depending on terrain slope and Earth curvature, it characterizes the radar viewing geometry. It is defined as the angle between the radar beam and the local surface normal. The radar sensor measures the radial line of sight distance between antenna and object. This line is called slant range, i.e. slant range distance. The true horizontal distance along the ground corresponding to each point measured in slant range is called ground range (**Figure 8**).

To form a two-dimensional image, the echoes are sorted by their arrival time in both directions. A SAR image consists of pixels that are associated with a small area on the Earth's surface called a resolution cell. Each pixel value represents the coherent sum of the echoes from all scatterers within the resolution cell, i.e.

$$s_p = \sum_{n=1}^{N} a_n e^{i\varphi_n}$$

where a_n represents the amplitude from scatterer n and φ_n is the phase of scatterer n within the p resolution cell.

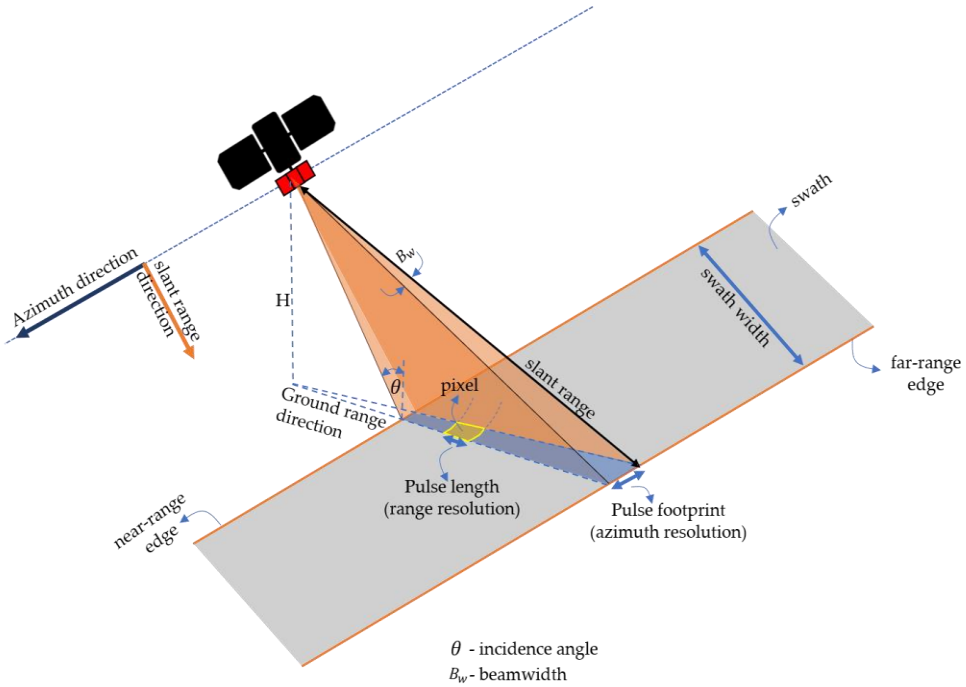


Figure 8 Geometric properties of radar

Pixel value is a complex number that contains amplitude and phase information of the backscattered echo within the corresponding resolution cell. Amplitude represents the magnitude of the scattered signal while phase describes the signal position within its oscillation cycle.

Different rows are associated with different azimuth locations, while different columns represent different slant range locations.

The radar image spatial resolution in slant range and azimuth direction is defined by pulse length and antenna beam width, respectively. Due to the different parameters that determine the spatial resolution in range and azimuth resolution, it is obvious that the spatial resolution in the two directions is different. For radar image processing and interpretation, it is useful to resample the image data to a regular pixel spacing in both directions.

In slant range direction the echoes from near-range swath edges arrive sooner than from far-range. The ability of radar to distinguish objects in range direction is defined by range resolution. The range resolution is defined as the distance that two objects on the ground have to be apart to give two different echoes in the return signal i.e. two objects will be resolved in range direction

if they are separated by at least half a pulse length. The range resolution depends on the bandwidth (B_w) or pulse duration (τ_p) of the transmitted signal, i.e.

$$\delta_r = \frac{c\tau_p}{2} = \frac{c}{2B_w}$$

The beamwidth is inversely related to the pulse duration, which represents the duration of a single transmitted radar pulse. Although it is independent of the range, the ground range resolution will depend on the local incidence angle, i.e.

$$\delta_{gr} = \frac{C}{2B_w sin\theta}$$

The equation shows that the range resolution is not constant across the swath; it degrades with increasing distance from nadir. This behavior is the opposite of that observed in optical systems.

The azimuth resolution refers to the capability of the radar system to distinguish between objects located at different angles in the horizontal plane. It is a function of the beam width θ_{ant} and the range r and can be expressed as:

$$\delta_a = \theta_{ant} r$$

Azimuth resolution linearly decreases as range increases. Based on the equation, it can be concluded that the azimuth resolution changes from the near-range to the far-range edge of the swath. Since radar beamwidth is inversely proportional to antenna length, a longer antenna will produce finer resolution.

4.3 Synthetic Aperture Radar

On one hand, to provide the azimuth resolution of a few meters, the RAR would require an antenna length of several kilometers. On other hand, the physical size of the antenna that radar platforms carry is limited. To overcome this limitation, the forward motion of the smaller antenna along the azimuth direction over time is used to simulate a very long antenna (synthetic aperture).

In 1951, Carl Wiely discovered that the resolution of radar images depends on the Doppler beamwidth of the echo rather than the long-trach footprint width of the beam pattern. The Doppler effect represents the shift in frequency of the wave caused by the relative motion of the sensor with respect to the target, i.e., a receding source (moving away from the observer) exhibits a lower frequency, while an approaching source (moving toward the observer) exhibits a higher frequency than the emitted frequency. Since two target points are separated in the azimuth direction they have slightly different speeds in any time point to the antenna. Consequently, the signal echoed from each target will have a frequency shift. The SAR uses the Doppler shift to identify target position.

The synthesis aperture concept is based on the fact that a target on the Earth surface is observed by many consecutive radar pulses. As the radar platform moves, the relatively small antenna of size L_{ant} (and corresponding azimuth resolution) illuminates the target from several positions. The target T at range R enters a beam when antenna is at p_{start} position and leaves the beam at p_{end} position (**Figure 9**). The backscattered signals from each radar pulse are recorded for as long as the target remains within the antenna beam. The echoes collected during this time are then coherently combined to simulate a much longer antenna—known as the synthetic aperture—than the actual physical one. The target that is offset by the x from the central antenna axis (R_0) will have Doppler frequency shift (f_d):

$$f_d = \frac{2v}{\lambda R} x$$

where v is the speed of the radar system platform and the target. The azimuth resolution is linearly related to the Doppler frequency resolution δ_{f_d} i.e.

$$\delta_a = \frac{\lambda r}{2v} \delta_{f_d}$$

The δ_{f_d} depends on the time that target spent within the beam i.e. the Doppler frequency shift can be more precisely determined if the total time duration that target stays within the radar beam is longer. Therefore, the δ_{f_d} can be approximate as

$$\delta_{f_d} \approx \frac{v}{L_{SA}}$$

where the length of synthetic aperture is equivalent to the distance the radar moves while the target stays within the beam and it can be calculated using the following expression

$$L_{SA} = \frac{\lambda}{L_{ant}} R$$

As a result, the maximum achievable azimuth resolution of the SAR is approximately equal to half the length of a real antenna and it is independent of the range and the wavelength. Although targets farther from the sensor stay in the beam longer (due to wider beamwidth), and closer targets are observed for a shorter time, this geometric effect is balanced so that the azimuth resolution remains constant across the entire image swath. The ability of SAR to achieve high and uniform resolution regardless of range is its key advantage, making it widely used in both airborne and spaceborne radar systems.

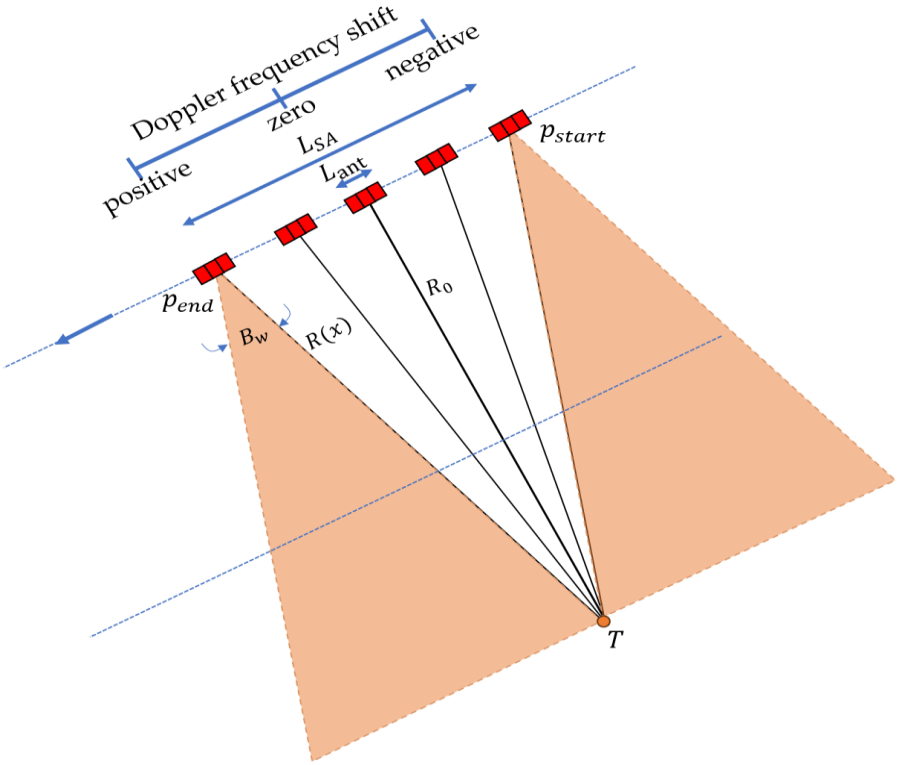


Figure 9 Geometry of observation using SAR for target T at along-track position 0 (Doppler shift 0). The p_{start} and p_{end} represents position when target T entered and leaved the radar beam respectively

4.4 Distortions in radar images

4.4.1 Geometric properties

Due to side looking viewing geometry, radar images exhibit geometric distortion which are caused by the relation between the local topography (surface slope and similar terrain features) and incidence angle. Radar imagery is typically affected by three geometric distortions: foreshortening, layover, and shadow (**Figure 10**).

In side looking geometry, if the radar beam reaches the base of a tall target before it reaches the top, the sensor-facing slope appears foreshortened i.e. a symmetrical mountain would appear as leaning toward the sensor side. As radar measures distance in slant range, the length of the slope will be compressed in the resulting image. The foreshortened depends on incidence angle θ and the slope angle (α), and it decreases with an increase in the incidence angle. The distortion is at its maximum if the radar beam is almost perpendicular to the slope. The echoes that come from sensor-facing slopes are stored into fewer pixels than it should be resulting in high digital numbers. Since echoes from different objects are combined the foreshortened areas in the radar image are very bright.

The layover is an extreme case of foreshortening and it occurs in areas where slopes are steeper than the incidence angle ($\theta < \alpha$). In layover situations the radar beam reaches the top of the slope earlier than the bottom, the slope is imaged upside down in slant range image. Due to that, the echo for slopes will overlay with image information at other areas. Layover effect decreases with increasing inclined angle and those areas are very bright on the image.

In the case of slops that are facing away from the sensor, the radar beam cannot illuminate the ground surface. Therefore, there is no energy that can be backscattered to the sensor and those regions remain dark in the image. Shadow effect increases as the incidence angle increases from near to far range.

The influence of all tree effects is related to the incidence angle. Although increasing the incidence angle can reduce foreshortening and layover, it also produces more shadow in the image.

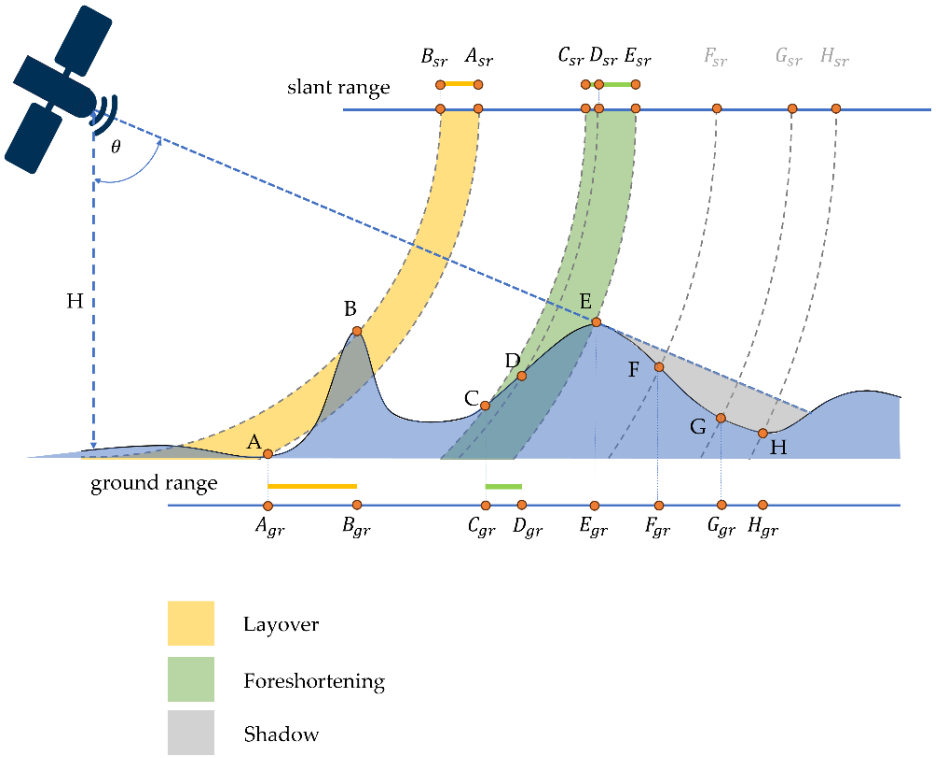


Figure 10 Geometric distortion of radar images

Due to that, the selected incidence angle should provide balance between foreshortening and layover on one side and shadow on another side. Foreshortened areas can be corrected by applying radiometric correction based on a digital elevation model.

4.4.2 Radiometric properties

In addition to geometrical distortion, all radar images, to some degree, exhibit a salt-and-pepper–like texture known as speckle. The speckle effect is caused by the interference of the different echoes within each resolution cell. The backscattered signal that forms one pixel comes from an area that contains numerous individual features that scatter radar beams. As a result, the backscattering signal from one pixel is a coherent sum of the thousand individual scattering contributions. With those scattering elements differing in position, orientation and height within the cell, the phase of the individual scatterers varies randomly, so that the scattering response of a single pixel results from the vector sum of numerous random contributions. The strength

of the summation vector depends on the relative phase of the scattered signal. If scatterers are in phase, constructive interference will produce large amplitude and bright pixels. On another hand, if two or more returning waves are of phase (phases differs by ~ 180°) destructive interference occurs, causing them to completely or partially cancel each other, which leads to lower amplitude and dark pixels. This interference causes both the amplitude and phase of the summed backscatter vector to vary randomly from pixel to pixel, resulting in the characteristic grainy appearance.

Speckle degrades radar image quality, making both visual and digital interpretation more complex. Although challenging to handle, several effective speckle filters and multi-look techniques have been developed to reduce it prior to image interpretation.

The multi-look technique divides the radar beam into several narrower sub-beams. Each sub-beam represents an independent look and the final image is created by averaging these multiple looks. It reduces speckle noise and it is performed during data acquisition.

On another hand, spackle filtering is performed on the output image to reduce local noise while trying to preserve lines and edges to maintain sharpness of image, preserve line and point target contrast, retain of mean values in homogeneous regions and texture information. Therefore, speckle filters need to balance between radiometric (noise removal) and spatial resolution (detail preservation). The type of target (homogeneous or point scatter) needs to be considered when designing the filters. For the targets with one or a few dominant radar returns within its resolution cell, such as buildings or poles, there is little or no random interference (**Figure 11**). Speckles are minimal, and the backscattered echo is a function of the reflection coefficient of the individual scatterer. In natural terrains (even in seemingly homogeneous areas such as grasslands or rivers) that contain many small scatterers randomly distributed within resolution cells and none of them provides a much stronger echo than others, the speckle noise is random and fully uncorrelated.

Taking that into consideration, two general frameworks are used: spatial filters and similar samples. The spatial averaging filter reduces speckle noise

by applying the standard algorithm to the target pixel neighborhood within a small moving window (usually 3x3, 5x5, 7x7, …).

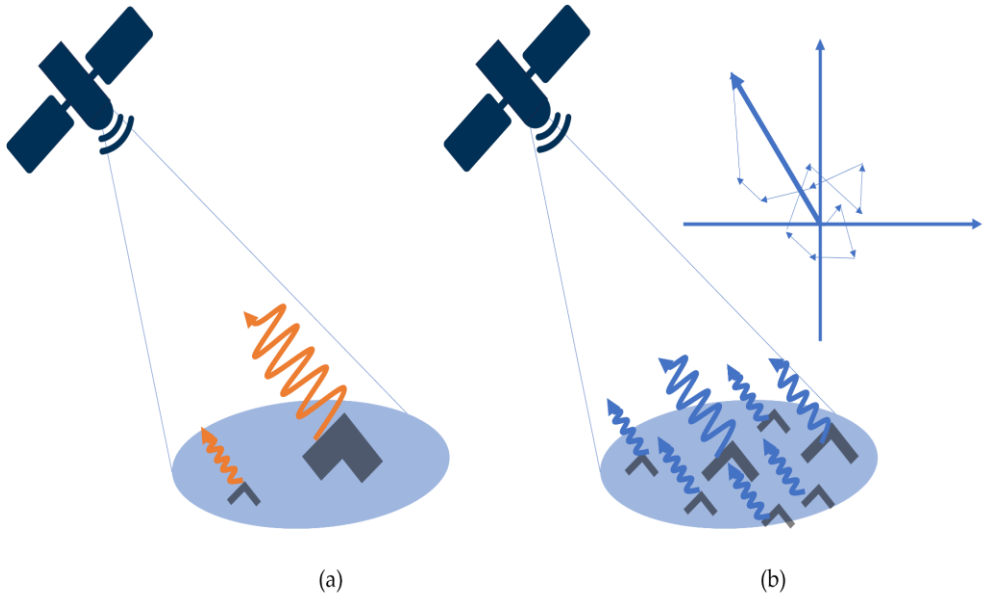


Figure 11 Modelling scattering mechanism inside a SAR resolution cell. (a) point scatterer - with one or more dominant scatterers within resolution, (b) multiple scatterers where there is no dominant scatterer.

For example, the Boxcar filter is a pure spatial averaging filter in which the target pixel value will be replaced by the mean of the pixels in a moving window. Although it has several advantages, including reduction of the standard deviation of noise in homogeneous areas, simple application, and preservation of mean value, it leads to loss of resolution. However, simple averaging cannot remove multiplicative speckle noise, and more sophisticated algorithms need to be used. They are usually based on a Bayesian technique that models radar backscattering as a product of true backscattering and multiplicative noise, based on prior information about the signal model or its distribution given as

$$y_p = x_p s_p$$

where y_p is the amplitude or intensity of the pth pixel in the noise SAR image, x_p represents a noise-free backscattering and s_p the speckle noise at the pth pixel. The s_p is modeled as a stationary random process, independent of x_p, with unit mean and a relative variance that characterizes speckle. Considering

that the real and imaginary parts of the collected complex noisy images are independent and identically distributed, they can be modeled as complex Gaussian variables with zero mean and variance of $\sigma^2/2$. Since the real and imaginary parts are Gaussian, the probability distribution function (pdf) of the amplitude can be described following the Rayleigh distribution model:

$$f_A(a) = \frac{2L^L}{\sigma^{2L}(L-1)!} a^{2L-1} exp\left(-\frac{La^2}{\sigma^2}\right)$$

where f_A is the Rayleigh pdf of amplitude, L is the number of looks, a represents pixel's values in an amplitude image (with a=y)

The y_p can be observed as a random variable whose mean is equal to x_p. Although the speckle noise does not significantly influence the mean value, it increases the variance. The primary aim is therefore to reduce variance and estimate the mean of noise-free backscattering. The expectation of y_p is equal to the expected value of true backscattering, i.e. $E(y_p) = E(x_p)$ since the speckle noise has a unit mean. The spatial averaging of similar pixels in the image can be used to approximate x_p as:

$$\hat{x} = \frac{1}{L}\sum_{i=1}^{L} y_{p_i}$$

One of the most widely used filters based on this methodology is the Lee filter.

The Lee filter uses the local mean and variance of all pixels within the moving window to estimate noise-free reflectivity by a linear combination of the local mean and the noise measurement as follows:

$$\hat{x} = a\bar{x} + by$$

where a and b are determined by applying the minimum mean square error criterion, i.e.

$$a = 1 - b \text{ i.e. } \hat{x} = \bar{x} + b(y - \bar{x})$$

here parameter $b = \frac{variance(x)}{variance(y)}$ is used to assess the heterogeneity of the local region and balance between local mean and original pixel value. In a homogeneous region, $b=0$ and the filter replace the pixel value with a local mean. In heterogeneous areas, $b \approx 1$, such as edges or textured regions, the filter preserves details by relying more to original pixel value. In regions with

moderate heterogeneity, the output pixel value will represent a linear combination between local mean and original pixel value. While Lee's filter works well in homogeneous areas, some noise around edges can still remain.

To address those limitations the Refined Lee filter is proposed. It uses a 7x7 moving window to detect edges by comparing means and variance along different directions. If an edge is detected the algorithm uses local gradients to estimate its orientation. Local mean and variance are calculated by using only pixels within edge-oriented window and then the $\hat{x} = a\bar{x} + by$ is applied; otherwise, the estimation of $\hat{x}$ is performed using all pixels in the local region. The filter provides good results on edges and in high-contrast areas.

Kuan filter relies on the same assumption as Lee filter but its weighting function is calculated based on the equivalent number of looks which can reduce the noise in the edge area.

In recent years, the Convolution Neural Network (CNN), which will be discussed in a subsequent chapter of the book, has been widely used to image classification and various image processing tasks including SAR image despeckling. Those approaches are data-driven since algorithms learn a mapping from noisy input images to output based on training data. Training data for despeckling with CNN contains the noisy SAR data and corresponding noise-free reference data. Since it is not possible to collect the noise-free SAR images, two strategies can be employed: temporal multilooking and synthesis strategies. The synthesis strategy uses optical images to simulate noisy SAR images by applying statistical spackle models while the temporal multilook strategy is based on reducing noise by temporal averaging images over a long time-series.

4.5 SAR Interferometry

SAR interferometry is a well-established remote sensing technique for precise measurement of geophysical parameters of the Earth's surface. It uses the phase difference between pairs of coherent radar signals to measure the range between radar and target. The phase of the radar signal represents the position within the wave cycle corresponding to the distance traveled by the radar signal to the target and back. Phase difference refers to the relative shift between two waves, either in time or space. Two waves of the same frequency are considered in-phase, if their peaks are perfectly aligned. The total phase of the returning echo is given by

$$\varphi = 2\pi \frac{2r}{\lambda} + \varphi_s$$

where $2r$ is round-trip distance, φ_s is the noise (speckle). An image of phase information is known as interferogram.

4.5.1 SAR interferometry geometry

A SAR satellite observes the same area from slightly different positions or at different times. The distance between satellites in the plane perpendicular to the orbit is called the interferometer baseline (B) which can be decomposed into horizontal/vertical components (B_h, B_v) while its projection perpendicular to slant range is the perpendicular baseline ($B_\perp$) (**Figure 12**). Taking into account the geometry of SAR configuration the following mathematical relationship can be obtained:

$$B_\perp = Bcos(\theta - \alpha) = B_h cos\theta + B_v sin\theta$$

$$B_h = Bcos\alpha$$

$$B_v = Bsin\alpha$$

where α is baseline orientation.

The signal of the resolution cell in the first image and the signal of the corresponding pixel on second image will be given as

$$i_1 = |i_1| e^{i\left(2\pi \frac{2R_1}{\lambda}\right) + \varphi_{S1}} \text{ and phase } \varphi_1 = arg(i_1) = 2\pi \frac{2R_1}{\lambda} + \varphi_{S1}$$

$$i_2 = |i_2|e^{i\left(2\pi\frac{2R_2}{\lambda}\right)+\varphi_{S2}} \text{ and phase } \varphi_2 = arg(i_1) = 2\pi\frac{2R_2}{\lambda}+\varphi_{S2}$$

The first phase component is deterministic (proportional to range distance) while the second is stochastic (speckle). The interferogram is created under the assumption that the phase difference is independent of the scattering mechanism (the scattering component remains unchanged: $\varphi_{S1} = \varphi_{S1}$) by coregistering two images and performing a pixel-by-pixel multiplication of their complex signals, i.e.:

$$i_1 i_2^* = |i_1 i_2^*|e^{-\frac{4\pi}{\lambda}\Delta r}$$

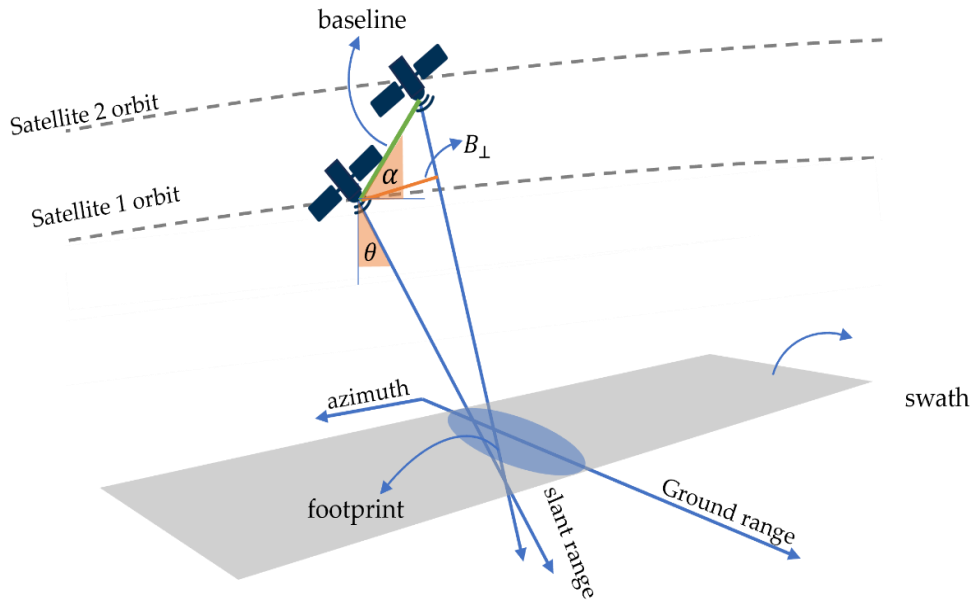


Figure 12 Geometry of a satellite interferometric system. $B_\perp$ represent the perpendicular baseline

This process multiplies the corresponding amplitudes and computes the difference of the corresponding phases at each point producing a new complex image called interferogram. The interferometric phase of each pixel depends only on the difference in path length between the two SAR images. This difference can be caused by elevation differences, motion or deformation. The interferometric phase is given by:

$$\Delta\varphi = 2\pi\frac{2\Delta r}{\lambda} + 2\pi N$$

where φ is interferometric phase, N represents the number of full wavelength cycles. The phase difference is the measure of the target displacement vector over the time interval between acquisitions and can be approximated by

$$\Delta r = Bsin(\theta - \alpha)$$

If the measurements have been made from different locations in space and at different times, the interferometric phase is proportional to the difference in the signal path lengths between the two acquisitions:

$$\Delta\varphi = \frac{4\pi}{\lambda}(\Delta\varphi_{flat} + \Delta\varphi_{topo} + \Delta\varphi_{disp} + \Delta\varphi_{atm}) = \frac{4\pi}{\lambda}\left(\Delta\varphi_{flat} + \frac{B_{\perp}h_T}{rsin\theta} + \Delta r\right)$$

where θ is local incidence angle. The $\Delta\varphi_{flat}$ represents the phase difference due to the Earth's curvature and satellite orbit. The orbit information provided by satellites can be used to estimate and remove the flat-earth interferometric phase components. This process is known as interferogram flattening. The $\Delta\varphi_{topo}$ represents the phase caused by the terrain elevation variation relative to reference height and it can be removed if the digital terrain model (DEM) is available. If two SAR images are not collected simultaneously, the propagation of radar beams is affected by differences in the atmosphere. The atmospheric delay is caused by spatial and temporal variation in the atmosphere conditions (such as temperature, humidity, pressure) affecting both elevation and deformation measurements. It can significantly impact the accuracy of deformation estimates, but can be mitigated using statistical methods or auxiliary data for atmospheric modeling.

4.5.2 DEM generation

The phase variation between two points on the flattened interferogram directly correspond to the actual change in terrain elevation. However, the flattened interferogram provides an ambiguous measurement of the relative terrain altitude since the interferometric phase is wrapped between $-\pi$ and π. The actual phase shift between two waves is usually larger than this range. The process of restoring continuous phase value by adding or subtracting the correct number of full cycles to the interferometric fringes - which represents

the full range of the phase values in an interferogram from 0 to 2π) is called phase unwrapping.

With reference to the geometry, the local height h_0 of the ground point is given as

$$h_0 = H - r_1 cos(\theta)$$

where H is the satellite flying height, r_1 is the distance between the scatters and radar antenna 1. Meanwhile, the interferometric phase difference

$$\Delta\varphi = 2\pi \frac{2\Delta r}{\lambda} + 2\pi N \text{ i.e. } \Delta r = \frac{\lambda \Delta\varphi}{4\pi}.$$

Taking into account the acquisition geometry (**Figure 13**) and applying the cosine's rule to the triangle ($r_2 = r_1 + \Delta r$) it is evident that

$$(r_1 + \Delta r)^2 = r_1^2 + B^2 - 2r_1 B cos(\pi/2 - (\theta - \alpha) \text{ i.e. } sin(\theta - \alpha) = \frac{(r_1+\Delta r)^2 - r_1^2 - B^2}{2r_1 B}$$

Where B is baseline.

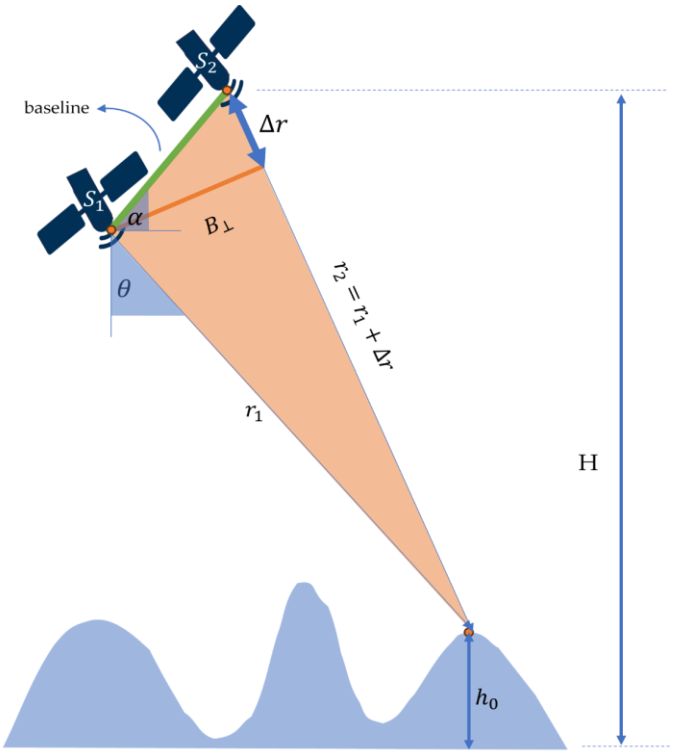


Figure 13 SAR interferometric geometry

The altitude difference corresponding to two adjacent fringes is called the altitude ambiguity (h_a). It is defined as the elevation difference that produces an interferometric phase change of 2π after interferogram flattening, and it is inversely proportional to the perpendicular baseline:

$$h_a = \frac{\lambda R sin\theta}{2B_\perp}$$

Based on the equation it can be concluded that the higher the baseline the more accurate the altitude measurement. However, an increase in the perpendicular baseline increases the decorrelation noise in interferometric phases.

If two observed targets T_1 and T_2 are located at the same range but different heights (**Figure 14**) then the difference in range implies a different interferometric phase at each point, caused by the height difference ($\Delta h = h_2 - h_1$). The phase-to-height sensitivity represents how much the interferometric phase difference changes for a given height difference. It increases with longer spatial baseline, shorter wavelength, and smaller incidence angles.

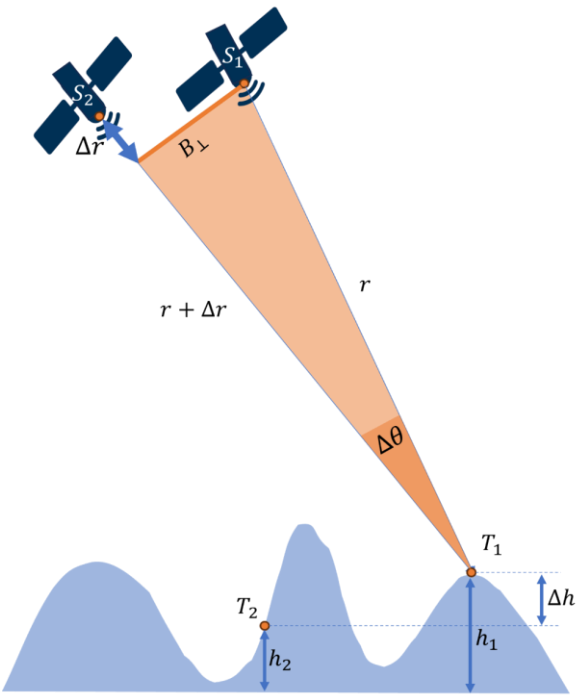


Figure 14 Illustration of the phase-to-height sensitivity in interferometric SAR.

4.5.3 Differential interferometry

If some of the point scatterers on the Earth surface slightly change their relative position between two SAR acquisitions (for example due to landslides, earthquakes, etc.), an additional phase term appears in the interferometric phase:

$$\Delta\varphi_d = \frac{4\pi}{\lambda} d$$

where d is relative scatter displacement projected on the slant range direction. This means that after the interferogram flattening, the interferometric phase contains both topographic and motion components, i.e.

$$\Delta\varphi_d = -\frac{4\pi}{\lambda}\frac{B_\perp h_T}{r sin\theta} + \frac{4\pi}{\lambda} d$$

where the first component represents the topographic phase while second represents the displacement phase. The fundamental equation for detecting and measuring changes is given by:

$$d \approx \frac{\lambda}{4\pi}(\Delta\varphi_d - \Delta\varphi_d \frac{B_{12}}{B_{01}})$$

where B_{12} is the baseline between SAR acquisitions 1 and 2 and B_{01} is the reference perpendicular baseline used for generating the modeled altitude phase. If a DEM is available, the differential interferogram can be obtained by subtracting the modeled phase contribution due to terrain elevation from the interferometric phase, thereby isolating the displacement signal. The measured displacement is not vertical but along the slant range and it represents the slant range component of the three-dimensional surface displacement vector under the assumption that the surface within a pixel deforms homogeneously. InSAR measures small-scale vertical movements, but large displacements cannot be detected directly because the phase difference is limited to half the radar wavelength. When displacements exceed this limit, phase unwrapping is needed to recover the total movement. The relative accuracy of detected displacements is on the order of millimeters, whereas the absolute accuracy of DEMs is much lower (e.g., 10–15 m for ERS data), since the differential phase is far more sensitive to displacement than to topography.

Differential interferogram has various applications including co-seismic and post-seismic displacement fields related to earthquakes, dynamics of glaciers and ice sheets, deflation and inflation of volcanoes, land subsidence (mining activity, exploitation of gas or oil.

5 PHOTOGRAMMETRY

Photogrammetry is the branch of remote sensing that extracts geometric information on a target area from a series of overlapping photographs taken from different positions. The main distinction between remote sensing and photogrammetry lies in their applications. Remote sensing typically uses data either from individual images, for tasks like classification, or from time series of images, for change detection, to derive information about the area of interest. It primarily relies on spectral analysis and the monitoring of large regions. In contrast, photogrammetry focuses on accurate geometric measurements and 3D reconstruction.

Analog photogrammetry is based on the stereo-pair, i.e. two images of the same area taken from two different viewpoints are used to directly derive 3 dimensional points, enabling measurement of heights. The basic principle of stereoscopy is binocular vision: by observing an object from two different perspectives, it is possible to perceive depth. By analyzing the parallax effect - the apparent shift of an object between the two images - it is possible to determine the depth of the object. More details can be found in [22].

In recent years, advancements in IT technologies and computer vision techniques, coupled with progress in sensor technology, have led to the development of Unmanned Aerial Vehicle (UAV) photogrammetry. UAV is a relatively new technology that combines the traditional photogrammetry principles with computer vision. Traditional photogrammetry contributes fundamental principles for accurate 3D reconstruction, such as collinearity equation, bundle block adjustment, camera calibration, etc. On the other hand, computer vision enhances the UAV photogrammetry with image matching and Structure from Motion (SfM) algorithms. SfM uses multiple series of overlapping photos from a variety of perspectives to create the 3D set of points (X, Y, Z coordinates) with associated RGB color information. Deep Learning (DL) and Artificial Intelligence (AI) enhance classification, object detection, and semantic segmentation, making UAV photogrammetry a powerful tool for a wide range of close-range applications. They enable near real-time processing, rapid data acquisition, and real-time transmission to ground stations, offering a flexible and low-cost alternative to traditional

photogrammetry. In addition, UAV images can be used for high resolution texture mapping on existing 3D models. However, there are limitations in the use of UAV that need to be considered. UAVs, particularly low-cost models, have limited payload capacity, which necessitates the use of lightweight sensors. As a result, small- or medium-format amateur cameras are often employed, requiring a larger number of images to achieve the same coverage and comparable resolution as large-format cameras. Additionally, payload limitations require the use of lightweight navigation units, leading to reduced sensor orientation accuracy. The flight range is also constrained by the radio link distance and the pilot's proficiency.

5.1 Basic principles of photogrammetry

To perform accurate 3D measurements based on 2D photos, the information lost in the acquisition process needs to be reconstructed. This is achieved by reconstructing a viewing ray for each ground point, often referred to as a feature. A viewing ray can be defined as the line from the feature, passing through the projective center of the camera to the corresponding pixel in the image sensor. However, a single viewing ray cannot unambiguously determine the feature's position, as the feature could lie anywhere along the line. To resolve this ambiguity, a second image, collected from a slightly different position, is used. Knowing the orientation and position of the camera, the distance of the feature (and its coordinate) can be computed by calculating the spatial intersection of two or more viewing rays. For faster processing and higher-quality 3D reconstruction, it is recommended—but not mandatory—to capture all photos simultaneously and use the same camera.

The perspective ray can be modeled as a pyramid with a rectangular base. Knowing the shape and size of the pyramid is essential for photogrammetry. Three parameters are needed for this purpose: width (SW) and height (SH) of the digital sensor (base of the pyramid) and focal length (height of pyramid) (**Figure 15**). However, the pyramid is not geometrically perfect, the ray passing through the camera lens follows a complex curved path, creating radial and tangential distortion. The exact shape of the distorted pyramid is determined during camera calibration. There are three main approaches to camera calibration: laboratory calibration, test-field calibration and self-calibration.

A digital camera consists of a two-dimensional array of charged-coupled device (CCD) elements, known as a full-frame sensor, which is mounted in the focal plane (image plane). Light rays from all points in the scene pass through the lens center before reaching the CCD element. During image acquisition, all CCDs are exposed simultaneously, producing a digital frame. Digital cameras are classified by the number of pixels in the digital image which corresponds to the number of CCD elements. Generally, the higher the number of CCDs (pixels) in the sensor, the more expensive the camera is. For example, a 16 megapixels camera has a sensor with 4000 pixels x 4000 pixels.

The optical axis (principal axis) is defined as a straight line joining the two centers of a lens's spherical surfaces. The optical axis intersects the image plane at the principal point. When the light rays that are parallel to the optical axis enter the lens they converge or diverge to a specific point called focus point. The distance from the focal point to the center of the lens is the focal distance. It determines the angle Field Of View (FOV) and magnification level (the longer focal length, the higher magnification will be). The FOV represents the extent of the real world that the sensor can capture at a given moment. For a given sensor size, the angular FOV increases as the focal length decreases. Conversely, a shorter focal length provides wider ground coverage at a given flight height. The horizontal FOV (α) is the function of sensor width while vertical FOV (β) is the function of sensor height. In photogrammetry wider angles are preferable since they provide a lower flight height and increased swath width. The focal length cannot be too short, as this would result in excessive distortion. Typically, the focal length should be similar to the sensor height. Information about sensor characteristics are saved in image metadata.

Ground coverage represents the ground surface area covered by a single photo. Ground coverage is a function of the sensor characteristics and the flight height (it increases quadratically with increase of height). It is not influenced by sensor resolution. If the ground is flat, the ground coverage is a rectangle with dimension a * b which can be calculated using the following expression:

$$a = 2 \cdot h \cdot tan\left(\frac{\alpha}{2}\right)$$

$$b = 2 \cdot h \cdot tan\left(\frac{\beta}{2}\right)$$

where the *h* is the distance to the object.

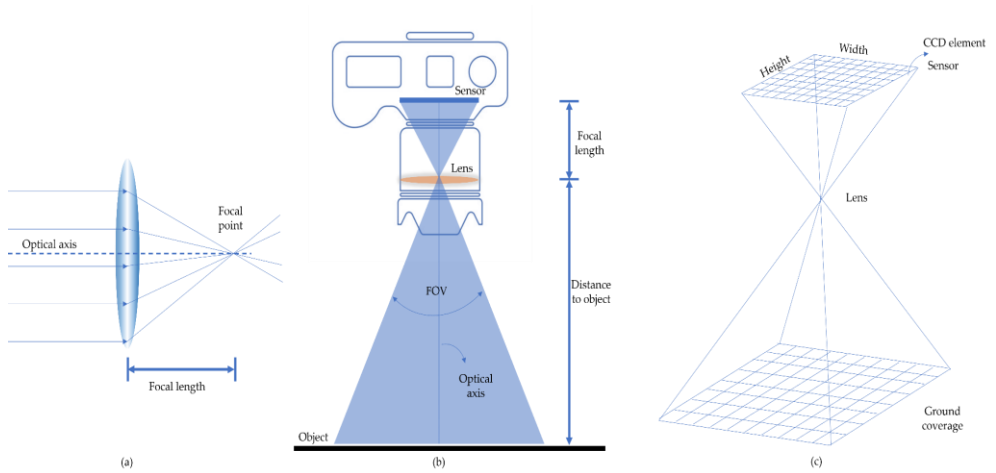


Figure 15 (a) lens, (b) (c) geometry of digital camera

The Ground Sampling Distance (GSD) is the distance between two consecutive pixel centers measured on the ground. The larger the GSD, the lower the spatial resolution of the image. It depends on sensor size, focal length and flight height. For standard cameras (FOV 72° and 16 MP) flying at 120 m, the GSD is approximately 3 cm. Increasing the flight height negatively affects the GSD in a linear manner, meaning that lower flight heights result in higher spatial resolution. In contrast, the relationship between GSD and megapixels is not linear; increasing the number of megapixels does not significantly improve the GSD. The GSD should be chosen based on the specific application, allowing the flight height and camera specifications to be adjusted according to project requirements.

5.2 Camera parameters

In modeling the geometry of camera three coordinate systems must be considered (**Figure 16**):

1. World coordinate system (WCS),
2. Camera coordinate system (camera frame (CF)) uses the camera center (C) as origin and optical axis as the Z-axis, and
3. Image coordinate system (image frame (IF)) measures pixel location in the image plan.

In order to represent the 3D point P of an object in an image, it is necessary to map the object position from the WCS location ($P(X, Y, Z)$) to camera coordinates ($p(u, v, w)$) and then to project camera coordinates onto the image plain to obtain the pixel coordinates ($p(x', y')$). To do so, following expression can be used

$$x = PX$$

where x is homogeneous image coordinates, P is camera matrix and X is homogeneous world coordinates.

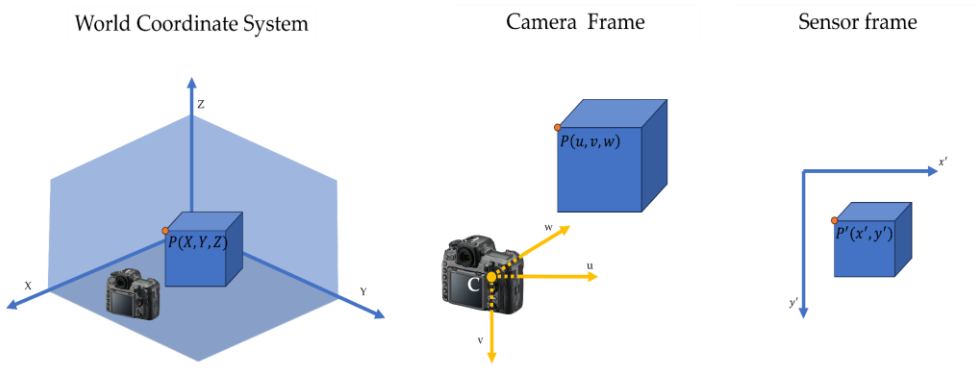


Figure 16 Image coordinate system (a) world coordinate system, (b) camera frame, and (c) sensor frame

5.2.1 Image coordinate system

A pinhole camera can be constructed by placing a barrier with a small hole between the 3D object and sensor. Because of the aperture only a fraction of light emitted by objects will hit a sensor allowing one-to-one mapping between points on 3D objects and sensors. The film is commonly referred to as the image plane, while the aperture represents the center of the camera (and it is denoted with C). Let $P = [x\ y\ z]^T$ be the point of an object visible to the pinhole camera. The 3D point P will be projected onto the image plane resulting in the point $P' = [x'\ y']^T$ on this plane Π'. Furthermore, the pinhole itself can also be projected onto the image plain resulting in a new point C'. The coordinate system $[u, v, w]$ centred at the pinhole and w axis coincided with the optical axis is known as camera coordinate system.

The point P defined with X, Y, and Z coordinate in world coordinate system is projected on the image-plain by dividing them by their Z component i.e.

$$P^{'} = [x' \quad y']^T = \left[f\frac{X}{Z} \quad f\frac{Y}{Z}\right]$$

In a digital camera, the lens collects the light form object over a wider area and directs it to an image plane where an array of CCD sensors converts light to a digital image. Thus, the relationship between the point P in 3D space and its corresponding point $P^{'}$ on the image plane can be expressed by

$$P^{'} = [x' \quad y']^T = \left[z'\frac{X}{Z} \quad z'\frac{Y}{Z}\right]^T$$

where z' is the distance between the focal point and the image plane (in the pinhole camera model $z' = f$, while in a lens-based camera $z' = f + z_0$). However, the pixel coordinates in the digital image are defined in a different reference system than the ideal image plane. In the image plane model, the origin is located at the principal point C′, where the optical axis intersects the plane. By contrast, in a digital image, the origin of the coordinate system is usually placed at the top-left corner of the sensor array. Because of this, the 2D coordinates on the image plane and 2D coordinates on the digital image are related by a translation vector $[c_x \quad c_y]$. Consequently, the mapping becomes:

$$P^{'} = [x' \quad y']^T = \left[f\frac{X}{Z} + c_x \quad f\frac{Y}{Z} + c_y\right]^T$$

Moreover, in a digital camera the image plane is discretized into pixels, i.e. the point location on the digital image is expressed in pixel coordinates while the points on the image plane are represented in physical measurements. As a consequence, two parameters, ρ_w and ρ_h that represent the width and height of the CCD sensor (usually expressed in μm), need to be introduced (**Figure 17** (b)). In most cases, pixels are squares, i.e. $\rho_w = \rho_h$. Under these conditions, the previous mapping can be expressed as

$$P^{'} = [x' \quad y']^T = \left[f\rho_w\frac{X}{Z} + c_x \quad f\rho_h\frac{Y}{Z} + c_y\right]^T$$

The transformation from $P \rightarrow P^{'}$ is not linear and therefore it cannot be expressed as a standard matrix-vector product. To overcome this problem, the coordinates are represented in homogeneous form. This means introducing the new coordinate $P^{'} = (x', y', 1)$ and $P = (X, Y, Z, 1)$ in an augmented space. Homogeneous coordinates are obtained by appending an additional

dimension to the vector, with the convention that the last coordinate is equal to 1 after normalization. Using homogeneous coordinates, the mapping can be expressed compactly as

$$P' = \begin{bmatrix} f\rho_w x + c_x z \\ f\rho_h x + c_y z \\ z \end{bmatrix} = \begin{bmatrix} f\rho_w & 0 & c_x & 0 \\ 0 & f\rho_h & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} P = MP$$

This can be transformed in

$$P' = MP = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix} [I \quad O]P = K[I \quad O]P$$

where K represents the camera intrinsic matrix, $\alpha = f \cdot \rho_w$ is focal length in x direction (in pixels), $\beta = f \cdot \rho_h$ is focal length in y direction (in pixels), while c_x and c_y represents the coordinates of the principal point in pixels. The camera matrix contains all essential parameters of the camera (two for focal length, two for translation and one for skewness) and these are collectively known as intrinsic camera parameters.

Moreover, in real cameras the image axes may be not perfectly orthogonal due to sensor manufacturing imperfections; in this case the camera intrinsic matrix is given by:

$$K = \begin{bmatrix} \alpha & -\alpha cot\theta & c_x \\ 0 & {}^{\beta}/_{sin\theta} & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The camera matrix describes the transformation of a 3D point in the camera coordinate system into its corresponding point $P^{'}$ on the 2D image plane. However, the location of objects in the real world is represented in a different system, namely the world reference system. Therefore, an additional transformation must be introduced to relate points from world reference system (P_w) to the camera frame (P). This transformation is defined by a rotation matrix R and translation vector **t** i.e.

$$P = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} P_w$$

where R is the rotation matrix of the world coordinate system defined in the camera frame and **t** is the position of the world coordinate system's origin in the camera frame.

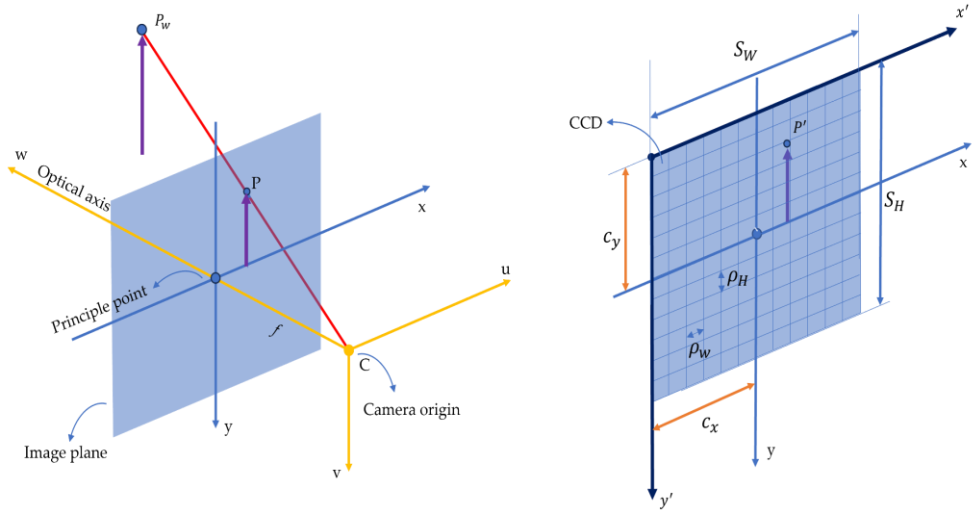


Figure 17 Intrinsic orientation (a) The central projection model. The image plane is a distance f of camera origin. (b) Image plane and discrete pixels

The R and **t** are known as extrinsic parameters and they do not depend on the camera characteristics (**Figure 18**). Consequently, the point P_w can be computed in image frame as

$$P^{'} = K[R,t]P_w = MP_w$$

where M is a 3x4 matrix known as a full projection matrix and includes both intrinsic and extrinsic parameters. It has 11 eleven degrees of freedom: 5 intrinsic parameters (focal lengths, principal point coordinates and skew), 3 parameters from rotation and 3 from extrinsic translation.

Rotation of points in a 3D space can be represented as a product of three successive rotations around the coordinate axes, a process known as a 3D Euclidean transformation. The rotation matrix R is orthonormal, satisfying the conditions: $RR^T = I$ and $|R| = 1$.

The camera extrinsic and intrinsic parameters are estimated through camera calibration. The underlying image model is based on an ideal projection in a pinhole camera, in which straight lines in reality are transformed into straight

lines in the photo; however, these lines can appear as curved lines due to lens imperfection, an effect known as lens distortion.

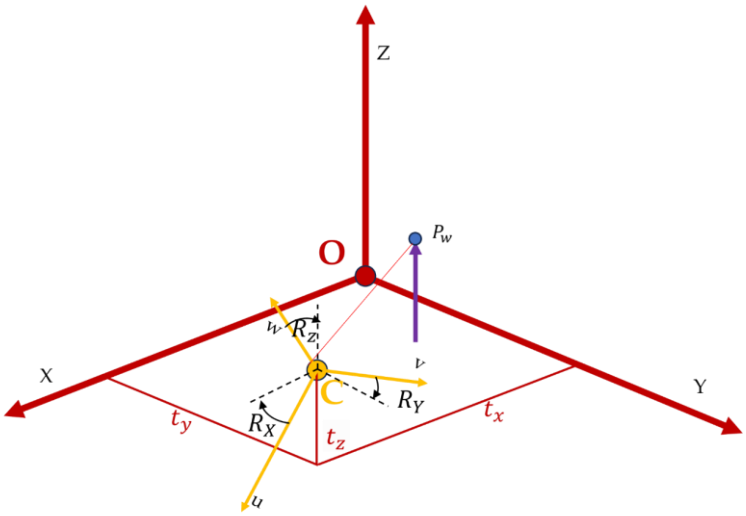
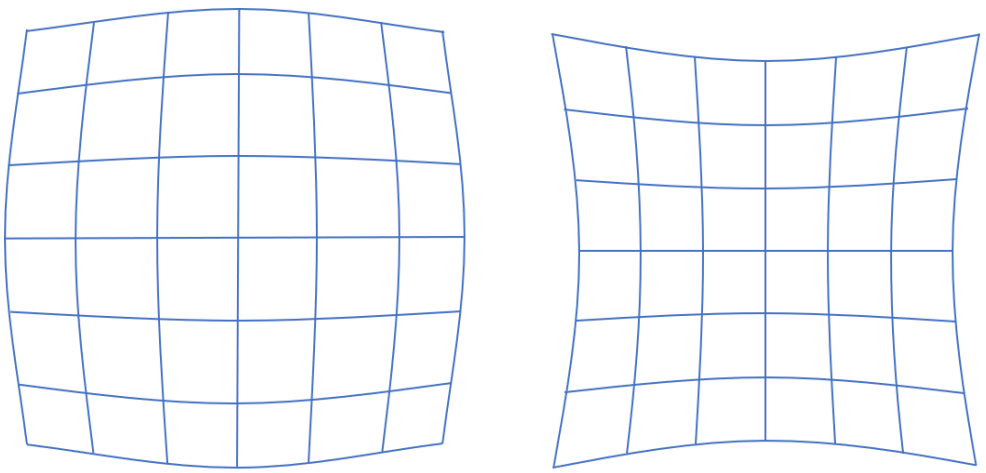


Figure 18 Extrinsic parameters

The most common geometrical distortion is the radial one, in which straight lines appear to be curved. It is caused by the spherical shape of the lens, and it increases with the distance from the optical axis. The image edges can be curved outward (barrel distortion (**Figure 19** (a))) or inward (pincushion distortion (**Figure 19** (b))) from the image center.



(a) (b)

Figure 19 (a) Barrel distortion, (b) Pincushion distortion

The tangential distortion occurs when the lens and the image plane are not parallel, and its effect is generally smaller than the radial distortion. Many wide-angle lenses have noticeable reflection, especially lenses with short focal lengths. Lens distortion alters the shape of the object, and therefore reduces the accuracy of the model. Therefore, it must be accounted for during camera calibration.

Camera calibration is typically performed by capturing several images of planar checkerboard patterns at different positions and orientations. The real 2D coordinates of corners and corresponding coordinates are then used to determine the intrinsic and extrinsic parameters.

5.3 Geometry of oblique images

Generally, the images can be classified based on the camera tilt relative to the vertical axis into: vertical photos (camera tilt under 3°), low oblique (horizon is not visible), and high oblique (horizon is visible). The oblique image has higher resolution and greater total area captured compared with vertical images (**Figure 20**). Also, they offer extensive information on the side view of the ground object that cannot be obtained from a vertical image, thus leading to greatly increased redundant information. In the past, redundant information represented a challenge for traditional photogrammetry since it struggled to match corresponding pixels across multiple images due to changes in perspective and varying light conditions. This is especially challenging for oblique images. However, the redundant information is the key to the success of the SfM algorithm, thus a combination of vertical and oblique images of target objects will improve 3D reconstruction but also increase processing time.

The principal plane of an oblique aerial image is the vertical plane that passes through the camera optical axis and the vertical line from the projection center; it intersects the oblique image plane at the principal line. The principal line passes through the image nadir point and the principal point. It is oriented in the direction of the biggest inclination in an oblique aerial image.

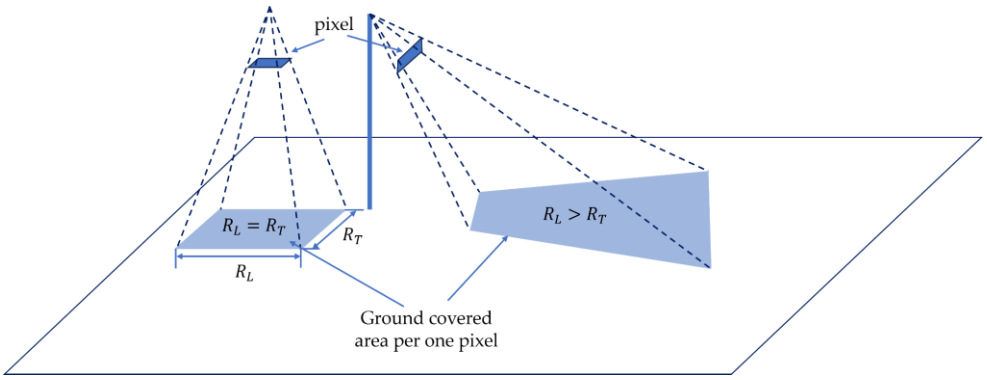


Figure 20 Comparison between the ground covered area per pixel on vertical and oblique images

The nadir point N′ (i.e., image nadir point) is the intersection of the vertical from the perspective center with the image. The ground nadir point (N) represents the intersection of the vertical from the perspective center with the ground surface. The geometry of the oblique image is shown at **Figure 21**.

The three angles of tilt (τ), swing (s) and azimuth (α) completely define the angular orientation of the oblique image. The azimuth is the clockwise horizontal angle measured from the ground Y axis (usually north) to the datum principal line. The tilt angle is the angle between the vertical and the camera optical axis; it determines the magnitude of tilt of an image. If the tilt angle is zero, the image is vertical. The swing angle is the clockwise angle measured at an oblique image plane from the positive y-axis to the nadir point.

The isocenter is the intersection of the bisector of the tilt angle and the oblique image plane. The isocenter lies on the principal line, the oblique image plane, and the plane of the equivalent truly vertical image. In an oblique aerial image, the displacement caused by tilt is radial with respect to the isocenter. This means that points on the image appear shifted along lines radiating from the isocenter. The key characteristic of this radial displacement is that angles measured from the isocenter in the image are true, meaning they are equal to the actual angles measured from the corresponding ground isocenter. This property is useful in photogrammetry because it allows certain angular measurements to be made directly on oblique images, despite the presence of tilt.

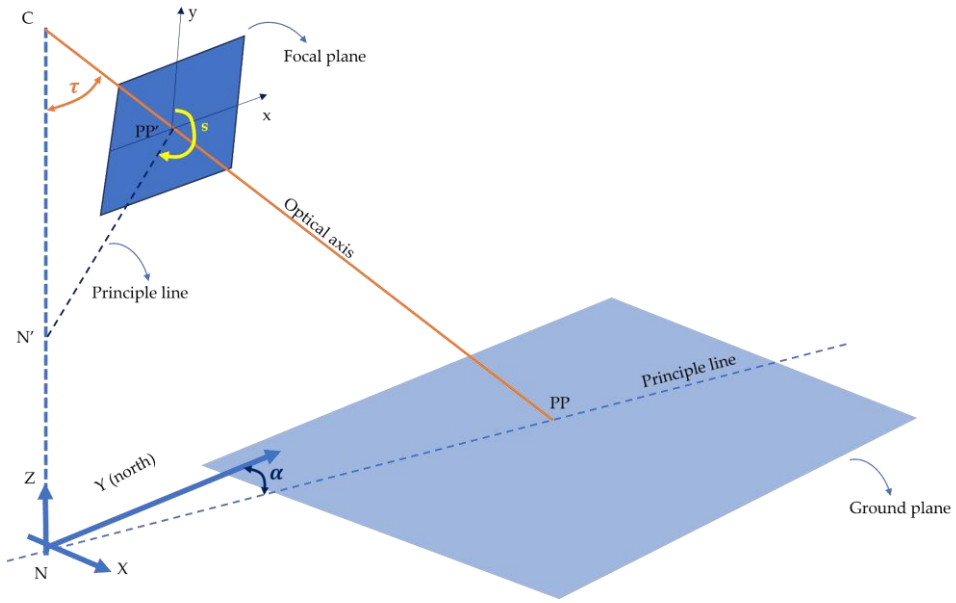


Figure 21 Geometry of oblique image

In vertical photos the object distance changes only due to variations in topography. In oblique images, the scale of the image is affected by the magnitude and angular orientation of the tilt as well as topography variation. Due to that, the scale of point on an oblique image is not the same in all directions i.e. x-scale (scale of lines perpendicular to the principal plane), the y-scale (scale of lines parallel to the principal plane) and z-scale (scale of vertical lines) are not the same. The shorter the object distance, the larger the scale. The scale of an oblique image varies along the principle line.

Image displacements caused by relief on an oblique image depend on the flying height of the aircraft, height of object above the reference ground level, tilt of the camera and the location of the object in the image (objects farther from the principal point or isocenter are displaced more). Suppose that point T is located on the top of the building corner and point B at the bottom of the same corner. On 2D maps those two points will coincide since their X and Y coordinates are the same. However, in the aerial image the top of the building appears shifted outward, i.e. point T moves away from nadir, while point B remains close to its true position. The distance between two photo points is called relief displacement and it is caused by the height difference between T and B. The direction of relief displacement (**Figure 22**) is radial with respect to the nadir point.

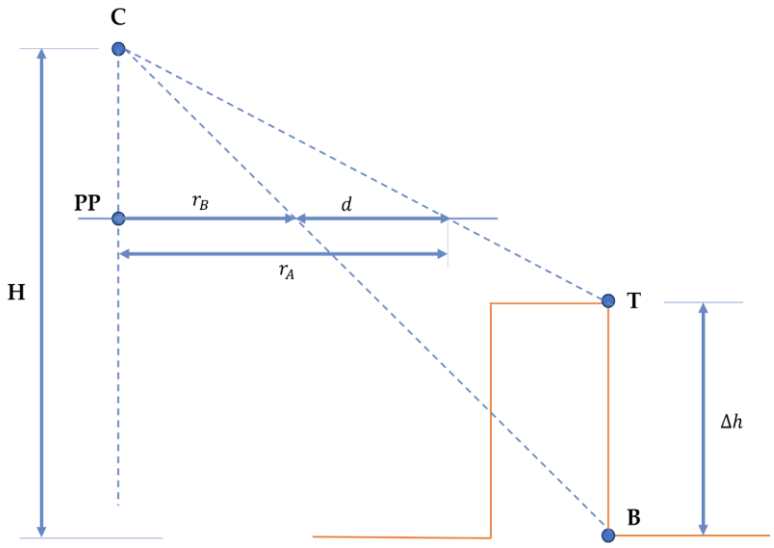


Figure 22 Relief displacement (d) (H is flight height [m], Δh is elevation difference between two points on vertical object [m] and r_T and r_B is radial distance from the principle point (PP) to displaced image)

5.4 Mission planning

The first step in the planning process is to understand the client's needs ensuring that the overall products are aligned with those requirements. A wide range of products can be delivered including aerial images, orthophotos, digital elevation models, digital surface models, cross sections, point clouds and digital maps for GIS. Beyond the type of the products and their accuracy, factors such as the location of the project, the size, shape, topography and vegetation cover, the availability of GCP etc. will influence the procedures, costs and scheduling of surveys.

Therefore, the project planning can be organized into following categories:

- Selection of instruments and methodology to achieve the needed accuracy,
- Mission planning, and
- Ground control planning

Typically, the selection process starts with selecting the platform as well as imaging and navigation sensor. Regarding the platform, the payload capacity, range and degree of autonomy must be considered. There are two main types

of UAV platforms: fixed wings and multi-rotor. Both of them have advantages and disadvantages. Multi-rotor unmanned aerial vehicles (UAVs) are relatively easy to operate and provide satisfactory flight autonomy. Their capability for vertical takeoff and landing enhances sensor safety and allows for greater operational flexibility, as no extensive area is required for launch or recovery. These characteristics make them particularly suitable for surveys conducted in complex environments. Their main limitation, however, lies in restricted flight duration, which in turn constrains the area that can be covered.

Fixed-wing UAVs, on the other hand, benefit from aerodynamic efficiency, which enables longer range, extended flight endurance, and wider spatial coverage. Nonetheless, they require a runway-like surface for takeoff and landing, as well as skilled piloting to ensure safe recovery and to minimize the risk of damage to both the platform and its sensors. Fixed-wing UAVs are generally larger in size, support higher payload capacities, and are associated with higher acquisition and operational costs.

The GNSS/INS system provides a real time precise positioning and orientation which allows the UAV to fly along a predefined path even in windy conditions, guaranteeing sufficient image coverage and overlap. Both UAV based GNSS and INS units are optimized for size, cost and power consumption limiting the transportation of high quality devices like those used in the airborne camera or LiDAR sensors. Although the professional surveying UAVs are equipped with RTK/PPK GNSS and IMU that can achieve centimeter-level accuracy, many lower-cost UAVs, which are often used in data collection, rely on the less-accurate, single-frequency GNSS receiver leading to accuracy of the final product in meter or decimeter range. Modeling errors in sensor position due to low-cost GNSS receivers is as important as camera calibration in photogrammetry. Poor GNSS accuracy can cause distortions, misalignments, and errors in 3D reconstruction, affecting the overall spatial accuracy of the resulting product. Therefore, compensation for GNSS-induced errors—through error modeling, the use of ground control points (GCPs), or post-processing techniques such as PPK or RTK corrections—is just as essential as accurate camera calibration. In applications with lower metric accuracy requirements, the raw accuracy of direct GNSS/INS measurements may be sufficient.

In photogrammetric surveys the variety of cameras, from low-grade customer cameras to digital single-lens reflex cameras, can be used. The type of used camera and the image resolution can influence the final product accuracy. UAVs most often use non metric digital cameras with low quality lenses and shutters that are not typically designed for photogrammetric survey due to their light weight and low cost. Those cameras have good radiometric quality but low geometrical quality due to lens distortion. Wide lenses (shorter focal length) are generally recommended for photogrammetric surveys. A camera with a time-lapse function is required when operating most aerial platforms, unless the interval between photographs is manually controlled.

In order to achieve the homogeneous radiometric quality, the use of automatic white balance is not recommended since it can cause color shifts between images leading to inconsistencies in the orthophoto. The dominant use of automatic exposure control should be avoided. Exposure settings will change for each image, leading to varying brightness and colors, which reduce the quality of products. Additionally, if shutter speed is reduced (longer exposure time) and flight speed is increased, image might become blurred due to the UAV movement, affecting feature detection and image matching in the processing phase. The camera with manual settings will provide better control over exposure and focus. On the other hand, the manual control require understanding and proper adjustment of:

- exposure time - to avoid motion blur, exposure time should be shorter that the time required to cover one GSD,
- Sensor sensitivity - need to be adjusted to maintain correct exposure without noise,
- Aperture (F-number) - Lower F-number (larger apertures) allow more light to reach the sensor. This is useful in low-light conditions but may affect depth of field.

A typical survey with UAV systems requires a mission planning and GCPs measurement (required for georeferencing). The mission planning is an important step in photogrammetry as the image geometry has a strong influence on the quality of the resulting product. Mission planning includes analysis of: application, the study area, the sensor to be used (resolution and focal length), the flight characteristics (law and legal limitations, technical

limitations, flight height, flight speed, front and side overlap, camera orientation, resolution and accuracy), and GCP consideration.

Based on quality requirements (the expected resolution and accuracy) defined in line with clients' requirements, several parameters are defined before flight. The flight attitude determines the spatial resolution of images, flight duration, area covered and number of images per area unit. Flight attitude is influenced by the value of the GSD and the sensor parameters. The following equations are used to calculate the above ground level (AGL) and smallest value of both equation is chosen:

$$AGL_1 = \frac{f * GSD * HR}{SW}$$

$$AGL_2 = \frac{f * GSD * VR}{SH}$$

where f is the focal distance [mm], HR and VR are the horizontal and vertical resolutions of the sensor [px], SW is sensor width [mm] and SH is the sensor height [mm]. The low flight height results in high spatial resolution, minimizes the effect of the altitude error, reduces covered ground area, and increases the flight duration, data volume and processing time.

In flat or almost flat terrains UAV usually flies horizontally and maintains a constant height. In the complex terrain it is recommended to use terrain following i.e. to maintain a relatively constant AGL in each line since the vertical Root Mean Square Error (RMSE) can increase. The change in the distance between sensor and object of interest results in the overlap reduction and can become critically low in very steep areas with fewer images overall in steeper. Although terrain following provides more uniform spatial resolution the accurate terrain model is required for flight planning.

In conventional photogrammetry, a front overlap of 55 - 60 % and a side overall of 15 to 25 % is typically recommended. However, SfM benefits from image redundancy and a higher degree of overlap increases the accuracy of the generated product. Usually it is recommended at least 80 % of front overlap and 60-70 % of side overlap. Front overlap defines distance between consecutive images and it depends on shutter speed assuming that the flight speed is constant. Side overlap influences the distance between flight lines. Front (o_f) and side overlap (o_s) can be calculated using following expression:

$$o_f = \left(1 - \frac{d_f * f}{h * SH}\right) * 100$$

$$o_s = \left(1 - \frac{d_s * f}{h * SW}\right) * 100$$

where d_f is the distance between consecutive images [m], d_s is distance between flight lines [m], f is the focal length [mm], h is distance between the sensor and the object [m], SW and SH are the sensor width and height [mm] respectively.

There is a positive relationship between the image overlap and accuracy of the resulting product. The increase of the front overlap can significantly reduce the root mean square error (RMSE) while increasing the side overlap increases the time of the flight and data processing but has a lower impact on the height accuracy. However, exaggerated overlaps lead to increased processing time without improving the quality of the final product.

The UAV flight speed is an important parameter that affects the image quality and power consumption. In order to define the optimal UAV speed, it is necessary to analyse several factors such as flight speed regulations, the wind speed and direction, the camera's shutter speed, and vertical overlap. Maximum wind speed at which the UAV is sensitive is usually defined by the manufacturer since it increases power consumption. Additionally, the high wind speed tilts the UAV leading to large pitch and roll angles and decreases the overall UAV stability. However, the flight speed has a critical impact on the image quality, primarily due to motion blur. Motion blur, which reduces image sharpness and detail, can negatively impact the photogrammetry process. It is usually expressed as a percentage of the GSD. Roth et al. [26] suggest that flight speed should be determined using the following expression:

$$S = \frac{GSD * \delta}{l_t}$$

where S is the UAV speed [m/s], δ is the maximum motion blur [px] and l_t is shutter speed [s]. The same authors recommended motion blur to be kept below 50 % [26].

The flight pattern determines the path that the UAV will follow. It is usually designed as parallel flight lines as predefined. Flight patterns can be automatically generated in flight planning software by specifying a few basic parameters. However, the single look direction in gridded image blocks typically lacks oblique images and therefore sufficient geometric information in complex scenes, leading to artificial doming due to error accumulation in the SfM process.

Various flight configurations, such as single grid (**Figure 23** (a)), double grid (**Figure 23** (b)), circular mission (**Figure 23** (c)) or a combination of them have been used. Single grid missions are recommended for generating 2D products, while double grid missions are beneficial for generating precise 3D reconstruction.

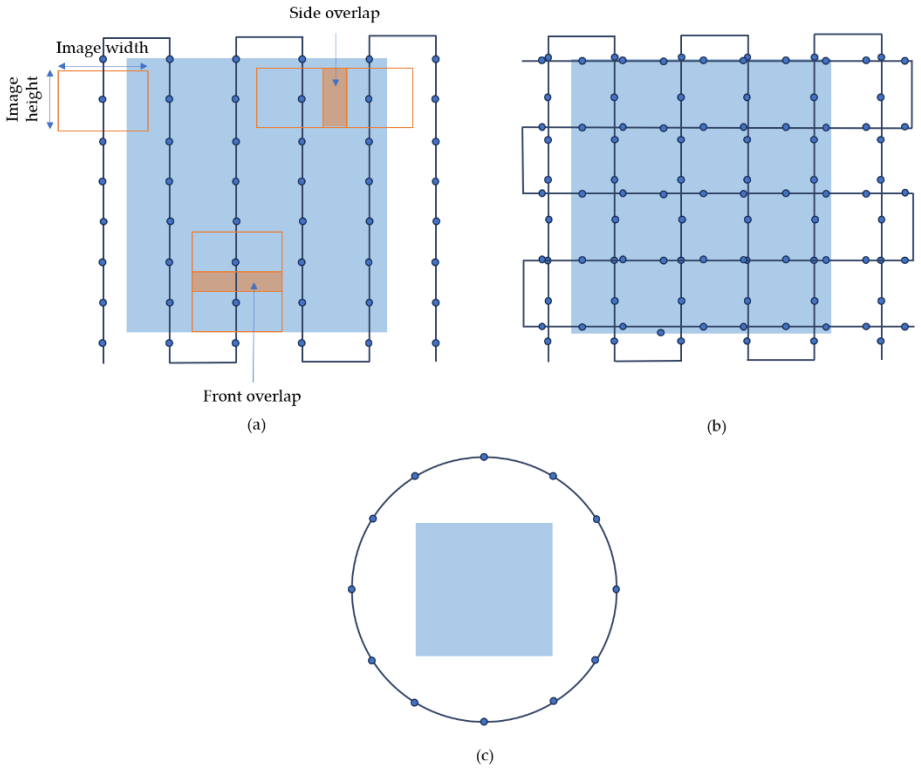


Figure 23 (a) single grid mission, (b) double grid mission, and (c) circular mission. The blue dots represent the position where the image will be collected. Frontal and side overlap are also shown.

The GCPs are used for georeferencing resulting data and to improve the estimation of intrinsic and extrinsic parameters of the camera in the SfM

process. The ground control involves several factors, such as the number and distribution of GCP and their accuracy.

GCP needs to be easily identifiable in an image, distinct from the surrounding area, and visible on multiple images. The size of the reference target needs to match the model resolution in order to allow precise identification on the images. Some software provides preconstructed coded targets as GCP, enabling automatic detection.

The accuracy of ground control points (GCPs) is primarily determined by the measuring instruments used. It is essential to ensure that a sufficient number of GCPs is employed to achieve the desired photogrammetric accuracy. Increasing the number of GCPs generally improves the resulting accuracy, but beyond a certain point, additional GCPs provide little to no further benefit. The spatial distribution of GCPs also plays a critical role: targets should be evenly distributed across the study area, both horizontally and vertically. A combination of edge distribution and stratified distribution is often considered best practice. In some cases, direct georeferencing—without GCPs—can be performed if the platform is equipped with a survey-grade GNSS/RTK receiver.

According to the ASPRS standard [27], several conditions must be satisfied in order to assess the product accuracy: the coordinates of the GCPs must be independently surveyed with at least three times higher accuracy than the tested product; at least 20 GCP (depending on study area size) should be used, regardless of the project size. For an orthophoto with an horizontal accuracy of 1 cm, the GCP points should be surveyed with horizontal RMSE = 0.25 cm and vertical RMSE = 0.5 cm. Clearly, RTK GNSS cannot deliver this level of accuracy. However, if the spatial resolution of the orthophoto is 15 cm, the GCPs should have $RMSE_{XYZ}$ of 2.5 cm, considering the required aerial triangulation $RMSE_{XYZ}$ of 7.5 cm (i.e., ½ of the orthophot's pixel size) and therefore RTK GNSS may suffice [27].

Table 6 shows the horizontal accuracy for planimetric data [27] and orthophotography. $RMSE_r$ is the radial accuracy, i.e. $RMSE_r = \sqrt{RMSE_X^2 + RMSE_Y^2}$, while **Table 7** presents the vertical accuracy for DEM

Table 6 The horizontal accuracy for planimetric data [27] and orthophotography

Accuracy class	RMSEz in non-vegetated terrain [cm]	NVA at 95 %	VVA at 95%	Appropriate contour interval supported by the RMSEz	MNRD [pts/m^2] / MNPS [m]
I	1.0	2.0	2.9	3 cm	20/0.224
II	2.5	4.9	7.4	7.5 cm	16/0.250
III	5.0	9.8	14.7	15 cm	8/0.354
IV	10.0	19.6	29.4	30 cm	2/0.707
V	12.5	24.5	36.8	37.5 cm	1/1.000
VI	20.0	39.2	58.8	60 cm	0.5/1.414
VII	33.3	65.3	98.0	1 m	0.25/2.000
VIII	66.7	130.7	196.0	2 m	0.1/3.162
IX	100.0	196.0	294.0	3 m	0.05/4.472
X	333.3	653.3	980.0	10 m	0.01/10.000

Table 7 Vertical accuracy examples for DEM [27] where NVA at 95 % is non-vegetated vertical accuracy at 95% confidence level, VVA is vegetated vertical accuracy at 95th percentile, MNPD is minimum nominal return density, and MNPS is maximum nominal pulse space

Accuracy class	RMSE$_z$ in non-vegetated terrain [cm]	NVA at 95 %	VVA at 95%	Appropriate contour interval supported by the RMSE$_z$	MNRD [pts/m^2] /MNPS [m]
I	1.0	2.0	2.9	3 cm	20/0.224
II	2.5	4.9	7.4	7.5 cm	16/0.250
III	5.0	9.8	14.7	15 cm	8/0.354
IV	10.0	19.6	29.4	30 cm	2/0.707
V	12.5	24.5	36.8	37.5 cm	1/1.000
VI	20.0	39.2	58.8	60 cm	0.5/1.414
VII	33.3	65.3	98.0	1 m	0.25/2.000
VIII	66.7	130.7	196.0	2 m	0.1/3.162
IX	100.0	196.0	294.0	3 m	0.05/4.472
X	333.3	653.3	980.0	10 m	0.01/10.000

5.5 SfM

SfM operates on the same principle as stereoscopic photogrammetry i.e. if a ground point, referred to as a feature, is identifiable in two or more overlapping, offset images, its 3D coordinates can be computed. However, there are fundamental differences compared to conventional photogrammetry since the geometry of the scene (K), camera position and orientation (R,t) are estimated simultaneously This is achieved through iterative bundle adjustment applied to highly redundant sets of matching features that are automatically extracted from multiple images.

5.5.1 SfM workflow

As mentioned, the SfM reconstructs camera orientation and scene geometry simultaneously through automatic identification of matching features in multiple images (**Figure 24**). Due to that, the key problem is that the SfM address is detection and marching features on images from different angles. SfM can be categorized to incremental methods and global methods. Incremental SfM starts with a minimal subset (two or three views) for the initial reconstruction and progressively integrates new images from associated 3D structure while performing bundle adjustment ensuring reconstruction accuracy. On another hang, global SfM solves all camera poses and 3D points simultaneously avoiding accumulation error. The estimated camera pose is then used to initialize triangulation followed by global bundle adjustment. Global SfM is faster but less robust than increment, especially if camera intrinsic parameters are not known in advance. SfM consists of several stages (**Figure 25**):

1) automatic identification of homogeneous features in individual images. Most commonly the Scale Invariant Feature Transform (SIFT) [28] algorithm is used.
2) Match corresponding features between images,
3) Choose two image that provide stable estimation of relative camera pose,
4) Estimate Essential matrix/Fundamental matrix and extract camera pose (R, t),
5) Construct 2D viewing rays from images,
6) Use triangulation to estimate 3D position and generate sparse point cloud,

7) Refine the SfM model (poses and point) using bundle adjustment, and
8) Use multi view stereo to densify the model generating a dense point cloud.

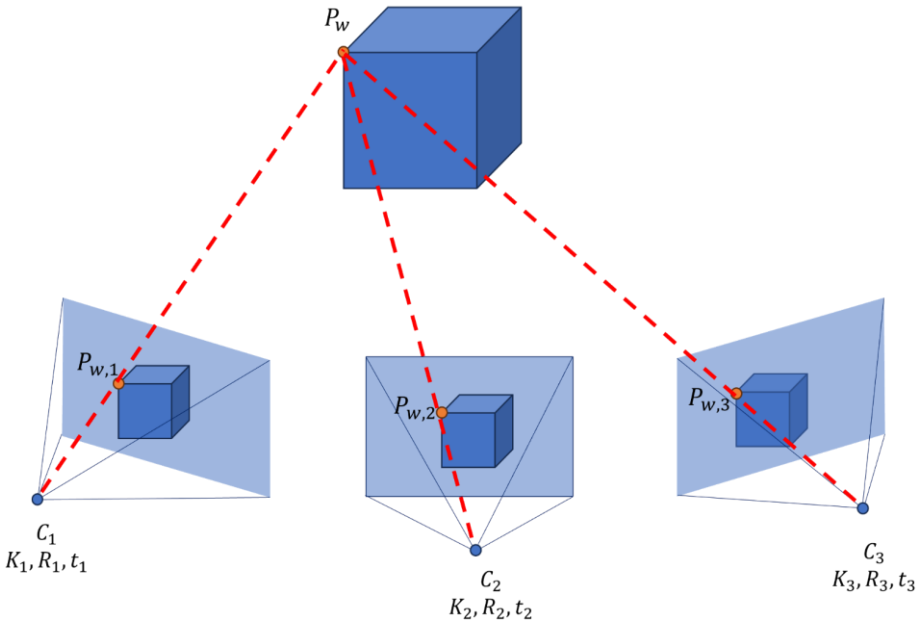


Figure 24 Structure from Motion

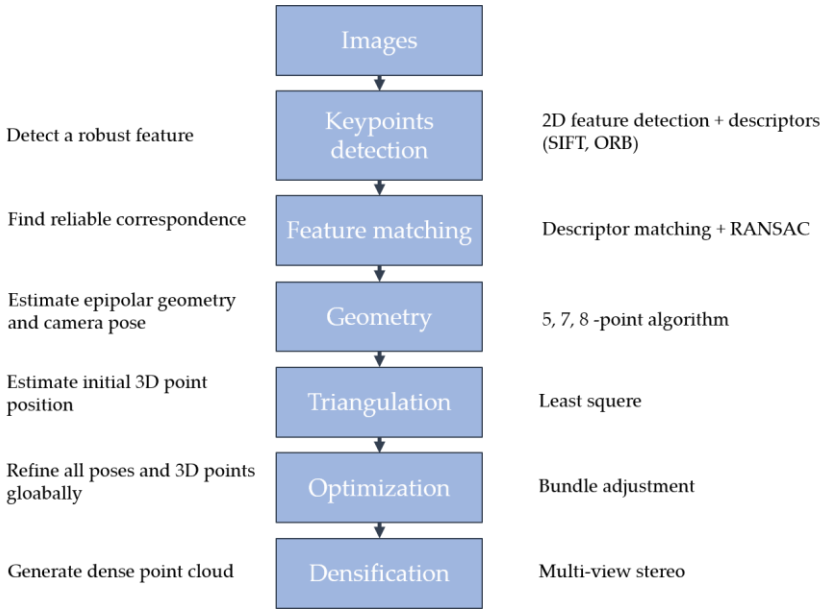


Figure 25 Structure from Motion workflow

5.5.2 SIFT

SIFT consists of four stages: feature detection, feature description, indexing and matching and model verification. The feature detection starts with detection of stable features across all possible scales. For each octave the initial image is repeatedly convolved with Gaussian functions ($G(k^i\sigma)$) to generate a set of blurred images, each separated in scale space by constant factor k. Adjacent Gaussian images are subtracted to compute the Difference of Gaussians (DoG) and this procedure is repeated for all octaves. The Gaussian image is down-sampled by a factor of 2, and the process is repeated (**Figure 26**). Local extrema of the DoG are found by comparing each pixel to its 26 neighbors in the 3x3 region at the current scale (8 pixels) and at adjacent scales (9 pixels per scale) (**Figure 27**).

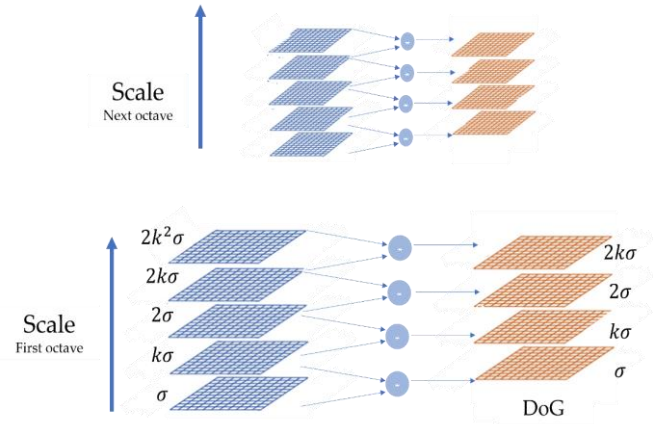


Figure 26 Convolving the initial image with a Gaussian to create a set of scales (on left) and subtracting the adjacent image to generate a DoG (on right)

If the pixel value is the maximum or minimum among all compared pixels, it is selected as a candidate for a keypoint. Since the location of an extremum is unlikely to coincide exactly with a pixel and is more likely to lie between neighboring pixels, sub-pixel localization is performed using a Taylor series expansion.

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x$$

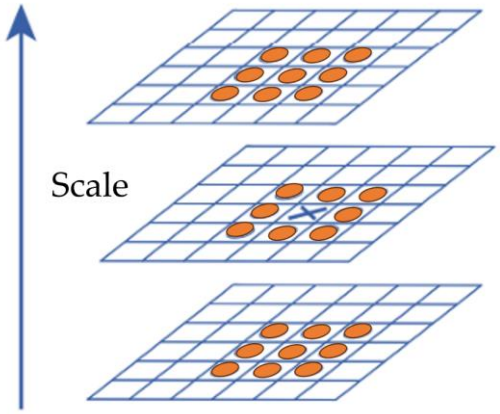


Figure 27 Detecting maxima and minima of the DoG image. The X represents a pixel that is compared with 26 neighboring pixels (marked with an orange circle)

The derivative is set to 0 to find the location of the extremum $\hat{x}$

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x}$$

The points with low contrast are generally less reliable than the high for feature points. Due to that $D(\hat{x})$ is used to filter candidate key points with low contrast by setting a threshold (discard all points with $D(\hat{x})$ < threshold). In addition, edge points that have high contrast in one direction and low in other, are filtered to ensure stability.

To achieve rotation invariance, a consistent orientation is assigned to each keypoint based on local image properties. An orientation histogram is created from the local gradient orientation of the sample point within the keypoint's neighborhood. The peak of the oriented histogram that corresponds to the dominant direction is identified, and the orientation and sum of magnitude is assigned to the keypoint (**Figure 28**).

Due to that, the keypoints are invariant to image rotation and scale and robust across a range of affine distortion, noise, and change in illumination conditions. The SIFT can automatically extract thousands of key points from images over all ranges of scales, ensuring robustness even in extracting small objects in a cluttered environment.

Assigning an image location, scale, and orientation for each key point is followed by the computation of a local feature descriptor which needs to be invariant to those transformations. The created descriptor needs to be highly distinctive to allow features to be matched in large datasets but invariant on changes in illumination or 3D viewpoint.

To compute the descriptor, for each keypoint 8x8 neighborhood from DoG levels is identified (Figure 27) and subtracted by the orientation of key points (i.e. align orientation of neighborhood to x-axis). After that gradient magnitude and orientation at each image sample point in the region around the keypoint are calculated and weighted by Gaussian, Sum of the weighted gradient magnitude at the near direction and orientation histogram for each 4x4 region is created. The histogram array represents the image descriptor.

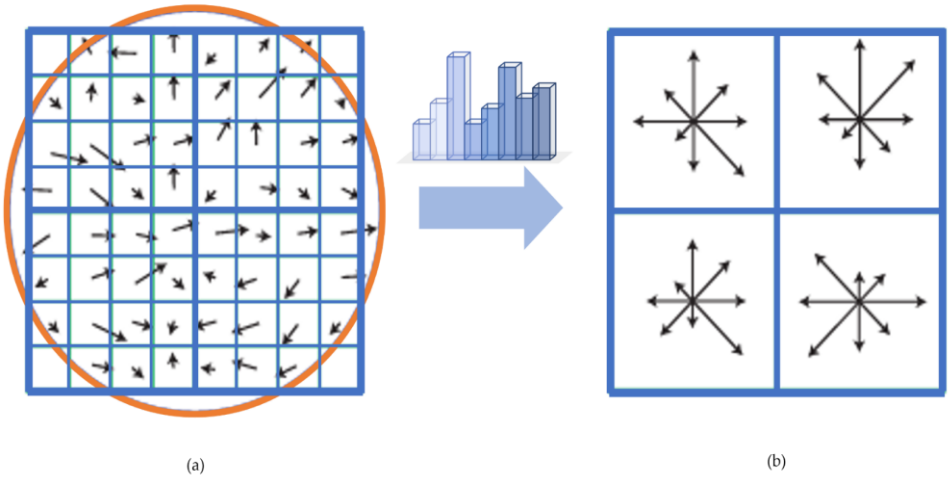


Figure 28 Feature description (a) image gradient magnitude and orientation (b) keypoint descriptor

The number of detected key points is a function of the number of scale samples, image texture and resolution. Therefore, the sharpness, textures, and resolution of images determine the quality of the resulting product. The higher spatial resolution, the higher number of keypoints will be detected and therefore the higher quality of products.

Feature matching for feature in image A is performed by finding its nearest neighbor in image B. Since nearest neighbor search may be too slow for large databases the SIFT uses a best-bin-first algorithm that returns the closest

neighbor with high probability. Descriptor vectors may match more than one reference pose database. For each pose 4D Hough transform is used to identify clusters. Each key point votes for all pose(s). The correct interpretation is ensured by counting votes for the same pose of an object within clusters of keypoint . In the end each keypoint contains 4 parameters: 2D location, scale, orientation and each matched keypoint within the database.

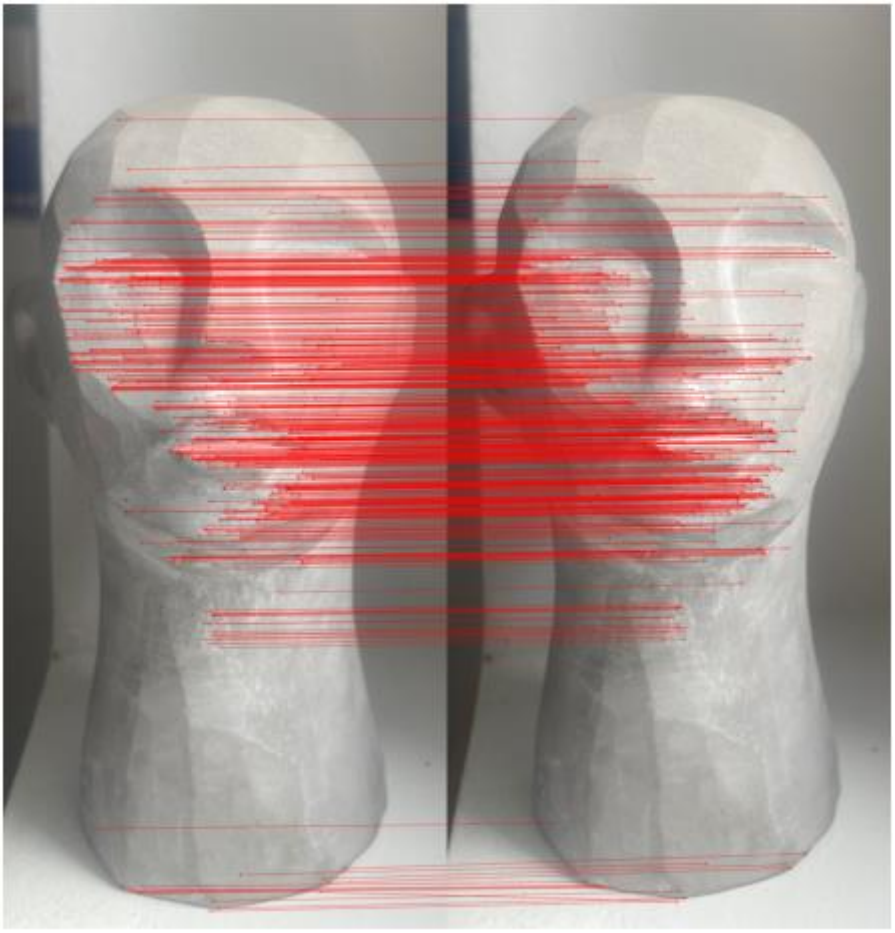


Figure 29 SIFT feature matching

Matched points are based on the assumption that the features have a similar appearance in both images, and due to that this process usually results in outliers (i.e. wrong matches due to repetitive features, change in viewpoints, image noise, occlusion, blur etc). For accurate 3D reconstruction, outliers must be removed. Matching of corresponding points between two images is a 2D search problem (**Figure 29**). However, it can be reduced to a 1D search by

using an additional constraint that the corresponding points must lie on an epipolar line on another image.

5.5.3 Epipolar geometry

In multi-view geometry, the relationship between the camera position, a 3D point, and its projection onto the image plane is known as epipolar geometry. The single point P_w is observed from two different viewpoints (C_1, C_2). The line that connects the camera's center represents the base line. The camera centers C_1 and C_2 and point P_w will lie in a single plane, known as the epipolar plane. The real point P will be projected onto the image planes of the two cameras P_1 and P_2. The intersection of the epipolar plane and image plane is an epipolar line (l_1 and l_2). The point in which the baseline (the distance between two camera centers) intersects the two image planes is known as the epipole e_1 and e_2. Moreover, the epipolar lines intersect the baseline in the epipole, and all epipolar lines intersect in the epipole (**Figure 30**). In practice, it is more efficient to coincide the scanlines with epipolar lines, making the correspondence search very effective.

Rectification is the process of projecting each image onto a common plane, parallel to the baseline, by rotating the original cameras about their optical centers. If image planes are parallel to each other (i.e. there is no relative rotation between cameras), the baseline will be parallel to the image plane, epipoles will be located at infinity and the epipolar lines are parallel to an axis of each image (usually x-axis). Since epipolar lines are horizontal and parallel, the corresponding point will be located along the horizontal lines (i.e. they must have the same y coordinate) of the rectified image and search becomes faster and computationally less expensive.

The horizontal displacement between corresponding points is called disparity. The disparity associated with each corresponding pixel of the image is called a disparity map. It is a grayscale image without any texture. Disparity is inversely proportional to the distance from the camera. The objects that are closer to the camera will have larger disparity and will appear brighter on the disparity map.

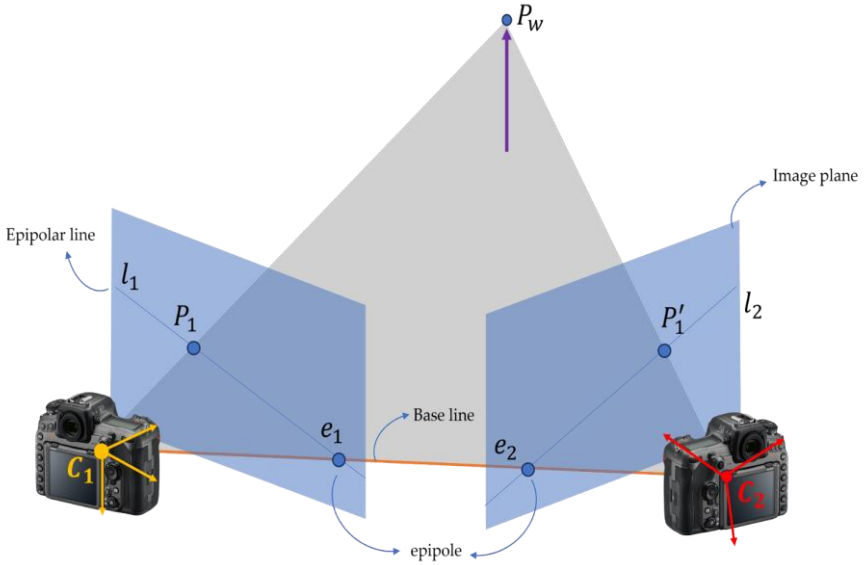


Figure 30 Epipolar geometry

Epipolar geometry depends only on the relative pose (position and orientation) and internal parameters of the two cameras, i.e. the position of the camera centres and image planes. It does not depend on the scene structure (3D points external to the camera). However, the exact location of point P_w is not known, but its projection to image planes can be determined. Based on that and known camera location (C_1, C_2) the epipolar plane can be defined. Epipolar planes and image planes will determine the epipolar lines. By definition, potential matches for P have to lie on the corresponding epipolar line , l_2 and potential matches for $P^{'}$ have to lie on the corresponding epipolar line l_1. Due to that, it is possible to determine strong constraints between image pairs.

The epipolar constraint can be expressed by

$$P^T E P^{'} = 0$$

where E is the Essential matrix and P and $P^{'}$ are conjugate points in the image coordinate. The E is defined by the following expression

$$E = [T_x]R$$

where $[T_x]R$ represents the relative pose of camera one with respect to camera two.

It has five degrees of freedom and it is completely defined by 3 rotations and 2 translations.

Moreover, the relationship between the corresponding set of points in two images from different views can be written as

$$P^T F P^{'} = 0$$

where F is a 3x3 fundamental matrix, which is similar to the Essential matrix, but it contains the 7 degrees of freedom i.e., it contains information about camera matrices (K_1 and K_2), relative translation T and rotation R between the cameras i.e. $F = K_2^{-1}[T_x]RK_1^{-1}$.

The epipolar line along which the correspondent points in the image must lie is expressed by $P^{'T} l_2 = 0$ where $l_2 \simeq F$ (The epipolar line on the second image is a function of the point P_1 in the first image).

The relationship between the Fundamental and Essential matrix is given by

$$E = K_2^T F K_1$$

5.5.3.1 Eight-point algorithm

A fundamental matrix can be estimated if the number of corresponding points between two images is higher than 8 without knowing extrinsic and intrinsic parameters by using the eight-point algorithm. The correct point correspondence is essential for fundamental matrix estimation. Each pair of correspondent points $P = (x, y, 1)^T$, $P^{'} = (x', y', 1)^T$need to meet epipolar constraint $P_1^T F P_2 = 0$ resulting in a homogeneous linear system, i.e.

$$[x \quad y \quad 1]\begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix}\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = 0$$

Each corresponding pair will result in one independent equation, given as

$$x'xf_1 + x'yf_2 + x'f_3 + y'xf_4 + y'yf_5 + y'f_6 + xf_7 + yf_8 + f_9 = 0$$

Since this constraint is linear, it only constrains one degree of freedom. Due to that, we need a minimum of eight correspondences to determine the Fundamental matrix, resulting in a homogeneous linear system with nine unknowns. It is given as following:

$$\begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_8x_8 & x'_8y_8 & x'_8 & y'_8x_8 & y'_8y_8 & y'_8 & x_8 & y_8 & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \end{bmatrix} = 0$$

This can be compactly written as

$$Af = 0$$

The least-squares solution of F is computed by performing singular value decomposition (SVD) on image coordinates. To do so, we *minimize* $||Af||^2$ subject to $||f||^2 = 1$ where A is an 8 x 9 matrix built from the correspondences and f is a vectorized form of the F matrix

The eight-point algorithm is extremely sensitive to outliers, i.e., for small variations in input variables, there are large changes in the F matrix. Due to that, the estimated F matrix may not be precise. The main challenge is that the correspondence keypoint will have large values for coordinates, such as $P_i = (1758, 2048, 1)$. If the keypoints are located in the same part of the image than both vectors P_i and $P_i^{'}$ will be similar, and therefore, A matrix will have one very large singular value. To solve this problem the normalized image coordinates can be used. First the origin is shifted to image center (translation) and distance of image point from origin is scaled by the factor $\frac{\sqrt{2}}{mean\ distance}$. The matrix F is estimated by using a regular least-squares eight-point algorithm and then denormalized results to obtain the F matrix for regular coordinate space.

5.5.4 RANSAC

The Random Sample Consensus (RANSAC) algorithm is used to obtain a better estimation of the fundamental matrix. RANSAC [29] is an algorithm for robust fitting of models that uses a minimal number of points from which a model can be computed (for example minimal number of points to compute line is 2), rather than using all data points and then enlarge this set with points that fit with a predefined tolerance (inliers). It is very efficient at finding the

inlier set even in the presence of a large number of outliers. Due to that, RANSAC is applied to compute Fundamental matrices to detect the uncorrected correspondences, matched by SIFT algorithm, and only point pairs that satisfy the epipolar constraints are used.

RANSAC loop consists of six steps (**Figure 31**):

- Randomly select n points,
- Calculate the model parameters that fit the data in the sample,
- Calculate the residual error for each data point,
- Select the data that support current hypothesis,
- Repeat this process from 1 to k times, and
- Select the transformation with the maximum number of inliers obtained within *k* interactions.

The higher the number of interactions k, the lower the number of outliers will be. The number of iterations can be determined by using an expression

$$k = \frac{log(1 - p)}{log(1 - w^n)}$$

where *p* is the probability of finding set of point that don't contain the outliers, *w* is the proportion of inliers in the data, and *n* is the number of points needed to estimate the model. For example, if we assume that 50 % of the corresponding points are inliers ($w = 50\%$), and the desired probability is $p = 99\%$. The needed number of interactions for the eight-point algorithm is 1177. So the number of corresponding points does not influence the number of iterations. Since RANSAC is based on the random hypothesis generation process it is non-deterministic, i.e., for each run, different results will be obtained.

The RANSAC is very effective at finding inlier, even in the presence of a high number of noise. Due to that, the eight point algorithm and RANSAC are applied together for SfM to separate outliers and provide the best estimation of the fundamental matrix.

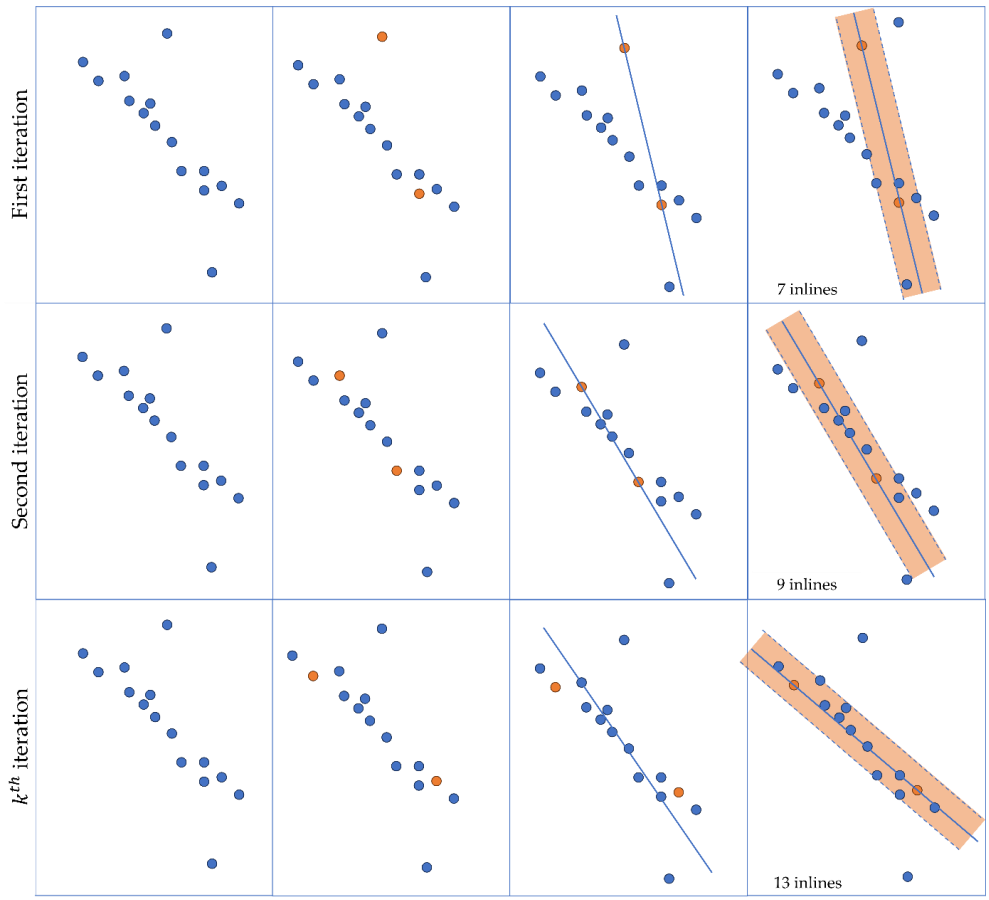


Figure 31 RANSAC algorithm steps

As already mentioned, these inputs are the corresponding key points created by the SIFT algorithm that need epipolar constraint. First, the eight points are randomly selected in order to estimate the fundamental matrix. The model is tested against all other points and those that fit (i.e. inliers) support this model. The model is repaired k times and the model that has the most inliers will be selected.

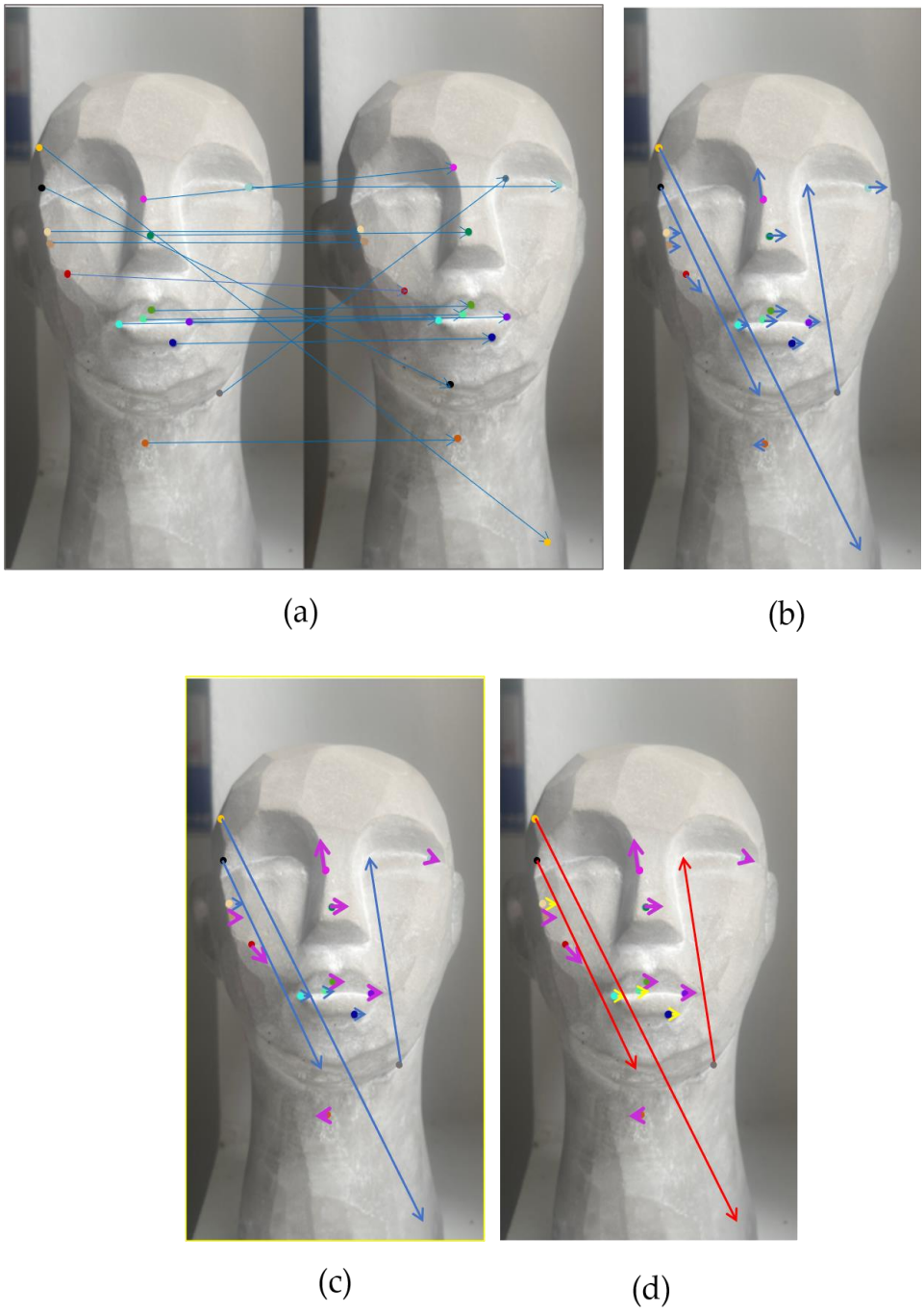


Figure 32 One interaction of eight point RANSAC (a) the image pair and corresponded key points donated by arrow, (b) overlapping keypoint from image 2 to image 1 (the arrows donate the motion vector of keypoint) (c) randomly selecting 8 corresponding points (marked by magenta vector) and using them to estimate the F (d) using estimated F matrix to detect inliers (yellow arrow) outliers (red arrow).

First, a viewing ray for that feature must be reconstructed from each image. A viewing ray can be defined as the line from the feature, passing through the projective center of the camera to the corresponding pixel in the image sensor. Second, considering that we know the orientation and position of the camera, the distance of the feature (and its coordinate) can be computed by calculating the spatial intersection of several viewing rays (**Figure 32**).

5.5.5 Stereo vision and triangulation

Stereo vision is a technique for 3D reconstruction based on obtained disparity between corresponding pixels from two images taken from slightly different viewpoints, and then apply the principle of triangle similarity to calculate depth information between object and the cameras. It is based on a binocular vision that our brain uses to perceive depth from the left and right image i.e. we perceive depth by using disparity from the left and right eyes, resulting from the eyes' horizontal separation.

Stereo vision consists of two main steps: matching and reconstruction. Matching resolves the correspondence problem i.e., finding the corresponding pixel between images, which is a challenging task due to variation in illumination, blurring, and noise. The process of detection features, finding correspondents and estimation of the fundamental matrix is already explained in previous sections. Reconstruction uses the camera intrinsic parameters (K and $K^{'}$) and extrinsic parameters (R and t) to reconstruct a viewing ray (l and $l^{'}$) defined by camera center and image locations (C_1 and C_2). The 3D location of the point P^w can be computed as the intersection of l and $l^{'}$ based on triangulation. On the **Figure 33**, the position of point P^w is defined by (x, y, z) coordinates, the P^w is observed by camera with pose C_i and p_i^w represents the image-plan projection of observed point while x_1 and x'_1 are pixel coordinates in the left and right image respectively. Based on geometry, the depth of the point can be computed from the properties of similar triangles

$$\frac{Z}{b} = \frac{Z - f}{b - x_1 + x'_1}$$

where f is focal length, b is baseline.

Due to feature uncertainty, precision of camera calibration, noise and numerical errors, they will not intersect exactly. Instead, triangulation usually computes the best approximation of a 3D point, by minimizing the distance between viewing rays. Different algorithms such as linear triangulation, midpoint method or nonlinear triangulation can be used.

The resulting 3D points are estimated locally from a subset of views and therefore are not globally consistent.

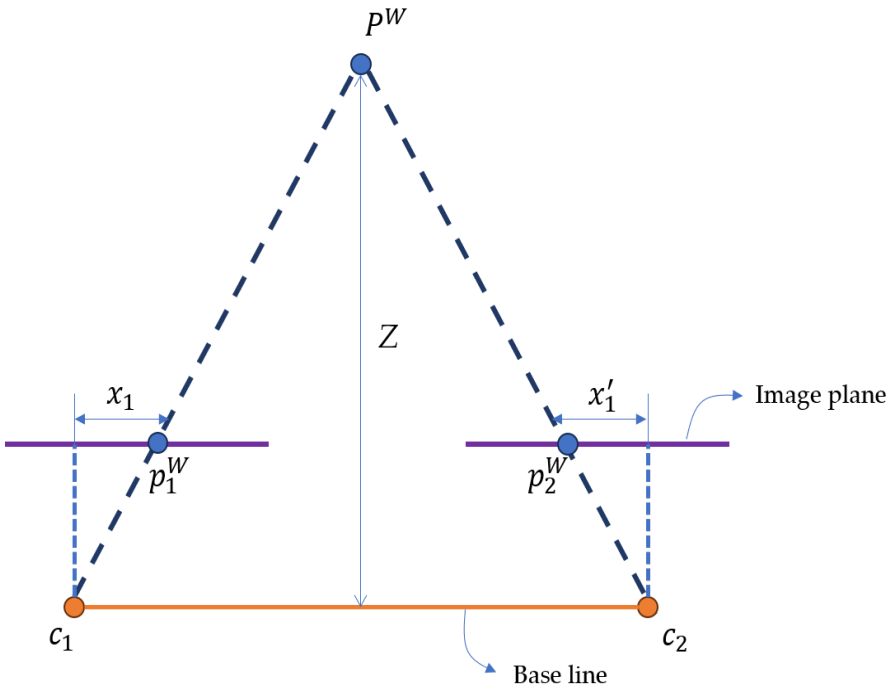


Figure 33 Triangulation with rectified images - top-down view

SfM uses similar principles as stereo and triangulation across multiple images to build a global sparse point cloud of the scene. The aim of the stereo vision is to reconstruct the 3D from images assuming that K, R, t are known while SfM aims to reconstruct 3D scenes and estimate camera poses simultaneously from multiple images. It can be categorized into incremental, global and hybrid approaches. In incremental SfM, the process starts with selection of the initial pair of images. There are several factors that need to be considered when choosing the stereo paras for initial SfM estimation: (1) if almost identical views are selected it will result with high uncertainty in triangulation, (2) very different view will lead to low overlap and high camera uncertainty (3) the larger baseline often provides better triangulation

accuracy. Usually, the two views are selected based on the maximum number of triangulation points (many overlapping cameras) for initial reconstruction. The selection of the initial pair is crucial for SfM quality of reconstruction and performance. After that the 8-point algorithm (or 5-point if using Essential matrix) is applied to estimate the F or E matrix followed by triangulate 3D points from corresponding points in the two images forming initial sparse 3D reconstruction and the pose of two cameras. Then the next image is added to the initial reconstruction sequentially. Based on given 2D-3D correspondence (from existing 3D points that are visible in the added image) the new camera pose is estimated followed by triangulation to compute new 3D points and optimization of existing points that are visible on this image. This process is repeated incrementally until all images are registered and the spares point cloud is completed. The application of incremental SfM for reconstructing large-scale scenes is not always ideal due to drift (due to accumulation of errors) as the number of images increases leading to low efficiency and time-consuming repetitive bundle adjustment.

In contrast to incremental SfM, global SfM solves the position of all images simultaneously using the view graph as input. The process starts with image-based feature extraction and matching followed by pairwise pose estimation and construction of the initial view graph of the input images where each node represents the camera and each edge links cameras that have enough matching points (relative orientation). The problem is usually solved in two separate steps i.e. radiation and translation averaging steps. The all relative rotations are used to compute global orientation for all cameras by applying rotation averaging that solves global rotation so that the inconsistency between relative and global rotation are minimized. After that, translation averaging is performed to determine the global camera position that is maximally consistent with pairwise relative translation.

The increment methods provide highly accurate and robust results, but it is computationally intensive. Global SfM is much faster and efficient, but does not as effectively remove outliers, resulting in lower accuracy and robustness.

The SfM is subject to some unique ambiguities since it tries to recover both object structure and the camera motion without any prior knowledge. First of all, ambiguity in object shape due to small viewpoint variation. If camera intrinsics are not known or not used in reconstruction, the resulting model is ambiguous by arbitrary 3D projective transformations. Due to that,

reconstructed models may appear distorted i.e. lines may not remain parallel, and angles may not be preserved. To resolve projective ambiguity, a calibrated SfM approach or self-calibration can be used. If constraints on the camera calibration matrix or scene are Affine, ambiguity can be observed as more restricted projective ambiguity when the SfM assumes an affine camera model (orthographic projection). In this access, reconstruction is only accurate up to an affine transformation, i.e., it preserves parallel lines and volume ratio but not true angles and lengths.

The next most often ambiguity is the scale (for a perspective camera) or depth (for an orthographic camera) ambiguity. It has some unique disadvantages such as ambiguity in the absolute scale of the scene that cannot be determined. Without a reference measurement, it is impossible to recover the absolute scale of the scene. For example, the bigger object at a longer distance and the smaller object at a closer distance may yield the same projection.

5.5.6 Bundle adjustment

Bundle adjustment is an optimization process that simultaneously adjusts all camera poses and all 3D feature coordinates (scene geometry) to minimize the total projection error in order to provide the most accurate reconstruction of structure and motion. The "bundle" refers to viewing rays that are adjusted optimally together in one bundle. The camera model and estimated camera pose can be used to reproject the estimated 3D feature coordinates onto the image plane. The reprojected error is the image-plane distance between the reprojected and observed position of the feature in the image plane (**Figure 34**).

It optimizes the structure and motion (R and t) by minimizing the sum of squared reprojection error, and it is commonly used after least square estimation of R and t (after the 8-point algorithm). I.e., bundle adjustment extends two-view reprojection minimization to multi-view scenarios. Since not all corresponding points are visible for each camera, it calculates the reprojection error for only the observations that are visible from all cameras. The most common approach is the Levenberg-Marquardt algorithm. It combines the Gauss-Newton algorithm and gradient descent, resulting in fast convergence and robustness.

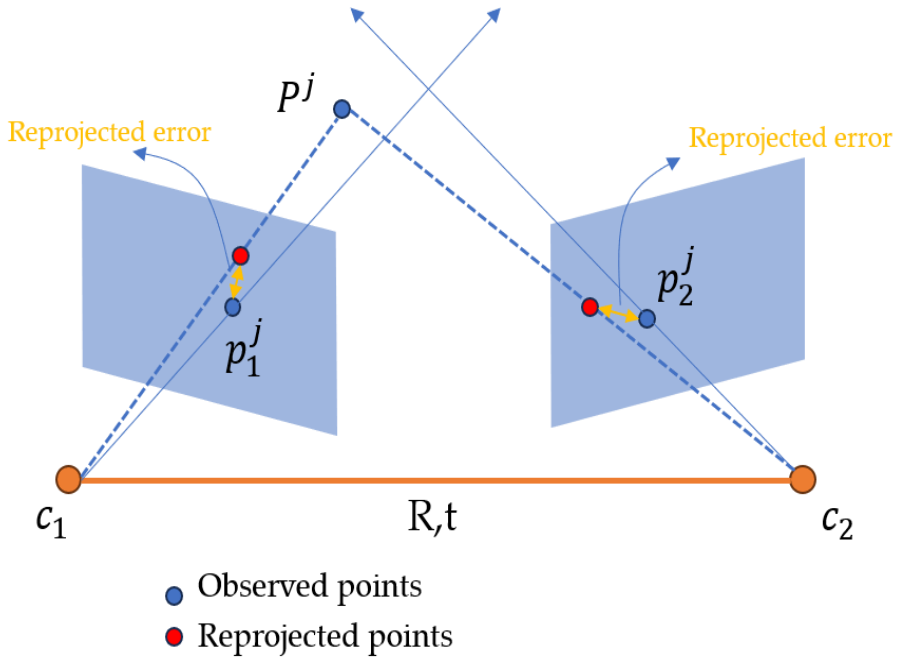


Figure 34 Bundle adjustment

To formalize the multi-image alignment, let's consider the N different camera poses with known camera parameters and a collection of n feature points. The camera at the pose C_i observes point P^j and image-plane coordinates are represented by p_i^j The estimated value of the image-plane projection of the jth feature point in the i^{th} image donated by P_i^j. The refined position of the feature point $\widehat{P^j}$ is calculated by minimizing the sum of reprojection errors, i.e.:

$$\widehat{P^j} \simeq arg \min_{p^j} \sum_{i,j} \left|\left|\pi(C_i \cdot P^j, K_i) - p_i^j\right|\right|^2$$

The number of unknown parameters in this system is $6 \cdot N + 3 \cdot n$, i.e., six unknowns for each camera pose and three unknowns for each feature point. Since one camera is considered the reference, the number of unknowns is is $6 \cdot (N-1) + 3 \cdot n$, while the number of measurements is $2 \cdot N \cdot n$, as the projection of each feature point onto the image plane is measured.

Bundle adjustment calculates the optimal relative pose and position not absolute position. In addition, scale ambiguity also applies here since changes in focal length and z-axis translation.

5.5.7 Multi-view stereo

In SfM the sparsity of feature matching points leads to spares point cloud. However, for most applications highly detailed and robust 3D point clouds are beneficial. To overcome this limitation, the multi-view stereo technique is often used to improve reconstruction.

Multi-view stereo has the same principles as the classical stereo, but it benefits from a large number of images and images with more varied perspectives. The multi-view stereo can be formulated as the simultaneous estimation of depth maps at key frames while optimizing not only photometric consistency (intensity or color similarity) and smoothness consistency but also the geometrical consistency (consistency of estimated disparity across multiple views). It provides more accurate depth maps and complete dense 3D scene models since it attempts to find a match for almost all pixels in an image. For example, if pixel size on the ground level (ground sampling distance) is 5 cm for state-of-the-art systems, a point density of hundreds of points per square meter can be achieved. This is much higher compared to the traditional LiDAR point clouds.

In SfM, multi-view stereo can be observed as a post-processing stage. The process began with taking the camera parameters and the sparse model obtained by SfM as input. After that, each image is considered as a reference and pair selection is performed where for each reference image, a subset of neighboring views with good overlaps and viewing angles is selected to improve depth estimation quality. A high number of matching points across images is essential, along with large angles between the sparse 3D points and the optical center of the image, to ensure accurate and stable reconstruction. The multi-view algorithm is used to compute the depth map by comparing multiple neighboring views with a reference image. This involves pixel-wise or patch-wise photogrammetric matching, where each pixel is matched across the views to find its most likely depth estimation using stereo principles such as plane-sweeping stereo, patch matched stereo or deep learning based MVS. The depth maps from multiple views are merged, and dense point clouds are obtained. This fusion step involves filtering out noise or inconsistent depth estimation and combining overlapping measurements to create a unified 3D surface.

6 LiDAR

Light Detection and Ranging (LiDAR) is a remote sensing technology that uses laser pulses to measure the distance between the sensor and target. A laser (Light Amplification by Stimulated Emission of Radiation) is a device that generates a coherent beam of light through the process of stimulated emission. When atoms or molecules in the active laser medium (gain medium) absorb external energy, their electrons transition to an excited state, moving to higher energy levels.

Each electron orbit corresponds to a specific energy level. For an electron to move to a higher orbit, it must absorb a photon with an energy exactly equal to the difference between its current and target energy levels. However, electrons do not remain in an excited state for long; they quickly return to their ground state, releasing a photon. The emitted photon is identical to the photon that was absorbed. This emitted photon can then stimulate other excited atoms to release additional photons of the same wavelength, phase, and direction. This chain reaction leads to the amplification of light, resulting in a powerful, coherent laser beam.

The key characteristics of a laser are:

- Coherence – laser beam exhibits spatial and temporal coherence, meaning that light waves have a constant phase over both space and time. This is an important property since high coherence results in an extremely high power.
- Monochromatic – A laser emits light in a single wavelength (or very narrow range of wavelengths), unlike ordinary light sources that consist of multiple wavelengths.
- High intensity – in contrast to natural light that spreads in all directions, laser beams are highly directional and spread minimally over long distances; the intensity of the laser beam reaching the target is high. This enables different applications such as cutting, welding, and long-distance measurement. In remote sensing, high intensity provides a high-flying height, ensuring that enough energy returns from the target to the detector.

- Collimation – Laser beams are highly directional and spread minimally over long distances, making them useful for applications that require precision. The narrower beam ensures better range accuracy.

6.1 Principles of LiDAR

The laser scanners can be categorized by the principle of the distance measurement on:

- time of flight principle
- phase-based principle.

Time of flight (**Figure 35**) measures the difference in time between emitting a pulse and detecting its return (i.e. echo). The distance to the target of interest can be determined by the following expression:

$$D = \frac{c \cdot t}{2}$$

where D is the distance between scanner and target object that reflects the laser pulse (range), c is the speed of light, and t is the time-of-flight.

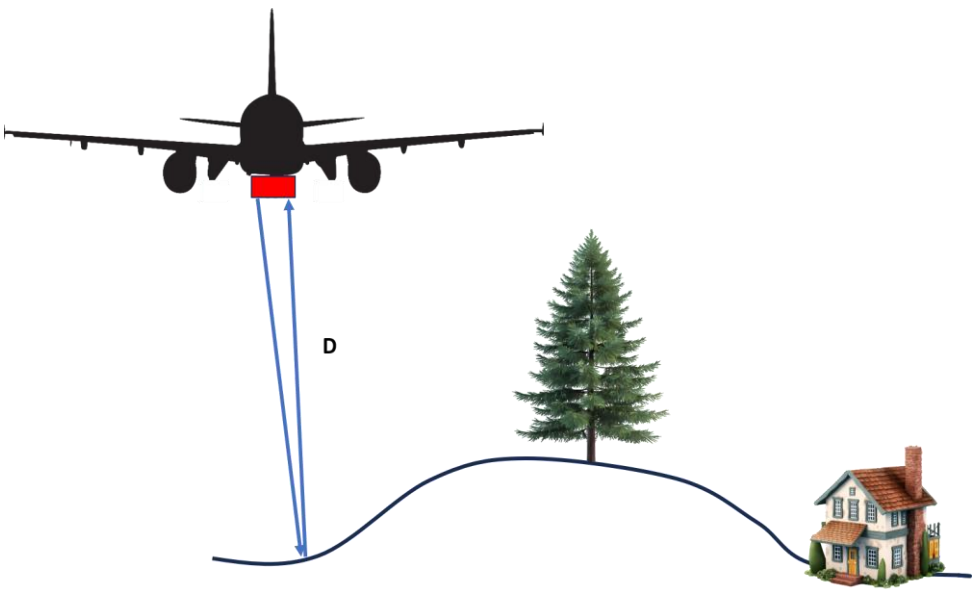


Figure 35 LiDAR distance measurement based on the time-of-flight principle

These scanners use discrete laser pulses for distance measurement and they are commonly known as pulse scanners. The pulse frequency is limited because the transmitter cannot send the next pulse until the receiver detects the previous one. Electronic measurement of time is crucial for this type of distance measurement. For instance, since the speed of light is approximately 300 000 km/s, it can be calculated that 3 nanoseconds (ns) are required for light to travel 1 meter. This means that to achieve a length measurement accuracy of 1 millimeter, it is necessary to measure a time interval of just 3 picoseconds (ps).

Phase-based principles (**Figure 36**) modulate the amplitude of the emitted laser in sine-wave-like patterns. When the laser light is reflected by the object and received by the scanner, the waves seem to be delayed or shifted compared to the waves that are currently emitted. This shift is the basis for the distance measurement; it is directly proportional to the distance of the object.

The x, y, z coordinates of each point are then calculated by using angle encoders to measure the mirror rotation and horizontal rotation of the scanner i.e.

$$t = \frac{\Delta\phi}{2\pi \cdot f_m};\ D = \frac{c \cdot t}{2} \Rightarrow D = \frac{c \cdot \Delta\phi}{4\pi \cdot f_m}$$

Where the c is the speed of light, t is period of time from laser emission to detection of the reflected signal, ϕ is phase shift and f_m is a modulated frequency.

As sine waves form an identical repeating pattern, ambiguity can arise if the phase shift is larger than one complete cycle of the wave. Because the sine waves repeat, any actual phase shift that is larger than one sine wave will appear identical to a corresponding phase shift within a single cycle.

Therefore, in this type of system max distance is limited to that in which phase shift is smaller than one sine wave. The equation for max distance that guarantee the uniqueness of results is:

$$z_{amb} = \frac{c}{2 \cdot f_{modulated}}$$

Meaning that if the wavelength is 300 m the max distance from objects is 150 m. Therefore, the long sine waves are used to avoid this ambiguity.

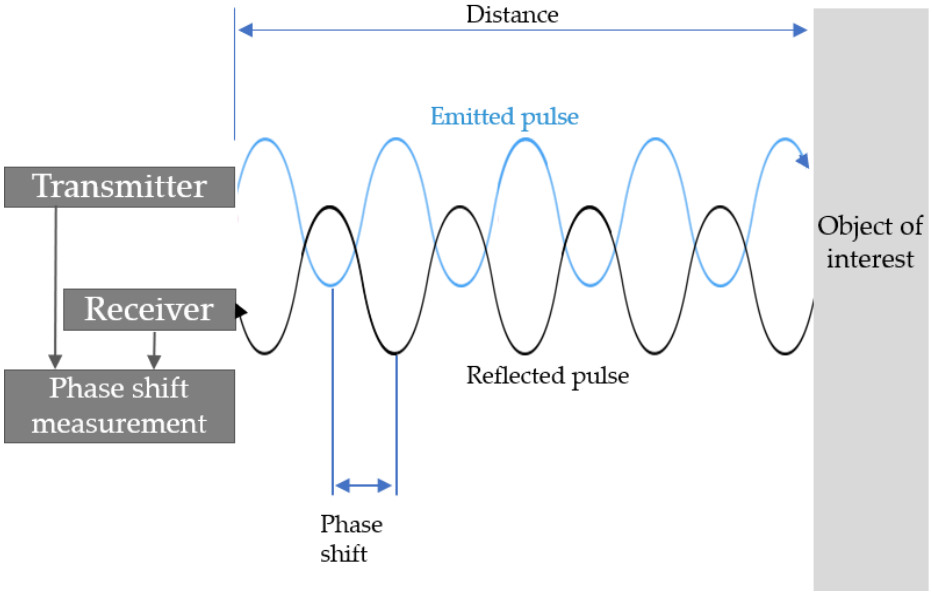


Figure 36 Phase-based distance measurement

Unfortunately, it is more difficult to determine the phase shift accurately if the sine waveforms a long slope. It can be determined more accurately if the slope is short and steep. This means, for accuracy reasons, short sine waves are preferred. To provide a larger max distance two or more frequency modulations are used. An approximate measurement is made based on the low frequency i.e. large wavelengths, and then a precise measurement is made by using high frequency. The largest wavelength provides an unambiguous measurement, while the shortest wavelength defines the precision that can be provided. Scanners based on phase-time measurement are faster and have a better resolution but lower precision compared with the time of flight-based scanners.

The accuracy of phase-based scanners is limited by several factors:

- Frequency Modulation: Variations in frequency modulation can introduce errors in measuring the phase shift, affecting the overall accuracy of the distance measurement.
- Accuracy of phase shift measurement: The precision of the phase shift measurement directly impacts the accuracy of the distance

calculations. Any errors in this measurement will influence the final distance result.

- Stability of the oscillation modulation: The stability of the modulator that generates the oscillation is crucial. Fluctuations or instabilities in the modulator can lead to variations in the emitted signal, causing inaccuracies in the phase measurement.
- Air turbulence: Air turbulence can distort the propagating wave, leading to variations in the phase measurement.

LiDAR scanners can be mounted on a static (tripod) or dynamic platform. When a scanner is placed at one position during data acquisition, that is static laser scanning. If the scanner is positioned close to the Earth's surface, this method is known as terrestrial laser scanning (TLS). The advantage of static laser scanning is high precision and high density of the resulting point cloud.

In contrast, dynamic laser scanning the device is mounted on a mobile platform. The examples of platforms for dynamic laser scanning are plains, cars, Unmanned aerial vehicle (UAV), trains etc. This approach allows for the rapid collection of data over large areas, although it may result in slightly lower precision compared to static scanning.

Additionally, handheld LiDAR scanners are becoming increasingly popular and even some of the latest smartphones have a LiDAR sensor. While handheld scanners may offer lower precision than tripod-mounted systems, their ease of use, due to their compact size and effectiveness, makes them ideal for use in certain situations. Making them valuable tools in various applications.

6.2 Components of LiDAR

The LiDAR systems are used for rapid measurement of the Earth’s surface, achieving a sampling rate greater than 150 000 pulses per second. The resulting product is a highly dense and accurate georeferenced point cloud. The three-dimensional coordinates (e.g., X, Y, Z or latitude, longitude, and elevation) of the target object are computer-based on the: (1) time of flight, (2) the angle at which the pulse was emitted, and (3) the absolute location of the sensor on or above the Earth's surface.

To ensure high accuracy, it is essential to determine these components within a centimeter or so. This presents a challenge, as aircraft typically fly at speeds of 160 to 320 kilometers per hour, while experiencing altitude fluctuation and keeping track of hundreds of thousands of lidar pulses per second. Due to that, apart from the laser scanner, the LiDAR system uses a GNSS and an Inertial Navigation System (INS) to provide precise positioning and orientation data (**Figure 37**).

GNSS provides the global position and orientation of the laser scanner, needed for georeferencing, i.e., to convert distance measurement into 3D points measurement in a global coordinate system such as WGS84. To achieve a highly accurate global position, differential GNSS (DGNSS) is employed. DGNSS enhances the accuracy of GNSS by using a ground-based reference station with a precisely known position. These stations continuously compare their actual position with the position determined by GNSS and broadcast the difference as a correction signal. The aircraft receives these correction signals from nearby DGNSS stations and applies them to refine its own GNSS position. By using DGNSS, the accuracy of the GNSS position can be improved from several meters to a few centimeters, significantly enhancing the precision of LiDAR-derived geospatial data.

To obtain the accurate orientation of the laser scanner, the INS of the aircraft is used. The INS accurately measures the aircraft's rotation around the X, Y, and Z axes using an inertial measurement unit (IMU). The X axis (roll) is aligned with the direction of the aircraft's flight, the Y axis (Pitch) lies in the horizontal plane and it is perpendicular to the X axis, and the Z axis (yaw) is in the vertical plane and it is perpendicular to the X and Y axes. Once the orientation of the laser scanner is determined, the direction in which the laser pulse is emitted can be accurately calculated.

By combining the global position and orientation of the laser scanner with the distance measurement from the laser pulse, the georeferenced 3D coordinates of the points on the target object that reflected the laser pulse can be computed. In a typical commercial system, distance is measured with an accuracy of 2-3 cm, GNSS error is between 5-10 cm, and IMU error is 27 cm at a flight height of 3000 m. Consequently, the absolute accuracy of LiDAR-derived elevation is between 10 and 20 cm, while relative accuracy is even higher.

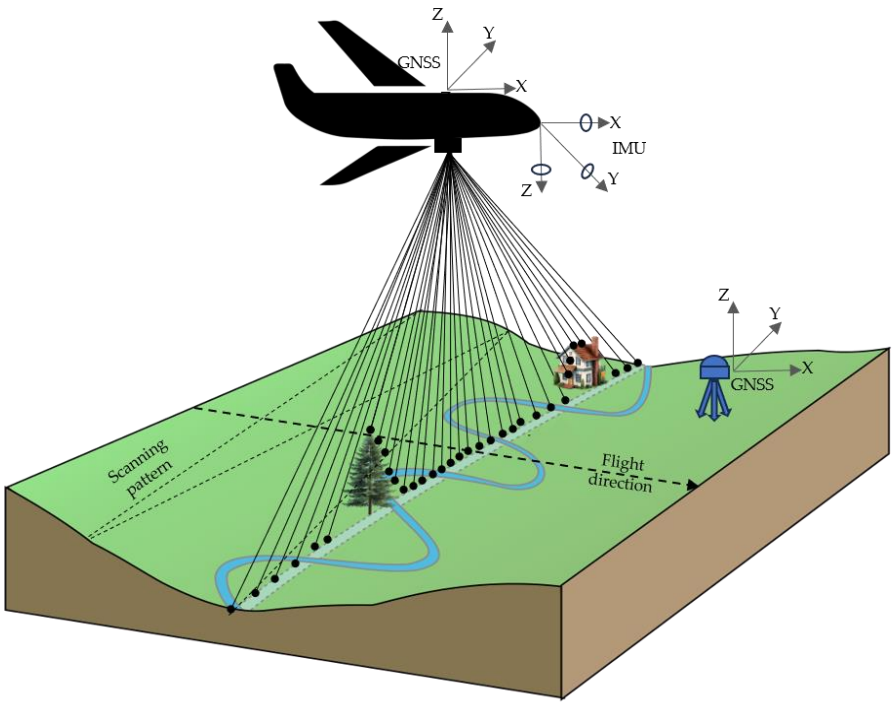


Figure 37 An airborne LiDAR system

6.3 Echo detection

In a LiDAR, one emitted pulse can lead to multiple echoes since a single laser pulse can interact with multiple surfaces. Since the laser pulses are typically emitted with slight divergence, their footprint on the ground spans several centimeters in diameter. This means that when a pulse encounters an object, part of the energy may be reflected while the remaining energy continues past the object, allowing it to interact with additional surfaces.

Once the reflected pulse is received by the sensor, it forms a waveform that represents the received signal power (amplitude) over time (t). The system detects echoes based on a predefined threshold—an echo is recorded whenever the signal power exceeds this threshold. This enables LiDAR to capture multiple returns from a single pulse, which is essential for mapping vegetation, buildings, and terrain with high accuracy.

An echo can also be referred to as a return. For each return, the return number and the number of returns are recorded. Return number refers to the position of a specific return within a single laser pulse, i.e., the first return is the first echo received from an emitted laser pulse, and the last return is the last received echo (see **Figure 38**). The return number can, in some cases, be used to determine if an echo was reflected on vegetation or ground (ground should then be the last return). The number of returns is the total number of returns for a given pulse.

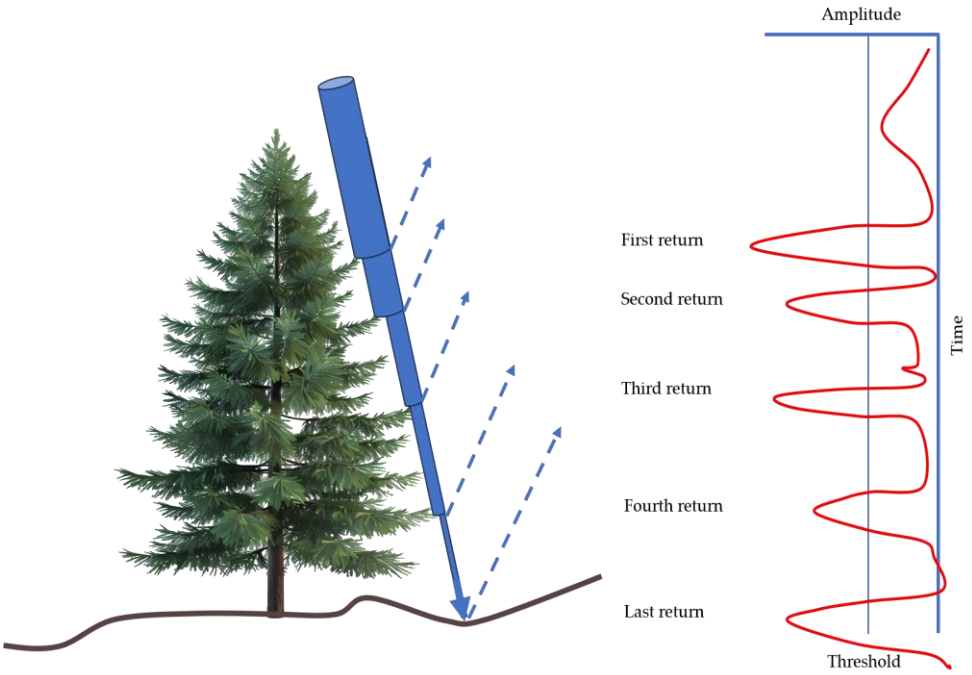


Figure 38 Multiple return LiDAR system

Multiple return systems can capture up to five returns per pulse, increasing the amount of data and ability to comprehensively analyses 3D structures such as forest canopy. There are typical for semi-transparent objects, overhanging objects (such as power lines), or abrupt surface discontinuities (building edges) A tree is particularly interesting because it often causes multiple echoes (one or more on branches and one on the ground below).

6.4 Laser properties

As mentioned above, the LiDAR operates using laser pulses at specific wavelengths. To select the optimal wavelength several factors need to be considered such as:

- Atmospheric scattering - how much of the signal is lost by traveling through the atmosphere,
- Absorption capacity of vegetation - how much of the signal is lost because it is absorbed by vegetation,
- Sunlight interference - Undesired light in the LiDAR signal due to direct sunlight or scattered sunlight, which can introduce noise in the measurements,
- Safety - The degree to which a laser wavelength is absorbed by the human eye. The wavelength must be chosen so that it is not absorbed by the eye so much, and
- Application - as different wavelengths interact with surfaces in a unique way.

Near-infrared wavelengths (typically between 1040 and 1550 nm) are most widely used in LiDAR systems, as they provide strong reflection from vegetation and the built environment. The attenuation of near-infrared pulses in the atmosphere is minimal when the atmosphere is cool, dry and clean. However, the presence of water vapor and carbon dioxide increases attenuated severely. Green LiDAR is used primarily for bathymetric applications due to its ability to penetrate water efficiently.

Scanning frequency represents the number of pulses emitted by laser in one second. It is directly related to the density of echos. The higher frequency at constant aircraft speed and a standard height above target will result in a higher number of returns and higher accuracy due to increased number of measurement points collected over area. Moreover, the high-frequency system can finer detail by operating on an aircraft that flies higher and faster than an aircraft equipped with a lower frequency system, thereby reducing flying time and acquisition costs.

The laser scanner is fixed to the aircraft and it typically includes rotating optical elements, such as mirrors or prisms. The mirrors are used to direct

emitted laser pulses in a cross-trac direction i.e. perpendicular to the direction of flight. This scanning mechanism significantly increases the ground covered per flight distance. The different configurations of rotating optics, and therefore different scanning patterns, can be used. A few common types are: zig-zag, parallel line, and elliptical patterns. Scanning patterns represent the spatial distribution of returns that would be expected from a flat surface. The zig-zag pattern (**Figure 39** (a)) uses oscillating mirrors that direct laser pulses across the swath width in both directions of the scene. Although the spacing between points is preserved, it yields a much higher density on scan edges (non-uniform density). In the parallel line pattern (**Figure 39** (b)), a rotating polygonal mirror is used to direct pulses across swaths in one direction of the scan only. The elliptical pattern is generated by using a rotation mirror that revolves about an axis perpendicular to the rotation mirror. With this pattern, the point density at the swath edge is higher (**Figure 39**(c)), which is beneficial for connecting neighboring swaths. It is clear the point density is affected by the scanning pattern. In practice, uniform patterns such as parallel lines and ecliptic patterns are preferred.

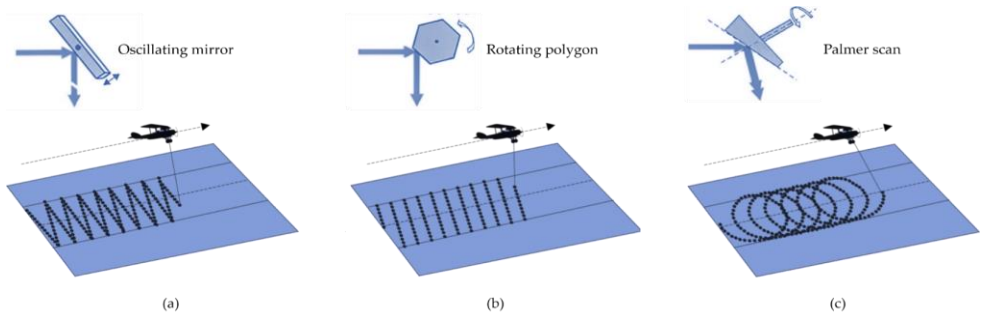


Figure 39 (a) zig-zag pattern, (b) parallel pattern, (c) elliptical pattern

The scanning angle is the angle at which the beam axis is directed away from the "focal" plane of the LiDAR instrument. It controls an area of ground being observed in a single flight line. The maximum scan angle is up to 30° in both directions from the vertical. Positive and negative angles represent pulses emitted to the right (starboard side) and to the left (port side) of the nadir, respectively. The scanning angle and flight height determine the scan swath. A narrow scan angle range limits the swath width, increases costs, and reduces efficiency. On the other hand, larger scan angles can introduce distortion, especially on steep terrain. Additionally, the laser footprint

increases at higher angles, affecting spatial resolution. For example, a study shows that for DEM production in steep terrains, the scan angle should be kept less than 15° [30]. For forest application, the often recommended scan angle is between 15 to 20 degrees [31]. However, using a wider scan angle can be beneficial in forest applications since the chance of detecting the ground and penetrating deeper into dense forests is higher for vertically incident beams (especially with newer systems capable of detecting multiple echoes with lower intensity).

The swath width is given by:

$$Sw = 2h \cdot tan\left(\frac{\theta}{2}\right)$$

where θ is scan angle range. At flight height of 500 m and scan angle range of 40 the swath width will be ~364 m.

Laser beam forms the footprint, the ground area illuminated by laser beam, known as Instantaneous Field of View (IFOV). Although the laser beam has low divergence (i.e. very narrow beam), with increase of the distance from source the size of the beam also increases (beam divergence). Due to that the laser footprint is not a point but an area. If the aircraft is perfectly horizontal and the laser beam is perpendicular to the ground, IFOV will be circular. As laser signal moves from vertical, the beam will be elongated in the direction of scanning, forming an ellipse. Additionally, the distribution of pulse energy is not uniform over the extent of IFOV and it decreases radially from the center.

The IFOV is a measure of spatial resolution of the LiDAR system. Thus, the point density within the point cloud will relate to the IFOV. Large IFOV will reduce the density but increase the coverage, leading to a lower signal-to-noise ratio. If IFOV is small, the sensor can determine the point position more precisely since it covers a smaller area.

The diameter of IFOV on the ground can be computed by:

$$IFOV = \frac{h \cdot \gamma}{cos(\theta_{inst})}$$

where h is the altitude of the aircraft above ground level, θ_{inst} is the instantaneous scan angle, and γ is the divergence of the laser beam. Typical

laser beam divergence ranges from 0.1 to 1.0 millirad. The change of IFOV for different scan angles at laser divergence of 0.5 millirad, and distance of 500 m from the instrument is shown in the **Table 8**.

Table 8 Change of IFOV considering different scan angles

$\boldsymbol{\theta_{inst}}$	$\boldsymbol{h_{inst}}$	**IFOV**
0°	500	0.2500
10°	500/cos(10)=507.63	0.2538
20°	500/cos(20)=532.38	0.2662
30°	500/cos(30)=577.35	0.2887

The minimum detectable object within the footprint depends primarily on its reflectivity rather than on the object size. For example, a power cable with a diameter of 1 cm is detectable, while a dark tree branch with a diameter of 3 cm will not be detectable. Moreover, the return pulse can be detectable even if it covers a small area of the target within the laser footprint if that area has high reflectivity (**Figure 40** (a)). In addition to reflectivity, several factors like range, laser power, atmospheric conditions, terrain inclination, 3D structure, and type of reflectivity (diffuse, spectral or both) influence detectability and accuracy. For flat surfaces with high homogenize reflectivity in the laser footprint, the reflected pulse will be very similar to received on. There will not be range averaging of various targets, resulting in good accuracy. On another hand, if multiple irregular surfaces close to each other reflect the received pulse, the reflected pulses are combined to a wider pulse with lower magnitude and longer rising time, resulting in range averaging and lower range accuracy. In addition to surface roughness, the measured range depends on surface slope. The laser pulses on the steep slope will be wider than on the flat surfaces and the measured range is an average of the range of the laser footprint. For example, if a beam is emitted with a scanning angle of 12 degrees down the 100-percent slope (45 %) the incident angle to the ground

will be 57 degrees i.e. the IFOV will be 0.5446 compared with 0.2555 on the flat surface (**Figure 40** (b) and (c)).

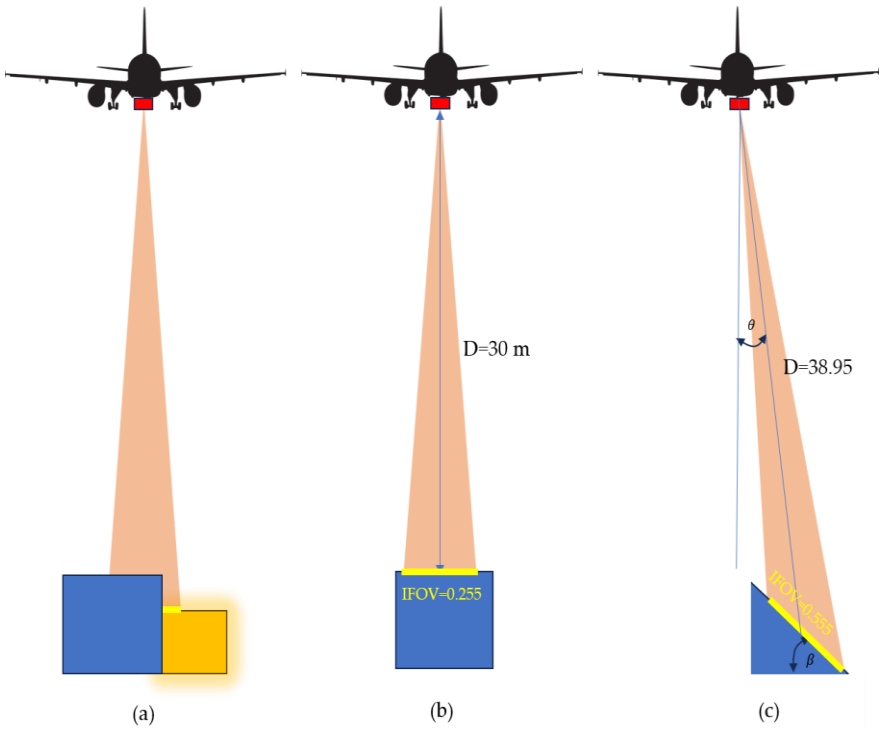


Figure 40 Interaction between laser beam and target (a) laser footprint partially covers a highly reflective object (b) laser pulse on a flat surface, and (c) laser footprint on a steep surface

Minimum separation between objects along the pulse path defines the vertical resolution of the Laser data. Usually, the returned pulse has a lower amplitude and wider width compared with the received pulse. The minimum vertical separable objects will be a function of pulse width (duration of a laser pulse as it travels from the sensor to the target and back).

Footprint spacing (**Figure 41**) is the nominal distance between the centers of consecutive beams along and between scanning lines, which, along with the beam divergence, determines the spatial resolution of LiDAR data.

The footprint spacing is a function of:

- Scanning frequency (number of emitted pulses per second [Hz]),

- The flight height - the maximum flight height depends mainly on transmitted power, while the minimum depends on national regulations and eye safety distance
- The speed of aircraft, and
- Scanning angle.

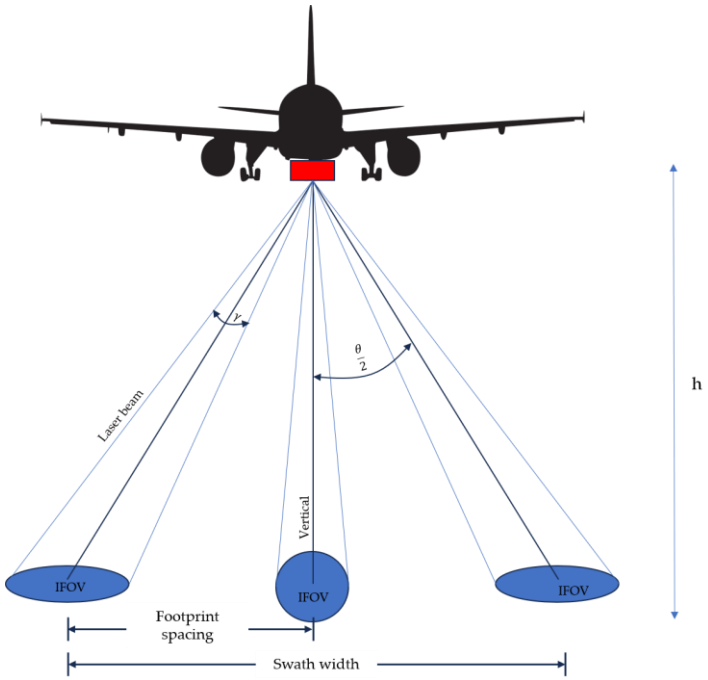


Figure 41 LiDAR horizontal resolution

6.5 Data characteristics

In addition to the previously mentioned number of returns and return number, several attributes are usually available for each detected point.

Pulse density is the direct function of footprint spacing over a hypothetically flat plane, and it can be calculated by pulse density = 1/ footprint spacing. It is the most consistent measure of LiDAR spatial resolution.

Return density represents the number of returns per unit square area (for example, 7 points per square meter). It is usually defined based on the application for which data is being collected. Return density is controlled by

specifications and operation mode of a LiDAR system (flight height, flying speed, scan angle, scan frequency, scan pattern, variation in acceleration and attitude) and type of target (geometry and reflectivity). For example, return density generated by 5 returns capable systems over forest will be much higher than the density generated over pasture or water surfaces. Recommended return density for the specific DEM accuracy class is presented in Table 7

Intensity is an attribute that describes the peak intensity value of the returned laser signal received by the sensor. It represents the reflective properties of the target, surface roughness, spreading loss, and atmospheric attenuation. High reflective surfaces, such as roads, dry soil, have higher intensity. Intensity is often used in feature detection and extraction, and can be used as a substitute for aerial imagery when they are not available. However, intensity is relative and values off the same target will vary from flight to flight or elevation to elevation, due to:

- Dependence on bidirectional reflectance distribution effect - describes how EM radiation is reflected at different angles. Since intensity depends on the angle of incidence and viewing geometry, its value may change significantly even for the same object,
- The distance of the laser instrument - the intensity will decrease with increasing distance due to atmospheric attenuation and energy dispersion,
- The total number of returns identified - the intensity varies depending on which return is recorded (first, second, last), and
- The rank of the return.

Moreover, to prevent hardware damage in case of extremely high amounts of backscattered energy due to the presence of high reflective targets, the receiver sensitivity is reduced practically instantaneously. However, increasing sensitivity for weak returns usually takes several seconds. Taking that into account, the presence of a single high-reflectivity target in one flight line can lead to a substantial discrepancy in the mean intensity of returns on the overlapping part of two adjacent flight lines. The fluctuations in the energy emitted by the laser also introduce variability in intensity values. Those fluctuations are likely more pronounced for high-frequency systems. Due to that, the application of intensity in point cloud classification is

challenging. Intensity is recorded in 8 bits (value from 0 to 255), 12 bits (0 to 1023), or in 16 bits (0 to 65535).

GPS time is an indication of the precise time that the pulse was emitted. This attribute can be used as a unique identifier for a pulse.

End-of-scan-line is a binary attribute (true or false) indicating whether the parent beam marked the edge of scanning lines.

Scan angle attribute indicating an instantaneous scan angle. Usually recorded in degrees.

6.6 Stripes and blocks

Data in airborne laser scanning is acquired strip-wise. The strip represents the set of points collected during one flyout. In order to scan larger areas, multiple flyouts, i.e., multiple strips next to each other with an overlap to avoid gaps, are needed (**Figure 42**). Larger overlaps are preferred to increase accuracy in strip adjustment and increase return density. Usually, an overlap between 20-30% is needed, depending on the geomorphological characteristics of the region being surveyed. Due to that, an overlap area of two overlapped strips will be surveyed twice, resulting in higher point density, increased data volume, and non-uniform distribution of points.

The set of all scanning strips represents the scanning blocks. The block design should include additional crossing flight strips and control areas. Their number is a function of the size and shape of the block. Each data strip must be covered by one crossing strip. The GNSS/IMU drift can result in misaligned or shifted point clouds between strips. Higher precision can be achieved by applying the strip adjustment. To increase redundancy and confidence in adjustment, it is recommended that control areas be covered by a crossing strip. A longer data strip is covered by more than one strip to reduce errors caused by GNSS accuracy variations. For control areas, usually long and wide flat areas (such as football fields) are required so that the uncertainty of the horizontal position would not affect its vertical component. Control points on the surface are measured with high precision in the local control network. The coordinates of control points should have at least 3 times better accuracy than

a LiDAR scanner. Those points are used for absolute accuracy analysis, which is necessary to calibrate the LiDAR system.

If there are discrepancies between overlapping laser scanner strips in height and planimetry, it is necessary to perform the strip adjustment. Those misalignments in the absolute and relative orientation of the point cloud are caused by systematic errors.

The discrepancies in ground control features imply errors in absolute orientation. Absolute orientation refers to the correct georeferencing of a LiDAR dataset in a global coordinate system. Relative orientation describes the orientation of strips with respect to each other, maintaining internal consistency. It does not consider absolute positioning but ensures that overlapping areas do not have mismatches. Relative accuracy is calculated between data overlapping swaths. The higher the swath overlap, the more precise the assessment would be.

Various strip adjustment methods have been used to eliminate the inconsistency in elevation, position, and accuracy. Those methods can be roughly categorized into two categories: data-driven approach (strip adjustment) and sensor system-driven (calibration) approach.

Sensor system-driven methods are based on the physical sensor model relating the system parameters to the ground LiDAR coordinates. These methods require the original observations (GPS, IMU, and the laser measurements) or at least the trajectory and the time-tagged point clouds. This data is not always available to the end-user.

Data-driven approaches rely solely on the LiDAR point cloud itself. In this approach, the systematic errors are removed by applying translation and rotation models e.g., a seven-parameter rigid body transformation or even a simple vertical shift. In order to provide both planimetric and vertical constraints for LiDAR strips, the relationship between matched conjunction features, such as lines and surfaces, needs to be established in order to calculate transformation parameters. This process includes the segmentation process, feature extraction, feature matching on the target point cloud, and then performing strip adjustment calculation.

Both sensor system-driven and data-driven methods need control elements or tie elements. Currently used control or tie elements are mainly lines and surfaces.

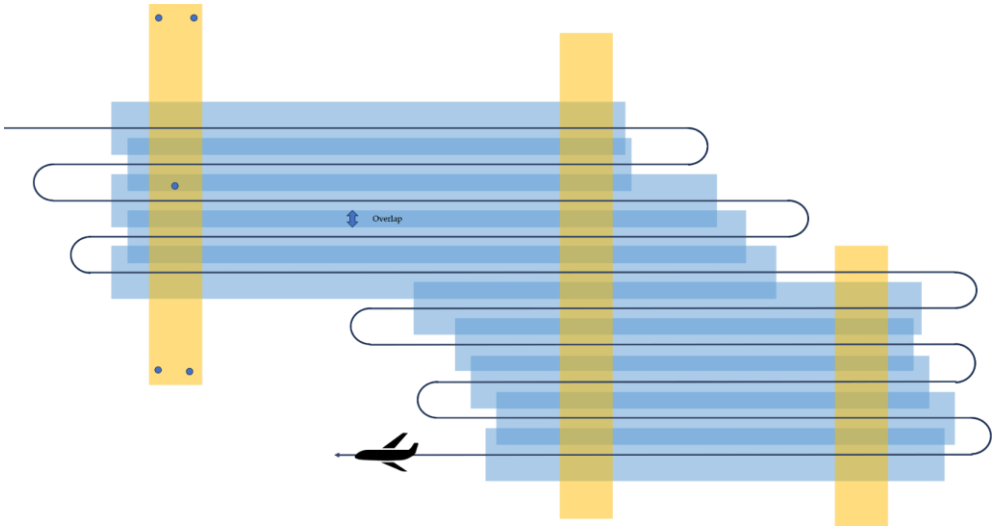


Figure 42 Strip configuration (blue) of the block, including three control areas (yellow)

1.1 Data Quality Control

Quality control of LiDAR data involves evaluation of return coordinates accuracy and precision, compliance with acquisition specifications, and data spatial consistency and completeness. Each sensor in an airborne LiDAR system can potentially generate errors, contaminating the laser data with systematic and random errors. The sources of errors in airplane LiDAR surveys are:

- Distance measurement error,
- Scan angle measurement errors,
- Error of aircraft position,
- Error in measuring aircraft orientation
- Device installation error,
- Geoid normal error, and
- Error caused due to time deviations (syndication error and interpolation error).

Random errors in LiDAR systems are most often caused by random fluctuation in measurement, such as inherent sensor noise, noise in recording scanning angles, range noise, platform instability, etc.. The magnitude of random errors is calculated during system calibration. Due to that, periodic calibration of LiDAR systems is needed. Random errors affect the absolute accuracy of returned coordinates, but also the relative accuracy. For example, the influence of attitude noise in the GNSS/IMU derived orientation will vary with scan angle, affecting less nadir regions compared with the off-nadir regions. Randomness can be reduced by averaging or repeating measurements.

Bias in GNSS, aircraft attitude, scanning angle, and time measurement course systematic errors. The device installation error, i.e. the three-dimensional offset between the GNSS unit and the laser pulse emission, would cause the systematic error independent of the flight height, but dependent on the flight direction. In multi-pass LiDAR, the strips may not align properly. An error in measuring the scanning angle can cause the return coordinate errors that increase with flight height and flight direction. This type of error produces a fixed amount of bias that can not be reduced by averaging, but it can be reduced when their sources are known.

Typical error values that occur in an aircraft laser scanning system are presented in Table 9.

Spatial completeness is a key quality factor of airborne LiDAR data. It represents the lack of continuity or scanning uniformity in LiDAR datasets. The scanning uniformity is represented by variability in pulse density or return density over the same objects. There are two types of data density: local and mean density. Mean density is calculated by dividing the all point number with the whole project area. High return density does not guarantee a uniform point distribution. Mean density is a general measure of dataset resolution, but it can not reflect the spatial completeness, so local density is needed. The local density computes the number of points within a defined neighborhood. Its variations occur due to variations in flight height, the distance between adjacent flight lines, or instability of the aircraft's attitude. For example, turbulence during data collection causes continuous variations of aircraft pitch and roll, causing the aircraft to deviate from its ideal scanning path, leading to gaps in data over certain areas despite the swath overlap. The

local density highly influences the performance of data processing techniques and the quality of the created products. All the local density values in an area form a density distribution map.

Table 9 Typical error values that occur in an aircraft laser scanning system

Error	**Typical Value**
Geoid normal error	0.017°
IMU error	0.01°
Device installation error (rotation between laser scanner and IMU)	0.3° / 0.01°
Scan angle measurement errors	ε=0.02° $\Delta\tau$=0.03°
Distance measurement error	5-10 cm
Device installation error (translation between laser scanner and IMU)	3 cm
Device installation error (translation between GNSS and IMU)	3 cm
GNSS error	10 cm
Error caused due to time deviations	1 cm

Accurate assessment of the error budget requires field surveying on the ground and of objects that are clearly observable in the resulting point cloud.

The vertical accuracy depends on the quality of INS, integration with GNSS and methods of post-processing. As already mentioned, the effect of attitude error on the 3D accuracy increases with flying height and the scan angle. In recent years, vertical accuracy is measured against the LiDAR surface in GCP. The LiDAR surface can be modeled using Triangulation Irregular Network or other interpolation methods. The orthogonal distance between the GCP and LiDAR surface represents vertical error. Using multiple GCP, the statistics

such as the maximum vertical errors per DEM accuracy class are presented in Table 7.

The positional accuracy depends mainly on the quality of DGNSS post processing, but other factors such as GNSS satellite constellation during flight, the number of satellites, the distribution and distance of ground reference stations, the accuracy of IMU (roll, pitch, and heading), the flight height, and scanner accuracy are also included. The assessment of positional accuracy is usually based on comparison between GCP coordinates and their conjugate points from the LiDAR dataset. However, the assessment of positional accuracy is not an easy task since point clouds are abstract and cannot be easily compared with discrete objects.

6.6.1 Data format

The result of the airborne LiDAR survey is a point cloud. A point cloud is a discrete set of data points in a 3D coordinate system. It allows the individual spatial measurements to be combined into a meaningful 3D representation of objects or environments. The point cloud is characterized by an enormous number of points (from a few thousand to a few million points). In addition to X, Y, and Z coordinates, each point can be described by several attributes such as R, G, B color, classification etc. Due to its characteristics, several formats can be used for point cloud storage. Point clouds differ fundamentally from classical raster and vector formats, meaning that traditional processing and analyzing methods of raster and vector data cannot be applied directly on point clouds. The two main categories of point cloud formats are ASCII and binary. ASCII uses human-readable text characters to represent X, Y and Z coordinates of each point. The benefit of text-based format lines is the fact that content of text files are usually well supported and easily accessible via a text editor. For example, the use of a delimited format, where each line contains data for single return makes it easy to integrate data into popular SQL databases where it can be queried and rearranged as needed. However, they are large in size resulting in very slow reading and interpretation even for a small number of points. In addition, the metadata is lost. ASCII file format includes xyz, obj, ptx, asc.

The American Society for Remote Sensing (ASPRS) introduced the LAS file format as an alternative to the ASCII file format. LAS is a public binary file

format for the interchange of 3D point clouds between different data users. It maintains metadata specific to the LiDAR while not being overly complex. A LAS file contains information about points in one of the point data record formats defined by the specification. The current version, LAS 1.4, allows 11 point data record formats (from 0-10), the preferred formats are 6-10 since they have improved several aspects of the core information in the point data records (improvement of classification scheme and waveform storage). All points must be of the same format within the files. The various formats differ in the available data fields.

The LAS 1.4 defines Variable Length Record and Extended Variable Length Record. The number of bytes used per point data record is explicitly given in the public header block. The Extended Variable Length Record enables the definition of additional user-defined fields in "extra bytes" to the field given by specification. The LAS file format is not compressed. However, an open-source project, LASzip, defines the open file format LAZ, significantly reducing file size up to 90% while maintaining precision, making it ideal for large-scale applications such as terrain modeling, forestry analysis, and autonomous navigation. LAS and LAZ formats provide interoperability with various software used in the analysis and visualization of LIDAR data.

7 REFERENCE FRAME

A reference frame is a three-dimensional coordinate system that describes the Earth's body and provides the foundation for determining position on Earth and in space. Many applications require precise global frames, such as geodetic survey, engineering, cadaster, photogrammetry, geophysics, hydrology, climatology, etc. Also, it allows the quantification of changes due to geodynamic processes and climate change. For example, in order to assess sea level variation, an accurate global reference that remains stable in the long term is needed. The global reference frame ensures a uniform basis for geospatial data. Geodata doesn't have value without a reference frame; thus, it has an elementary role in modern society.

Reference frame includes the following concepts: reference surface, datum, and coordinate system.

All human activities and all measurements are performed on the Earth's surface. The primary aim is to present collected information on various maps. However, due to the irregular distribution of the third dimension, i.e., vertical variations between mountains and valleys, it is impossible to approximate the shape of the Earth with any reasonably simple mathematical function. To overcome this, all measurements must be projected orthogonally onto a mathematical figure that closely approximates the Earth's shape and dimensions. Once this projection is made, the measurements can then be transformed onto a two-dimensional plane (i.e. map).

As a first-order approximation, the sphere could be used for small-scale mapping purposes (from 1: 1 000 000) or when very high accuracy is not required. However, the sphere does not meet accuracy requirements when high precision is required and ellipsoids need to be used.

The Earth's ellipsoid is any ellipsoid that approximates the Earth's surface. In geodesy and cartography, a rotation ellipsoid with small flattening is used. It is a surface resulting from rotating an ellipse around its semi-minor axis. The size of the ellipsoid can be defined by using a semi-major axis (a), a semi-minor axis (b), and flattening (f). Flattering is a parameter that emphasizes the difference between the sphere and the ellipsoid, and it is equal to $f = \frac{a-b}{b}$.

Some of the most used ellipsoids are: Bessel (1841), WGS84, and GRS80.

The most accurate approximation is the geoid. The geoid is an equipotential surface of Earth's gravity field, ideally still sea surface extended under land mass, and it is always orthogonal to Earth's gravity direction at every point. It can be simplified by imagining that the entire Earth's surface is covered by water. If the current and other influences, such as winds and tides, were absent, the result would be an ideally still sea whose surface is affected only by Earth's gravity and rotation.

The geodetic datum describes the orientation and position of the ellipsoid in relation to Earth. At least seven parameters are needed to define a global datum: three for the determination of origin, three for the determination of the orientation of the reference ellipsoid, and a scale factor (usually in parts per million (ppm) units). The horizontal geodetic datum is a reference for defining 2D coordinates on a reference surface (ellipsoid or sphere). A vertical geodetic datum is a basis for the definition of heights.

7.1 Vertical geodetic datum

Since Earth's gravity field is irregular due to variations in mass distribution, the geoid is undulated rather than a perfectly smooth surface. Depending on the mass deviation in land mass, the geoid will be below (where mass deficiency exists) or above (where a mass surplus exists) the reference ellipsoid. The deviation between the geoid and reference ellipsoid is called geoid undulation (N) (**Figure 43**).

$$N = h - H$$

where N is the geoid undulation, h is ellipsoidal height, and H is orthometric height.

The geoid is used to describe height and a reference surface for the measurement of land elevation and water depths on Earth (vertical datum). Since the geoid is a theoretical surface and the actual sea surface is constantly affected by tides, currents, and waves, tide gauges (mareographs) are used to record sea levels over several years. By averaging these long-term measurements, short-term fluctuations are eliminated, resulting in the Mean

Sea Level (MSL). MSL serves as an approximation of the geoid and is commonly used as a reference surface for measuring elevations.

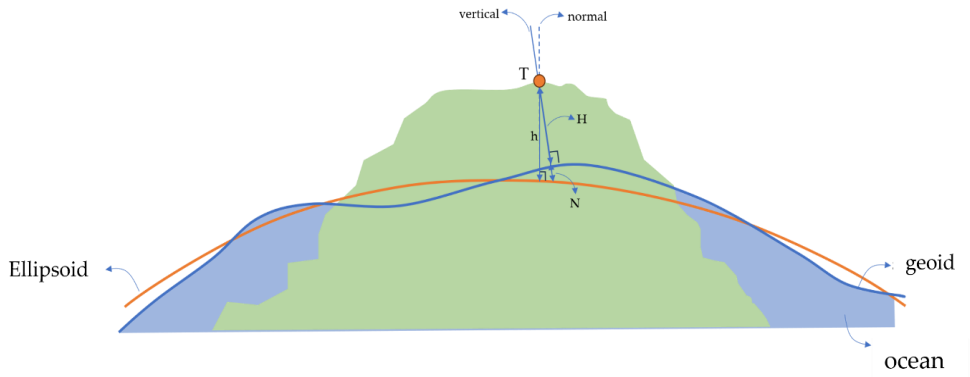


Figure 43 Vertical datum

Every country or a group of countries has established the mean sea level measuring points located near the area of concern (local vertical datum). For ex-Yugoslavia, the local mean sea level is derived through the mareograph in Trieste (zero height). All elevations in ex-YU are measured relative to the Trieste tide gauge using geodetic leveling. The result is orthometric heights, i.e., heights above local sea level. The orthometric height, the distance between the geoid and a point on the topographic surface, measured along a normal, is crucial for many engineering and geoscience applications.

There are several realizations of the local vertical datum (the height reference system for specific regions). They are approximately parallel to the geoid but offset by a couple of meters due to local conditions (currents, tide, temperature difference, etc.) at the specific location.

The use of the Global Navigation Satellite System (GNSS) to determine height has been widely used. However, the GNSS determines height with respect to the reference ellipsoid, i.e., ellipsoidal heights. To convert ellipsoid heights to orthometric, the geoid undulation is required.

In addition to local measurement, satellite missions, such as Gravity Recovery and Climate Experiment (GRACE) and Gravity field and steady-state Ocean Circulation Explorer (GOCE), have significantly improved geoid determination by providing global gravity field measurements. Those

satellite-derived global geoid models, such as EGM 2008 or EGM 96, are widely used in geodesy and navigation to convert GNSS measurements to orthometric heights. However, the global gravity model doesn't completely represent the Earth's gravity field because their resolution and accuracy are limited. Typically, a combination of GNSS leveling and global vertical datum provides absolute accuracy from centimeter to decimeter [30], [31].

7.2 Horizontal datum

There are two primary types of horizontal datums: the global horizontal datum and the local horizontal datum.

As the demand for interoperability in geodetic measurements increases, there is a growing need for a global reference surface that ensures coherent and consistent results across various disciplines, such as astronomy, geophysics, and other Earth sciences. The parameters of this ellipsoid, its placement, and orientation within the Earth should closely match the shape of the global geoid. Therefore, during their determination, specific conditions are set to ensure an accurate approximation. Among these conditions, the rotational axis of the ellipsoid must align with the Earth's axis of rotation, and the geometric center of the ellipsoid must coincide with the center of the Earth's gravity (geocentric). In addition to these requirements, if the ellipsoid's volume is made equal to that of the geoid and the sum of the squared distances between the geoid surface and the ellipsoid is minimized, such an ellipsoid is referred to as the global reference surface.

In contrast to global datum, the local horizontal datums are reference surfaces that apply to specific countries or regions. Local datum positioning the ellipsoid to provide the best fit to the geoid in the area of interest. In this way, the difference between the reference ellipsoid and geoid could be ignored. The orientation of the ellipsoid is defined by the fundamental point (φ, λ, h) and an azimuth to an additional point. The local horizontal datum is determined through a triangulation network. There are several hundred local horizontal datums in the world. In ex-YU, the fundamental point is Hermannskogel, located near Vienna, and the underlying ellipsoid is the Bessel ellipsoid.

A global reference frame provides a uniform basis for geospatial data, ensuring interoperability, consistency, and the combination of measurements

collected by ground or space-based sensors. It is needed for the precise orbit of satellites, all navigation, and change detection. A geodetic reference system provides a basis for a uniform global coordinate system in which every point on the Earth can be described by unique coordinates.

7.2.1 ITRS

The fundamental global geodetic reference system in use today is the International Terrestrial Reference System (ITRS). The ITRS is a geocentric coordinate system (its origin is at the center of mass of the whole Earth, including oceans and atmosphere) with three orthogonal coordinate axes (X, Y, Z) co-rotating with the Earth in its diurnal motion in space. The Z-axis points to the historical direction of the Earth's mean axis of rotation. The X-axis is oriented towards the mean Greenwich meridian, and it is orthogonal to the Z-axis. The Y-axis completes the right-handed reference coordinate system. The scale unit is the meter.

The ITRS is realized through the International Terrestrial Reference Frame by determining the 3D coordinates of firmly anchored points. Those points represent observation stations distributed worldwide. Their coordinates are computed consistently with the system definition by space geodetic observation techniques.

ITRF serves as the foundation for many global and local geodetic datums such as WGS84 and ETRF (European Terrestrial Reference Frame). The ITRF2020-u2023 is the latest realisation of ITRS. It represents the update of ITRF 2020 [30], extending its data coverage by including observations from 2021 to 2024. The frame parameters (origin, scale, and orientation) are the same as ITRF2020 [32].

7.2.2 WGS84

WGS84 is a global geodetic reference system used in various applications, including positioning, navigation, and mapping. It is the standard coordinate system used by the Global Positioning System (GPS), and it is maintained by the US National Geospatial-Intelligence Agency. WGS84 defines a global Cartesian coordinate system (X, Y, Z) with its origin at the Earth’s center of mass. The Z-axis points the direction of the BIH Conventional Pole (epoch

1984). The X-axis passes through the intersection of the Greenwich meridian, and the plane passes through the origin, and it is normal to the Z-axis. The Y-axis completes a right-handed orthogonal coordinate system. The origin of WGS 84 is periodic refinement. The current realization WGS84 (G2139) aligns closely with ITRF 2020 to within one centimeter in each 3D component [33].

7.3 Coordinate systems on an ellipsoid

The geodetic coordinates are geodetic latitude, geodetic longitude, and height. They are also referred to as ellipsoidal coordinates (**Figure 44**).

Geodetic latitude is defined by the angle from the equatorial plane to the normal to the ellipsoid at the given point. It is usually marked with a Greek letter φ and it can have a value in the interval between 0 and 90 on North and South ($-\frac{\pi}{2} \leq \varphi \leq \frac{\pi}{2}$).

Geodetic longitude is defined by the angle that the prime meridian (the meridian of the Greenwich observatory near London) makes to the meridian plane of a given point. It can have value in the interval [0,180] on the East and West and is marked with the Greek letter λ.

Geodetic coordinates uniquely define the position of any point on the ellipsoid. However, to determine a point's position on the actual Earth's surface, the distance from the ellipsoid to the point along the ellipsoidal normal is required. This distance is known as the ellipsoidal height (h).

In this coordinate system:

- Latitude and longitude define the horizontal position of a point on the ellipsoid.
- Ellipsoidal height (h) represents the vertical distance of the point above or below the ellipsoid.

While latitude and longitude provide a reference for horizontal positioning, ellipsoidal height is essential for determining elevation relative to the mathematical model of the Earth's shape.

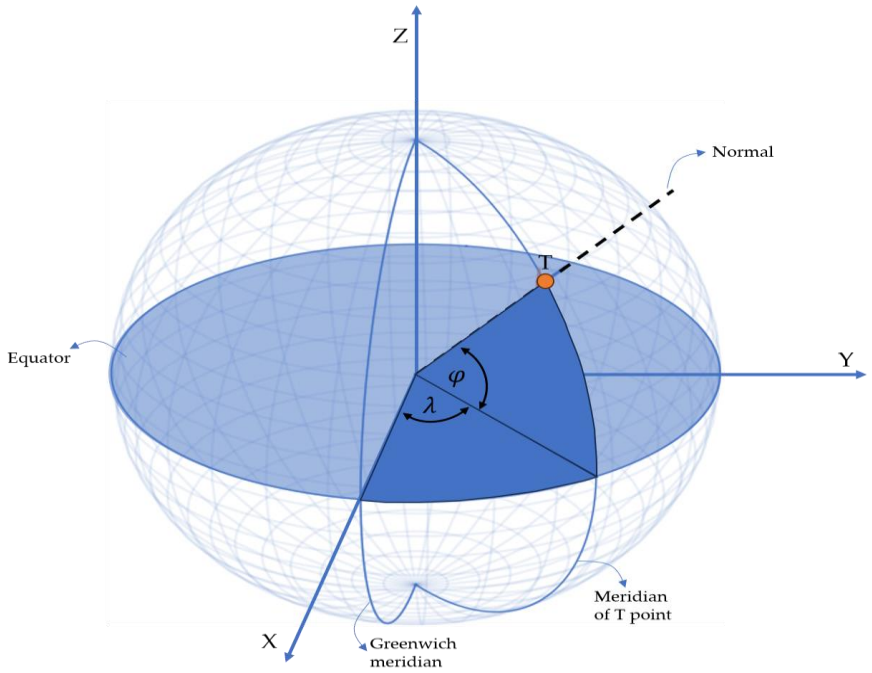


Figure 44 Geodetic coordinates latitude (φ) and longitude (λ)

7.3.1 3D Cartesian coordinate system

Geodetic coordinates φ, λ, h can be transformed to an Earth-centred, Cartesian three-dimensional system using the following equations (**Figure 45**):

$$X = (N + h)cos\varphi cos\lambda$$

$$Y = (N + h)cos\varphi sin\lambda$$

$$Z = (N(1 - e^2) + h)sin\varphi$$

where $N = \frac{a}{\sqrt{1-e^2 sin^2\varphi}}$, $e^2 = \frac{a^2-b^2}{a^2}$, a is the semi-major axis, b is the semi-minor axis of the ellipsoid.

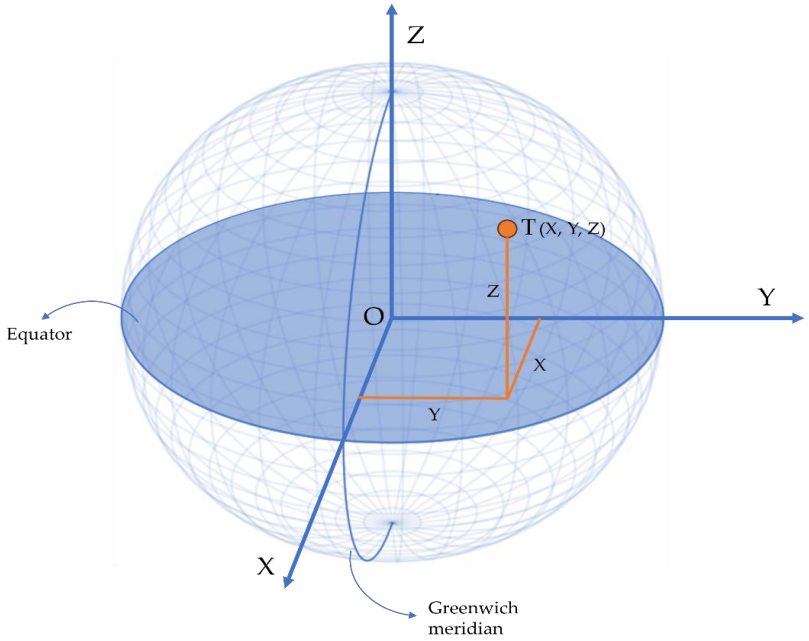


Figure 45 3D Cartesian coordinate system

7.3.2 2D Cartesian coordinate system

In geodesy, the 2D Cartesian coordinate system in the plane is often used. It consists of two perpendicular coordinate axes that intersect at a single point (O), called the origin. The horizontal axis is called the abscissa axis and is denoted by the symbol 'X'. The vertical axis is called the ordinate axis and is denoted by the symbol 'Y'.

The position of a point in this system is defined by two perpendicular coordinates: the ordinate (y) and the abscissa (x), which are measured along their respective coordinate axes from the origin.

7.4 Cartographic projection

One of the primary tasks in geodesy and cartography is map production. To represent part of Earth on a flat paper map or computer screen, the curved surface must be mapped onto the 2D plane. Meaning that each point on the reference surface with geodetic coordinates (φ, λ) must be transformed to a set of Cartesian coordinates (x, y). This mathematical transformation, known as cartographic projection, establishes the functional relationship between

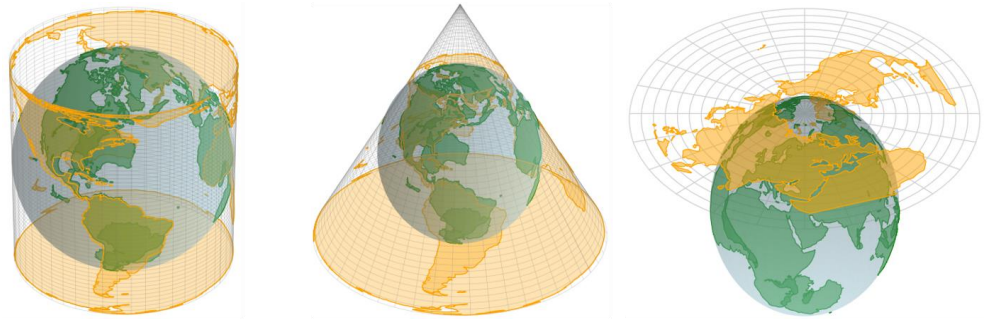
points' coordinates on an ellipsoid and their corresponding coordinates in the plane. However, this process inevitably includes distortion since it is impossible to represent a curved surface on a map without distortion of angles, length, or area. Based on the type of distortion, projection can be classified as conform (equality of angles, i.e., shape preservation), equivalent (there is no deformation of area), and conditional (most often equidistant, i.e., there is no deformation of length). Cartographic projection allows for the computation and analysis of distortion, assessment of spatial distribution, and effective management of these factors in mapping.

Historically, maps were created using geometric projections—tangible methods where one could literally imagine projecting the Earth's curved surface onto a flat sheet or developable surfaces using a light source inside a globe. The map projection can be classified based on the: type of surface (cylindrical, conical, or azimuthal (**Figure 46**)), point of secancy (tangent or secant), and aspect (normal, transversal, or oblique). In normal projection, the main orientation of the projection surface (the rotation axis of a cone or cylinder, or normal on a plane and ellipsoid in tangent point for azimuthal projection) is parallel to the Earth's rotation axis. In transversal projection, the main orientation of the projection surface is perpendicular to the Earth's rotation axis, while in oblique projection, they form an angle between 0^o and 90^o.

These projections maintained an intuitive, visual link between the round Earth and the flat map, preserving certain geometric properties such as distances, angles, or areas depending on the projection type. Geometric projections were widely used in the past because they could be physically constructed or measured, making them easy to understand and interpret.

Modern cartography, however, relies on analytical (or analytic) projections, which are defined by mathematical formulas rather than physical constructions. Analytical projections do not necessarily have a direct geometric interpretation, but some can be seen as refinements of older geometric projections—for example, the Gauss-Krüger projection can be derived from a transverse Mercator cylindrical projection. The main advantage of analytical projections is their precision, computational efficiency, and flexibility, allowing maps to be tailored to preserve specific properties such as area, shape, or distances in a given region. As a result,

analytical projections have become the standard in today's digital cartography and GIS, replacing the geometric methods of the past while building conceptually on their foundations.



(a) (b) (c)

Figure 46 Developable surfaces used in cartographic projection (a) cylinder (cylindrical projection), (b) cone (conical projection), and (c) plane (azimuthal projection)

7.4.1 Universal Transverse Mercator projection

The Universal Transverse Mercator (UTM) projection is an analytical projection that can be considered a derivation of the transverse secant cylindrical conformal projection. It represents the modification of the Gauss-Krüger projection. It allows for the representation of the entire Earth within a single coordinate system with limitations for polar areas (north of 84^oN and south of 80^oS), for which the Universal Polar Stereographic projection should be used.

UTM specifications:

- The whole world is divided into 60 zones, each 6 degrees of longitude wide, starting at the International Date Line (180^o) and moving eastward. Zones are numbered from 1 to 60, with Zone 1 covering longitudes from 180^o to 174^oW (**Figure 47**).
- It covers an area from 84^o north latitude to 80^o South latitude. Each zone is divided into subzones, each 8 degrees of latitude, starting at 80^oS and moving northward. The subzones are marked by letters of the English alphabet from C to X. Letters I and O are omitted due to their similarity with 1 and 0. The first row covers the latitude from 80^oN to 72^oN. Letters A and B and W and Z are used for polar areas.

- A scale factor is 0.9996 on the central meridian of the zone, i.e., the scale error is 0.4 m/km.
- Each zone has a coordinate system. The origin is defined at the central meridian and equator. The position of points is defined by coordinates easting and northing (E, N). The easting indicates distance in meters from the central meridian, while the Northing indicates the distance from the equator.
- To avoid negative coordinates for positions located west of the central meridian, the false easting of 500,000 m is assigned. The equator has been given a Northing value of 0 m for positions north of the equator, and a false northing value of 10 000 000 m for positions south of the equator.

UTM is one of the most important cartographic projections, which was primarily developed by the military in the 20th century. However, UTM is widely used in Earth observation, photogrammetry, cadastral, etc. UTM is particularly effective for large-scale mapping as it maintains relative accuracy and minimizes distortion within each zone. Due to relatively small deformation, precision georeferencing of images and precise navigation, UTM became very popular.

One disadvantage of the UTM is that multiple coordinate systems must be used for large areas, which can lead to confusion.

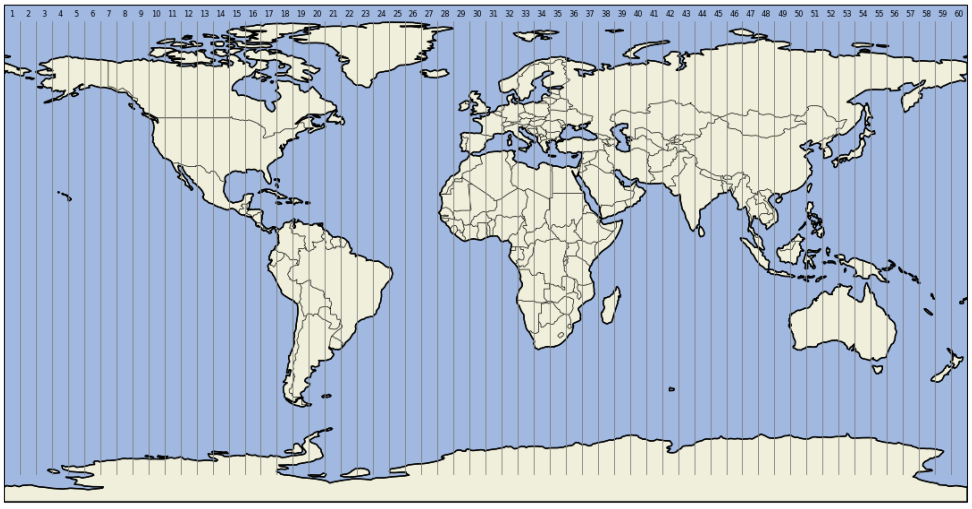


Figure 47 UTM zones of the World

7.4.2 EPSG code

The EPSG code is a unique identifier assigned to a specific coordinate reference system, projection, or datum. EPSG stands for European Petroleum Survey Group and is a scientific organisation that maintains a geodetic parameter database. Each entity is assigned a unique EPSG code between 1024 and 32767, ensuring consistency across different GIS applications. The EPSG code is very important since there are many different coordinate systems defined for different areas. By using the EPSG code, there is no need for manual specification of parameters, ensuring spatial accuracy, consistency, and efficiency in geospatial data processing.

The EPSG code represents a different geodetic parameter set, which includes: Coordinate reference system, projection, datum, and units. Some of the frequently used codes are:

- EPSG:4326 - WGS 84, a geographic coordinate system with latitude and longitude in degrees,
- EPSG:3857 - Pseudo Mercator, used in web mapping applications (Google Maps, OpenStreetMap, Bing Maps, etc.) with coordinates in meters,
- EPSG:32633 - UTM Zone 33N (WGS84), and
- EPSG: 32733 - UTM Zone 33S (WGS84).

8 UNSUPERVISED LEARNING

Four of the most common aims of remote sensing data analysis are clustering, dimension reduction, regression, and classification.

8.1 Clustering

Clusters are groups of objects that share common characteristics. Clustering is the process of dividing a dataset into such groups, so that the objects within each cluster are as similar as possible to one another, while being as different as possible from objects in other clusters. In practice, this means minimizing the distance between data points within the same cluster and maximizing the distance between points belonging to different clusters. The main challenge lies in defining what "similarity" means. Clustering is one of the most widely used techniques in unsupervised learning, where the goal is to discover hidden patterns in data that are not labeled in advance.

All clustering techniques consist of two main steps: calculating the similarity measure between data samples (distance) and applying a clustering algorithm to group similar objects. The similarity measure can be defined as the distance between different data points, and it represents the strength of the relationship between two data samples. It can be defined in different ways depending on the particular application, and it is significant for clustering. The most commonly used distance measures are Minkowski distance, Euclidean distance, Manhattan distance, Chebyshev metrics etc. (**Figure 48**). Minkowski distance is a generalized similarity metric and can be used for both ordinal and nominal variables. It is given by

$$D_{(X,Y)} = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

where X, Y are independent variable vectors of n length and parameter p controls the type of distance used. If $p = 1$ the distance becomes the Manhattan distance, for $p = 2$ the Euclidean distance and for $p = \infty$ the Chebyshev distance. Euclidean distance measures the straight-line (shortest) distance between two points. Manhattan distance, also called city-block or taxicab distance, is the sum of the absolute differences across all dimensions.

Instead of taking the direct path, it measures distance as if movement is restricted to a grid, like walking along city blocks. Chebyshev distance, sometimes referred to as the maximum metric, is determined by the largest absolute difference along any single dimension. It can be visualized as the number of moves a king would need on a chessboard to travel from one square to another. Cosine similarity, instead, measures the cosine of the angle between two vectors, capturing their orientation while ignoring their magnitude.

The clustering is an optimization problem. The aim is to find clusters that optimize an objective function that is subject to the same constraint. i.e., it is necessary to define an objective function that minimizes the distance within the cluster. However, pure minimization of distance is not a solution since the result can be that each data point represents a cluster. Therefore, constraints such as defining the number of clusters in advance or setting a minimum distance between clusters are often required.

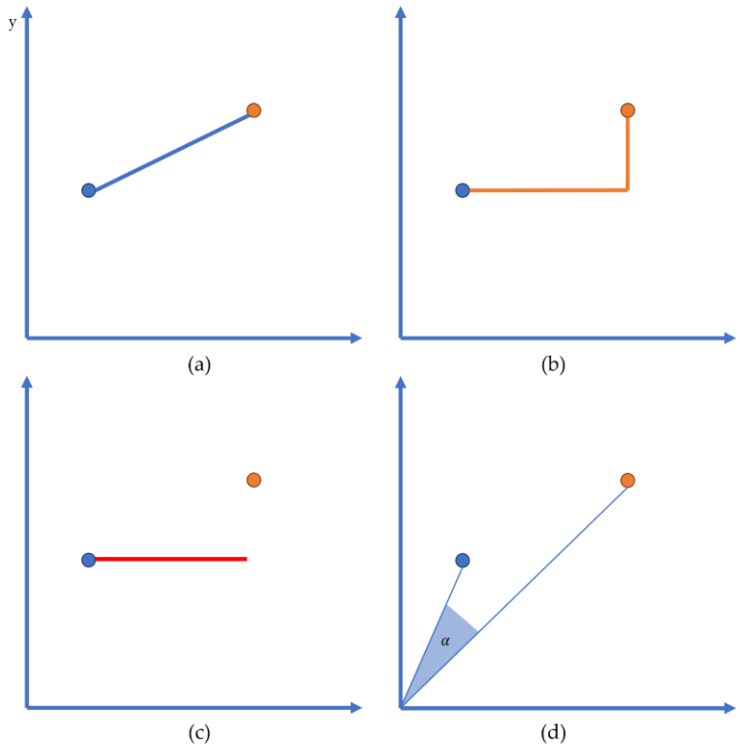


Figure 48 Similarity measures (a) Euclidean distance, (b) Manhattan distance, (c) Chebyshev distance, (d) Cosine similarity.

Cluster algorithms can be classified into several categories, including:

- Exclusive clustering: each observation belongs to only one definite cluster (such as K-means),
- Overlapping clustering, which uses fuzzy sets to cluster data so that each point can belong to two or more clusters with different degrees of membership (for example, fuzzy c-means),
- Hierarchical clustering, which has two versions: agglomerative clustering and divisive clustering. Agglomerative clustering works in a bottom-up approach. It is initialized by setting each sample as its own cluster and merging the closest (most similar) pairs in each step. Divisive clustering uses a top-down principle i.e. it starts from one cluster containing all data points. At each step, the clusters are successively split into smaller clusters according to some dissimilarity,
- Density-based clusters, which are based on the assumption that the clusters are dense regions in space separated by sparse regions. It is used to find non-linear shape structures (for example, Density-Based Spatial Clustering of Applications with Noise (DBSCAN)), and
- Probabilistic clustering: data points are clustered based on the likelihood that they belong to a particular distribution. The Gaussian Mixture Model is one of the most used methods.

8.1.1 K-means

K-means clustering is an interactive centroid-based process that groups unlabeled data points into K non-overlapping clusters by using mathematical distance measure. It is one of the most widely used unsupervised ML algorithms. The aim is to minimize the sum of distances between data points and their centroids. The k-means clustering algorithm consists of a few steps:

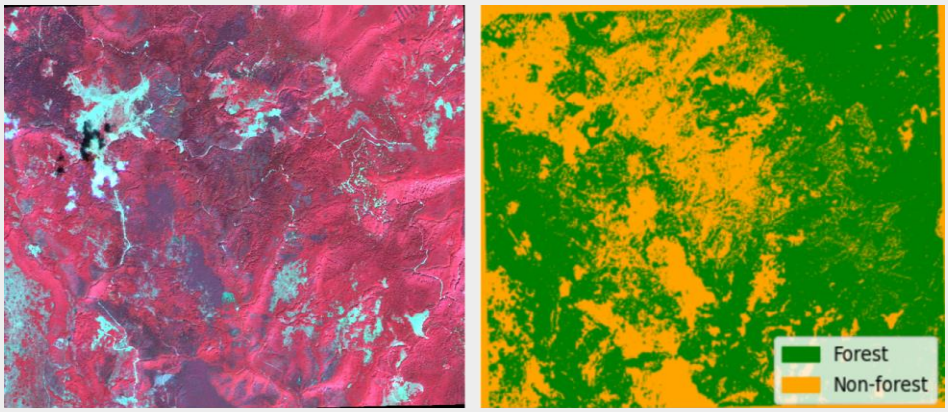
1. Users specify a number of clusters K $\{c_1^0, \ldots, c_K^0\}$.
2. Algorithms initialize K centroids so that they are placed as far as possible from each other. Calculate the centroids of clusters such as $u_j^i = \frac{1}{|c_j^i|}\sum_{x \epsilon c_i^j} x$ where i represents the *i-th* iteration.
3. Each data point is assigned to the closest centroid based on a distance metric (usually Euclidean distance) i.e., minimizing an objective

function. For the objective function, which represents the quality of clustering, the sum of error (SE) is often used $J = \sum_{i=1}^{K} \sum_{x \epsilon c_i} (x - u_i)^2$ where x represents the data sample and u_i represents the centroid to which x has been assigned $u_i = \frac{1}{m_i} \sum_{x \epsilon c_i} x$.

4. After points are assigned to clusters, the centroids are recalculated. The new centroid is the mean of all the data points within the cluster.
5. This process is repeated iteratively until centroids no longer change or a predefined number of iterations is reached.

K-means is an efficient, relatively fast, and scalable clustering algorithm. One of the biggest challenges is selecting the number of clusters K and sensitivity to randomly chosen centroids. If initial centroids are poorly selected, algorithms can get stuck in local optimums that are very different from the global optimum. Due to that, usually multiple runs are performed, and a solution that minimizes the similarity measure is chosen. Another key issue in clustering is selecting the optimal number of clusters. The appropriate number of clusters can be chosen based on prior knowledge about data or by using a heuristic approach.

Example: Classify the Sentinel-2 image into 2 classes: forest and non-forest.



As seen in the previous chapters, a satellite image consists of multiple bands such as Red, Green, Blue, Near-infrared etc, and each pixel is represented as a vector of its spectral values across different bands. Application of the k-means algorithm starts with defining the *k* number of clusters (in this case, the number of classes that we want to identify on the satellite image). The k

clusters' centroids are randomly initialized in spectral space. Each pixel is assigned to a cluster whose centroid is closed based on Euclidean distance in spectral space. After that, the centroids are updated by averaging the spectral values of all pixels in each cluster. The process is repeated until the centroids no longer change significantly.

8.2 Hierarchical clustering

Hierarchical clustering is implemented through a splitting (divisive) or merging (agglomerative) approach. It uses distance measures where the number of clusters is unknown. It shows the hierarchy of merging or division of clusters via a tree-based diagram called a dendrogram. In the dendrogram, each level represents one interaction of the algorithm. The agglomerative algorithm works by grouping data one by one based on the distance measure of all pairwise distances between the data points. For a given set of data points x_i where $i = 1, \ldots, N$

- Start by assigning each point to a cluster $c_i = x_i$. The number of clusters will be equal to the number of data points $K = \{c_1, \ldots, c_N\}$.
- Find the pair of nearest clusters (c_i, c_j) such that $D(c_i, c_j) \leq D(c_i^{'}, c_j^{i})$, $\forall c_i^{'} \neq c_j^{i} \, \epsilon \, K$ and merge them into one cluster c_k. Delete c_i, c_j from K and insert c_k so that cluster number is equal to *N-1*.
- Compute the similarity between the new clusters and each of the old clusters.
- The process continues until all items are classed into a single class of size *N*. By using the dendrogram, the number of actually present clusters is determined.

The distance can be defined in different ways, distinguishing:

- Single linkage clustering (**Figure 49** (a)) - The distance between one cluster and another cluster is equal to the shortest distance (shortest edge between two nodes in the graph) from any member of one cluster and any member of the other cluster. It provides good performance for long, elongated clusters and nonconvex shapes, but it is sensitive to noise.
- Complete-linkage clustering (**Figure 49** (b)) - The distance between one cluster and another cluster is equal to the greatest distance from

any member of the cluster to any member of another cluster. It is less sensitive to outliers, but it tends to create a compact and spherical-shaped cluster, and

- Average-linkage clustering (**Figure 49** (c)) - The distance between one cluster and another cluster is equal to the average distance from any member of one cluster to any member of another cluster.

The main advantage of hierarchical approaches, beyond their ease of implementation, is that they do not require prior knowledge of the number of clusters. However, they are highly sensitive to noise and outliers and struggle to handle clusters of different sizes or complex shapes. In addition, hierarchical methods are computationally expensive, which limits their scalability to large datasets.

Another challenge of hierarchical clustering is the absence of an explicit global cost function, which makes it difficult to objectively determine the correct number of clusters from the dendrogram. In a dendrogram, the bottom nodes correspond to individual data points, while internal nodes represent clusters formed through successive merging. The vertical axis indicates similarity (also referred to as cluster height): shorter branches reflect higher similarity, whereas longer branches indicate lower similarity. Clusters are defined by drawing a horizontal cut across the dendrogram at a chosen similarity level, with all points connected below the cut belonging to the same cluster.

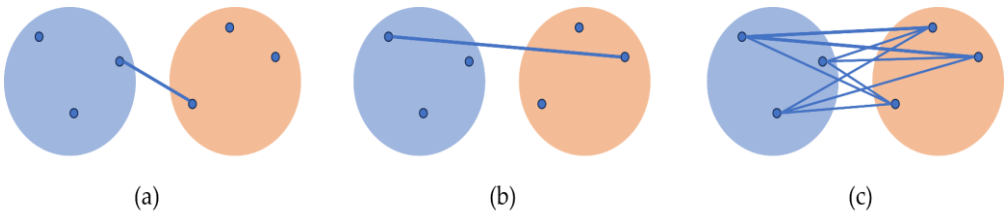


Figure 49 Similarity measures in hierarchical clustering (a) single-linking, (b) complete-linking, and (c) average-linking

8.2.1 Density-Based Spatial Clustering of Application with Noise (DBSCAN)

DBSCAN is a simple and effective density-based clustering algorithm that enables the identification of clusters with random shapes and sizes. The principle is that clusters correspond to regions with a high density of points,

whereas noise is associated with areas of low point density. The DBSCAN is defined by two parameters ε and *minPts*. ε represents the radius used to define neighbors, i.e. if the distance between two points is lower or equal to ε, the points are considered as neighborhood points, while minPts defines the minimal number of points within the ε region.

For a given data set X containing a total of $i = 1, \ldots, N$ objects, DBSCAN formulates a local density as $density(x_i)$ in the neighborhood of the *i-th* point as the total number of points in its neighborhood $density(x_i) = count\big(N_\varepsilon(x_i)\big)$ where $N_\varepsilon(x_i)$ represents estimation of local density, i.e. $N_\varepsilon(x_i) = \big(x_j \forall j, distance(x_i, x_j) < \varepsilon\big)$. Based on the number of points in a neighborhood, objects can be classified into three categories (**Figure 50**):

- core points (x_{core}) - if at least *minPts* are in the neighborhood of object x_i,
- border point (x_{border}) - if the number object belongs to the neighborhood of x_{core} and local density is less than *minPts*, and
- noise point (x_{noise}) - if in the neighborhood of radius ε, there are fewer than minPts of an object and none of them is the core point.

The point x_i is density reachable if it is in the neighborhood of x_{core}. The process begins by picking an unvisited point x_i. If a point x_i is identified as a core point, it forms a new cluster (x_{c1}) and all points that are density-reachable from x_{c1} are added to the cluster. The cluster is then expanded recursively by examining the neighbors of each neighbor. Points that are not assigned to any cluster after the process are considered noise.

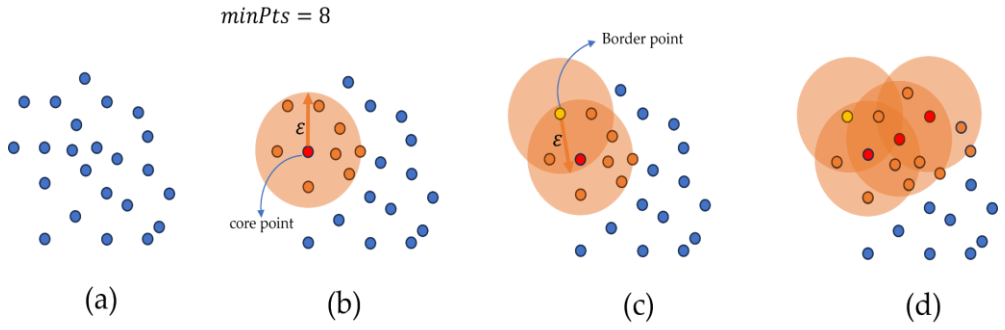


Figure 50 Graphical presentation of key definitions in DBSCAN (a) cluster, (b) core data point, (c) border data point, (d) density reachable object

One of the main advantages of DBSCAN is that it does not require specification of the number of clusters. Moreover, it is capable of efficiently handling noisy data, finding arbitrary-shaped and sized clusters, and it is mostly insensitive to point ordering within the database. However, it has some limitations.

The algorithm is highly sensitive to the user-specified parameters global density threshold ε. If ε parameter is too high, the algorithm may merge distinct clusters into a single cluster that should remain separate. On the other hand, low ε value results in misclassifying objects as noise if the density within the cluster is not satisfied. Therefore, accurate results can be obtained if the parameter is set to its optimal value. The chosen value plays a critical role in the clustering outcome. Another challenge is dealing with datasets with high-density variations since DBSCAN uses a fixed ε for all points. ε and *minPts* parameters are not flexible to handle large differences, resulting in missing sparse or merging dense clusters. Moreover, dealing with high dimensionality is limited since the data space grows exponentially with the increase of dimensionality (a phenomenon known as curse of dimensionality). As a result, most high-dimensional datasets are sparse, and since DBSCAN relies on a fixed density threshold, many points may be incorrectly labeled as noise. High dimensionality also increases computational cost and can degrade performance. To address these challenges, preprocessing steps such as dimensionality reduction can be applied before clustering.

8.3 Dimensionality reduction

The number of input features in a given dataset is known as dimensionality. Although having multiple features can help distinguish between different objects, handling high-dimensional datasets is challenging due to the curse of dimensionality. Redundant or irrelevant variables can degrade algorithm performance, and the computational cost increases significantly. Dimensionality reduction aims to reduce the number of features that are used to represent objects while preserving their essential structure and patterns. For example, in remote sensing, high-dimensional data such as hyperspectral images often contain redundant or irrelevant information that can lead to overfitting and require more memory and longer processing time. Dimensionality reduction helps mitigate these issues by reducing the number

of features to a manageable size while maintaining the integrity of the original data set.

The reduction of feature number can be made by:

- Feature selection - aims to identify the most relevant features while discarding the less significant. Common approaches include filtering, wrapper methods, and embedded, and
- Feature extraction - aims to create new features by combining the existing ones, providing the dimensionality reduction without missing relevant information.

Filtering methods use statistical techniques such as correlation or Chi-square tests to select optimal features. In terms of computation, they are very fast and computationally inexpensive.

Wrapper methods are based on using classification techniques (SVM (see Section 12), decision tree (see Section 13). The model is trained by iteratively using a subset of features. The feature subset used for training can be created by using methods such as forward selection (Start with no features and iteratively keep adding one at a time) or backward elimination (it starts with all features and after each iteration it removes the least significant). It will continue until the addition/removal of new features does not improve model performance. This approach allows the creation of an optimal subset for model training, thus resulting in higher accuracy, but it is computationally intensive since it requires training and evaluating the model multiple times.

Embedded methods combine feature selection into the training process. The model identifies the most relevant features based on the built-in mechanisms such as Lasso (L1 regularization) and Elastic net (*L1* and *L2* regularization) regularization or tree methods (see Section 9.14). Embedded methods are more computationally efficient than wrapped methods and more accurate than filter methods.

An underfitted model occurs when it does not have enough parameters or complexity to capture the patterns in the underlying system, resulting in poor performance even on the training data. On another hand, including too many features can lead to overfitting (the model works well on training data, but it fails during testing). One common solution to reduce overfitting is

dimensionality reduction, which eliminates redundant or irrelevant features, helping the model focus on the most informative variables and improving its generalization ability.

8.3.1 Principal component analysis

Principal component analysis (PCA) is one of the most commonly used dimension reduction methods. It represents the process of identifying the principal components of samples, i.e. the variables that capture the most important patterns in the data. These components are linear combinations of the original features and are designed to explain the maximum possible variance in the dataset with fewer dimensions. Essentially, it transforms the data into a smaller set of features while preserving as much information as possible. Each component is independent and orthogonal, i.e., each principal component (PC) does not depend on another. PC is based on the analysis of the correlation or the covariance structure in the set of measurements X on m variables for n observations. If the covariance is positive, then the two variables increase or decrease together (correlated), while a negative covariance means that one variable increases when another decreases (the two variables are inversely correlated).

Eigenvectors of the covariance matrix are computed to determine the PCs of the data. Each eigenvector represents a direction in the feature space along which the data varies the most, and the corresponding eigenvalue indicates the amount of variance captured along that direction. Let A be an $m \times m$ covariance matrix. If there exists a nonzero vector x in R^n such that $Ax = \lambda x$, the scalar λ is called the eigenvalue of A and x eigenvector of A. λ represents the amount of variance explained by the corresponding PC, while the vector x ($mx1$) represents the direction of that component in the feature space. We can rewrite $Ax = \lambda x$ as $(A - \lambda I_m)x = 0$ where I_m is the mxm identity matrix.

For a nonzero solution ($x \neq 0$) this homogeneous system must have no unique solution for x i.e., $det(A - \lambda I_m) = 0$. The determinant gives polynomial equation of degree m in λ. Solving this polynomial gives all the eigenvalues $\lambda_1, \ldots, \lambda_m$ of A. The determined λ values are substituted back into the equation $(A - \lambda_i I)x_i = 0$ to calculate the corresponding eigenvectors x. The eigenvector with the largest eigenvalue λ_1 is the direction of greatest

variation (also known as first PC), while m^{th} larges λ_m is m^{th} PC ($\lambda_1 > \lambda_2 > \lambda_3$...) (**Figure 51**).

Let's consider a image with two attributes NDVI and NDWI each with 3 samples.

$$NDVI = \begin{bmatrix} NDVI_1 \\ NDVI_2 \\ NDVI_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}, \quad NDWI = \begin{bmatrix} NDWI_1 \\ NDWI_2 \\ NDWI_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$$

These four attributes can be combined into matrix of attributes S

$$S = \begin{bmatrix} NDVI_1 & NDWI_1 \\ NDVI_2 & NDWI_2 \\ NDVI_3 & NDWI_3 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ -1 & 0 \end{bmatrix}$$

From the matrix, we can compute a covariance matrix A

$$A = S^T S = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

$$\det(A - \lambda I) = \begin{bmatrix} 2-\lambda & -1 \\ -1 & 2-\lambda \end{bmatrix} = (2-\lambda)^2 - 1 = \lambda^2 - 4\lambda + 3 = 0$$

$\lambda_1 = 3$, and $\lambda_2 = 1$

To find x, we substitute the two eigenvalues into matrix

$$\begin{bmatrix} 2-\lambda & -1 \\ -1 & 2-\lambda \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

For $\lambda_1 = 3$, we will have

$$\begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$ i.e.

$x_1 + x_2 = 0 \Rightarrow x_1 = -x_2$, therefor all vectors $[1 \quad -1]^T$ are solutions.

For $\lambda_2 = 1$, we will have

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$ i.e.

$x_1 - x_2 = 0 \Rightarrow x_1 = x_2$, therefor all vectors $[1 \ 1]^T$ are solutions.

The matrix of eigenvectors can be explained as a rigid rotation in a high-dimensional space in which the covariance matrix is diagonal. When this transformation is applied to the original data, it is projected onto a principal

direction (the axis corresponding to the highest variance). The projection of original data onto each eigenvector gives an idea about the importance of the feature to the object. The larger values in the corresponding component of the principal vector mean that the feature is important in explaining the variability of the data.

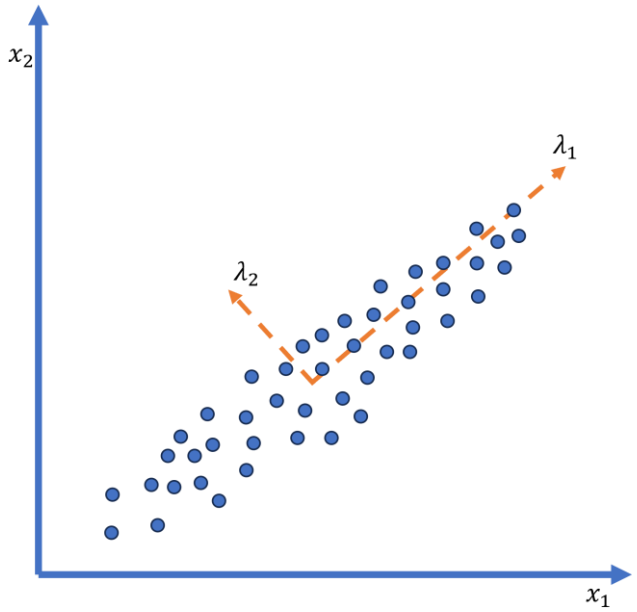


Figure 51 Graphical representation of a PCA transformation in 2D.

Principal components are new variables formed as linear combinations of the initial variables. Those combinations are done in such a way that PCs are uncorrelated (i.e., orthogonal to each other). There are as many PCs as variables in the data, but they are designed in such a way that most of the information within the initial variables is compressed into the first components. The first PC corresponds to the direction of maximum variance in the dataset, while the second PC is orthogonal to the first and captures the maximum remaining variance, and so on. To reduce dimensionality, the PCs are ranked in order of decreasing eigenvalue, and only enough components are retained to preserve a desired percentage of the total variance—commonly around 85–95%. In this way, dimensionality is reduced without losing much information since components with low information are discarded. However, a key limitation is that PCs are often less interpretable, as they are combinations of original variables and do not have a direct, intuitive meaning.

Principal component analysis (PCA) is based on analyzing the variance in the data and does not require any tuning parameters. It is non-iterative and does not suffer from issues related to local optima. Additionally, directions with small variance typically capture mostly noise, so retaining only the top principal components can effectively reduce noise in the data. However, PCA is limited to linear projections and therefore may not perform well when the underlying structure of the data is nonlinear.

9 OPTIMIZATION

Imagine standing on a hill, trying to figure out which direction to throw the ball to get downhill the fastest. To determine that direction, we need to know how steep the ground is and in which direction it goes down. The slope of a line is the rate of the vertical change (change in the y-coordinate (rise)) for the corresponding change in the x coordinates (run) for points on the line. This is the basic idea of gradient-based optimization. In reality, the slope of the terrain is not constant, and it changes from point to point. Due to that, for nonlinear functions, the rise over run presents the average rate of change between points. The main idea is to measure the slope between two points ($f(x)$ and $f(x+h)$) that are close together to find the rate and direction of the change (**Figure 52**), i.e.

$$slope = \frac{raise}{run} = \frac{\Delta y}{\Delta x} = \frac{f(x+h) - f(x)}{h}$$

As two points get closer (i.e. $h \to 0$), the limit is the derivative at point x, which represents the exact slope of the curve at the point.

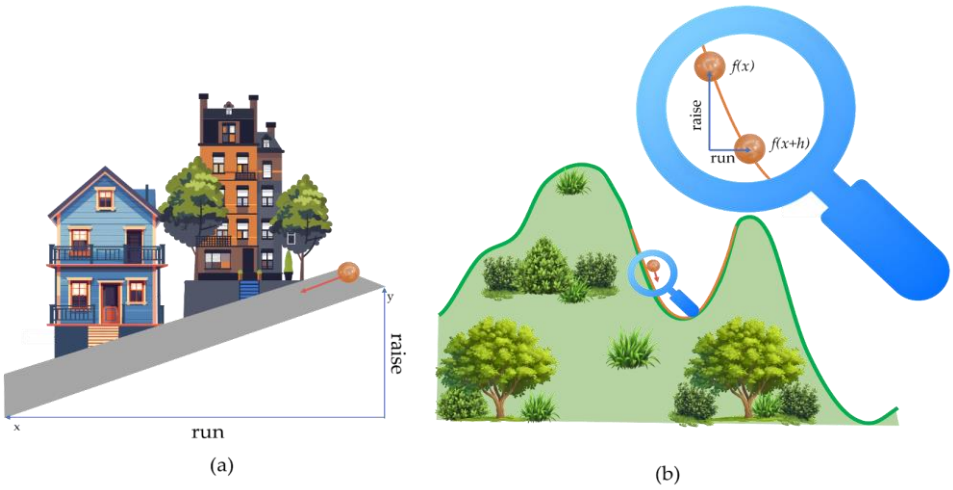


Figure 52 Run-over-rise (a) constant slope (b) slope constantly changes (non-linear function)

Let us consider the terrain slope (orange curve in **Figure 52**) represented by the function $f(x) = x^2$ and suppose we want to determine the slope at the point x=1. The first derivative is equal to $f'(x) = 2x$ which represents the equation of the tangent and the slope at the point x is equal to 2. Meaning that

the function increases at a rate of 2 units (rise) for every 1 unit increase in x (run).

Let f be a predictor that maps an input x to an output y. Most machine learning algorithms involve optimization by minimizing or maximizing a function of $f(x)$. The $f(x)$ is referred to as the objective function or criterion. The objective function is minimised through the loss function, also known as the cost function. The loss function $L(f(x,w),y)$ measures the difference between the prediction (obtained by applying the model with parameters w to make a prediction on x) and the true value y. It measures the model's performance by calculating the training error and reflects how far the prediction is from the actual outcome y. The ultimate goal is to minimize the loss function during the training phase to optimize the model parameters, i.e.

$$\min_{w\in\Omega} \sum_{i=1}^{n} L_i(f(x_i, w_i)\, y_i\,)$$

where x_i are the input data, y_i are the output data, and w_i are the model parameters that need to be optimized. The optimizer determines how the network updates its parameters based on the loss function. To solve the $min_w L(f(x,w),y)$, a gradient descent algorithm is frequently used.

9.1 Gradient

Let $f: \Omega \subset R^n \to R^m$ be a function that maps n *dimensional* vectors to m dimensional space, where Ω is a subset of an Euclidean space. Formally, if there exists a unique linear map $Df(x_0): R^n \to R^m$ such that

$$\lim_{||h||\to 0} \frac{||f(x_0+h)-f(x_0)-Df(x_0)[h]||}{||h||} = 0\,,$$

the function f is differentiable at x_0 and $Df(x_0)$ is called the derivative of f at x_0. In the case of a univariate function (*m=n=1),* the derivative $Df(x_0) = f'(x_0)$ corresponds to the slope of the tangent line at the point x_0. Moreover, the best linear approximation of f around x_0 is defined as:

$$g(x) = f(x_0) + f'(x_0) \cdot (x - x_0)$$

and it geometrically corresponds to the tangent line at the graph of f at the point $\big(x_0, f(x_0)\big)$. Similarly, in the case of a multivariate function (for any

$m, n \geq 1$), the derivative $Df(x_0)$ defines the tangent plane of f at $(x_0, f(x_0)) \in R^{m+n}$ which is defined by the equation

$$g(x) = f(x_0) + Df(x_0) \cdot (x - x_0).$$

This represents the best linear approximation of f around the point x_0, i.e., it provides the slope of $f(x)$ at that point.

If $Df(x)$ exists for all $x \in \Omega$, we say that *it is differentiable*. If addition, if $Df(x)$ is continuous in x, the f is continuously differentiable.

Since $Df(x_0)$ is the best linear map that approximates f, it can be represented as a matrix multiplication

$$Df(x_0) = J_f(x_0) \cdot h$$

where $J_f(x_0)$ is known as the Jacobian of f.

If $f(x) = f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)$ is the vector-valued function, then its Jacobian is

$$[Df(x_0)]_{i,j} = \frac{\partial f_i}{\partial x_j}(x_0) \text{ i.e., following } mxn \text{ matrix}$$

$$Df(x_0) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x_0) & \cdots & \frac{\partial f_1}{\partial x_n}(x_0) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(x_0) & \cdots & \frac{\partial f_m}{\partial x_n}(x_0) \end{bmatrix}$$

If f is real-valued, the Jacobian $Df(x_0)$ reduces to a row vector:

$$Df(x_0) = \left[\frac{\partial f}{\partial x_1}(x_0) \ \ldots \ \frac{\partial f}{\partial x_n}(x_0) \right].$$

The transpose vector consisting of partial deviation with respect of each input is called the gradient of f at $x_0$and it is often denoted as$\nabla f(x_0)$ i.e.

$$Df(x_0)(h) = \nabla f(x_0) \cdot h$$

The gradient has several important properties that are relevant to optimization:

- The direction of the gradient $\nabla f(x_0)$ at a particular point x_0 indicates the direction in which f increases the most locally around x_0 (steepest ascent direction). Consequently, the opposite direction, $-\nabla f(x_0)$,

points toward the direction in which f decreases the most (steepest descending direction),

- The magnitude of the gradient vector represents the rate of the fastest change (i.e. how steep the slope is). If the slope is negative, it means that a small change in x results in a decrease in $f(x)$. Similarly, if the magnitude is positive, a small change in x results in an increase of $f(x)$, and
- At any point, the gradient vector $\nabla f(x)$ is orthogonal to the level set of f at x. In 2D, these level sets correspond to contour lines. Consequently, moving along the contour line does not change f, whereas moving perpendicular to it results in a maximum or minimum change in f.

9.1.1 Basic geometrical properties of functions

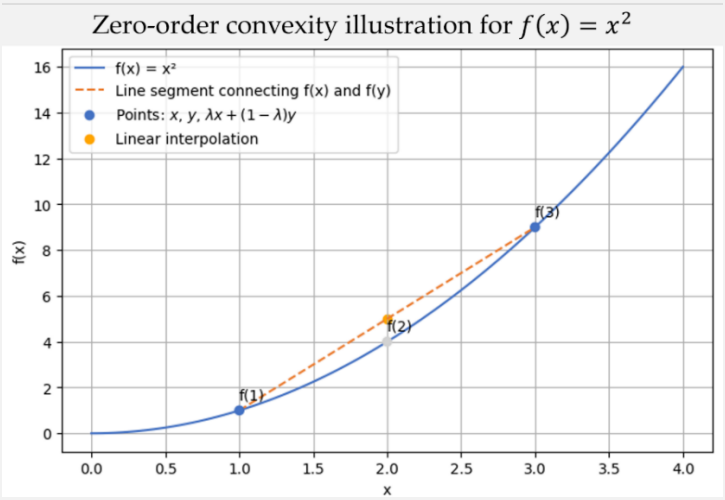
Convexity: The zero-order definition states that a function f is convex if its domain Ω is a convex set, i.e., if for every $x, x_0 \in R$ and $0 \leq \lambda \leq 1$ the inequation

$$f(\lambda x + (1-\lambda)x_0) \leq \lambda f(x) + (1-\lambda)f(x_0).$$

Let consider $f(x) = x^2$ and pick two points $x = 1$, $x_0 = 3$, and $\lambda = 0.5$.

$$f(0.5 \cdot 1 + 0.5 \cdot 3) \leq 0.5 \cdot f(1) + 0.5 \cdot f(3)$$

$f(2) \leq 0.5 \cdot 1 + 0.5 \cdot 9$ i.e. since $4 \leq 5$, the zero-order condition convexity holds.



Geometrically this mean that for each pair of points $(x, f(x))$ and $(x_0, f(x_0))$ that lying of the graph of f, the connecting line segment remains above the graph.

In addition, it is possible to define convexity in terms of first and second-order conditions. Let function $f: R^n \rightarrow R$ be differentiable. The f is convex if and only if for every $x, x_0 \in R^n$the inequality $f(x_0) \geq f(x) + \nabla f(x)^T (x_0 - x)$ is satisfied.

A twice differentiable function $f: R^n \rightarrow R$ is convex if the second derivative is always greater than 0. Let f be a function and x_0 is a point in the domain of f.

If $\frac{d^2 f}{dx_0^2} > 0$ the f is convex at x_0 , if $\frac{d^2 f}{dx_0^2} < 0$ then f is concave T x_0, if $\frac{d^2 f}{dx_0^2} = 0$, then x_0 is a candidate for a local maximum, a local minimum of an infection point (a point in which f changes from being convex to concave, also known as a saddle point).

For example, the function $f(x) = x^2$ is convex since $f^{'} = 2x$ is a monotonically increasing function and $f^{''} = 2 > 0$.

There are two important properties of the convex function:

- Any local minimum is also a global minimum. Due to that, the local search algorithms are effective for optimization, and
- For each pair of points (x, $f(x)$) and (x_0, $f(x_0)$) lying on the f, the connecting line segment must be above f everywhere. If a function is differentiable, this tangent is the linear function, i.e.
  $l(x) = f(x_0) + \langle \nabla f(x_0), x - x_0 \rangle$.

Convex functions are desirable because they are easier to optimize, as any local minimum of a convex function is also a global minimum. For example, the squared error is a convex loss function. Many loss functions that are used in ML involve norms, and all norms are convex.

Strong convexity: A function f is α-strongly convex for $\alpha > 0$ if for all $x, x_0 \in \Omega$

$$f(x_0) \geq f(x) + \nabla f(x)^T (x_0 - x) + \frac{\alpha}{2} \left|| x_0 - x \right||^2.$$

A strong convex function always lies above a quadratic approximation (parabola) that passes through the point $(x, f(x))$. If a function is strongly convex, then it has one global minimum.

Lipschitzness: A differentiable function f is L-Lipschitz if the norm of the gradient is bounded by L, i.e.

$\left|\left|\nabla f\right|\right| \leq L$, implying $|f(x_0) - [f(x) + \nabla f(x)^T(x_0 - x)]| \leq \frac{L}{2}\left|\left|x_0 - x\right|\right|^2$ for all $x, x_0 \in \Omega$.

The function that satisfies a Lipschitz condition on Ω is uniformly continuous on Ω. Geometrically, the graph of L-Lipschitz stays within a cone with slope $\pm L$, i.e., if for all x the function never rises above or falls below the boundaries defined by this slope. This limits how quickly the function output can change: a smaller L means a slower change.

Smoothness: A differentiable function $f: \Omega \to R$ is β-smooth if its gradient is β-Lipschitz continuous, i.e.

$\left|\left|\nabla f(x) - \nabla f(y)\right|\right|_2 \leq \beta\left|\left|x - y\right|\right|_2 \qquad \forall x, y \in \Omega$

The β-smoothes ensures that the gradient cannot change too quickly and must be bounded by β value. The smoothness implies a quadratic upper bound on the function, i.e., in a fixed point x_0 the convex function lies above its tangent line, and the smooth convex function always lies below the parabola which passes through the point $(x_0, f(x_0))$ (**Figure 53**).

The function $f(x) = x^2$ is L=2 smooth since $f' = 2x$. For the point $x_0 = 1$ the tangent part (lower bound) is
$f(1) + f'(x - 1) = 1 + 2(x - 1) = 2x - 1$, while the upper band is

$$f(1) + \nabla f(1)(x - 1) + \frac{L}{2}(x - 1)^2 = 1 + 2(x - 1) + (x - 1)^2 = x^2$$

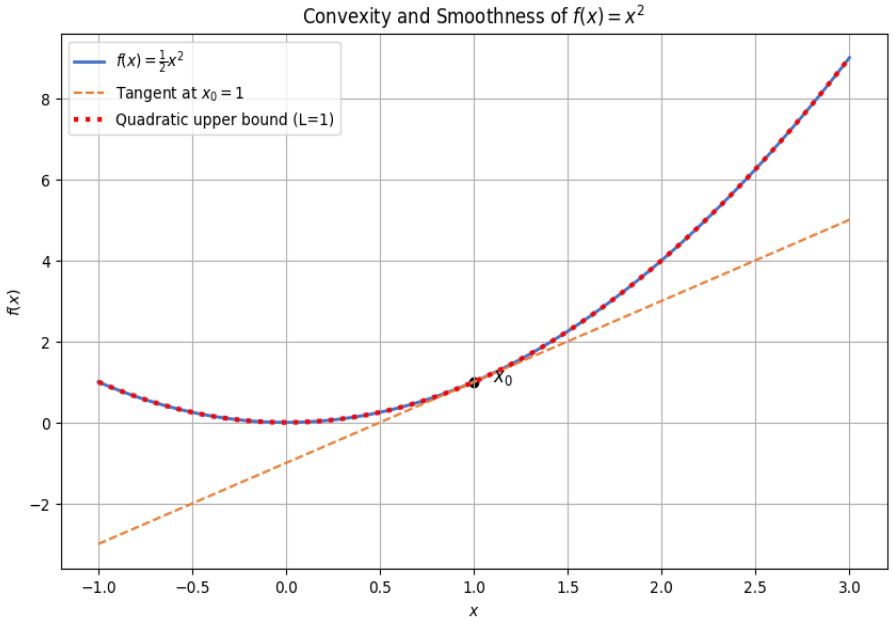


Figure 53 Convexity and smoothness of f(x)=x².

For a twice differentiable function, an equivalent condition for β-smoothness is

$$0 \preceq \nabla^2 f(x) \preceq \beta I_d$$

where I_d is the identity matrix, the lower bound comes from convexity, and the upper bound comes from β-smoothness of function. Moreover if x^* is the minimum point of a β-smooth function f, then for all $x_0 \in \Omega$ $\left|\left|\nabla f(x_0)\right|\right|_2 \leq \beta\left|\left|x_0 - x\right|\right|_2$, i.e., if point x_0 is close to the x^* then the gradient of x_0 must also be small. This means that any algorithm following the gradient of the function should slow down as it approaches the minimum. Due to that, the smoothness of the function determines the simplicity and efficiency of minimization by using gradient descent.

9.1.2 Chain rule

The chain rule is essential in optimization, especially when working with deep learning and composite functions. The chain rule is often written as $\frac{df}{dx_0} = \frac{df}{dg}\frac{dg}{dx_0}$ where f is a function of g, which is itself a function of x_0. It enables the

correct calculation of the gradient when a function is made up of layers of other functions.

For example, suppose you want to monitor changes in the water level during the day. The water level W_l depends on the amount of rainfall r, and they are related through a function $W_l = f(r)$. The first derivative $\frac{dW_l}{dr}$ can be used to determine the rate of change of water level with respect to the rainfall. However, the rainfall changes over time t, and it can be modeled as a function of time, i.e., $r = g(t)$. The derivative $\frac{dr}{dt}$ represents the amount of rainfall accumulated over time. The change of the water level during the day is represented as a composite function $W_l = f(g(t))$.

To determine the derivation of the function, it is necessary to determine the derivation of the composite $f \circ g$. If f and g are functions such that g is differentiable at x_0 and f is differentiable at $g(x_0)$, then the composite $f \circ g$ is differentiable at x_0 and

$$D(f \circ g)(x_0) = D(f)\big(g(x_0)\big) \circ Dg(x_0).$$

If $D(f)\big(g(x_0)\big)$ and $Dg(x_0)$ are linear, their composition is also linear. The chain rule for invariant functions has a direct analogue in multivariate functions, and it can be written in the form of Jacobians as

$$D(f \circ g)(x_0) = (Df)\big(g(x_0)\big)Dg(x_0).$$

The chain rule can also be directly applied to the inverse of a function f. Let f and f^{-1} be differentiable. Then for any $x \in range(f)$

$$Df^{-1}(x_0) = (Df)^{-1}\left(f^{-1}(x_0)\right)$$

9.1.3 Extreme points

Finding the extreme points of a function is of exceptional importance. The minimum of the function f can be found between points that satisfy $f'(x) = 0$ (also known as stationary points). Assume that the function $f: \Omega \subseteq R^n \to R$, is convex and differentiable and defined on a subset Ω of R^n. The $x_0 \in \Omega$ is a local minimum of f if $f(x_0) \leq f(x)$ for all x within a small neighborhood of x_0. The x^* is a global minimum of f if $f(x^*) \leq f(x)$ for all $x \in \Omega$.

In optimization problems, the aim is to reach the global minima. However, in many cases, the problem can be so challenging that finding a local minimum is considered a successful outcome.

If $x_0 \in int(\Omega)$ is a local minimum, then $\nabla f(x_0) = 0$; therefore local/global minimum belong to the set of stationary points defined as $\{x \in \Omega \mid \nabla f(x) = 0\}$.

As already mentioned, the derivative is the best linear approximation of a function. The first-order Taylor expansion can be used to formulate this concept rigorously as

$f(x) = f(x_0) + (\nabla f(x_0), x - x_0) + o||x - x_0||$, i.e., the approximation error satisfies $\lim_{x \to x_0} \frac{\xi(x-x_0)}{||x-x_0||} = 0$. The little-o notion $o||x - x_0||$ is used to express that one function grows much more slowly than another. The first-order Taylor expansion is a linear function that approximates the function around a certain point. In 1D space, this is tangent to the curve, while in multidimensional space, it is a hyperplane that is tangent to the hypersurface at that point. However, to find local minima in multidimensional space, simply finding a point where the gradient is zero is not enough, and the Hessian matrix is needed.

The Hessian matrix encodes how the gradient changes with respect to each input variable, i.e., it describes the curvature of the loss landscape function in every direction. It is used to build the quadratic approximation of the function, enabling methods to find critical points efficiently, such as saddle points or flat regions.

In order for a critical point x_0 to be a local minima in multiple dimensions, the matrix of second derivatives $D^2 f(x_0) = \nabla^2 f(x_0)$ (i.e., Hessian matrix) of the objective function f at the point x_0 must be positive definite, that is all its eigenvalues are positive (**Figure 54** (a)). A local maxima has the largest value in the neighborhood and a negative definite Hessian (**Figure 54** (b)), If the Hessian is indefinite (eigenvalues of mixed sign) x_0 is a saddle point (**Figure 54**(c)), while the singular Hessian (zero determinant) indicates that the test is inconclusive.

One of the most important features of convex function is that any local minima guarantees a global minimum. Although some convex functions have

flat regions rather than a single point, any point within that region is a suitable solution. On the other hand, nonconvex functions (which are very often in neural networks) have many local minima. The local minima that have a similar value of the cost function are not problematic for optimization. However, if local minima with a high cost in comparison with global minima are common, that can represent a challenge for gradient-based methods. Moreover, in high-dimensional space, saddle points are more common and more likely to have a higher cost than local minima. The large flat regions (plateau) in the loss landscape represent a major challenge for optimization since the gradient and the Hessian are all zero and this slows or stalls the optimization.

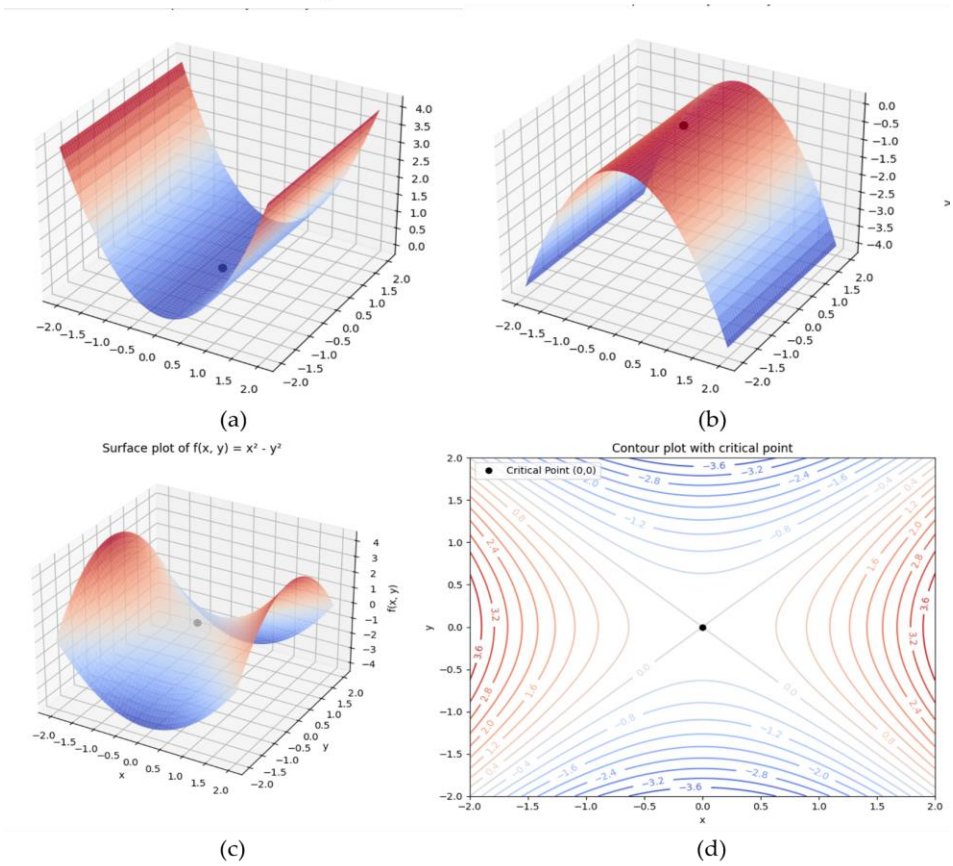


Figure 54 (a) Hessian is positive defined, (b) negative defined, and (c) Hessian is indefinite. (d) The critical point is (0, 0) since $\nabla f(x)=0$. Since the Hessian is indefinite, the critical point represents the saddle point

9.2 Gradient Descent

In Machine Learning, we aim to find the analytically set of parameters that minimize the loss function. In general, function minimization is performed under the structural assumptions that the function is convex and smooth.

The gradient descent algorithm iteratively approaches the point at which the function *f* achieves its minimum by taking the steps in the direction of steepest descent. The gradient at the current position is scaled by a learning rate and subtracted from the value of the current position (makes a step). The gradient is subtracted because we want to minimize a function. This process can be written as:

$$w_{n+1} = w_n - \eta \nabla f(w_n)$$

where η is the learning rate that scales the gradient and thus controls the step size.

The main steps of the gradient descent method are:

1. Initialize weights randomly w_0,
2. Determine a descent direction by computing the gradient at this point,
3. Along that direction, make a scaled step in the opposite direction of the gradient and update the weights with the objective to minimize the losses (**Figure 55**),
4. Repeat steps 2 and 3 until either the maximum number of iterations is reached or the algorithm converges to a local minimum within a specified tolerance (threshold).

For most functions, the gradient will not reach exactly 0 in a reasonable amount of time. Therefore, stopping criteria need to be defined. Ideally, the algorithm stops once the gradient is sufficiently close to 0 i.e. if the norm of the gradient is below some predefined threshold $||\nabla L(w)|| < threshold$. Gradient descent involves 5 parameters: the starting point (is often an initial guess or randomly initialised), the gradient function (computes the gradient of the specified original loss function), the learning rate (which scales the step size), the maximum number of iterations and the tolerance (which conditionally stops the algorithm when convergence is achieved).

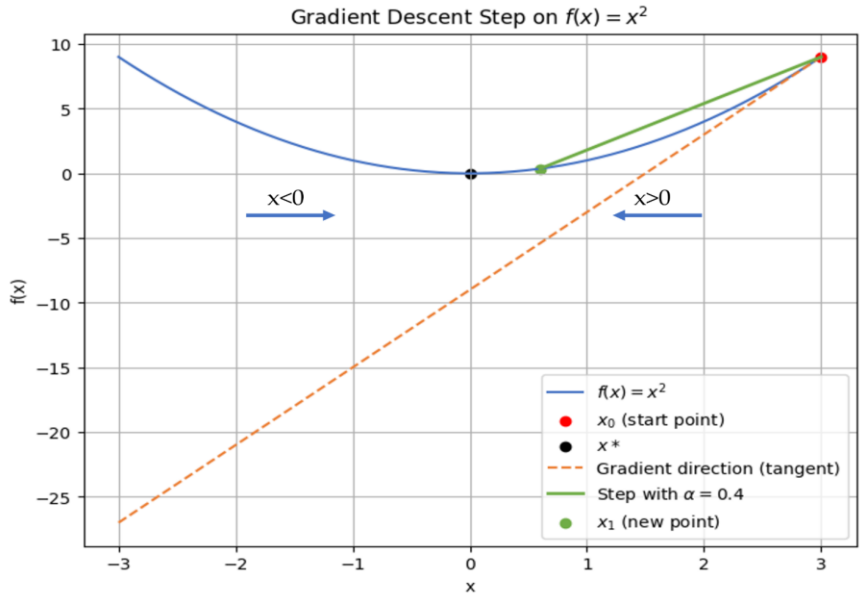


Figure 55 Gradient decent for function f(x)=x² with learning rate η=0.4. The global minimum is at x=0. Since $f'(x) = 0$ the gradient stops here. For x<0 the $f'(x) < 0$ so we can decrease f by moving to the right. For x>0 the $f'(x) > 0$ so we can decrease f by moving to the left. The starting point is set to $x_0 = 2$, the slope of the tangent line at that point is 4. The new point will have coordinates $x_1 = 2 - 0.4 \cdot 2 \cdot 2 = 2 - 1.6 = 0.4$ and $y_1 = 0.4^2 = 0.16$.

9.2.1 Choosing the step size

The learning rate determines the step size taken in the direction of the gradient. There are several approaches that can be used to determine the step size (learning rate) based on characteristics of the loss function: fixed step size, exact line search, and backtracking line search. In the first, the fixed learning rate is selected. However, the learning rate has a strong influence on the algorithm performance since:

- the smaller the learning rate, the slower the gradient descent converges. If the step size is small compared to the local curvature, the gradient direction w_{n+1} is very similar to or the same as w_n. As a result, the algorithm takes more steps or may reach the maximum number of iterations before arriving at the optimal point (**Figure 56** (a)). In addition, it could get stuck in the local minimum.

- If the learning rate is too big, the algorithm may not converge to the optimal point (jumping around) or even diverge completely to random locations on the curve.

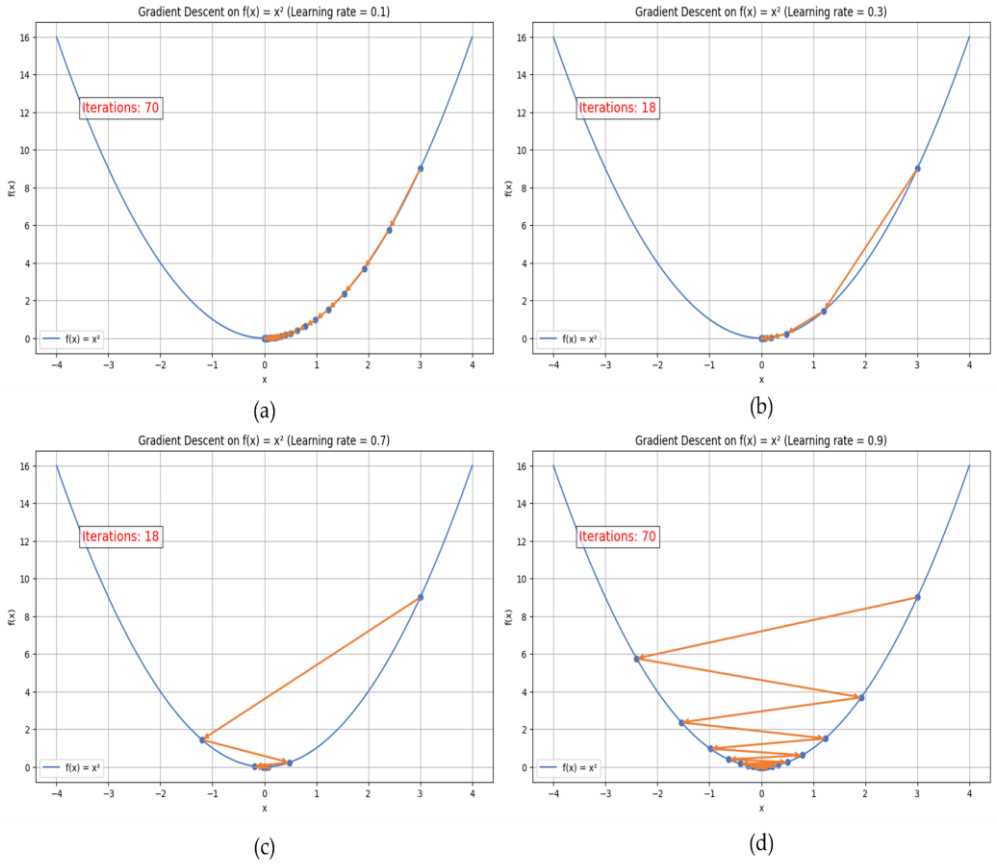


Figure 56 the gradient descent steps for minimizing the function $f(x)=x^2$ with different learning rates.

The best step size depends on the local curvature of the function. In gradient descent, the ideal learning rate is inversely proportional to the curvature. If curvature is high, the small step size should be used, while for wide curvature, the larger learning rate is preferable. Consequently, the optimal learning rate to ensure convergence is defined as

$$\eta < \frac{1}{L}$$

where $L = max f"(x)$ is the Lipschitz constant.

For example, if the function is quadratic $f(x) = x^2$, the second derivative is constant across all x ($f'' = 2$) so the curvature is also constant. The safe learning rate in that case would be $\eta < 0.5$. Figure 5 shows the trajectories and number of interactions for different learning rates and a given threshold ($||\nabla f(w)|| < 10^{-6}$).

In higher dimensions, things are more complicated since curvature is given by the Hessian matrix. The optimal learning rate is inversely proportional to the largest eigenvalue of the Hessian:

$$\eta = \frac{1}{\lambda_{max}}$$

where λ_{max} corresponds to the steepest curvature direction. Any smaller or slightly larger value will yield slower convergence (Figure), while the learning rate two times larger than optimal will cause divergence. However, computing and storing Hessian matrices in large learning models come with extreme computational costs. Fortunately, the Hessian-vector product can be approximated without computing the full matrix by using finite differences. The fixed-size method is commonly used for low-dimensional problems.

In *exact line-search*, the best learning rate is resolved as a 1D optimization problem. Given a starting point and the direction of the gradient, the learning rate is chosen to minimize the function in that direction, i.e.

$$\eta^k = argmin_{\eta \geq 0} f\left(x^k - \eta \nabla f(x^k)\right).$$

Solving this equation exactly is very time-consuming since minimization problems need to be solved in each step. The main advantage of this method is that information about the function's smoothness is not needed in advance. On the other hand, an exact line search is not used very often since it is more computationally demanding and not much more efficient compared with backtracking.

Backtracking line search is based on the idea of starting with a large step size and reducing it approximately. It starts by choosing two parameters $0 < \beta < 1$ and $\alpha \leq 1$. At iteration k, starting from $\eta = 1$, while

$$f\left(x^k - \eta \nabla f(x^k)\right) > f(x^k) - \alpha\eta \left|\left|f(x^k)\right|\right|_2^2$$

i.e., the Armijo condition is not satisfied, shrink the step size by setting $\eta = \beta\eta$ and repeat the step. Otherwise, perform the Gradient Descent update. This method is simple and is frequently used in practice.

9.2.2 Convergence of gradient descent

As already mentioned, gradient descent begins with a random initialization of parameters, followed by an an iterative optimization process aimed to trying to find the stationary point of the objective function. The convergence depends on the selected learning rate and the structural properties of function itself. While establishing convergence is fundamental, the convergence rate is equally critical. Convergence rate quantifies how fast the algorithm reach a specific error tolerance and it is usually measured by the number of iterations needed to converge. In practice, guaranteeing efficient function minimization is only possible by making certain assumptions about convexity and smoothness.

Let $f: R^n \to R$ be differentiable, convex, and β-smooth function with β>0, i.e.,

$$\left|\left|\nabla f(x) - \nabla f(y)\right|\right| \leq \beta\left|\left|x - y\right|\right|.$$

Then, Gradient descent with fixed step size $\eta = 1/\beta$ satisfies

$$f\left(x^k\right) - f(x^*) \leq \frac{\left|\left|x^0 - x^*\right|\right|_2^2}{2\eta k}$$

where $f(x^*)$ is the optimal value. The gradient descent is guaranteed to converge and it has a convergence rate of $O(1/k)$, where k is the number of iterations.

Reaching the global minimum may require too many iterations, so in practice we commonly aim to reach ϵ-suboptimal point $f\left(x^k\right) - f(x^*) \leq \epsilon$. The value of ϵ represent the tolerance (how close we want to be to global minimum $f(x^*)$), and it depends on the specific application. To reach ϵ-suboptimal point we need $O(1/\epsilon)$ iterations. The very small ϵ will lead to a large number of iterations. For example, if $\epsilon = 10^{-6}$ the 10^6 iteration is needed.

If function f is differentiable, possibly non-convex and β-smooth, finding the global minimum under this assumption is not guaranteed. Under this

assumption, the aim is to reach ϵ-stationary point x i.e. we want to find $x^* \in R$ such that

$$\left|\left|\nabla f(x^*)\right|\right|_2 \leq \epsilon \quad \text{where } \epsilon > 0$$

Gradient descent with fixed step size $\eta = 1/\beta$ satisfies

$$min_{t=0,..,k}\left|\left|f(x^t)\right|\right|_2 \leq \sqrt{\frac{2\beta}{\eta}\left(f(x^0) - f(x^*)\right)}.$$

The gradient descent rate for the optimization of a nonconvex function is $O(1/\sqrt{k})$ or $O(1/\epsilon^2)$. For example, if $\epsilon = 10^{-6}$, then the number of iterations is 10^{12}. This is the worst-case scenario, and convergence in practice is faster. However, there is no deterministic algorithm that can guarantee better performance.

Additionally, if function f is differentiable, β-smooth and α-strong convex, then the gradient descent with fixed step size $\eta = 2/(\alpha + \beta)$ or with backtracking line search satisfies

$$f\left(x^k\right) - f(x^*) \leq \gamma^k \left|\left|x^0 - x^*\right|\right|_2^2 \text{ where } \gamma^k = \left(1 - \frac{\alpha}{\beta}\right)^k \cdot \frac{\beta}{2}\, 0 < \gamma < 1.$$

The ratio $\kappa = \frac{\beta}{\alpha}$ represents the condition number of f. The condition number can be more precisely interpreted by using the Hessian matrix, i.e., $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$ where λ_{max} is the largest and λ_{min} is the smallest eigenvalue of the Hessian. The smaller the condition number is, the faster the convergence. Consequently, the larger β the slower the convergence. Geometrically, a high condition number produces elongated contour lines, a situation known as ill-conditioning. As the contours become more elongated, gradient descent deviates more from the optimal direction toward the minimum. (**Figure 57**). As a result, gradient descent tends to bounce back and forth along directions of high curvature, producing a zig-zag pattern and slower convergence. For small condition numbers, the contours are well-rounded, and gradient descent moves more directly toward the minimum. The convergence rate under strong convexity assumption is exponentially fast $O\left(\gamma^k\right)$. Therefore, the ϵ-suboptimal point is reached in $O(log(1/\epsilon))$ iterations. For example, if $\epsilon = 10^{-6}$ the $O(log(10^6)) \approx 14$ iteration is needed. So, the assumption of

strong convexity leads to much faster convergence and guarantees the reaching global optima.

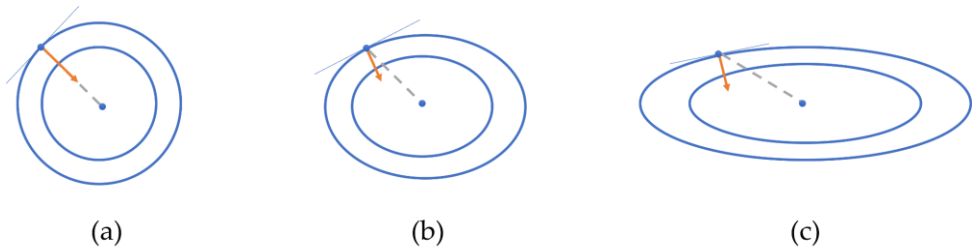


Figure 57 Diversion of gradient (orange) from optimal direction (dashed grey line) to the minima (blue dot) as countries get more elongated.

9.3 Subgradient method

A vector $g_x \in R^n$ is subgradient of $f: R^n \rightarrow R$ at $x \in \Omega$ if for all $x_0 \in \Omega$

$$f(x_0) \geq f(x) + g_x^T (x_0 - x)$$

Any vector that satisfies the above condition is called a subgradient of f at x. If function f is differentiable at x, then its gradient is the only subgradient at that point. On another hand, if f is a non-differentiable function, there exists a set of subgradients at point x. The set of all subgradients is known as the subdifferential of f at x. A point x* is a minimum of a nondifferentiable function f if one of subgradients of f at x* is equal to 0 i.e. if the subdifferential contains zero.

For example, the function $f(x) = |x|$ at the point $x = 2: f(x) = x$ is differentiable and gradient $\nabla f = 1$. However, at the point $x = 0$ the function is non-differentiable (it is not possible to find a parabola that always lies above the function). Nevertheless, any $g_x \in [-1,1]$ satisfies inequation $|x_0| \geq g_x \cdot x_0$ for all $x_0 \in R$. Therefore, the subdifferential of f at $x=0$ are all values between -1 and 1, i.e., it is possible to draw many lines with slope in this range that will stay below the function.

From an optimization point of view, a non-smooth function is one that is non-differentiable everywhere. Consequently, if the function is not-smooth, i.e., non-differentiable, we cannot rely on gradients. Instead, we replace the gradient with a subgradient to perform iterative optimization:

$$w_{n+1} = w_n - \eta g_n.$$

The subgradient is not a descent method in general since the update is not necessarily performed in the descent direction, i.e., in contrast to GD, it doesn't guarantee that the objective function decreases at every iteration. Instead, the subgradient uses different directions and it ensures convergence by tracking the best iteration found, i.e.,

$$w_{best} = argmin_{n\in\{0,\dots,k\}} f(w_n)$$

9.4 Stochastic gradient descent

Gradient descent is computationally expensive and demands a lot of effort (especially since only small steps are made in that direction). The stochastic gradient descent (SGD) can address those limitations by sampling the gradient. Rather than computing all gradients at each interaction, the SGD randomly pick a single instance (or a small batch of instances) and update the weights based on the gradient of the loss for that instance only

$$w_{n+1} = w_n - \eta \nabla f(w_n; x_i).$$

The instance is chosen uniformly at random, allowing SGD to provide an unbiased estimation of the gradient. Although the computation is much faster, it can introduce noise in the estimation of gradients and make a step in the wrong direction. However, if the learning rate is sufficiently small the errors tend to average out. Consequently, the gradient computed from one instance's loss can be seen as an approximation of the true gradient.

The SGD is not limited to a single instance. Instead, a a small subset of training data can be sampled randomly $I_k \subset \{1, \dots, n\}$. The parameter update then becomes:

$$w_{n+1} = w_n - \eta \frac{1}{m} \sum_{i\in I_k} \nabla f(w_n)$$

approach known as mini-batch SGD. After processing one mini-batch, another randomly chosen subset of samples is used for the next update. This process continues until all training samples have been used once, completing one epoch of training. The procedure is then repeated for multiple epochs until convergence.

For example, suppose that the training dataset contains 16000 data samples and the mini-batch size is set to 16. Each gradient will use 16 data points instead of 16000. That speeds up the gradient computation 1000 times. Although the resulting gradient estimate will be noisier, it provides a direction close enough to the true gradient according to the law of large numbers.

The mini-batch has two main advantages: it reduces the noise in gradient estimation (lowers the variance) and enables the advantage of fast matrix operations and parallelism.

In machine learning, parameters and hyperparameters refer to different types of model variables. Parameters are learned directly from the training data, and used to define the model's internal, adaptable state. They are not predefined but rather iteratively optimized during the training process, since learning them is the primary goal. For example, weights and biases in neural networks are parameters that are adjusted during model training: they encapsulate the model's knowledge and directly determine its prediction on unseen data.

On the other hand, hyperparameters are manually selected before training begins and used to control the learning process and the models overall behavior. They influence crucial aspects such as convergence speed, stability, and generalization ability. Examples of hyperparameters include the regularization strength in a regression model, the learning rate, the number of layers in a neural network, the number of trees in Random Forest. Selecting appropriate hyperparameters often involves systematic search strategies such as grid search.

The mini-batch size is one of the most important hyperparameters and it varies for different applications, architectures and available computing power. Large batches will result in less noisy estimation of gradient ensuring convergence in fewer epochs but it can lead to lower generalization and a higher risk of overfitting. On the contrary, small batches require less computation and perform more frequent weight updates. The batch size should be a power of 2 to fully exploit the potential of the GPU.

The SGD convergence behavior typically exhibits three stages. At higher levels, the SGD generally points in the same direction resulting in a positive gradient. In the stationary phase gradients are smaller but the noise level remains almost the same, causing updates to point in varying directions and sometimes yielding a negative inner product. gradients from different samples point in almost random directions, causing the parameters to bounce back and forth around the minima. As a result, convergence with a fixed learning rate is not guaranteed. To address it, a large learning rate is used early in the training to get close to the optimum, followed by a gradual decay in the learning rate to reduce the fluctuations (reducing the noise by reducing the step size). When SGD gets close to the optimum the learning rate needs to be small enough to average out noise and allows SGD to settle into a stable state. Batch size and learning rate are highly correlated: increasing the batch size reduces gradient variance, enabling the use of a larger learning rate. On the contrary, small batch size leads to noisy gradient estimation and a smaller learning rate needs to be used to prevent unstable updates. The small batches require less computation (since fewer samples are used), converse to flat minimum, and perform more weight update than large batches but parallelizing small-batch SGD is highly limited. The large batch enhances the computation parallelism and can speed up convergence. However, it typically leads optimizers to converge toward sharp minima, which can reduce generalization ability. Imagine a loss landscape as a 1D curve, a minimum is flat if the loss changes slowly in the wide region around the optimum (indicated by small eigenvalues of Hessian) (**Figure 58**). In contrast, a sharp minimum occurs where the loss increases rapidly around optimum (high eigenvalues of Hessian matrix). Models that converge to sharp minimum are more sensitive to small changes in the input data or model leading to the lower generalization ability.

Let f be a β-smooth convex function. For SGD with fixed step size $\eta = 1/\beta$, we have $E\left[f\left(x^k\right)\right] - f^* \leq \frac{\beta\left||x_0 - x^*|\right|^2}{2k} + \frac{\sigma^2}{2\beta}$

where $\frac{\left||x_0 - x^*|\right|^2}{2k}$ represents optimization term, and $\frac{\sigma^2}{2\beta}$ represents a variance floor that is constant and cannot be reduced by continuing iterations with the same fixed step.

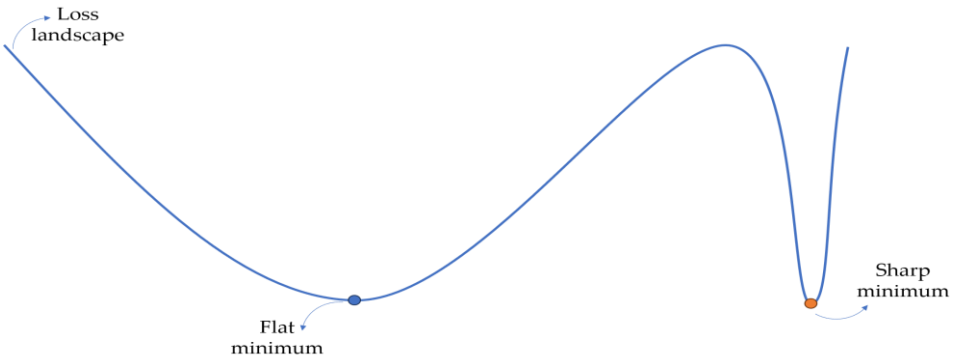


Figure 58 Visualization of flat and sharp minimum

The optimization term converges as $O(1/k)$. Using the diminishing rate $\eta = 1/\beta\sqrt{k}$ removes variance floor and lead to convergence rate of $O(1/\sqrt{k})$.

When f is α-strong convex and β-smooth, using the fixed step size $\eta < 1/\beta$ we obtain $E[f(x^k)] - f^* \leq O(e^{-\alpha k}) + O(\eta\sigma^2)$ where σ^2 is gradient variance. The first term indicates that SGD converges rapidly to a neighborhood of the optimum but then it oscillates around it due to variance in the gradient. The common practice is to use a fixed learning rate until progress stalls and then reduce it by some factor. If the diminishing learning rate $\eta = 1/\alpha k$ is used than $E[f(x^k)] - f^* \leq O(1/k)$, i.e., to get ϵ-suboptimal solution the $O(1/\epsilon)$ is needed.

The SGD reduces computational cost per iterations which is important especially for large training sets, but also it will make much slower convergence per iteration (increases the number of iterations) compared with the GD. The comparison of computational cost between GD and SGD for a strong convex function is given in **Table 9**.

Table 9 Comparison of complexity of GD and SGD for strong convex function. The d represents the number of feature per sample

	Number of iterations	**Cost per iteration**	**Total cost**
GD	$O(log(1/\epsilon))$	$O(n \cdot d)$	$O(n \cdot d \cdot log(1/\epsilon))$
SGD	$O(1/\epsilon)$	$O(d)$ or $O(b \cdot d)$ for mini batch	$O(d/\epsilon^2)$ or $O(b \cdot d/\epsilon^2)$

The SGD is more sensitive to the ϵ-accuracy, while the GD is more sensitive to dataset size. For large n, the SGD results in lower computational cost.

9.5 Accelerated SGD

The accelerated versions of SGD can significantly improve the convergence speed and the quality of the obtained local minimum. One prominent example is the SDG with momentum. Momentum addresses two issues with the SGD: convergence speed and local minima. On a loss surface with narrow ravines, the gradient can oscillate across the steep directions, slowing progress towards the minimum. So, if the learning rate is too large relative to the curvature in one direction, updates may oscillate, effectively reducing the progress. However, in directions with shallow curvature, the gradients are more consistent in sign, leading to smoother and faster convergence.

In the momentum method, the gradient oscillations are addressed by introducing the velocity vector that averages the past gradient updates. So instead of looking just at the current value of the gradient, the velocity vector memorizes the direction where the gradient has been consistent (low curvature direction) over interactions, i.e.

$$w_{t+1} = w_t + v_{t+1}$$

$$v_{t+1} = \mu v_t - \eta \nabla w_t f$$

where t is the number of interactions and μ is the momentum coefficient that controls how much of the velocity (previous gradient) is carried into the current update. In this way, the momentum tries to guide the gradient path toward the flat direction. The μ can have values between 0 and 1, the larger the coefficient, the greater the influence of the previous update, which means the "ball" keeps moving in the same direction longer. If $\mu = 0$ that the plain SGD is recovered.

The analogy of a rolling ball can be used to visualize how SGD with momentum behaves during optimization. Imagine a ball rolling down the hilly surface that represents the loose function. This surface has both local and global minima (**Figure 59**). Therefore, moving to the left or right around local minimum results in an increase in the loss. The SGD optimization can be visualized as if we release the ball at a certain point and let it roll on the loss

curve. In that case, the ball velocity is based only on the current acceleration during that step (i.e., current slope value). When the ball reaches a local minimum, the slope becomes flatter, the ball loses its velocity, and can be trapped in the local minimum, unable to escape the dip. If momentum is implemented, then the ball is not completely stopped at each point. Instead, it continues to roll along the loss curve, building some speed. In that case, the momentum of the ball will be equal to the current acceleration plus the current velocity resulting from past acceleration. Thus, even when the slope becomes flatter (like in local minima), the ball reduces velocity, but momentum enables escape from local minima and continues to move toward the global minimum.

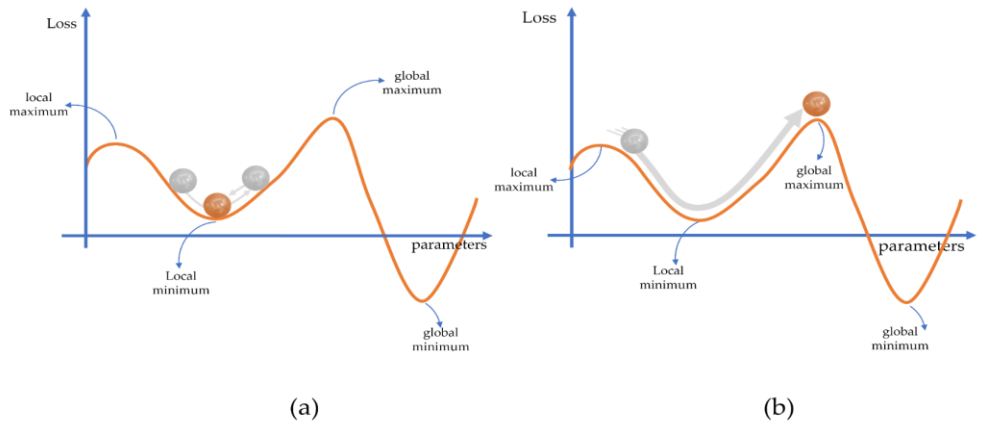


Figure 59 (a) SDG, (b) SGD with momentum

So, the intuition behind momentum is that if we repeatedly move in the same direction, then we will become more confident and start to take bigger steps in that direction. However, on ill-conditioned surfaces (Figure 60 (a), (b)), momentum can build too much velocity, leading to overshooting and oscillating around minima.

Nesterov's momentum, also known as Nesterov Accelerated Gradient Descent (**Figure 60** (c)), is an improved version of the traditional momentum introduced to reduce those oscillations by including a look-ahead feature into the update rule. The core intuition is to first anticipate where the accumulated momentum is leading—by taking a "peek ahead"—then compute the gradient at that estimated future position and adjust the update accordingly.

Neserov momentum calculates the gradient at a position slightly ahead in the direction of the accumulated moment. This enables the optimizer to correct its

direction quicker than momentum-based methods and reduce the oscillations, i.e.,

$$w_{t+1} = w_t - \eta v_{t+1}$$

where $v_{t+1} = \mu v_t - \eta \nabla(w_t + \mu v_t)$, μ is momentum coefficient that controls the decay of the velocity vector v, and has value $0 < \mu < 1$ (typically 0.9).

Let f be convex and β-smooth function in such a way that $f(x^k) - f(x^*) \leq \frac{2\beta\left||x^0 - x^*|\right|_2^2}{k^2}$, which means we reach ϵ-suboptimal point after $O\left(\frac{1}{\sqrt{\epsilon}}\right)$ iterations. The $O\left(\frac{1}{\sqrt{\epsilon}}\right)$ represents a significant improvement compared with GD. For an α-strong convex and β-smooth function, the Nesterov momentum needs $O\left(\sqrt{\kappa} \cdot log(1/\epsilon)\right)$ iterations to achieve ϵ-suboptimality. Nesterov is exponentially fast, reaches the optimal rate among first-order methods and can be applied to any convex problem. Nesterov momentum provides faster convergence (**Figure 60** (c) and (d)).

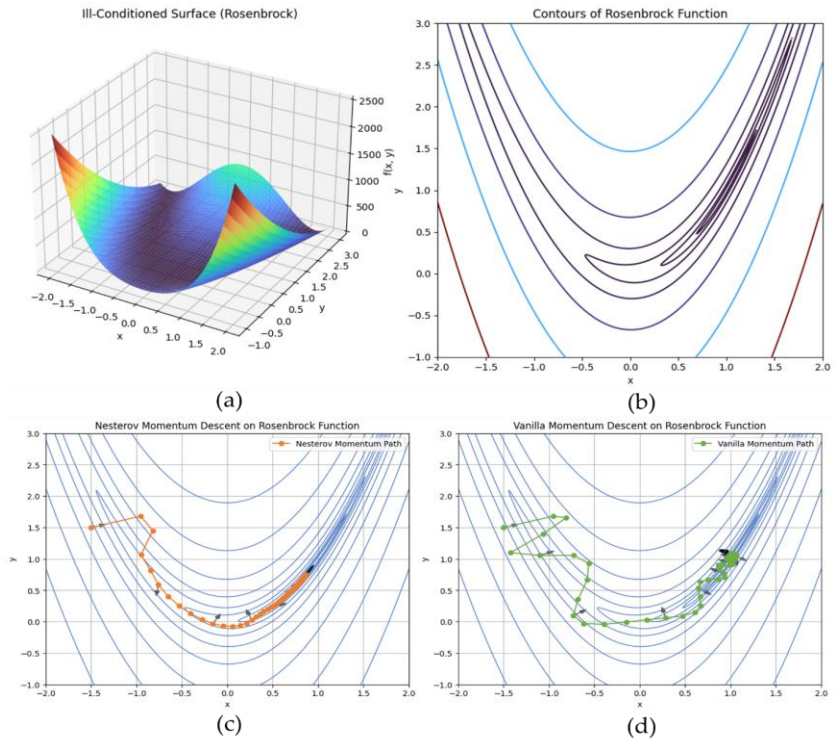


Figure 60 (a) Rosenbrock function - nonconvex function with global minima located in the narrow, curved valley (b) contour lines of Rosenbrock function (c) performance of Nesterov momentum and (d) performance of momentum (vanilla) on Rosenbrock curve

9.6 Newton's method

The Newton method is a second-order method used for smooth convex optimization. It minimizes a convex, twice differentiable function by iteratively minimizing its quadratic approximation since it uses more information about the function via the Hessian. Therefore, Newton's methods move in the direction of the negative Hessian inverse to the gradient. By multiplying with the inverse Hessian, the optimizer takes larger steps along directions of low curvature and smaller steps along directions of high curvature. This means the Newton method uses second-order information to "spherize" the contours in each step and moves in a direction orthogonal to the transformed contours, resulting in a more direct path: it naturally takes bigger steps along flat directions and smaller steps along steep ones. Moreover, the Newton method is independent of linear scaling of the input, providing automatic adjustments for axis stretching. Set the initial point to an arbitrary value and update it by

$$w_{t+1} = w_t - \nabla^2 f(w_t)^{-1} \nabla f(w_t)$$

until a stopping criterion is met.

The vector $-\nabla^2 f(w_t)^{-1} \nabla f(w_t)$ is called the Newton step. Notice that the above equation does not contain any learning rate hyperparameter, which theoretically represents a great advantage compared to first-order methods. This is the pure Newton method (equivalent to $\eta = 1$). However, it does not always guarantee a descent direction. The dumped Newton method is used to ensure a descent direction by scaling the Newton's step, with learning rate typically determined through backtracking line search.

Let f be a twice differentiable, α-strong convex and β-smooth function. To reach an ϵ-suboptimal point we need at most $O\left(log\ log\ \frac{1}{\epsilon}\right)$ iterations. Notice that the rate of convergence is quadratic, and therefore much faster, compared with the linear convergence of GD under the same assumptions. A visual comparison of the Newton and quasi-Newton methods is shown at Figure 8.

However, there are two main challenges with using Newton's method:

- It is sensitive to initial conditions, especially if the loss function is non-convex. Unlike the GD that ensures a descent direction, Newton fits a

paraboloid (second-order approximation) at the local curvature and proceeds to move to the stationary point of that paraboloid. Depending on the local behavior of our initial point, it can end up in a maximum or a saddle point instead of a minimum. Because of this, the Newton method has a local convergence guarantee that holds when the initial point x_0 is sufficiently close to the optimum. This is because the accuracy of Newton depends on the accuracy of the second-order approximation and, since f is twice differentiable, the quadratic model of f will be accurate if the initial point is close to the optimum. If x_0 is far from the optimum, the method may diverge. Achieving a global convergence guarantee is much harder to obtain but it is possible when the function is both α-strongly convex and β-smooth (**Figure 61**).

- Although Newton's method can significantly accelerate the optimization of moderate size problems where the quadratic approximation is accurate, often computing and inverting the Hessian can be computationally expensive. In contrast to computing the gradient that scales as $O(n)$, computing of the Hessian requires $O(n(n+1))/2)$ operations, since it is a symmetrical matrix, and inverting it scales as $O(n^3)$.

For example, in 100 dimensions, we have to calculate the 100 values for the gradient and 5050 values for the Hessian at each step, and additional 100^3 operations for inverting it. It is evident that the higher convergence rate will quickly be overweighted by large computational costs, especially when n is large.

To keep the efficiency of the second-order optimization and avoid computational cost, the Quasi-Newton method is used. The main idea is to avoid a full computation of the Hessian matrix across iterations by just approximating it with a positively defined matrix B, which is updated in each step by using information from previous steps. As a result, the computation costs have been significantly reduced.

Different quasi-Newton methods, such as Symmetric Rank-One (SR1), Davidon-Fletcher-Powell (DFP), or Broyden-Fletcher-Goldfarb-Shanno (BFGS), compute B_{k+1} matrix by imposing additional constraints. However,

all these methods need to satisfy the quasi-Newton condition (also known as the secant equation) given as follows:

$$B_{k+1}s_k = y_k \text{ or equivalent for inverse form } H_{k+1}y_k = s_k$$

where $s_k = w_{k+1} - w_k$ is the step taken in the parametric space and

$y_k = \nabla f(w_{k+1}) - \nabla f(w_k)$ represents the change of the gradient after taking the step s_k.

In the Newton method, we use $s_k = -\nabla^2 f(w_k)^{-1}\nabla f(w_k)$ to compute the update. In quasi quasi-Newton method, we just know the approximation of Newton B_k and it is necessary to ensure that it behaves as Hessian, i.e. $y_k \approx \left[-\nabla^2 f(w_k)^{-1}\right]s_k$. Therefore, the B_k must predict that the moving in s_k direction causes the gradient to change by y_k.

One of the most popular methods is the BFGS. In addition to ensuring the symmetry and positive-definiteness of B, the update of B_{k+1} is obtained by minimizing the matrix norm of the difference between B_{k+1} and B_k , i.e.,

$$min_{B_{k+1}}\left|\left|B_{k+1} - B_k\right|\right|_W$$

subject to the symmetry condition $B_{k+1}^T = B_{k+1}$ and the quasi-Newton condition $B_{k+1}s_k = y_k$

where $\left|\left|\cdot\right|\right|_W$ denotes the weighted Frobenius norm.

The solution for B_{k+1} is given by

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$$

If f is twice continuously differentiable, the Hessian is positively defined and β-smooth function around the optimum, then BFGS has a superlinear convergence rate, which lies between $O(log(1/\epsilon))$ and $O(log\ log(1/\epsilon))$

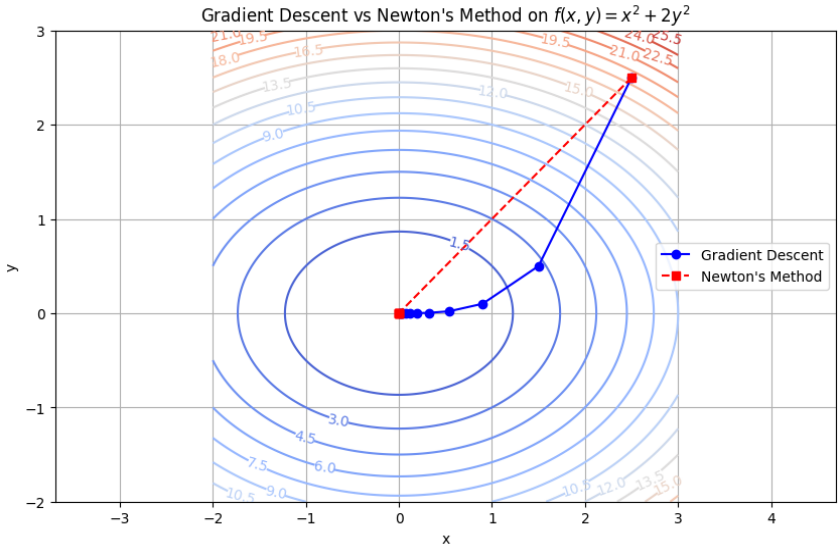


Figure 61 Visual comparison of GD and Newton methods

9.7 Adaptive learning rate

In the optimization model, the scale invariance is very important. For example, consider a problem with two variables: elevation (measured in meters) and NDVI (a pure number ranging from -1 to 1). If we change the elevation unit to centimeters, the elevation variable becomes 100 times larger, making the objective function 100 times more sensitive to changes in elevation. Moreover, the curvature in that direction is scaled 100^2, leading to poor conditioning (the condition number κ becomes too large) and an elongation of contour lines (**Figure 62**). Consequently, the cost function becomes very sensitive to some directions in the parameter space and almost insensitive to others. So, the gradient updates with respect to NDVI become inefficiently small, while the gradient update for elevation may explode, producing a net effect of slowing the convergence (moving very slowly in the flat direction and oscillating in the steep direction) if the same learning rate is applied.

It is desirable for models to be invariant to this type of change. Although this is guaranteed by the Hessian preconditioning (due to the affine invariance property), it is not for gradient descent. This can be addressed by adjusting

the learning rate dynamically for each parameter at each iteration based on observed statistics of the historical gradient.

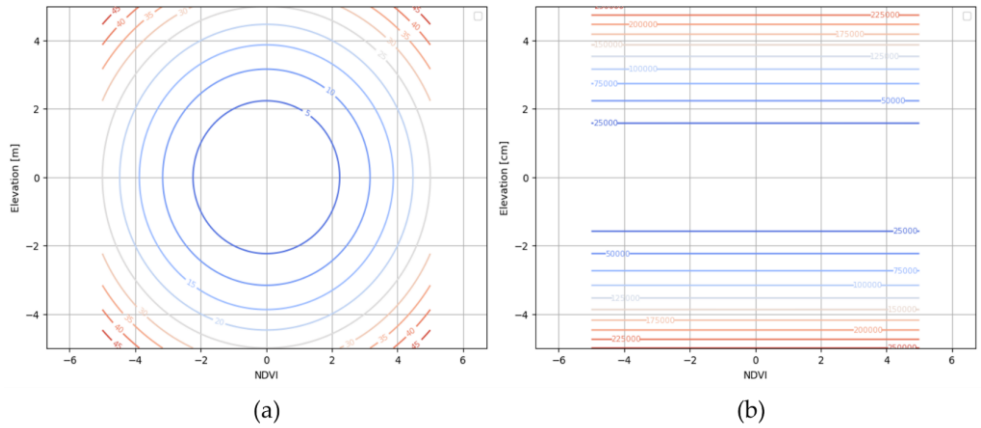


Figure 62 (a) Elevation in meters - couture lines are circle, loss surface is well-conditioned, the gradient descent progress smoothly, (b) Elevation in centimeters - the contours are highly elongated, loss surface is ill-conditioned

The **Adaptive Gradient Algorithm (AdaGrad)** adjusts the learning rate for each parameter individually by scaling it inversely proportional to the square root of the cumulative sum of past squared gradients (i.e., the sum of all previously observed squared values for that component).

Algorithm AdaGrad

Input: learning rate η, initial parameters w
Initialize $r \leftarrow 0$
loop
  sample a stochastic gradient $g \leftarrow \nabla f_{i_t}(w)$
  accumulate the second momentum estimate $r_j \leftarrow r_j + g_j^2$
  update model $w_j \rightarrow w_j - \frac{\eta}{\sqrt{r_j}} g_j$
end loop

This means that parameters that are not frequently updated will have a large learning rate, while parameters that are frequently updated will have a smaller learning rate.

The AdaGrad has great success when the gradient is sparse in nature (i.e. most gradient components are equal to 0) because it accumulates squared gradients. However, it may not work well in the nonconvex setting since the

learning rate depends on the whole history. As a result, the step size can be very small in certain directions, slowing down convergence.

Root Mean Square Propagation (RMSProp) represents the modification of the AdaGrad algorithm for a nonconvex setting. Unlike AdaGrad, which uses a sum of square gradients, RMSProp uses an exponential moving average of the squared gradients. This ensures that the effective step size generally does not go to zero, allowing the optimizer to continue making progress.

Algorithm RMSProp

Input: learning rate η, decay rate ρ, initial parameters w
Initialize $r \leftarrow 0$
loop
sample a stochastic gradient $g \leftarrow \nabla f_{i_t}(w)$
accumulate the second momentum estimate $r_j \leftarrow \rho r_j + (1-\rho) g_j^2$
update model $w_j \rightarrow w_j - \frac{\eta}{\sqrt{r_j}} g_j$
end loop

It has proven to be an effective and widely used optimization algorithm in DL.

Adaptive momentum (Adam): represents the variation of RMSProp that uses a moving average of momentum with exponential weighting and correction for bias to estimate the first-order (the mean) and second-order moments (the unscented variance) of the gradient. Recommended initial settings for ML are: $\eta = 0.001$, $\rho_1 = 0.9$, $\rho_2 = 0.999$ and $\epsilon = 10^{-8}$. Under the assumption that gradient magnitude is bounded and distance between any parameters generated by Adam is bounded, Adam achieves a $O(1/\sqrt{k})$ convergence rate. Adam has several important properties: the updates are invariant to rescaling of the gradient, it performs well with sparse gradients and non-stationary objectives, it is straightforward to implement, efficient to compute, and requires little memory. Compared with SGD with momentum, Adam shows marginal improvements.

Algorithm Adam

Input: learning rate η, exponential decay rate ρ_1, $\rho_2 \in [0,1)$, initial parameters w
Initialize $s \leftarrow 0$ (first moment vector) $r \leftarrow 0$ (second moment vector)
Initialize timestep $t \leftarrow 0$
loop

$$t \leftarrow t + 1$$

sample a stochastic gradient at time t $g_t \leftarrow \nabla f_{i_t}(w)$
accumulate biased first momentum estimation $s_t \leftarrow \rho_1 s_t + (1 - \rho_1) g_t$
accumulate second momentum estimate $r_t \leftarrow \rho_2 r_t + (1 - \rho_2) g_t^2$
correct first momentum bias $\hat{s} \leftarrow \frac{s_t}{1-\rho_1^t}$
correct second momentum bias $\hat{r} \leftarrow \frac{r_t}{1-\rho_2^t}$
update parameters $w_t \rightarrow w_{t-1} - \frac{\eta \cdot \hat{s}_t}{\sqrt{\hat{r}_t + \epsilon}}$

end loop

9.8 Nonconvex optimization

If f is nonconvex, it can have many local minima, saddle points, very flat regions, or widely varying curvature, making optimization hard. Gradient descent stops naturally when $\nabla f(x) = 0$. When f is nonconvex, this happens not only when x is a minimum but also when it is a maximum or a saddle point. The saddle points are stationary points ($\nabla f(x) = 0$) but Hessian is undefined. In high dimensional spaces, saddle points are more frequent than local minima. Additionally, there can be some flat regions, where the gradient is very small or zero. The choice of the initial point and the step size determines the point the algorithm converges to. Therefore, to minimize the loss, the gradient-based methods need to efficiently avoid maxima, flat regions, and saddle points. Many convex optimization methods can be applied to nonconvex optimization problems. However, theoretical guarantees for global convergence are limited or non-existent.

9.9 Loss function - review

The loss function quantifies the prediction error that represents the difference between the model prediction and the actual ground truth data. The primary

objective in ML is to optimize the model parameters by minimizing the total loss. The loss function should enable ML models to effectively learn from training data, even in challenging situations such as class-imbalanced or noisy data.

The choice of the loss function depends on the type of algorithm being optimized, the nature of the task, and the available dataset. Choosing a loss function that is closely aligned with the task objective is crucial, because the model will exploit the easiest way within the data and architecture to minimize the loss. As a result, a model may achieve low loss values yet still fail to solve the problem effectively in practical terms. For example, suppose we want an order to be delivered both quickly and accurately. We could train an AI model to select the best delivery services using a loss function that minimizes delivery time. However, the model will exploit the easiest way to reduce this loss, potentially favoring companies that deliver packages quickly but frequently make mistakes. This behavior would not align with our actual goal. Therefore, a loss function that fully reflects the objective should account for both on-time delivery and accuracy.

As already mention, there are several desirable properties of a loss function that should be taken into consideration during selection process:

- Differentiability - the loss function must have a derivative for each point within the domain and does not contain any breaks or gaps,
- Convexity - the local minimum is the global minimum,
- Robustness - it should be able to handle outliers and not be affected by a small number of extreme values,
- Smoothness - the function doesn't have sharp transitions, ensuring stable and efficient training, and
- Monotonicity - if loss function values decrease as the predicted output approaches the true output. It ensures that the optimization process is moving toward the correct solution.

These properties directly affect the rate of convergence, which measures by how fast the algorithm reaches a predefined threshold. A lower bound enables estimation of the best possible upper band for the class of problem under consideration, while an upper bound on the convergence rate allows the estimation of the number of steps that are needed to reach a predefined

threshold. If the loss function is only Lipschitz continuous, gradient descent takes cautious steps, but if the loss function is also smooth, the gradient will converge faster compared with only L-Lipschitz. Additionally, the assumption of strong convexity and smoothness leads to exponential fast convergence, also known as linear convergence. Influence of loss function assumptions on convergence rate and optimal step size is shown at **Table 10**.

Table 10 Influence of loss function assumptions on convergence rate and optimal step size

Function assumption	Convergence rate	Optimal step size
convex + L-Lipschitz	$O(1/\sqrt{k})$	$\frac{\left\|\|x_0 - x^*\|\right\|_2}{L\sqrt{k}}$
convex + β-Smooth	$O(1/k)$	$1/\beta$
α-strong convex + L-Lipschitz	$O(1/k)$	$\frac{2}{\alpha(k+1)}$
α-strong convex +β-Smooth	$O(e^{-k})$	$\frac{2}{\alpha+\beta}$

The loss functions are categorized based on the type of the task, such as regression, classification, or object detection. In addition to that, the performance metrics are used to evaluate how well the model generalizes to new data and how accurate the prediction is. The most commonly used loss function and performance metrics, depending on the task's characteristics, is presented in the **Table 11**.

Table 11 Review of the most commonly used loss functions and performance metrics for different tasks

Task	Loss function	Performance metrics
Regression	Mean Squared Error (MSE)	Mean Squared Error (MSE)
	Mean Absolute Error (MAE)	Mean Absolute Error (MAE)

	Huber Loss	Root Mean Squared Error
	Quantile loss	Mean Absolute Percentage Error
		R^2
		Adjusted R^2
Binary classification	Binary Cross-Entropy	Accuracy
	Weighted Cross-Entropy	Precision
	Focal Loose	Recall
	Hinge Loss	F1-score
		Kappa coefficient
Multi-class classification	Categorical Cross-Entropy	Accuracy
	Weighted Cross-Entropy	Precision
	Focal Loss	Recall
		F1- score
		Kappa coefficient
		Intersection over Union (IoU)
Semantic segmentation	Pixel-wise Cross-Entropy	IoU
	Focal Loss	F1- score
	Dice Loss	
Object detection	Focal Loss	Average precision
	IoU loss	Average recall
	GIoU Loss	

9.9.1 Loss function in regression

The regression model aims to predict the continuous variable, so popular loss functions are error-based, i.e., they measure the residuals to optimize model parameters.

Mean Absolute Error (**MAE,** also known as ***L1*** **loss**) (**Figure 63** (a)) measures the average of the absolute difference between predicted and true value, i.e.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2$$

where $\hat{y}_i = f_w(x_i)$. Since MAE is an absolute value, it is always positive, and errors follow a linear behavior, so it is less sensitive to outliers. It is not differentiable at 0 i.e., when $y_i = \hat{y}_i$. Due to that, the optimization of MAE can be done by using the subgradient

Mean Squared Error (**MSE,** also known as ***L2*** **loss**) (**Figure 63** (b)) represents the average of the squared difference between predicted and true value, i.e.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

It is derived under the assumption that residuals have a Gaussian distribution. Due to the squared error, it is always positive, but also sensitive to outliers. In neural networks, large errors have a larger impact on the computed gradient, leading to suboptimal weight updates. The MSE is differentiable for both parameters and prediction, enabling optimization by using gradient-based methods. However, it is scale-dependent.

Root Mean Squared Error (RMSE) is defined as the square root of MSE. For a dataset with n samples, predictions $\hat{y}_i$ and true values y_i , RMSE is given as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Due to squaring, RMSE is always positive and penalizes large errors heavily. It is sensitive to outliers, since a few large errors can significantly increase its values. RMSE has the same unit and scale as the target values, making it easier to interpret. However, due to scale dependency, comparison between datasets

with very different scales is difficult. To address this limitation, the **Normalized RMSE** can be used (see Section 9.17.1.).

Huber loss (**Figure 63** (c)) combines the advantages of MAE and MSE, i.e., it is less sensitive to outliers than MSE but smoother than MAE. It is defined as

$$L_{Huber}(y_i, \hat{y}_i) = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2, & if |y_i - \hat{y}_i| \leq \delta \\ \delta\left(|y_i - \hat{y}_i| - \frac{1}{2}\delta\right), & otherwise \end{cases}$$

where δ is a user-defined threshold, which is critical, and it can be adjusted dynamically during the training process. If $\delta \to 0$ the function behaves more like MAE, while for large δ it behaves more like MSE. The method is robust to outliers because it applies a linear penalty for errors larger than the threshold, while a quadratic one for small errors. The function is differentiable everywhere except in $|y_i - \hat{y}_i| = \delta$ so subgradients can be implemented.

Quantile loss (Figure 10 (d)) is frequently used to estimate the conditional quantiles in regression. The main idea is that minimization of asymmetrical weighted absolute residuals (asymmetrical error penalties, i.e., giving different weights to positive and negative residuals) will lead to quantiles. If the model underestimates ($y \geq \hat{y}$), the residuals are weighted by τ while for the overestimations ($y < \hat{y}$) the residuals are weighted by $(1 - \tau)$ i.e.

$$L_\tau(y, \hat{y}_i) = \begin{cases} \tau \cdot |y - \hat{y}|, & if\ y \geq \hat{y} \\ (1 - \tau) \cdot |y - \hat{y}|, & if\ y < \hat{y} \end{cases}$$

where τ is quantile level ($0 < \tau < 1$). Based on the previous equation, it can be concluded that for lower quantiles ($\tau = 0.25$) the overestimations are penalised more. Consequently, for higher quintiles, the underestimations are penalised more. If $\tau = 0.5$, the quantile loss reduces to the mean absolute error (MAE), corresponding to the solution that minimizes the conditional median of y. The function is not differentiable at residual 0. The overview of the benefits and limitations of the most frequently used loss function in regression is presented in **Table 12**.

Table 12 The overview of the benefits and limitations of the most frequently used loss functions in regression

Loss function	**Best use case**	**Strength**	**Limitations**
MSE	No outliers, Gaussian noise	Efficient, differentiable	Highly sensitive to outliers, scale-dependent
RMSE	Data with moderate outliers	Same unit and scale as the target, penalizes large error	Sensitive to outliers, does not distinguish error types
MAE	Outliers, skewed distribution	Robust, interpretable	Non-differentiable at 0, not scale invariant
Huber	Mixed noise, gradient descent	Convex and differentiable	Need tuning parameter δ, not scale-dependent
Quantile	Predicting quantiles, intervals, and risk-sensitive applications	Handles asymmetry, robust to outliers	Need to choose quartile values, Non-differentiable at 0, harder for interpretation

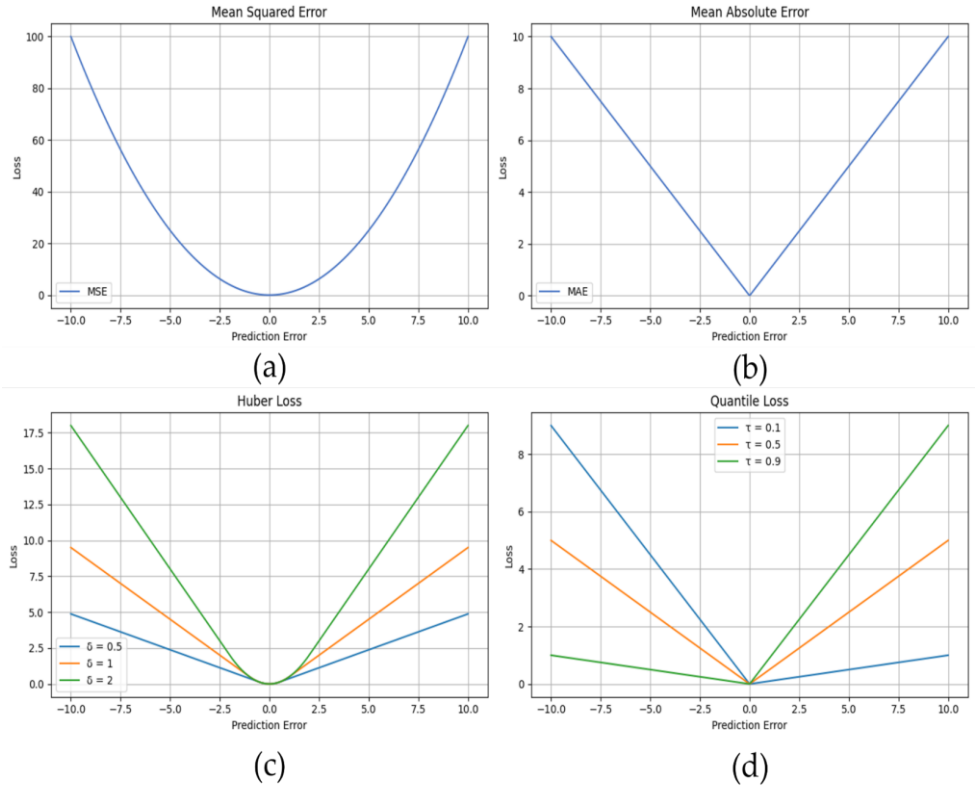


Figure 63 Illustration of the most commonly used loss functions in regression (a) MAE, (b) MSE, (c) Huber loss, and (d) Quantile loss

9.9.2 Loss function in classification

In classification tasks, models map input features to the labels that correspond to dataset classes. Based on the number of classes, the classification task can be categorized into: binary classification (data classified into two classes) and multiclass classification (data classified into K classes). For classification tasks, the probabilistic loss functions are commonly used. Let q be the probability distribution of the dataset, and $p(y|x; w)$ is the distribution of the model that predicts outputs. Probabilistic loss functions measure how the prediction probability distribution matches the true distribution. Usually, models trained with this type of function provide a measure of how likely a sample is labeled with one class compared to another, providing margin-based information.

Most neural networks are trained by using maximum likelihood estimation (MLE). Formally, given a dataset dataset *D*, we are maximizing the likelihood of the observed data:

$$p(D \mid w) = \prod_{i=1}^{n} f_w(x_i)^{y_i} \cdot \left(1 - f_w(x_i)\right)^{1-y_i}$$

or equivalently, to maximize the log-likelihood:

$$log(p(D \mid w)) = \sum_{i=1}^{n} \left(y_i log f_w(x_i) + \ (1 - y_i) log\left(1 - f_w(x_i)\right)\right)$$

The loss function is obtained then by taking the negative of the log-likelihood

$$L = -\sum_{i=1}^{n} \left(y_i log f_w(x_i) + \ (1 - y_i) log\left(1 - f_w(x_i)\right)\right)$$

This is also known as cross-entropy loss. One of the main advantages of using cross-entropy is that we are training models that best explain the dataset under a known or assumed probabilistic model. That means that the loss function does not need to be specifically designed for each model. Rather, specifying a model $p(y \mid x)$ automatically a cost function $logp(y \mid x)$is determined.

Binary cross-entropy (**BCE** also known as **Log loss**) (**Figure 1** (a)) is frequently used for binary classification. Since binary classification involves two classes (e.g.: cat vs dog, spam vs not spam), the maximum likelihood approach is based on the Bernoulli distribution. The Bernoulli distribution is a discrete distribution with two possible outcomes

$$P(y = 1 \mid x) = p \text{ and } P(y = 0 \mid x) = 1 - p$$

If we have a dataset where each label is $y_i \in \{0, 1\}$, then the BCE loss for a single instance is given by

$$L(y_i, \hat{p}_i) = -[y_i log(\hat{p}_i) + \ (1 - y_i) log(1 - \hat{p}_i)]$$

where $\hat{p}_i$ is the model's predicted probability that $y_i = 1$. The loss value is inversely proportional to the probability of the sample being correctly predicted, meaning that the greater the probability, the smaller the loss. The model prediction $\hat{p}_i$ must belong to the interval [0, 1] to be a valid probability.

If a simple linear output $w^T x + b$ is used, the predicted probability can be outside [0, 1], and the gradient of the model output would be 0. This is challenging for optimization because learning algorithms lack guidance on how to optimize the parameters. In order to ensure that the output is between 0 and 1, a sigmoid activation unit is applied to the linear output. A sigmoid output unit is given as

$$\hat{y} = \sigma(w^T x + b)$$

where σ is the logistic sigmoid function, defined as $\sigma(y) = \frac{1}{1+e^{-y}}$. This function maps any real-valued input to the range [0, 1], making it suitable for modeling probabilities in binary classification.

Hinge loss function (**Figure 64** (b)) represents an alternative to binary cross-entropy, which is frequently used in margin-based classifiers such as Support Vector Machine (SVM), but it can also be applied effectively to neural networks. This approach defines a soft-margin *m* (usually set to 1) around the decision boundary and enforces that the prediction score for the correct class is at least *m* units higher than for incorrect classes. The hinge loss function penalises both incorrect ($y_i \hat{p}_i \leq 0$) or not confident enough predictions (where the resulting argument is lower than the margin $y_i \hat{p}_i < m$). It is defined as

$$L_{Hinge}(y_i, \hat{p}_i) = max(0, m - y_i \hat{p}_i)$$

where y_i is the ground truth class label and $\hat{p}_i$ is the predicted output from the neural network, which is often mapped to a value in the range $\{-1, 1\}$ using a suitable activation function, such as hyperbolic tangent (tanh). If a prediction is both correct and confident enough (i.e., outside the margin) the loss is zero.

Also, it can be used for multi-class classification, by implementing a one-versus-all or one-versus-one approach. The Hinge loss is convex but non-differentiable at the hinge point $y_i \hat{p}_i = 1$. To address this limitation, a frequently used variant is the squared Hinge loss, defined as

$$L_{Hinge} = max(0, 1 - y_i p_i)^2.$$

This allows its usage in higher-order optimization algorithms and also penalizes predictions more strongly as they approach the margin.

Categorical Cross-Entropy Loss (**Figure 64** (c)) is often used for multi-class classification tasks where we need to classify an instance into one of *K* classes $y \in \{1, \ldots, K\}$. It is defined as

$$L(y_i, \hat{p}_i) = -\sum_{j=1}^{K} y_{i,j} log \hat{p}_{i,j}$$

where $\hat{p}_{i,j} = [\hat{p}_{i,1}, \ldots, \hat{p}_{i,K}]$ is the predicted probability distribution for sample *i*.

A one-hot encoded vector is a way of representing categorical data (like class labels) as a binary vector, where only one element is 1 (indicating the correct class) and all other elements are 0. $y_{i,j}$ is a one-hot encoder vector (1x*K*) representing the true class label for sample *i* (for example, if K=3 and the true class of sample *i* is class 2, then $y_i = [0,1,0]$).

In neural networks, to extend the cross-entropy loss to multi-class classification problems, the softmax activation function must be applied to the output layer. The softmax function transforms the raw output of a classifier (logit) into a probability distribution over *K* classes. Given a vector of logits $z = [z_1, \ldots, z_K] \in R^K$, the softmax activation function computes the probability that the input belongs to each class:

$$\hat{y}_i = P(y = i \mid z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

This ensures that each output value $\hat{y}_i$ belongs to the interval $[0, \ldots, 1]$ and that the vector sum is equal to 1, satisfying the properties of a valid probability distribution.

To maximize the likelihood, we need to maximize the predicted probability for the true class, which is equivalent to minimizing the negative log-probability. Given:

$$P(y = i \mid z) = log\ softmax(z)_i,$$

the categorical cross-entropy with softmax is defined as

$$L = -z_i + log \sum_{j}^{'} e^{z_j}.$$

Applying softmax is essential since the categorical cross-entropy measures the dissimilarity between the true distribution (typically one-hot encoder) and the predicted probability distribution $\hat{y}_i$.

The first term represents the score for the correct class, i.e., it penalizes the model if it assigns a low logit to the correct class (if $-z_i$ is small, the loss will be low).

The second term penalizes overconfidence in the wrong class, ensuring the model does not assign high probability to incorrect classes. Therefore, when the loss is small, the logit corresponding to the correct class z_i is high while the logits for all other classes z_j are low. Conversely, the loss becomes large when the correct class logit z_i is low, and one or more of the incorrect class logits z_j are high.

The categorical cross-entropy is continuous, differentiable, and convex with respect to the model outputs and this makes it suitable for gradient-based optimization methods.

Weighted cross-entropy (WCE) (**Figure 64** (d)) is a modification of the cross-entropy loss created to address the class imbalance by assigning higher weights (greater importance) to minor classes. In standard cross-entropy, all classes are treated equally. However, in tasks where classes are imbalanced (some classes appear more frequently than others), the gradient of the loss function will be dominated by the majority class, leading to poor model performance on minority classes.

For binary classification, the overall WCE loss is given by:

$$L_{WCE} = -\frac{1}{n}\sum_{i=1}^{n} w_1 y_i log(\hat{p}_i) + w_0(1-y_i)log(1-\hat{p}_i)$$

where w_1 and w_0 are weights for the positive and negative classes, respectively. The following expression can be easily expanded for multi-class classification, i.e.

$$L_{WCE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{K} w_j y_{i,K} log(\hat{p}_i)$$

where w_J represents the weight for class $j = \{1, \ldots, K\}$, $\hat{p}_i = [\hat{p}_{i,1}, \ldots, \hat{p}_{i,K}]$ is the predicted probability distribution over classes for sample i, and y_i is the one-hot encoded true label.

The weights can be determined based on the:

- Inverse class frequency, i.e. $w_K \approx 1/f_K$ where f_K is the relative frequency of class *K*. Consequently, less frequent classes will have higher weights.
- Weight normalization - helps stabilize gradients when some weights are very large.

Dice loss has been widely used in image segmentation tasks to handle class imbalance. It is based on the Dice coefficient maximization. The Dice coefficient measures overlap between the predicted segmentation and ground truth annotation. It is defined as:

$$L_{Dice} = 1 - Dice = 1 - \frac{2\sum_{i=1}^{n} p_i y_i + \epsilon}{\sum_{i=1}^{n} p_i + \sum_{i=1}^{n} y_i + \epsilon}$$

where p_i is the predicted probability for pixel *i*, and y_i is the corresponding ground truth while ϵ is a small constant added to ensure loss function stability by preventing division by zero. For multi-class segmentation, Dice loss can be computed per class and averaged, i.e.

$$L_{Dice} = 1 - \frac{1}{K}\sum_{j=1}^{K} \frac{2\sum_{i=1}^{n} p_{i,j} y_{i,j} + \epsilon}{\sum_{i=1}^{n} p_{i,j} + \sum_{i=1}^{n} y_{i,j} + \epsilon}$$

where $p_{i,j}$ and $y_{i,j}$ are the predicted probability and ground truth for class *j* at pixel *i*.

Dice loss is continuous and differentiable but not convex.

Focal loss (**Figure 64** (e)) is a variant of the cross-entropy loss designed to address class imbalance by down-weighting the contribution of easy examples and focusing more on hard misclassified examples. The focal loss is defined as

$$L_{focal} = -(1 - p_i)^{\gamma} log(p_i)$$

where p_i is the predicted probability for the true class, and $\gamma \geq 0$ is the focusing parameter that controls the strength of the modulation. When $\gamma = 0$,

the Focal Loss reduces to the standard binary cross-entropy loss. It is Lipschitz continuous and convex for the predicted probabilities, making it suitable for gradient-based optimization.

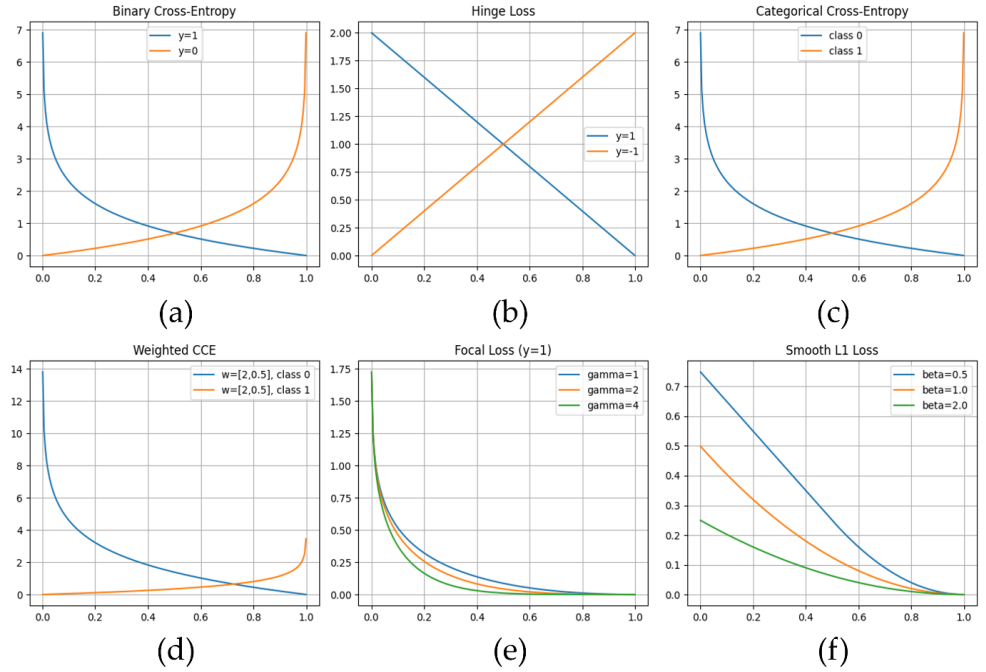


Figure 64 Loss function in classification (a) BCE, (b) Hinge, (c) CCE, (d) WCE, (e) Focal and (f) Smooth L1 loss

Object detection includes both the classification of the object type and the accurate prediction of the coordinates of a bounding box around the object of interest. Because of this dual objective, a composite loss function is typically used, containing both a classification component (measuring classification errors, such as CE) and a regression component (measuring prediction errors in the precise location and dimension of the boundary-box, such as Smooth L1 or IoU loss). Jointly, they analyze the misclassification and boundary-box inaccuracy, enabling accurate and robust detection across various data distributions.

Smooth L1 (Huber-like) (**Figure 64** (f)) is commonly used in bounding-box regression, combining the benefits of both *L1* (Mean Absolute Error) and *L2* (Mean Squared Error) functions.

For a single coordinate, it is defined as

$$L_{L_{1smoot}}(y_i, \hat{y}_i) = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2, & if|y_i, -\hat{y}_i| \leq \beta \\ |y_i, -\hat{y}_i| - \frac{1}{2}\beta, & otherwise \end{cases}$$

where β controls the transition between *L1*-like (linear) and *L2*-like (quadratic) regions. Typical value of the hyperparameter β is between 0.5 and 1. If $\beta = 0$ Smooth loss is equivalent to L1 loss. The smooth L1 is equivalent to $Huber\ loss/\beta$. Due to that, the difference can be defined as:

- for $\beta \to 0$ Smooth L1 converges to L1 loss, while Huber loss converges to a constant 0,
- for $\beta \to \infty$ Smooth L1 loss converges to a constant 0 loss, while Huber loss converges to MSE.

When the absolute error is small, the function behaves like L2 loss to ensure smooth optimization, while for large errors are penalized like L1 loss. It is less sensitive to outliers than MSE.

Intersection over Union loss (**IoU loss**) (**Figure 65** (a)) is based on the IoU metric. IoU (also known as Jaccard similarity index) measures the overlap between the predicted bounding box (B_p) and the ground truth box (B_t) i.e.

$$IoU = \frac{Area(B_p \cap B_t)}{Area(B_p \cup B_t)}$$

where $(B_p \cap B_t)$ is the intersection area of the predicted and ground-truth boxes and $(B_p \cup B_t)$ is their union.

If $IoU = 1$ the two boxes are perfectly overlapped. Compared with *L1/L2 losses*, IoU considers all shape properties, including location, size, and orientation at the same time, encodes relationships between all parameters, and it is scale invariant. On the other hand, it is non-differentiable if there is no overlap between the ground-truth and predicted bounding box (i.e. $IoU = 0$). In that case, IoU does not reflect if two boxes are close or far from each other, i.e., the model doesn't learn when boxes do not overlap, making optimization challenging. Additionally, if boxes partially overlap, the IoU changes slowly.

The IoU loss is given by

$$L_{IoU} = 1 - IoU$$

Since $0 < IoU < 1$, then the L_{IoU} is also bounded between 0 and 1.

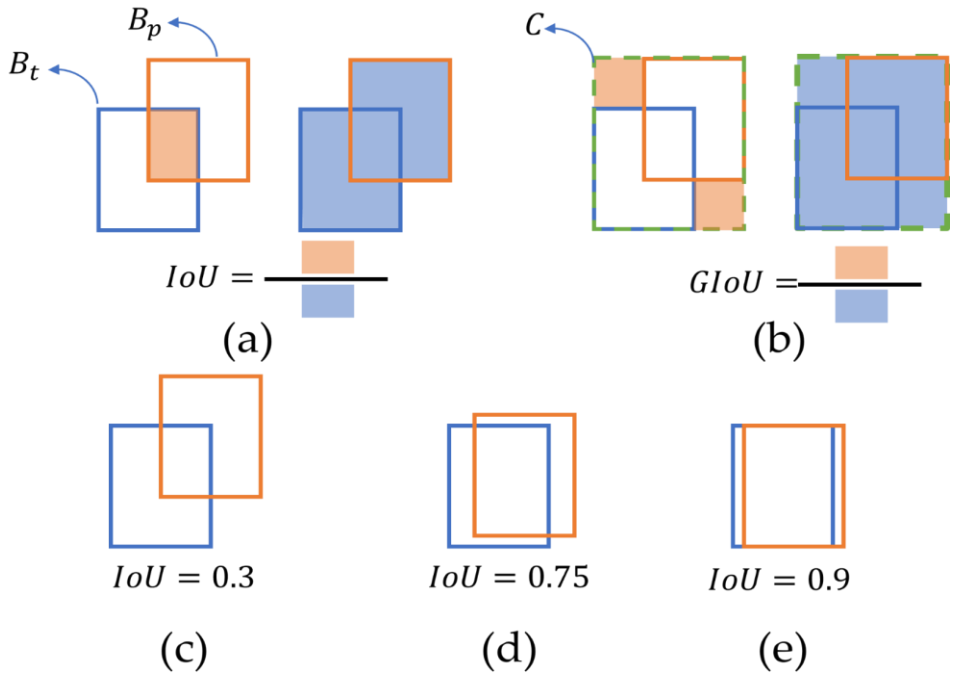


Figure 65 (a) IoU, (b) GIoU, (c) example of poor alignment, (d) example of good alignment, (e) example of excellent alignment

Generalized IoU (*GIoU*) (**Figure 65** (b)) is introduced to address the limitations of IoU. Let $B_p, B_t \subseteq S \in R$ be two arbitrary convex shapes. The $C \subseteq S \in R^n$ is the smallest convex shape that encloses both B_t and B_p and have the same shape type. The $GIoU$ is calculated as follows:

$$GIoU = IoU - \frac{Area\left(C/\left(B_p \cup B_t\right)\right)}{Area(C)}$$

The $GIoU$ loss is defined as:

$$L_{GIoU} = 1 - GIoU$$

The L_{GIoU} penalizes the boxes that do not overlap by emphasizing the empty area/volume outside the B_t and B_p but inside the smallest enclosing box. Due to that, it provides the gradient updates in all stages, improving the

convergence. It is non-negative, symmetrical, and scale invariant. The $GIoU$ always represents the lower bound for IoU.

The comparison of the advantages and disadvantages of the loss function commonly used in classification and object detection tasks is presented in **Table 13**.

Table 13 The overview of the benefits and limitations of the most frequently used loss functions in classification and object detection

Loss	**Usage**	**Advantages**	**Disadvantages**
BCE	Binary tasks, logistic regression	Probabilistic, strong penalty for confident mistraces	Sensitive to imbalance, overconfident if not regularized
Hinge	Margin-based classification	Zero loss beyond margins, simple	Not probabilistic, non-differentiable at the margin
CCE	Multi-class classification	Differentiable, strong penalty for misclassification	Sensitive to class imbalance, requires more memory (due to one-hot encoder)
WCE	Binary/Multi-class Imbalanced datasets	Address the imbalance by introducing weights, suitable for a cost-sensitive context	Weight tuning can be challenging. Large weights destabilise training. Fixed weight may not be optimal if class distribution changes (new data source)
Focal	Binary/Multi-class, Address severe imbalance	Imbalanced data, Hard samples in detection, minor classes are better characterized	Tuning the parameter γ can be challenging

Dice	Semantic segmentation, an imbalanced data set	Robust for imbalanced datasets	Overconfident prediction
Smooth L1	Boundary-box regression	Robust to outliers, less sensitive to the MSE, sensitive to small errors	Requires β tuning
IoU	Segmentation, object detection	Scale invariant,	Non-differentiable, sensitive to partial overlap, hard to optimize
GIoU	Object detection when partial overlap is frequent, box regression	Scale invariant, differentiable, and more stable gradient,	Don't handle centroid distance or aspect ratios, non-overlap sensitive, more computationally intensive

9.10 Activation functions

Deep neural networks are often used to approximate complex, non-linear relationships between input and output. However, the output of neurons is linear despite having several layers. Due to that, non-linearity needs to be introduced in the network.

Activation functions are predefined mathematical functions that introduce non-linearity to the output of individual neurons in each layer of the neural network before passing it to the next layer. Activation functions should satisfy several important properties. They must add non-linear curvature into the loss surface to improve convergence, they must be computationally cheap since they are calculated millions of times in deep neural networks, and avoid

saturation. Different types of activation functions, including sigmoid, Tanh, ReLu, etc., can be used.

Logistic sigmoid (**Figure 66** (a)) function and tanh activation functions have been widely used for shallow networks. The sigmoid function (defined in Section 9.10) converts the neuron output into the interval [0, 1]. The main drawback of the sigmoid function is that it is saturated for both low and high inputs, which causes the gradient with respect to parameters to approach zero, a mechanism known as the vanishing gradient. Hence, the update during training with SGD is very low, leading to slow convergence. Moreover, the sigmoid function is not zero-centred, and it is a bit computationally intensive.

The **hyperbolical tangent** (Tanh) (**Figure 66** (b)) activation function is defined as follows

$$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

It outputs zero-centred values between -1 and 1. It can be regarded as an extended sigmoid function, and it has the same drawbacks as the sigmoid.

ReLU [34] (**Figure 66** (c)), the rectified linear unit has become a state-of-the-art activation function in DL. It is a piecewise-linear function $ReLu(x) = max(x, 0)$ that outputs 0 for all negative inputs, while for positive inputs, it returns their value. The derivative in the active region is equal to 1, and therefore, the gradient is unscaled and consistent. Moreover, the second derivative is equal to 0 (except for $x=0$, where it is undefined), making the loss landscape simpler and easier for optimization. It addresses the limitations of sigmoid and Tanh functions as it does not saturate for positive values; it is more computationally effective, and enables faster convergence. However, its output for all negative values is 0, meaning that there is no gradient flow through those inactive neurons, leading to the problem of "dead ReLu". This problem is addressed by several variants of ReLu.

Leaky ReLu (LReLu) (**Figure 66** (d)) adds small fixed positive gradients for negative inputs to prevent saturation. It is given by $LReLu(x) = max(x, 0.01 \cdot x)$. One of the main drawbacks of LReLu is finding the right slope in a linear function for negative inputs, since different slopes can be suited for different problems and networks.

The **parametric ReLU** (PReLu) (**Figure 66** (e)) represents an extension of LReLu by making the slope for negative inputs as a learnable parameter, i.e. $PReLu(x) = max(x, \alpha \cdot x)$ where α is slope for inputs less than zero; however, it can quickly overfit.

The **Maxout** (**Figure 66** (f)) [35] can be used to further generalize both ReLu and LReLu. Instead of applying a fixed nonlinearity (such as ReLu, sigmoid, tanh), Maxout divide input vector *x* into k groups and outputs the maximum value within each group. This allows network to learn piecewise linear function, partitioning input space into several regions with local linear behavior. Mathematically it is given by

$$z_i = \max_{j \in \{1,\dots,k\}} \left(x^T w_{ij} + b_{ij}\right)$$

where k is the number of linear pieces, w_{ij} and b_{ij} are learnable parameters of the $j-th$ component of the $i-th$ neurone. This allows Maxout to approximate any approximate. If $k = 2$ the maxout is defined as $Maxout(x) = max\left(w_1^T x + b_1, w_2^T x + b_2\right)$ and both ReLu and LReLu are special case of Maxout. However, Maxout is parameterized differently, i.e., each maxout unit is parametrized by a *k*-weight vector. The intersection point and slopes on each side are learned rather than fixed, like in ReLu (intersection point in 0, slope on positive side 1, and slope on negative side 0). The maxout does not saturate or die, but it doubles the number of parameters and therefor increases computational and memory cost. Maxout pairs particularly well with dropout regularization.

Exponential linear units (ELU) (**Figure 66** (g)) [36] address the vanishing gradient effect seen in ReLu and LReLu. It is defined as

$$ELU(x) = \begin{cases} x, & if\ x > 0 \\ \alpha(e^x - 1), & if\ x \leq 0 \end{cases}$$

where α is a hyperparameter that controls the saturated values for negative inputs (i.e. controls the level of nonlinearity for those values). ELU uses an exponential function to smooth negative values and asymptotically approaches to $-\alpha$. When $\alpha = 0$, the network behaves like ReLu, while higher values allow more negative activations.

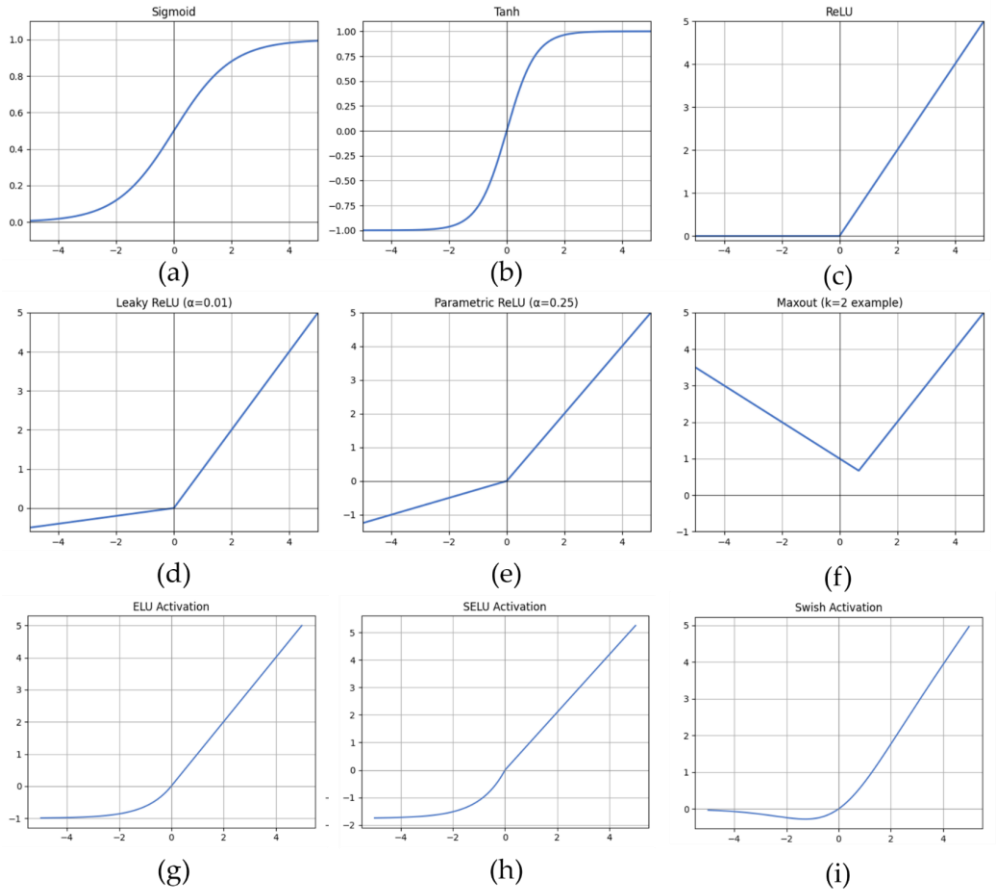


Figure 66 Activation functions in deep learning (a) sigmoid, (b) Tanh, (c) ReLu, (d) LReLu, (e) PReLu, (f) maxout, (g) ELU, (h) SELU, and (i) Swish activation function

The negative values push the mean of the activation function closer to 0, which helps to reduce bias shift and enables faster convergence during the training. ELU also becomes saturated for small inputs and decreases the information passed to the next layer, resulting in a noise-robust and low-complex representation.

The **Scaled Exponential Linear Unit** (SELU) (**Figure 66** (h)) [37] represents an improvement of ELU by introducing self-normalization which ensures that the output remains normalized. It is defined as

$$SELU(x) = \lambda \begin{cases} x, & if\ x > 0 \\ \alpha(e^x - 1), & if\ x \leq 0 \end{cases}$$

where $\lambda > 1$ is a scaling constant and α is a hyperparameter. It has an output range in $[-\lambda, \infty]$. The values of λ, α can are chosen in such a way that activations automatically normalize to have zero mean value and unit variance. This can prevent vanishing gradients or exploding gradients, stabilizing the learning process and enabling faster convergence. However, it is sensitive to initialization.

Swish is an adaptive activation function (**Figure 66** (i)) introduced by [38]. It is defined as follows

$$Swish(x) = x \cdot \sigma(\beta x) = \frac{x}{1 - e^{-\beta x}}$$

where β is a learnable parameter that controls the amount of non-linearity based on the dataset and network architecture complexity. If $\beta = 0$ Swish becomes a linear function $f(x) = x$, while for large values it becomes like ReLu. Swish is smooth, differentiable, non-monotonic, and one-sidedly bounded at zero, properties that often lead to superior performance compared to ReLU and other standard activation functions in deep neural networks. However, it is more computationally expensive and less interpretable than ReLU.

9.11 Normalization

In ML, normalization is a crucial preprocessing step, especially when input features have different scales. For example, one column has values from 0 to 1, and another has values from 1000 to 5000; the learning algorithm may become biased to the input with higher magnitudes. Normalization eliminates this issue by rescaling all features to a common scale (for example, 0 to 1 or -1 to 1) without losing the discriminative strength while maintaining the general data distribution. In addition to increasing the model training speed and generalization ability, it improves overall model stability. It can be done at the function level or the batch level.

The most widely used normalization method if the input data are normally distributed, is z-score normalization, given as

$$z = \frac{x - \mu}{\sigma}$$

where μ is the mean and σ represents the standard deviation computed for each feature. It rescales the data to have zero mean and unit variance. The method is often used in logistic regression, PCA, and SVM.

The MinMax normalization rescales every feature to the interval [0, 1] using the following formula

$$z = \frac{x - min(x)}{max(x) - min(x)}$$

It is sensitive to outliers, which can significantly affect the scaling.

As already mentioned, deep neural networks consist of multiple layers of neurons that apply piecewise linear/non-linear functions, enabling models to learn complex non-linear mappings between input and output. However, due to increased learning capacity, training of DL models is difficult due to the highly non-convex nature of optimization. In contrast to ML, where the normalization is a data preprocessing step applied to input features, in DL, it is mostly performed inside the network itself. Normalization techniques in DL can be broadly categorized into batch-based, layer-based, instance-based, group-based, and weight-based normalization. The layer normalizations are commonly used in language applications, while Batch Normalization (BN) has been extensively used for computer vision tasks.

When training deep neural networks, the inputs are passed through multiple layers, and each layer applies transformations. After each transformation, the activations may vary widely in magnitude. Additionally, the distribution of inputs to each layer can change during training—a phenomenon known as covariate shift—which often necessitates a lower learning rate and careful parameter initialization. When using the stochastic gradient descent SGD, training proceeds in steps, and each step considers a mini-batch of size m, allowing better gradient estimation and parallel computation. BN [39] deals with the reduction of covariate shift by normalizing the internal activations of the network. It is typically applied per feature dimension before applying the activation function to the outputs of a layer. The BN transformation applied over activation is presented in the algorithm.

Algorithm BN

Input: values of x over a mini-batch $B = \{x_1, \ldots, x_m\}$
learnable parameters γ, β
Output: $y_i = BN_{\gamma,\beta}(x_i)$

compute mini-batch mean $\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$
compute mini-batch variance $\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$
normalize $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
scale and shift $y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$

The γ and β are learnable parameters and ϵ is a constant added for numerical stability. BN normalizes the inputs so that, within every mini-batch, they have zero mean and unit variance.

BN improves gradient descent by stabilizing the distribution of layer inputs, enabling the use of higher learning rates and reducing sensitivity to weight initialization. It can also act as a form of regularization. Furthermore, it significantly accelerates training because it allows larger learning rates and, in some cases, can partially replace dropout without increasing overfitting. However, BN is sensitive to the mini-batch size. During inference, i.e. the testing or prediction phase, the mean and standard deviation are not computed from the batch; instead, fixed empirical values calculated from the training phase are used to normalize the activations.

9.12 Training, Validation, and Testing dataset

In DL and ML, the most common practice regarding the use of available datasets is to split each dataset into three datasets:

- **training dataset** - data used for model fitting with multiple model parameters. It contains the larger portion of the data with both input features and output labels. In each interaction, performance measures are used to assess the errors of the model when applied to the training dataset (training error). The training error is used to optimize model

parameters (such as weights in neural networks or coefficients in linear regression),

- **validation dataset** - used to rank and select the best-fitted model. It is usually created by splitting the training set; it contains samples with known labels, but the label is not exposed to the model; instead, they are used to evaluate the model's performance. Based on the errors on the validation set (validation error), the optimal architecture and model hyperparameter set (the hyperparameters are not learned during training) are selected as those that achieve the lowest validation error, and
- **test dataset** - the unseen data used to assess the generalization ability of the final trained model. It is not used in any part of the training process. Notably, the accuracy estimated using this unseen test dataset provides an unbiased estimate of the model's performance on any new samples drawn from the same distribution as the training and test sets.

The split ratio between those sub datasets depends on the size of the available dataset and the complexity of the model. Typically, 70% of available data is used for training, 15% for validation, and 15% for testing, but ratios can vary. As the size of the available dataset increases, the percentage between the training and validation datasets can be smaller. The created test set cannot be too small since a small test dataset may not provide a reliable estimation of generalization abilities. However, a larger test set means a smaller training dataset, which can have a negative impact on model performances, especially for small datasets.

9.13 Capacity, overfitting, and underfitting

ML models must not only achieve low training error, but they also need to perform well on new, unseen data. The ability of algorithms to perform well on data that are not used during the training process is called generalization. The generalization ability reflects models' predictive capacity on unseen data. It is typically assessed using the test dataset (also known as test error).

Take linear regression as an example: the model is trained by minimizing the training loss

$$\frac{1}{m_{training}} \left|\left|x_{train}w - y_{train}\right|\right|_2^2 .$$

However, the test error is more important for evaluating the model's generalization:

$$\frac{1}{m_{test}} \left|\left|x_{test}w - y_{test}\right|\right|_2^2 .$$

The low training error does not guarantee good performance on unseen data, which is why the test error is the key measure of a model's predictive capability.

To relate training error to performance on a test dataset, we assume that the training and test data are independent and drawn from the same underlying probability distribution, often called the data-generating distribution D. Importantly, we do not assume any specific form for this distribution; we only require that the datasets are identically distributed. Let's say that we have a probability distribution D and every data point (x, y) in the dataset is independently drawn from the distribution, i.e. $(x_i, y_i) \sim D$. By repeatedly sampling from D, we can generate both a training set and a test set. Because both sets come from the same distribution, the patterns learned from the training data are expected to generalize to the test data. Typically, the training error—computed on the dataset used to optimize the model parameters—will be equal to or slightly lower than the test error. Therefore, a machine learning algorithm should aim to achieve both low training error and low test error to ensure good generalization. These two properties of the ML algorithm represent the two central challenges: underfitting and overfitting (Figure 14). Underfitting occurs when the model cannot achieve a low error rate even on the training set. Overfitting occurs when the model achieves very low training error but much higher test error (i.e., the gap between the training error and test error is large).

Underfitting and overfitting can be controlled by adjusting a model's capacity, which roughly corresponds to the number of trainable parameters. Capacity reflects the model's ability to capture complex relationships between input and output data. A model with low capacity may struggle to fit the training data, leading to underfitting. Conversely, a model with high capacity can overfit, memorizing details of the training set that do not generalize to the test set. Essentially, more trainable parameters allow the model to store more

information, which can include noise or patterns irrelevant for unseen data. In machine learning, capacity can be controlled by selecting the hypothesis space—the set of functions the model is allowed to use.

Consider the problem of predicting y from x. In a linear regression problem, we want to decide whether to fit the simple model, such as the linear regression model, i.e. $\hat{y} = b + wx$ or a mode complex model such as the 5-degree polynomial $\hat{y} = w_1x + w_2x^2 + \ldots + w_5x^5$ (**Figure 67**). ML models perform best when their capacity is well-matched with the true complexity of the task and the size of the available dataset.

The figure compares three types of models: linear, quadratic, and 5-degree polynomial estimators. The linear model, having very low capacity, is unable to capture the curvature inherent in the true relationship between x and y. As a result, it underfits, producing predictions that are systematically off across the dataset.

On the other end of the spectrum, the 5-degree polynomial model has high capacity and is capable of perfectly predicting y for all examples in the training dataset. While this may seem ideal at first glance, it comes at a cost: the model essentially memorizes the training data, including any noise or idiosyncrasies present. Consequently, it fails to generalize to unseen data points, producing poor predictions on the test set. This phenomenon is known as overfitting. The difficulty arises because when a model has such high flexibility, there exist many wildly different functions that can fit the training data exactly, making it hard to select one that performs well on new data.

In contrast, the quadratic model strikes the right balance. Its capacity matches the true underlying structure of the task, allowing it to capture the essential curvature without memorizing irrelevant details. This enables the quadratic model to generalize well, producing accurate predictions not only on the training set but also on unseen data. In other words, the quadratic function exemplifies a model that avoids both underfitting and overfitting, achieving strong generalization.

Although simpler models are generally more likely to generalize well, a model must also have sufficient complexity to achieve a low training error. As model complexity increases, the training error typically decreases, eventually approaching the minimum possible error.

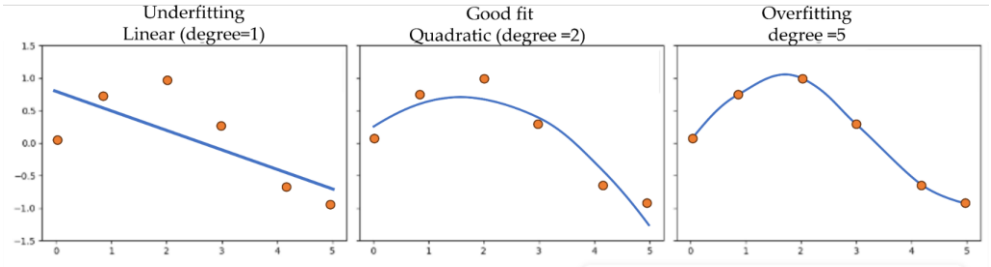


Figure 67 The fitted models to the training set

However, the effect of model capacity on test error is non-monotonic: initially, increasing complexity improves generalization and reduces test error, but beyond a certain point, further increases in capacity lead to overfitting, causing test error to rise. This phenomenon results in the familiar U-shaped curve of generalization error as a function of model capacity, where the optimal model lies somewhere in the middle—complex enough to capture the underlying patterns but not so complex that it memorizes the training data.

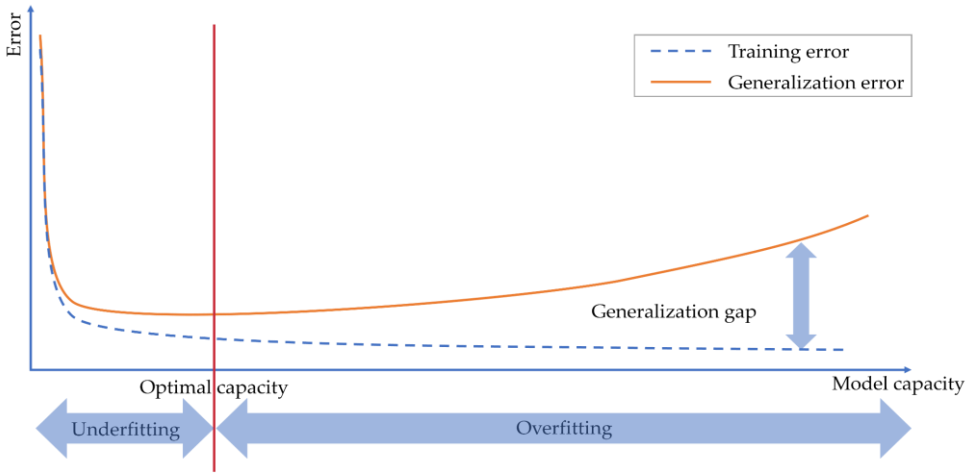


Figure 68 The relationship between model capacity and error

As model capacity increases, the training error decreases; however the gap between training and generalization error also increases (**Figure 68**). When this gap becomes larger than the gains from reduced training error, the model begins to overfit.

Moreover, the model trained on a larger dataset tends to generalize better. This is due to the fact that it is easier for a model to memorize a small dataset, including noise and sample quirks. Larger datasets contain fewer accidental patterns relative to their size, allowing the model to focus on learning the true underlying patterns in the data.

The high-performing ML algorithm should have sufficient capacity to learn true data patterns, but not too much to memorize the training dataset and accidental patterns.

Because the relationship between model capacity and test error is non-monotonic, one strategy to control model capacity is to tune hyperparameters using a validation dataset. Grid search and manual search are the most widely used strategies for hyperparameter optimization. In manual search, the users use hand-tuned hyperparameters based on intuition or experience and a trial-and-error approach to determine appropriate hyperparameter values. The trial and error approach refers to training multiple models by using a random model configuration and choosing one with the lowest validation error. In grid search, the search space is a regular grid created by defining the set of possible candidate values for each hyperparameter. The model is trained by using all possible combinations, and the one with the lowest validation error is selected. The grid search is easy to understand and implement; it enables repetition of experiments with the same settings, but it is computationally intensive when the number of parameters is high.

9.14 Regularization

Regularization is a key technique in machine learning and deep learning, as it helps reduce a model's generalization error without increasing the training error. Regularization encompasses techniques designed to reduce a model's tendency to overfit the training data, thereby improving its accuracy on unseen data.

In traditional ML, the regularization refers to constraining the loss function of the training model. In DL, regularization includes several techniques that can be divided into: loss-based regularization, data-based regularization, and architecture-based regularization. For example, let x be an independent variable and y be a dependent variable. The linear regression will provide

high accuracy if the relationship between variables is linear. The linear regression has a hyperspace that contains a set of functions that can be used for this problem. Regularization is used to introduce desirable preferences into training, i.e., it constrains the type of function that can be used for the solution.

9.14.1Loss-based methods

Loss-based methods work by modifying the loss function to include an additional term that penalizes excessive model complexity.

$$L_{reg}(w) = L(w) + \lambda R(w)$$

where $L(w)$ is a loss function designed for a specific objective, $R(w)$ is a regularization term that is independent of the target and $\lambda \geq 0$ is a scalar that represents the importance of the regularization term that penalizes the model's trainable parameters. If $\lambda = 0$ the regularization term is removed and weights are close to their initial solution, while for large λ the regularization strongly penalizes the large weights, i.e., the loss term is insignificant, and the regularization term forces weights to be close to 0.

Minimization of the $L_{reg}(w)$, leads to the choice of weights that balance between model underfitting and overfitting and therefore improve the generalization ability of algorithms. Training the model using a modified loss function will result in model parameters with desirable properties that are defined by the regularization term. The two most commonly used types of regularization are L1 norm (also called *Lasso - Least Absolute Shrinkage and Selection Operator*) and L2 norm (also called ridge) regularization.

The *L1 norm* regularization encourages sparse solutions by setting the network weights to zero, effectively reducing overall network complexity. (let us remind that a sparse solution is one where most of the parameters - weights - are zero, and only a small number of them remain nonzero) .It is defined as

$$L_{lasso}(w) = L(w) + \lambda \left|\left|w\right|\right|_1$$

where $\left|\left|w\right|\right|_1 = \sum_i |w_i|$ is the *L1* norm of the weight vector w. The higher the value of λ is, the more likely L1 regularization will drive additional weights

to zero. *Lasso* is often used for feature selection, because it tends to assign zero weights to irrelevant features, effectively removing them from the model.

L2 norm, also known as weight decay, encourages large weights to shrink toward zero and can be interpreted as performing maximum a posteriori (MAP) estimation with a Gaussian prior on the weights. In linear regression, using MSE as a loss function, we often wish to reduce extreme values of model parameters. This can be achieved by modifying the loss function to include the weight decay. The regularized loss function becomes

$$L_{ridge}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w^T x_i)^2 + \lambda \sum_{j=1}^{d} w_j^2$$

where the $\sum_{j=1}^{d} w_j^2 = ||w||_2^2$ is the *L2 norm* of the weight vector. Finding the model weights that minimize the *L2* regularized loss is also known as ridge regression. Ridge regression is solved in three steps: select λ, minimize the ridge cost function $w = (X^T X + \lambda I)^{-1} X^T Y$ and record R^2 on the test set, and find the λ that gives the largest R^2. Selecting optimal λ is hard, and usually cross-validation is used.

From the Bayesian perspective, we start with a prior distribution over the model parameters (or hypotheses). As data are incorporated into the model, this prior is updated to form a posterior distribution. Specifically, L1 norm regularization corresponds to assuming a Laplace (double exponential) prior on the weights, which encourages sparsity, while L2 norm regularization corresponds to a Gaussian (normal) prior, which encourages smaller but nonzero weights.

9.14.2 Data-based regulation

The success of ML and DL models depends on the training data. The easiest way to increase the generalization ability of the modes is to train them on a large dataset. However, the creation of a training dataset is time-consuming and financially demanding, and the available trained data are always limited. This challenge, especially in computer vision and medical image analysis domains, can be addressed by augmenting the available datasets.

In classification, the model maps a high-dimensional input x into a single output y. To provide high generalization, the trained model needs to be

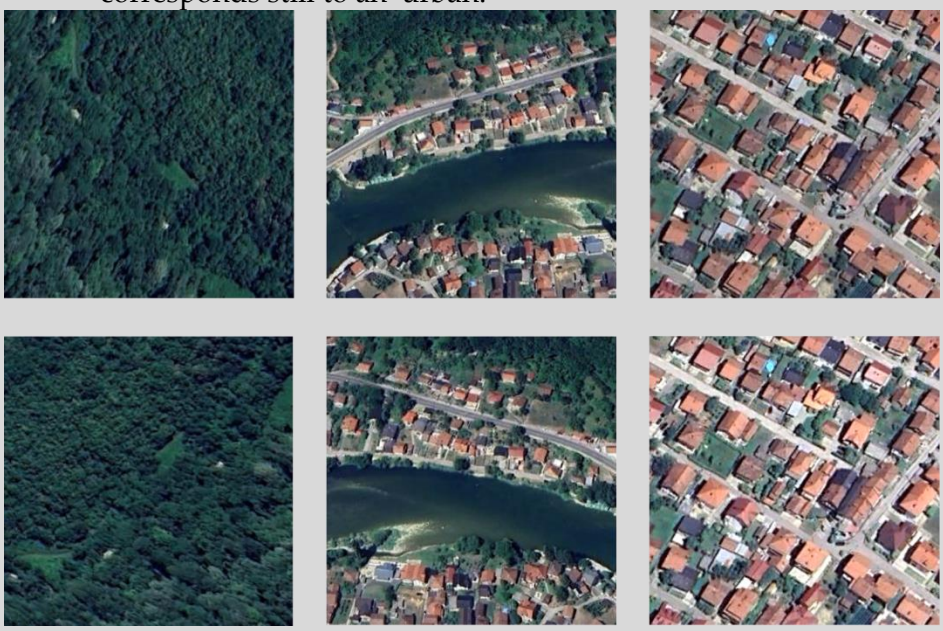
invariant to a wide variety of transformations. The easiest way to enlarge the dataset is to generate the new (x, y) pairs by transforming the existing input data. This approach is known as data augmentation.

Let us consider a simple example: we want to perform a satellite image classification task where the goal is to classify land cover types (e.g., forest, water, urban) from high-resolution images. Each image patch is an input x, and the corresponding land cover label is the output y.

To help the model generalize well, we want it to be invariant to transformations such as rotation, translation, or changes in lighting — for example, a forest looks like a forest whether the image is slightly rotated or shifted.

Data augmentation can help by generating new training samples from the existing images. For example:

- Rotate an image of a forest by 90 degrees corresponds still to a forest;
- Flip a river image horizontally corresponds still to a river;
- Slightly adjusting the brightness of an urban area image corresponds still to an urban.



These transformations create new (x, y) pairs without manually collecting more data. The augmented dataset is larger and more varied, which helps the model learn robust features and generalize better to unseen satellite images.

Data augmentation applies stochastic transformations to modify original training samples, creating new ones that enlarge and introduce variability into the dataset. So, the main aim is for a given dataset D consisting of x_i training samples with corresponding labels y_i apply the transformation T to create new training data x'_i without altering the corresponding label i.e. $T(x_i, y_i) \rightarrow (x'_i, y_i)$. Different transformation operations, such as geometric or radiometric transformation, can be used.

Geometrical transformations change the geometrical structure of an image by mapping pixels to new positions without altering the pixel value. The most commonly used geometrical methods are affine transformation, including rotation, translation, scaling (zooming and cropping), horizontal or vertical flipping, and mirroring. The same geometric transformations can be applied to entire point clouds or specific instances (such as vehicles). Moreover, non-affine transformations, like projective or perspective ones can be used. The geometrical transformations are simple, computationally effective, and usually used as primary data augmentation models in computer vision.

Returning to the previous example, the geometric transformations work well for creating more training data only if the new, transformed images still look realistic and represent the kinds of data the model will see in the real world. Moreover, translation or rotation suffers from a padding effect, i.e., new pixels need to be added to fill in the empty areas created by the transformation, which can lead to the omission of the target objects and loss of information.

Another type of data augmentation is introducing random noise into inputs to increase the robustness of the models. In the case of images, new data can be for instance generated by randomly perturbing the RGB information of the pixels. Commonly used noise types in remote sensing include Gaussian noise, salt and pepper noise, Jittering (for instance adding small spatial variation in point cloud data), and speckle noise. New training data can be generated by modifying the image sharpness. In remote sensing, this can involve: sharpening, which reduces blur by enhancing high-frequency components and making edges more distinct; blurring, which smooths the image by averaging the values of surrounding pixels, reducing noise and fine details.

Sometimes parts of objects in images are hidden (occluded), which makes it harder for models to learn. This challenge can be addressed by using the

cutout that removes contiguous sections of input images. This forces the model to rely on the surrounding context and learn to recognize objects even when parts are missing. Introducing these modified images into the training dataset helps the model become better at handling occlusions and encourages it to use more image context during decision-making. Also, Mixup [40], which randomly selects two images and mixes them in a certain ratio to form a new image, is used. This encourages the model to generalize better because it learns from blended examples instead of just single images. The same idea is applied for point clouds on the instance level (individual objects or samples in the dataset) by randomly selecting two samples and mixing them to create a new training sample.

The point cloud augmentation often includes randomly dropping out some data points, enabling the model to become more robust to missing or incomplete representations.

In recent years, the Generative Adversarial Network (GAN) [16] has become the most popular model for artificially generated image data. It consists of a generator that tries to create realistic data so that the discriminator that distinguishes between real and generated data cannot tell the difference. Competition between these two sub-models (the generator and the discriminator) continuously optimizes and enables the generation of high-quality data. For example, GAN can be used to transform the visual appearance of an image taken under one set of conditions (sunny) to a different set of conditions (haze). The conditional GAN (CGAN) is an extension of GAN where both generator and discriminator are conditioned on additional information, such as reference images or labels. This guidance encourages the model to generate data with specific desired features rather than purely random samples.

However, generative models are difficult to train on a limited dataset and they often require data augmentation to perform effectively. Additionally, they can suffer from mode collapse, a phenomenon in which the generator fails to produce diverse outputs and instead generates limited or repetitive samples.

9.14.3 Network-based regularization

Early stopping is one of the most used regularization methods in deep learning. It is considered a training strategy since it does not require changes of loss function or training procedures. During network training, the training error and validation error are interactively reduced. If a model has sufficient capacity to overfit, the validation error starts to increase once the network begins to memorize the training data. The main idea behind early stopping is to halt training when the validation error reaches its minimum, thereby improving the network's generalization ability. To implement this, early stopping stores the model parameters each time the validation loss decreases. Once training is complete, the parameters corresponding to the lowest validation error are restored and used, rather than the parameters from the final training iteration.

Early stopping controls model capacity by limiting the number of iterations used to fit training data. The number of iterations is determined based on monitoring the validation error, i.e., the algorithm is terminated when there are no at least minimal improvements in the validation error over a predefined number of iterations. Therefore, the two hyperparameters must be specified empirically: the delta hyperparameter, which represents the minimal change in validation loss to be considered as an improvement, and the patience hyperparameter, which defines the number of iterations with no improvements after which the training will be stopped.

Early stopping is a simple and effective regularization method that reduces training time by limiting the number of iterations. It can be used alone or in combination with other regularization techniques. However, early stopping has some limitations. It requires a validation set, which can be challenging when the available dataset is small. The size of the validation set also introduces a bias–variance trade-off: a small validation set may lead to unreliable stopping decisions, while a larger validation set reduces the data available for training. Additionally, running periodic evaluations on the validation set increases computational cost. Early stopping also requires storing a copy of the model parameters corresponding to the lowest validation error, although this storage cost is typically negligible.

Generally, the use of deeper neural networks, with a high number of units, enables models to learn very complicated patterns in input data. However, large and more complex architectures are prone to overfitting. One of the most frequently used regularization methods to prevent network overfitting is Dropout.

Dropout [41] randomly drops units from the network during each forward pass. Dropout can be observed as a way of reducing the model complexity by randomly setting individual neurons' output to 0. The probability of dropping each unit in a network is estimated using the fixed Bernoulli probability p. In that way, the unit and all its connections are temporarily removed from the network, and a new "thinned" architecture is sampled and trained (**Figure 69**).

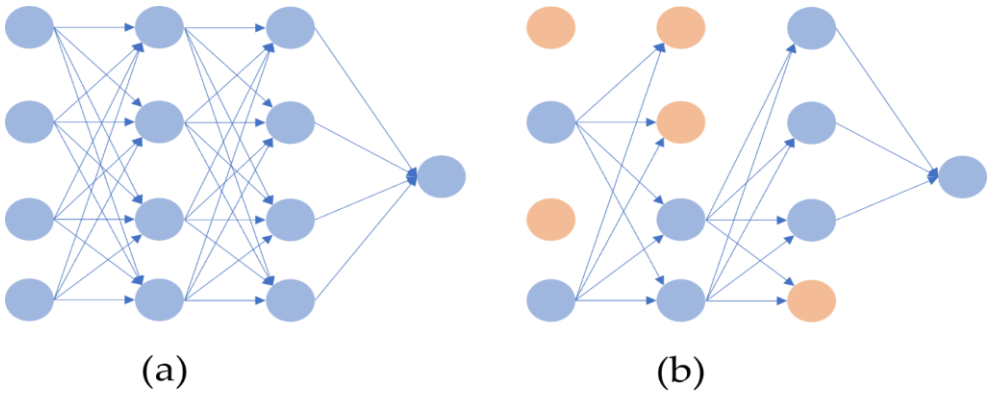


Figure 69 (a) Standard neural network architecture, (b) "thinned" network architecture by dropping orange neurons and all their connections.

Dropout can be interpreted as training 2^n "thinned" networks, each with a distinct, randomly selected subset of neurons, while sharing weights extensively. It leverages the well-established strategy of reducing model variance and overfitting by effectively averaging the predictions of multiple models. Consider a neural network with $l = [1, \ldots, L]$ hidden layers, where the input of each lth layer is denoted by x_i while y_l represents the vector of outputs from the layer l. The feedforward operation is given by

$$y_l^i = w_l^i x_l + b_l^i$$

$$x_{l+1}^i = f(y_l^i)$$

where f is the activation function;

y_l^i represents the linear transformation of the input x_l based on the trainable weights w_l^i and bias b_l^i associated with hidden unit *i* in the layer *l*;

x_{l+1}^i is the output of the hidden unit which forms a part of the input to the subsequent (*l+1*)-th layer in the network and which is given by the application of the activation function to the pre-activation output y_l^i.

With Dropout, the same feedforward operation may be expressed as follows:

$$r_l \sim Bernoulli(p)$$

$$\hat{x}_l = r_l \odot x_l$$

$$\hat{y}_l^i = w_l^i \hat{x}_l + b_l^i$$

$$\hat{x}_{l+1}^i = f(\hat{y}_l^i)$$

where r_l is a vector of independent Bernoulli random variables, each having a probability p of being 1 and *1-p* of being 0. This vector is sampled and multiplied elementwise (the operation is denoted with $\odot$) with the outputs x_l of that layer to create the thinned inputs $\hat{x}_l$. The thinned outputs $\hat{x}_{l+1}^i$ are then used as inputs to the next (*l+1*)th layer. This process is repeated for each layer, creating sub-networks. For learning, the weights are backpropagated through the sub-network. For testing, the full neural network is used without dropout. To account for the neurons that were dropped during training, the outgoing weights of each unit are scaled by the dropout probability p, i.e., $w_l^{test} = pw_l$. The probability p is a user-specified hyperparameter that needs to be tuned. The $p = 1$ means no dropout, and a small p leads to more dropout. It is typically set between 0.5 and 0.8. Dropout is relatively easy to implement, it provides a trade-off between overfitting and training time, and improves the performance of the neural networks in different domains. However, it also increases the training time and needs careful tuning of p.

9.15 Cross-validation

When the available dataset is too small to create a representative training/validation dataset for accurate estimation of generalization error, an alternative approach can be used. This approach is based on repeating the training and testing computation on different randomly chosen subsets of the original datasets.

The k-fold cross-validation is most commonly used. In k-fold cross validation, the original dataset is randomly splitter in k disjoint subsets. For each iteration, one subset is used as a validation set, and the rest *k-1* subsets are combined and used as training sets. This process is repeated k times, so each subset is used as a validation set. The approximation of validation error is estimated as the average validation error across k iterations. The algorithm is shown below.

Algorithm k-fold

Input: dataset D with $z_i = (x_i, y_i)$ elements, A learning algorithm, L loss function, k - number of subsets, learning algorithm and

Split D into k disjoint subsets D_i whose union is D

Return: vector of errors e

for i from 1 to k**:**

    $f_i = A(D/D_i)$

    for z_j in D_i:

        $e_j = L(f_i, z_j)$

    end

end

return e

The parameter k is chosen in such a way that the resulting subgroup is a representative sample of the data set, and based on the available computation resources. Usually, 10-fold or 5-fold is used. Once the k-fold cross-validation is done, the best model or hyperparameters are selected as the one that had the lowest averaged validation error. Then the final model is trained on the full training data and then evaluated once on the test set.

Although the k-fold provides a more reliable estimation of validation performance, in the case of imbalance, it can lead to unstable performance. The stratified k-fold is used to address that limitation by ensuring that each of the k subgroups has the same class distribution as the full dataset. The process can be parallelized to speed up the model evaluation.

9.16 Bias-variance trade-off

Let $\hat{\theta}_m$ be an estimator where m is the size of the dataset. For example, in the case of the linear regression, $\hat{\theta}_m = (X^T X)^{-1} X^T y$. The distribution of the $\hat{\theta}_m$ is called the sampling distribution. The bias and variance of the estimator $\hat{\theta}_m$ correspond to the first and second moments of this sampling distribution. They quantified two different sources of error in the estimator.

The bias measures how the average model prediction over the entire dataset differs from the true value of the function or parameter. The bias of an estimator is defined as

$$bias(\hat{\theta}_m) = E(\hat{\theta}_m) - \theta$$

where $E(\hat{\theta}_m)$ is the expected value (mean) of the model and θ is the true underlying value of θ used to define the data-generating distribution. An estimator $\hat{\theta}_m$ is unbiased if the bias is equal to 0 (i.e. $E(\hat{\theta}_m) = \theta$). An estimator $\hat{\theta}_m$ is asymptotically unbiased if $\lim_{m \to \infty} bias(\hat{\theta}_m) = 0$.

The variance ($Var(\hat{\theta}_m)$) reflects the extent to which estimators for individual data sets vary around their expected value. Therefore, it measures the extent to which the estimator is sensitive to the particular sampling of the data.

Alternatively, the standard error ($SE(\hat{\theta}_m)$) that represents the square root of the variance can be used. When we estimate statistics using a finite number of samples, our estimation of the true underlying population parameter (such as the mean) is uncertain. This is due to the fact that we may obtain different results if different samples have been used. The expected degree of variation in any estimator is a source of error that needs to be quantified. Under MSE, the bias and variance are related as

$$MSE(\hat{\theta}_m) = E\left[\left|\left|\hat{\theta}_m - \theta\right|\right|^2\right] = Bias(\hat{\theta}_m)^2 + Var(\hat{\theta}_m) + \sigma^2$$

where σ^2 is an irreducible error caused by inherent noise in the dataset, which cannot be eliminated. The aim is to minimize the expected loss, so it is preferable that estimators exhibit low bias and have relatively low variance. The bias and variance of an estimator are not necessarily directly related. In practice, many techniques used to reduce variance tend to increase bias, and techniques that reduce bias can increase variance. This phenomenon is called

the bias-variance trade-off. In **Figure 70** this concept is represented with four target diagrams that resemble shooting at a bullseye, where each shot represents a model's prediction and the center of the target represents the true value. The balance between bias and variance is essential for building a well-generalized estimator.

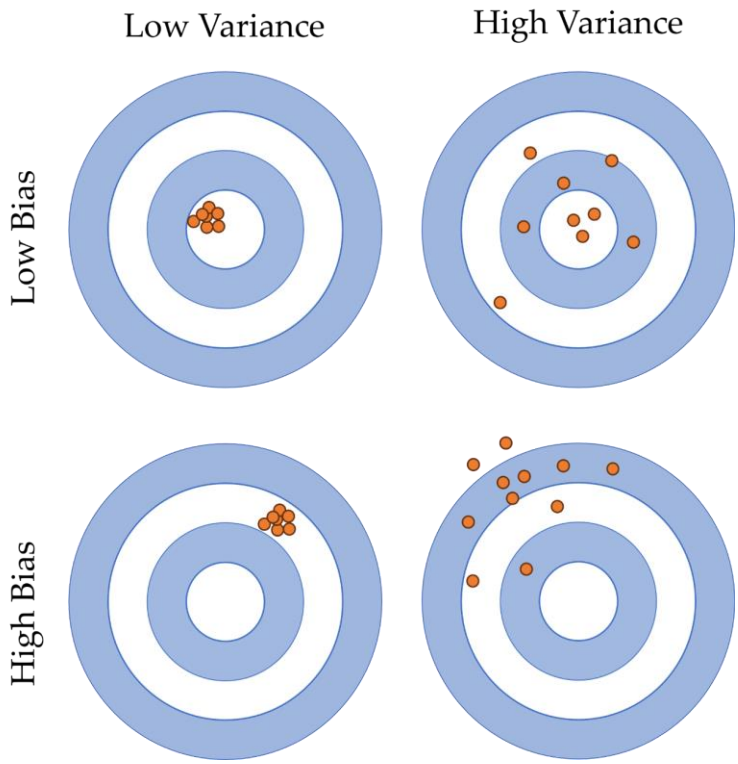


Figure 70 Graphical representation of Bias and Variance

In the case of two estimators, one may have low bias but high variance, and the other may have low variance but high bias. Which one should be selected? The most common approach to balancing between bias and variance is to use cross-validation. Alternatively, it is possible to compare the MSE of the estimators since it captures the overall expected prediction error, incorporating both the bias and variance. Desirable estimators have the lowest MSE, i.e., they manage to keep both bias and variance low.

The relationship between bias and variance in ML is tightly related to underfitting and overfitting. This is due to the fact that increased model capacity tends to increase variance and decrease bias, i.e.:

- Overfitting - the model or estimator has a high variance, i.e., the model is too complex (it memorizes training data but fails to generalize). To reduce overfitting, it is necessary to reduce the variance of the estimator by regularization, using more training data, reducing the number of features, or simplifying the model, etc.
- Underfitting - models/estimators have a high bias, i.e., the model is too simple to capture complex patterns in the data. So, reduction of underfitting is based on reducing bias by: reducing regularization, adding more relevant features, and increasing model complexity.

When evaluating a model, it is essential to take into account the training error and the cross-validation error simultaneously. The training error can be viewed as the measure of the bias in the model. If the model is unable to fit the training data accurately, then it is likely that the model has high bias (underfitting) (**Figure 71**). The gap between validation and training error provides an indicator of the model variance. The low training error and higher validation error indicate high variance (the model is overfitting the training data)(**Figure 71**).

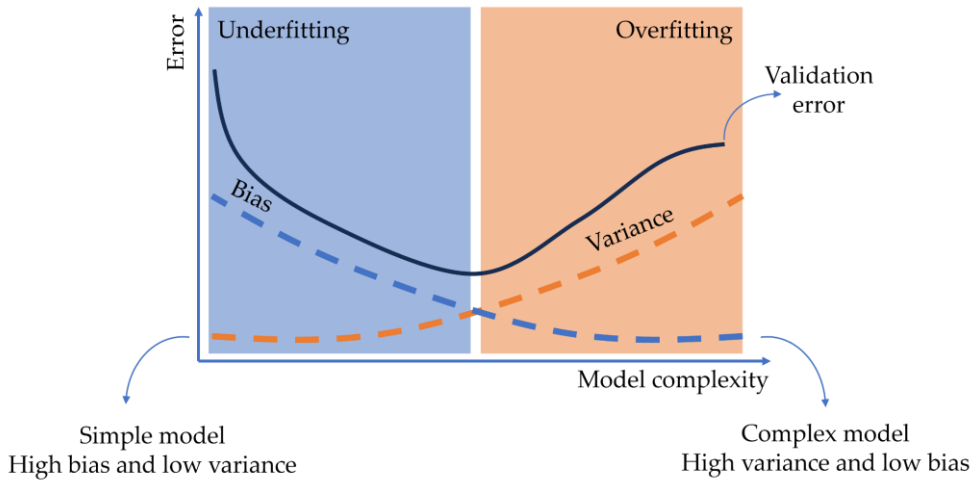


Figure 71 Validation error as a function of model complexity

9.17 Performance metrics

Performance metrics are used to assess the predictive power of the developed models. In ML, the selection of appropriate metrics is an important step. Various metrics can be used but they need to be chosen carefully depending on the type of application. The list of performance metrics used in different tasks is presented in Figure 71.

9.17.1 Performance metrics in regression

Selecting appropriate metrics is crucial for the accurate evaluation of regression models. Most commonly used metrics in regression are: *RMSE*, *MSE*, *MAE*, *MAPA*, R^2, and $Adjusted\ R^2$.

RMSE is defined as the square root of the MSE. For n data points, the RMSE is given by

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y_i is the true value and $\hat{y}_i$ is the predicted value. In general, the smaller the value, the better the model's performance. RMSE is easy to interpret; it has the same unit as the target y. As already mentioned, the main drawback of this metric is sensitivity to outliers (a few outliers can produce a significant increase in RMSE), and it does not differentiate between error types (underestimation or overestimation).

The **normalized *RMSE* (*NRMSE*)** represents the non-dimensional form of the *RMSE* that has been widely used for comparison of the regression models and algorithms of different scales. It is defined as follows

$$NRMSE = \frac{RMSE}{y_{max} - y_{min}}$$

where y_{max} is the maximum of the target true values and y_{min} is the minimum of true target values.

Mean Absolute Percentage Error (MAPE) represents the average relative error of prediction, expressed in percentages. It is given by

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|y_i - \hat{y}_i|}{y_i} \cdot 100\right)$$

MAPE is scale invariant, enabling the comparison between different tasks or datasets. Moreover, the percentages are a very intuitive interpretation of relative error. On the other hand, it is sensitive to zero and near-zero values (if y_i is 0 or close to zero the *MAPE* is undefined), and the presence of outliers. Additionally, over- and under-prediction of the same magnitude do not have the symmetrical impact on MAPA; for instance, +50% and -50% are both treated as 50% by MAPA, but their absolute effect is not the same.

For example, if NDVI goes from 100% to 150% it is a 50% increase in value, but if it reduces from 150% to 100% it is a decrease of -33.3% not -50%. This asymmetry occurs because relative change is calculated with respect to the original value, so increases and decreases of the same magnitude do not produce equal percentage changes. Consequently, metrics like MAPE, which rely on absolute percentage differences, can misrepresent the impact of over- versus under-prediction, potentially leading to misleading interpretations in remote sensing analyses.

The R^2 (coefficient of determination), It is the statistical measure of the goodness of fit of the regression line to the actual data, i.e., the portion of total variance in the dependent variable that is caused by the variation of the independent variables.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})^2}{\sum_{i=1}^{n}(y_i - \overline{y_i})^2}$$

It quantifies the predictive power of a regression model. The values range between 0 and 1, where 1 represents the perfect fit and 0 indicates that the model explains no variability of the dependent variable.

For example, the $R^2 = 0.65$ for biomass prediction based on NDVI means that 65 % of the variance is explained by independent variables (i.e., NDVI). The remaining 35 % is due to the variance of the dependent variable (y_i has variance σ^2), i.e., 35% of the variation is due to the variance of biomass, which is unexplained and arises from other factors or noise in the data.

Adding more variables to the model will always decrease $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$and therefore increase the coefficient of determination. So, trusting blindly to R^2 will always lead to the largest model and possible overfitting. Moreover, it does not detect bias, is sensitive to outliers, and can provide misleading values for small datasets due to unstable variance estimation.

The **adjusted** R^2 represents the modified version of R^2 by introducing the penalty for unnecessary prediction. The adjusted R^2 will only add new independent variables to the model if it improves the prediction accuracy. It is defined by

$$R^2_{ADJ} = 1 - \frac{\left(1 - R^2\right)(n-1)}{n - N - 1}$$

where n is the sample size and N is the number of independent variables in the model. The values range between -1 and 1. A high value of R^2_{ADJ} indicates that the model fits well the data and that the chosen variables contribute to explaining the dependent variable. The low or negative value suggests that model performance is not improved by adding more independent variables. The R^2_{ADJ} reduces the risk of overfitting and enables fair comparison for the model trained on the same datasets but using different parameters (nested models). However, its application in complex and nonlinear models is limited.

9.17.2Performance metrics in classification

A confusion matrix is a fundamental tool for the assessment of classification accuracy for both binary and multi-class classification. The confusion matrix represents counts of each combination of predicted and true values with respect to test data. There are four possible combinations:

- **True Positive (TP)** - indicates the number of instances that are correctly classified,
- **True Negative (TN)** - indicates the number of instances that are correctly rejected,
- **False Positive (FP)** - represents the number of instances that are incorrectly classified (type I error), and
- **False Negative (FN)** - number of intances that are incorrectly rejected (type II error).

The diagonal element of the confusion matrix (**Table 14**) represents the elements that are correctly classified, while the elements off the diagonal indicate the misclassification. The elements n_{ij} in the confusion matrix, indicate the instances belonging to i that had been classified as j. The size of the confusion matrix is equal to the number of classes, i.e., for a binary classification, the confusion matrix is 2x2, while for a multi-class classification, it is *k*x*k*, where *k* represents the number of classes.

Table 14 Confusion matrix

	Predicted positive	Predictive negative
Actual positive	**TP**	**FN**
Actual negative	**FP**	**TN**

Once the confusion matrix is created for a trained algorithm, the following metrics can be calculated: accuracy, recall, precision, f1-score, and kappa-coefficient.

Overall accuracy (OA) is used to reflect how often the model predicts the correct output. It is expressed in percentages, and it is given by

$$OA = \frac{TP + TN}{TP + FP + TN + FN} = \frac{correctly\ classified}{total\ number\ of\ samples}$$

Although OA provides insights into overall model performance, it can be misleading in the case of data imbalance.

> For example, let's say that we want to classify water/non-water and that water only represents 3% of pixels. If the model completely omits minor classes, the overall accuracy will be 97%. This high value is misleading, as the model entirely neglects the minority class.

Precision quantifies the fraction of predicted positives that are actually correct. It is defined as:

$$Precision = \frac{TP_i}{TP_i + FP_i}$$

In multi-class classification with *K* classes, *Macro-Precision* is calculated on a per-class basis and then averaged, i.e.

$$Macro - Precision = \frac{1}{K}\sum_{i=1}^{K}\frac{TP_i}{TP_i + FP_i}$$

Macro-Precision treats all classes equally, while *Micro-Precision* weights the classes by their frequency, and it is given as

$$Micro - Precision = \frac{\sum_{i=1}^{K} TP_i}{\sum_{i=1}^{K}(TP_i + FP_i)}$$

Precision is also known as *User accuracy* in classification terms. User's Accuracy reflects the viewpoint of the map user, who relies on the classified map to make decisions. It measures commission error, showing how reliable a mapped class is.

For example, if a user looks at a pixel labeled as "water," User's Accuracy tells them the probability that this pixel truly corresponds to water on the ground.

The *Precision* can have values from 0 to 1. Perfect precision of 1 means that all objects identified as positive are indeed positive and no false positive exists.

Recall (also known as TP rate) quantifies the portion of actual positives correctly classified by the model.

$$Recall = \frac{TP_i}{TP_i + FN_i}$$

Similarly to precision, *Macro-Recall* and *Micro-Recall* are defined as:

$Macro - Recall = \frac{1}{K}\sum_{i=1}^{K}\frac{TP_i}{TP_i+FN_i}$ and $Micro - Recall = \frac{\sum_{i=1}^{K} TP_i}{\sum_{i=1}^{K}(TP_i+FN_i)}$

Recall is essentially equal to the *Producer's accuracy* in classification terms. Producer's Accuracy represents the perspective of the map producer, assessing how accurately the classified map reflects the reference (ground truth) data. It quantifies omission error, indicating the proportion of real-world instances of a class that were correctly mapped.

For instance, if some actual water areas are omitted and incorrectly classified as urban, the producer's accuracy for water decreases.

Macro-Recall is also known as balanced accuracy. The *Recall* of 1 means that the model correctly identifies all positive cases as such, and no positive cases were ignored. However, both *Precision* and *Recall* can be influenced by class imbalance. The easiest way to maximize only *Precision* is to be very conservative in predicting positives, i.e., the model will only label an instance as positive when it is very confident, resulting in fewer positives and therefore reducing the *Recall*. On the other hand, maximization of recall can only be done by overpredicting a positive class, increasing the number of FP, and reducing *Precision*. Due to that, the *Precision* and *Recall* are analyzed together.

True Negative Rate (TNR) measures the proportion of actual negatives correctly classified as negative. It is given by the following

$$TNR = \frac{TN}{TN + FP}$$

It ranges between 0 and 1, the higher values are preferred. *TNR* is often used when the cost of FP is high, and in combination with Recall, provides a complete picture of model performance.

False Positive Rate (FPR) represents the proportion of actual negatives that are incorrectly predicted as positive, i.e.

$$FPR = \frac{FP}{TN + FP}$$

It ranges between 0 and 1, and lower values indicate fewer false alarms. The relationship between *FPR* and *TNR* is given by $FPR = 1 - TNR$. *FPR* has been extensively used when the cost of FP can be high, and also as the x-axis of a *ROC* curve.

F1-score balances *Precision* and *Recall* by taking their harmonic mean, given as

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

A higher value of *F1-score* indicates a more balanced classification, i.e., in order for an F1-score to be high, both *Precision* and *Recall* need to be high. It is very insightful for imbalanced datasets when the positive class is rare; however, it completely ignores how models classify the negative class (TN).

Cohen's Kappa Coefficient [42] is used in multi-class classification. It evaluates how well the model performs compared to just randomly assigned values, i.e., it measures agreement between prediction and ground truth, correcting for the agreement that could occur purely by chance. It is given by

$$\hat{\kappa} = \frac{p_o - p_e}{1 - p_e}$$

where p_o represent the observed agreement, and it is given as $p_o = \frac{\sum_{i=1}^{K} D_{ii}}{n}$ and p_e represents expected agreement $p_e = \sum_{i=1}^{K} \frac{R_i \cdot C_i}{n^2}$ where K is the number of classes, n is the total number of samples, $\sum_{i=1}^{K} D_{ii}$ sum of diagonal elements of the confusion matrix, R_i - row total for class i and C_i is the column total for class i.

The *Kappa Coefficient* ranges from -1 to 1. The negative value indicates that trained models perform worse than random classification. The interpretation of positive *Kappa coefficients* is shown in Table 15.

Table 15 Interpretation of Kappa statistics

Kappa	**Interpretation**
0.00 - 0.20	Poor agreement
0.21 - 0.40	Fair agreement
0.41 - 0.60	Moderate agreement
0.61 - 0.80	Substantial agreement
0.81 - 1.00	Almost perfect agreement

Although Kappa coefficients have been frequently used, they only measure the exact agreement and treat approximate agreements as disagreement. This limitation can underestimate the performance of models when small spatial or semantic deviations occur, which is common in geospatial and remote sensing applications where class boundaries are often uncertain or mixed.

The receiver operating characteristic curve (ROC) is a graph of the tradeoff between the TPR (on the y-axis) and FPR (on the x-axis) using different probability thresholds. The *ROC curve* provides a single measure of the

overall performance of the model across a full range of possible thresholds, allowing averaging their effect on accuracy. Due to that, it is a threshold-independent measure.

The **area under the ROC curve (AUROC)** (**Figure 72Figure 1** (a)) is used to determine scores, enabling the comparison of different ML algorithms. The score ranges between 0 and 1, with 1 being the perfect classifier. The 0.5 score indicates random guessing. So the classifiers with a curve close to the upper left corner are desirable, while classifiers with curves below the diagonal line perform worse than random. An *AUROC* of 0.935 means that the model has excellent discriminatory abilities, i.e., if we randomly choose one positive and one negative example, the model will score the positive one higher about 93.5% of the time. *ROC* curves can be misleading in the case of highly imbalanced datasets. That limitation can be addressed by using the Precision-Recall curve.

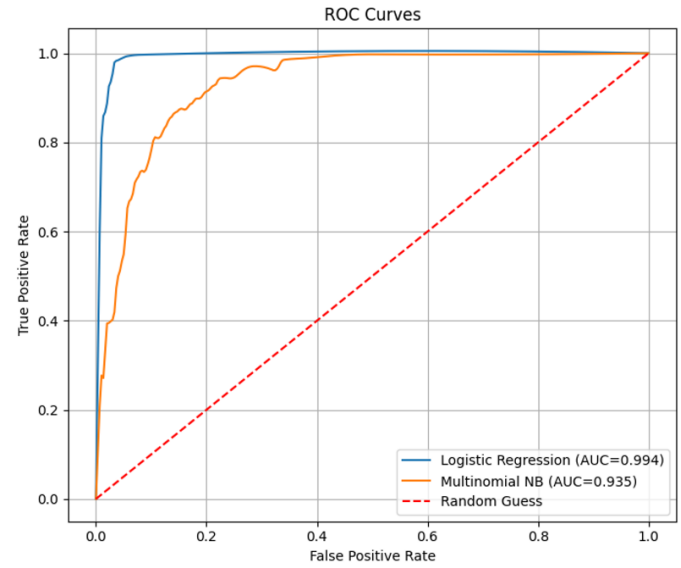
Precision-Recall Curves (PRC) (**Figure 72** (b)) visualize the tradeoff between Recall (x-axis) and Precision (y-axis). It is created by plotting the Precision-Recall pairs that are obtained using different thresholds on a continuous or probabilistic classifier. It is most often used for binary classifications, especially in imbalanced datasets.

Area under Precision-Recall Curve (AUPRC) has been used as a summary statistic when comparing the performance of different algorithms. The perfect model will have a *PRC* that passes through the upper right corner that corresponds to both *Recall* and *Precision* equal to 1. The closer the classifier is to this corner, the higher the *Precision* and *Recall* are. AUPRC score ranges from 0 to 1, a higher value indicates better overall performance on a positive class. In contrast to the ROC curve, where the baseline is fixed to 0.5, the baseline value for AUPRC is defined as

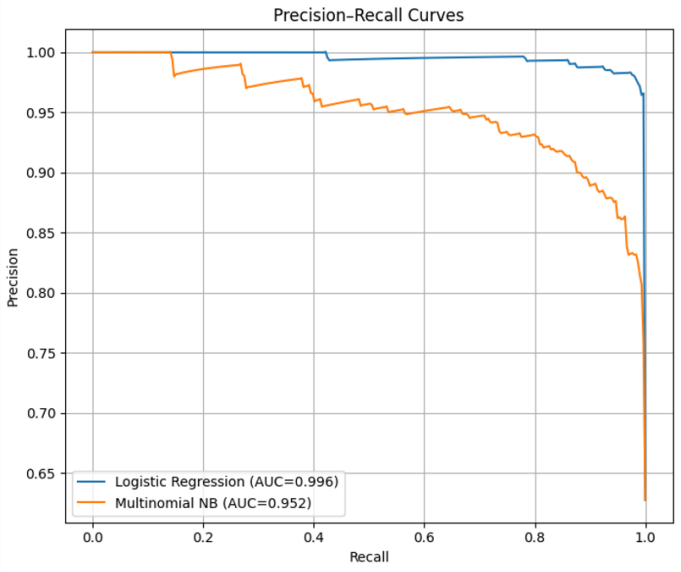
$$Baseline = \frac{positive\ cases}{positive\ cases + negative\ cases}$$

If AUPRC is lower than baseline, the model is doing worse than random guessing for positive classes. Since AUPRC does not use TN, it will not be affected by the large proportion of TN in the data.

Although it is primarily used for binary classification, it can be extended to multi-class problems by using a One-vs-All approach for each class (Class 1 vs Not Class 1, Class 2 vs Not Class 2, etc.).



(a)



(b)

Figure 72 (a) ROC curve for two models on the same dataset. The dashed red line indicates the random performance. (b) PRC curve for two models on the same dataset. The AUPRC is indicated in the legend.

Consider the semantic segmentation of a satellite image into three classes: water, forest, and urban, and we want to evaluate the model performance. The table shows a confusion matrix created based on the test set.

	Water	Urban	Forest	Total	Producer's accuracy
Water	21	6	1	28	0.75
Urban	8	51	3	62	0.82
Forest	2	7	44	53	0.83
Total	31	64	48	143	
Users Accuracy	0.68	0.79	0.92		

On Table 8, the rows of the matrix indicate the ground truth while the columns represent the classification results. Diagonal matrix elements represent the number of pixels that are correctly classified. In the above example, the 21 pixels of water in the test set are correctly classified. Off-diagonal elements represent the misclassified pixels, i.e., the classification errors. Off-diagonal row elements represent the ground truth pixels of a certain class that are excluded from that class during classification (FN). On the other hand, the off-diagonal column represents the true pixels of the other class that are included in a certain class (FP). For example, 6 ground truth pixels of water were excluded from the water class in the classification and ended up in the urban class, while 8 ground truth pixels of the urban class were included in the water class.

Recall (Producer's accuracy) represents the probability that any pixel in that class has been correctly classified. It is calculated for the water class as

$$Recall_{water} = \frac{21}{21 + 6 + 1} = \frac{41}{48} = 0.75$$

$$Macro - Recall = \frac{0.75 + 0.82 + 0.83}{3} = 0.80$$

$$Micro - Recall = \frac{21 + 51 + 44}{28 + 62 + 53} = 0.87$$

Therefore, the water class has a producer accuracy of 0.75, meaning that 75% of the water ground truth pixels also appear as water pixels in classified images.

Precision (User's accuracy) is defined as the portion of correctly classified positive pixels among all positive predictions made by the model, i.e., the reliability of classes in the classified image. For the water class, it is calculated as

$$Precision_{Water} = \frac{21}{21 + 8 + 2} = \frac{21}{31} = 0.68$$

Meaning that approximately 68% of the water pixels in the classified image actually represent the water on the ground.

The overall accuracy of the classified image is calculated as

$$OA = \frac{21 + 51 + 44}{143} = 0.81$$

meaning that 81% of samples were correctly classified.

The F1-score for the water class is $F1 - score = \frac{2*0.75*0.68}{0.75+0.68} = 0.71$.

Taking into account the metrics for water class, it can be concluded that the model effectively detects water class while maintaining a moderate rate of FP. Since the precision is less than the recall, the model tends to overestimate the water class.

9.17.3 Performance metrics in object detection

As already mentioned, object detection algorithms need to exactly localize an object and assign it to the correct class. To evaluate the object classification the TP, FP and FN are used. However, they are defined based on the IoU.

TP represents instances that are correctly identified and localized by the model, and the IoU score between the predicted boundary box and ground

truth box is higher than or equal to the predefined threshold. The threshold depends on specific applications. A low threshold is more flexible, while values close to 1 are more restrictive, demanding almost perfect overlap between predicted and true boundary box. Commonly used thresholds are:

- 0.25 - when a task is recall sensitive, such as medical images. Although due to a lower threshold value, even a small overlap between the predicted and ground truth boundary box will be considered as a correct prediction. This can lead to more FP and lower precision. However, in a medical context, this is desirable since FN is much riskier,
- 0.50 - when moderate localization precision is required, and
- 0.75 - when precise localization is crucial, such as in autonomous driving.

FP are instances that the model incorrectly identifies as an object that does not exist in the ground truth, or the IoU score is below the threshold.

FN represents instances where the model fails to detect an object that is presented in the ground truth.

Average Precision (*AP*) measures per-class performance and then averages over all classes (also known as **mean Average Precision (*mAP*)**). *mAP* measures the accuracy of object identification and classification. It can be used to compare different models or the different setups of the same model. For a dataset that contains the *K* classes, the *mAP* at an *IoU* threshold *t* is defined as

$$mAP = \frac{1}{K}\sum_{i=1}^{K} AP_i.$$

The *AP* for a specific class i represents the area under the precision-recall curve. To create the precision-recall curve General steps to calculate the *AP* includes: for each detected object calculate *IoU* with ground-truth objects, match objects if $IoU \geq t$ for each class, sort predictions by confidence score from highest to the lowest, forming the precision-recall pairs as threshold changes, employ interpolation methods to gain more detailed analysis of precision-recall behavior and calculate the area under the interpolated curve. Most often, 11-point interpolation, which uses 11 equally spaced recall levels (r) between 0 and 1 [0.1, 0.2, …, 0.9, 1] and calculates interpolated precision as the maximum precision for any *Recall>r*, or all point interpolation is used.

mAP uses both precision and recall, providing a balanced measure of the model's performance. Moreover, it is non-threshold dependent, providing a more comprehensive assessment. It ranges from 0 to 1. The values close to 1 indicate a reliable model that has a low number of FN and FP.

Average Recall represents the model's ability to successfully detect all ground-truth objects across different thresholds. General steps include: computation of IoU for each detected box, matching detection with ground truth if IoU is higher than a threshold, computing the recall, averaging the recall over various IoU thresholds, averaging over different maximum numbers of detected objects per image, and aggregating across all images in the dataset.

10 REGRESSION

In the following, we will introduce standard linear regression, in which we assume that the stochastic component of the model—the errors—is independent and identically distributed with constant variance (homoscedasticity). This implies that each observation is independent and equally precise, with no correlation between errors and the same level of uncertainty across all data points. The theory and properties described here hold under these assumptions.

The regression is a statistical technique used to assess the relationship between two or more variables. In the machine learning context, regression attempts to model the relationship between input variables (the independent) and labels (the target variable). Regression can be used for prediction, estimation, hypothesis testing, and modeling relationships.

Let y denote the output variable, which depends on several independent variables denoted by x. In the regression, it is assumed that the model (i.e., mathematical function) maps the input features (x) to the output (y), by using some parameters β in the following form

$$y = f(x, \beta)$$

The function y is called the regression function. The machine learning algorithm optimizes the set of unknown parameters such that the approximation error is minimized, i.e., the difference between predicted and true values given in the training set is minimal. In regression, the output variable is written as a function of independent variables, i.e., y represents the sum of a function of the input variable $f(x)$ and random errors ε i.e.

$$y = f(x) + \varepsilon.$$

The function $f(x)$ is unknown and it is approximated by an estimator $g(x, \theta)$ containing a set of parameters β. It is assumed that errors are random and follow a normal distribution with a mean of 0. The model can be expressed in a formula

$$\hat{y}_i = \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \varepsilon_i$$

where $\beta_j;\ j = 1,\ldots,p$ is a vector of unknown parameters (also called weights) which characterize the role and contribution of independent variables ($x_{ij};\ i = 1,\ldots,n$). The ε_i is the error vector, n is the number of observations, and p is the number of independent variables. The value of the desired θ minimize the following expression

$$E(\theta) = \sum_{i=1}^{n} \big(y_i - g(x_i, \beta)\big)^2$$

where β is a vector of p parameters $\beta_1,\ldots,\beta_P$. There are various types of models that can be used for regression. Those models are mostly categorized by using the following aspects:

- Number and types of independent variables - when there is only one independent variable, the model is known as a simple regression model, while multiple regression models involve more independent variables, and
- The shape of the regression line - linear regression fits a straight line while polynomial regression fits a polynomial equation to represent a non-linear relationship between input and output.

10.1 Linear regression

Simple linear regression contains only one independent variable. It defines the relationship between input and output by using the straight line (**Figure 73**) defined by

$$y = \beta_0 + \beta_1 x + \varepsilon$$

where β_0 represents an intercept term and β_1 represent the slope of the fitted line (i.e. $p = 2$).

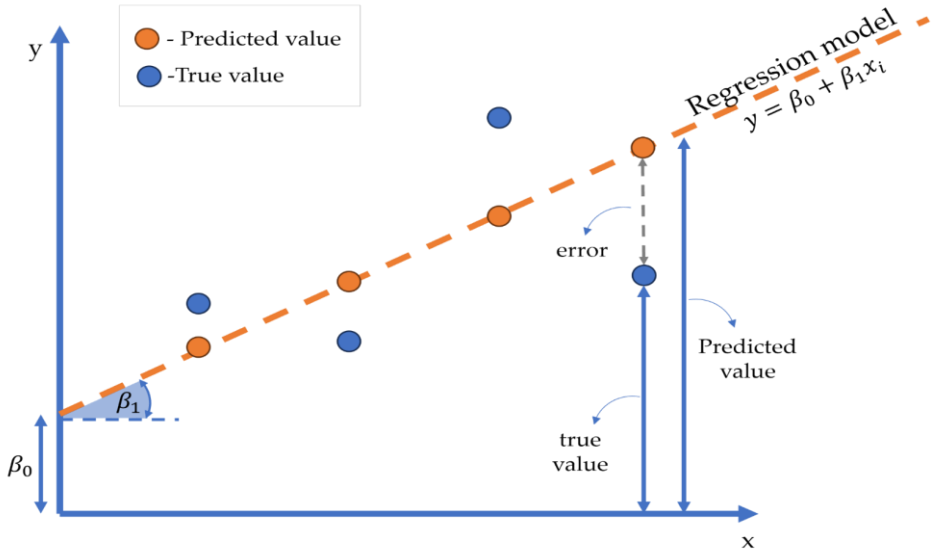


Figure 73 Linear regression model

10.2 Simple Linear Regression

To determine the optimal regression coefficients, the Ordinary Least Squares (OLS) method is used. In OLS, the intercept and slope are optimized to minimize the sum of the squares of the vertical distances between predicted and actual values, i.e.

$$E = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i)^2$$

The values can be determined by differentiating the loss with respect to each parameter and setting it equal to zero

$$\frac{\partial E}{\partial \beta_0} = -2 \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i) = 0 \text{ and}$$

$$\frac{\partial E}{\partial \beta_1} = -2 \sum_{i=1}^{n} x_i (y_i - \beta_0 - \beta_1 x_i) = 0$$

Thus β_0 and β_1 are the solution of the system of two equations

$$\sum_{i=1}^{n} y_i - n\beta_0 - \beta_1 \sum_{i=1}^{n} x_i = 0 \text{ and}$$

$$\sum_{i=1}^{n} x_i y_i - \beta_0 \sum_{i=1}^{n} x_i - \beta_1 \sum_{i=1}^{n} x_i^2 = 0$$

the means of x and y are given by

$\bar{x} = \frac{1}{n}\sum x_i$ and $\bar{y} = \frac{1}{n}\Sigma y_i$ and variance of x is given by

$Var(x) = \frac{1}{n-1}\sum(x_i - \bar{x})^2$ while the covariance of x and y is defined as

$$Cov(x, y) = \frac{1}{n-1}\sum(x_i - \bar{x})(y_i - \bar{y})$$

Then the values of β_0 and β_1 can be calculated using the following equations

$$\beta_1 = \frac{Cov(x, y)}{Var(x)}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

Example: The values of the crop coefficient (K_c) and corresponding NDVI values are provided in Table. Estimate the K_c in the plot if the NDVI value is equal to 0.82.

Plot	1	2	3	4	5	6	7
K_c	0.479	0.552	0.540	0.643	0.745	0.830	1.027
NDVI	0.730	0.739	0.760	0.767	0.786	0.798	0.845

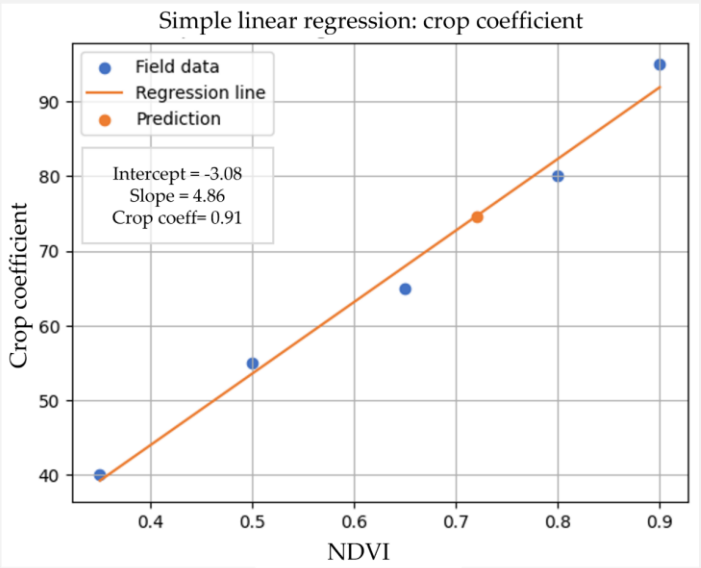
The first step is to examine whether a linear relationship exists between independent and dependent variables either through visual inspection of scatter plots or by applying statistical tests. As figures show, they lie approximately along a straight line, indicating that the crop coefficient tends to increase as NDVI increases. This suggests a clear linear relationship between variables. The next step is to fit the regression model.

n=7

$\bar{x} = 0.775$, $\bar{y} = 0.688$, $Var(x) = 0.001$, $Cov(x, y) = 0.006$

$$\beta_1 = \frac{18.85}{0.197} = 4.86, \beta_0 = 0.688 - 4.86 \cdot 0.775 = -3.08$$

$$Biomass = 4.86 \cdot NDVI - 3.08 = 4.86 \cdot 0.82 - 3.08 \approx 0.91$$



The Simple Linear regression model is based on a few assumptions that need to be fulfilled in order model to provide valuable results:

1. Linearity - the relationship between the independent and target value needs to be linear. If the relationship is non-linear, the model will show a poor performance.,
2. The errors are uncorrelated - there should not be correlation or patterns between errors.,
3. The independent variables x_i are exactly known - if $x_i$values contain measurement errors, the estimated regression coefficient β will be biased,
4. Errors should follow a normal distribution, and
5. Errors should have constant variances (homoscedasticity) across all values of the independent variable. If homoscedasticity is not met, the method of least squares becomes imprecise.

Assumptions 2, 3, 4, and 5 can be checked by using statistical tests. To test whether a linear relationship exists between independent and dependent

variables, the *t-test* or *F-test* can be used. The t-test examines an individual coefficient, i.e.

$$H_0: \beta_1 = 0 \text{ versus}$$

$$H_1: \beta_1 \neq 0$$

So, the *t-test* checks whether the slope is significantly different from zero. If the p-value is below the chosen significance level (usually 0.05), the H_0 is rejected, the is rejected and the relationship is linear. In contrast to the *t-test*, which is limited to two variables, the ANOVA test can handle multiple variables. It tests the whole model, i.e., it checks whether all slopes together in multiple regression are zero:

$$H_0: \beta_1 = \beta_2 = \ldots = \beta_P = 0 \text{ vs}$$

$$H_1: At\ least\ one\ \beta_j \neq 0$$

The H_0 states that all regression coefficients are 0, meaning that there is no predictive relationship between the x and y variables. On the other hand, the H_1 claims that at least one of the regression coefficients is not 0, i.e., there is at least one independent variable that affects y. The ANOVA uses the *F-test* that is define:

$$F = \frac{\sum_{i=1}^{n} \left(\widehat{y_i} - \underline{y}\right)^2}{\sum_{i=1}^{n} (y_i - \widehat{y_i})^2} \cdot \frac{n - N - 1}{N} = \frac{SSR}{SSE} \cdot \frac{n - N - 1}{N}$$

where the *SSR* is the regression sum of squares, *SSE* is the sum of square errors, *n* is the total number of measurements, *N* is the number of independent variables, $\widehat{y_i}$ is the predicted value for observation *i*, $\underline{y}$ is the mean of the observed dependent variable. If the computed F value is larger than the *F*-statistic for the desired level of significance, the null hypothesis is rejected, meaning that the variables (all together) have a significant linear relationship with the dependent variables.

10.3 Multiple linear regression

A multiple linear regression is an extension of simple linear regression. The model defines the relation between N independent variables and the target variable by using the following equation:

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_N x_N$$

The optimal values of the parameters $\beta_0, \beta_1, \ldots, \beta_N$ is estimated by using the ordinary least squares method. The matrix notation of the above model is

$$Y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_i \end{bmatrix}, X = \begin{bmatrix} 1 & x_{11} & \ldots & x_{N1} \\ 1 & x_{12} & \ldots & x_{N2} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{1i} & \ldots & x_{Ni} \end{bmatrix}, \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_N \end{bmatrix}$$

The simple least squares estimation of regression coefficients can be defined as

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Example: Estimate the soil moisture based on multiple spectral bands. The results of field measurement of soil moisture and spectral reflectance for corresponding bands are available in Table

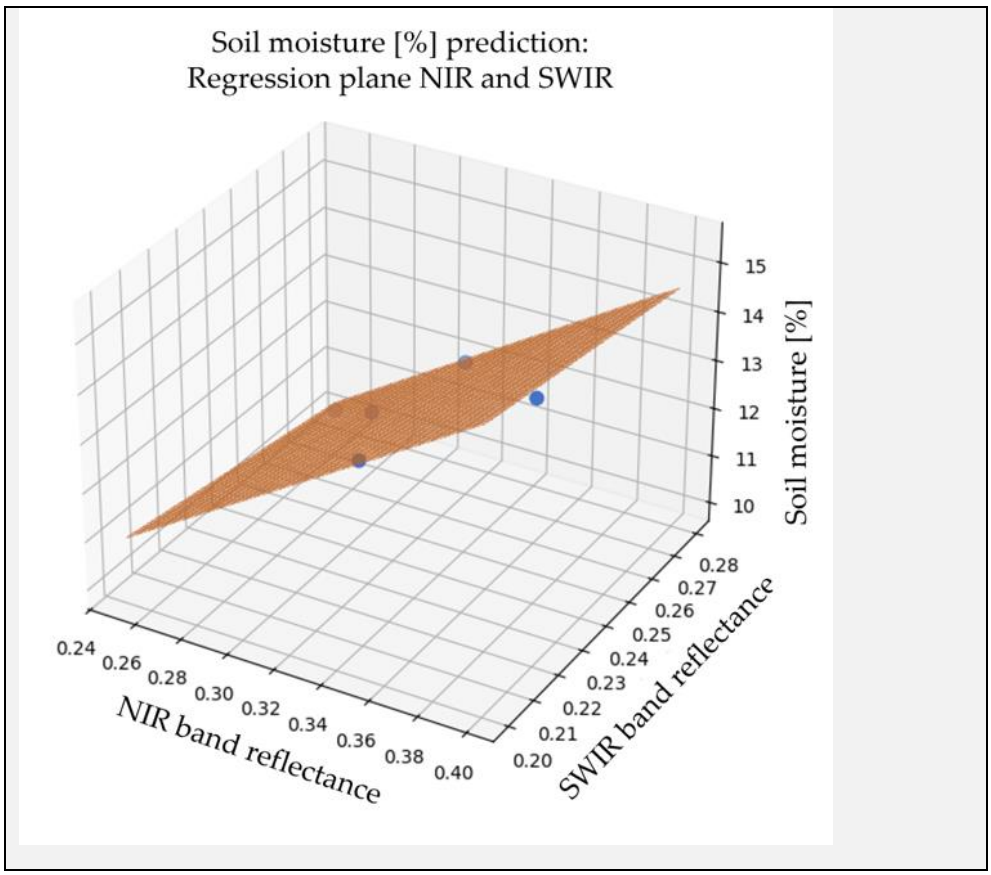
Sample	1	2	3	4	5
NIR	0.30	0.40	0.35	0.25	0.33
SWIR	0.25	0.22	0.20	0.28	0.26
Soil moisture [%]	12	15	14	10	13

Using the samples, fit the multiple linear regression model to estimate fore regression coefficients.

$\beta_0 = 5.65, \beta_{NIR} = 29.44, \ \beta_{SWIR} = -10.13$

The final mode is $y = \beta_0 + \beta_{NIR} x_{NIR} + \beta_{SWIR} x_{SWIR} = 5.65 + 29.44 \cdot x_{NIR} - 10.13 \cdot x_{SWIR}$

In multiple linear regression, the regression model is a hyperplane in a space with dimension *N+1*. In this case, the regression model is a plane in 3D space.



10.4 Polynomial regression

Polynomial regression is used for modeling non-linear relationships between independent and target variables by using the N^{th}-degree polynomial (the highest exponent in the polynomial). The polynomial regression model can be represented as follows

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_N x^N$$

Although the highest order of polynomials that can be fit with n data points is n-1. However, the polynomial curve will pass through all data points, providing a perfect fit with the training data and low generalization ability. This is interpolation not regression. Moreover, each new term (x^2, x^3, x^4) adds a coefficient, and more data is needed. Additionally, from a mathematical perspective, the hierarchy principle needs to be followed since only hierarchical models are invariant under the linear transformations. The model

is hierarchical if it contains all lower-order terms (i.e., if the highest order of the polynomial function is x^4 the model should include the x, x^2, x^3 in a hierarchy). In practice, the degree of the polynomial model is kept as low as possible. The degree higher than 4 is rarely used.

The optimal value of regression parameters is determined by applying ordinary least squares by minimizing the sum of squared errors

$$E = \sum_{i=1}^{n} \left[y_i - \left(\beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_N x^N \right) \right]^2$$

By differentiating the loss with respect to each parameter and setting it equal to zero

$$\frac{\partial E}{\partial \beta_i} = 0, \forall i = 0, 1, \ldots, N$$

the system of *N+1* linear equations is created

$$\sum y_i = \beta_0 n + \beta_1 \left(\sum x_i \right) + \ldots + \beta_N \left(\sum x_i^N \right)$$

$$\sum y_i x_i = \beta_0 \left(\sum x_i^2 \right) + \beta_1 \left(\sum x_i \right) + \ldots + \beta_N \left(\sum x_i^{N+1} \right)$$

$$\sum y_i x_i^2 = \beta_0 \left(\sum x_i^2 \right) + \beta_1 \left(\sum x_i^3 \right) + \ldots + \beta_N \left(\sum x_i^{N+2} \right) \ldots$$

$$\sum y_i x_i^N = \beta_0 \left(\sum x_i^N \right) + \beta_1 \left(\sum x_i^{N+1} \right) + \ldots + \beta_N \left(\sum x_i^{2N} \right)$$

The matrix representation of a linear system $\vec{y} = X\vec{\beta}$, where

$$\vec{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_i \end{bmatrix}, X = \begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^N \\ 1 & x_2 & x_2^2 & \ldots & x_2^N \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_i & x_i^2 & \ldots & x_i^N \end{bmatrix}, \vec{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_N \end{bmatrix} \text{ than}$$

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

where $\hat{\beta}$ is an unbiased estimation of β. The assumptions of the multiple regression model are similar to the simple linear regression, i.e.:

- The errors follow the normal distribution with a mean of zero and a standard deviation σ. The errors are uncorrelated with each other and independent of the errors associated with all other observations.

- The independent variables x are assumed to be measured correctly.

There are several strategies that can be used to build a polynomial regression model. The forward selection procedure successively fits the model with increasing order to test the significance of the regression coefficient at each step of model fitting. The order increases until the t-test for the highest order term is nonsignificant. On the other hand, the backward elimination starts with the highest order model and then deletes the highest terms one at a time.

Example. The values of the field measurement of biomass and NDVI are presented in Table. Build the biomass estimation model.

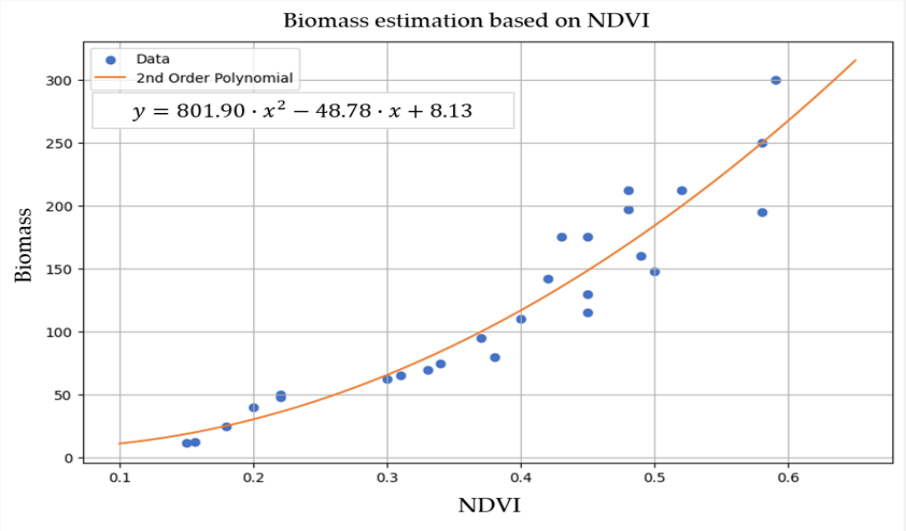
NDVI	Biomass	NDVI	Biomass	NDVI	Biomass
0.15	12.0	0.38	80.0	0.58	195.0
0.16	12.5	0.37	95.0	0.48	197.0
0.18	25.0	0.40	110.0	0.48	212.0
0.20	40.0	0.45	115.0	0.52	212.0
0.22	48.0	0.45	130.0	0.58	250.0
0.22	50.0	0.42	142.0	0.59	300.0
0.30	62.5	0.50	148.0		
0.31	65.0	0.49	160.0		
0.33	70.0	0.45	175.0		
0.34	75.0	0.43	175.0		

Using the measurements, fit the polynomial regression model to estimate three regression coefficients.

$\beta_0 = 8.13, \beta_1 = -48.78, \beta_3 = 801.90$

The final mode is $y = 801.90 \cdot x^2 - 48.78 \cdot x + 8.13$

The model is shown in the figure below.



10.5 Polynomial piecewise

In some situations, low-order polynomials don't provide a good fit. One possible solution is to use the higher-order polynomial. However, the use of a single high-degree polynomial may produce large errors if the function has different behavior in different regions of the independent variables. This type of problem can be solved by piecewise polynomials, where instead of using one global polynomial function that fits the entire data range, the range is split into sections and separate polynomials are fitted to each section. So, piecewise regression includes the two phases: divide the domain of independent variables into pieces and fit a polynomial function separately for each region. The join points of sections are called knots. However, fitted functions are not continuous. To force the continuity, the restriction on the parameter estimation is introduced, i.e., for the polynomial of order k, the function values and derivatives up to $k-1$ are equal at each knot. The piecewise polynomials with continuity constraints are called splines. To create a spline, it is necessary to determine the knots and to select the order of the polynomial.

The cubic spline, which uses the cubic function within each region, is often used. The cubic spline with K knots ($t_1 < t_2 < \ldots < t_k$) is defined by

$$E(y) = \sum_{j=0}^{1} \beta_{0j} x^j + \sum_{i=1}^{K} \beta_i (x - t_i)_+^3$$

where

$$(x - t_i)_+^3 = \{(x - t_i)^3, \;\; x > t_i \; 0, \qquad\qquad x \leq t_i$$

The four parameters are needed to describe each region, leading to a total $4 \cdot K$ degrees of freedom. It assumes that the position of knots is known.

The number and position of the knots have a significant influence on the fit. If a small number of knots are used, the regression is underfitted, and with too many knots, the regression is overfitted. Similarly, knot position is also important since uniformly distributed knots can lead to overfitting in regions with a low number of points or underfitting in regions with a high number of points. The position of knots can be determined by using equidistant knots, quantile-based knots, domain knowledge, and visual inspection of plots. On the other hand, the number of knots is determined by using a penalty approach such as the *B*-spline, smoothing spline, *P*-spline, or regularized spline.

The smoothing spline presents the solution of the minimization problem

$$arg\ min \sum_{i=1}^{n} \big(y_i - f(x_i)\big)^2 + \lambda \int_0^1 f^{''}(x)^2 dx$$

the first part represents the goodness of fit of f while the second term represents a penalty for the roughness of the function. The smoothing spline starts by putting the knot at each data point, and the overfitting is controlled by the penalty on the integral of the squared second-order derivative (second term).

10.6 Model building

A simple strategy for building of regression model consists of five basic steps:

1. Data collection and preparation,
2. Preliminary model investigation,
3. Reduction of independent variables,
4. Model refinement and selection, and
5. Model validation.

The data collection includes the field measurement of variables (forest biomass, water quality parameters, air quality measurements, information about classes, etc.) and integrating them with corresponding remote sensing data. It usually includes steps to verify that the assumptions of regression analysis are met, such as creating scatter plots, checking data distribution, identifying of outliers, encoding of categorical variables, normalizing data , etc. Preliminary model investigation includes identification of functional form for predictor variables based on statistics' prior knowledge or state-of-the-art studies to perform data transformation (such as logs).

The regression analysis depends on the independent variables that are present in the model.

Usually, a large set of potentially explanatory independent variables is available. However, some variables may not be fundamental for the problem, some may contain errors, or represent the duplication of another variable. Therefore, it is crucial to detect and use the variables that play a consistent role.

The aim of reduction is to select a subset of variables that are significant for the outcome to improve model accuracy and interpretability. The variable selection is performed assuming that the functional form of the independent variable ($x^2, \frac{1}{x}, log(x)$) is known, and that data contains no outliers or influential observations. Many methods have been proposed for the selection of suitable variables in regression, such as forward test-based, criterion-based, or screening-based procedures. Test-based methods, such as stepwise or autometrics, rely on statistical tests to select informative variables.

The stepwise procedure is the simplest and most straightforward approach to variable selection, in which forward selection or backward elimination is performed using the F-value as a criterion.

The F-value (or F-statistic) is a number to test whether a group of variables (or a single variable in the partial case) significantly improves the model. It compares two sources of variation: the variation explained by the model and the unexplained variation, contained into the residuals. A high F-value indicates that the variable(s) being tested contribute significantly to explaining the outcome, while a low F-value suggests they do not.

In forward selection, variables are added one by one to the model. At each step, the partial F-value is calculated to assess if that variable significantly improves the model. The process is repeated until the F-statistics are significant for a given significance level (i.e. the remaining variable does not improve the model significantly). The backward elimination starts with all variables (full model). At each step, the variable with the lowest F-value in the comparative test is removed. The process continues until all variables are significant. In practice, the stepwise selection combines both the forward (adding the best variable) and the backward (checking if any variable should be removed) selection procedures. It is intuitive and easy to understand but, it is poorly data-driven and can be unstable if variables are highly correlated.

Autometric represents a robust automated general-to-specific model selection procedure that starts with a general model and systematically eliminates insignificant variables using backward elimination and diagnostic tests.

If there are N independent variables, then it is possible to create 2^N models. In a criteria-based procedure, the best model is selected according to certain criteria. The the Mallows' CP, the Adjusted R^2Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) are most commonly used.

The Mallows' C_p statistics is an estimation of the total mean squared prediction error (bias+variance) of fitted models, which is averaged over the independent variables. It is used to compare models with different numbers of parameters. In addition to measuring goodness of fit, it explicitly considers bias (since leaving out important variables can lead to a biased model), which helps to detect the underfitting and introduces a penalty for adding unnecessary variables, discouraging overfitting. C_p is given by

$$C_p = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sigma^2} - n + 2N$$

where σ^2 is external estimation of the variance calculated for the full model (using all variables) or from prior knowledge of the measurement errors. For the full model C_p is exactly equal to N. The models with low bias and appropriate complexity have small C_p value or value close to N. If the C_p value is much greater than the number of independent variables; the bias is substantial.

The AIC the model that balances the goodness of fit with model size, i.e., it penalizes a model for adding variables that do not significantly improve the model performance. The primary aim is to find a model that best explains the dependent variable with a minimum number of independent variables. The suitability of the model is measured by maximizing the log likelihood of the predictor coefficient and error variance, i.e.

$$AIC(\alpha) = -2logL \; + \; \alpha \cdot N$$

where L is the likelihood, and α is constant in the penalty term, typically set to 2. AIC decreases with an increase in model performance, i.e., the model that minimizes AIC should be chosen. AIC is efficient when the sample size n is large relative to the number of variables N; in contrast, it can lead to overfitting, favoring the complex models. [9] suggested the bias-corrected version AIC_C by calculating the Kullback–Leibler information for normal distributions, assuming the true model is among the candidate models. It is defined as

$$AIC_C = AIC + \frac{2N(N+1)}{n - N - 1}$$

it is clear that for a large sample size (n) the AIC_C will converge to AIC.

Similar to AIC, BIC is a criterion used for model selection from a finite set of models. It combines the goodness of fit with a penalty for model complexity. The BIC is defined as

$$BIC = -2logL + nlogN$$

In contrast to *AIC*, which has a fixed penalty, in BIC, the penalty grows with the number of variables. Due to that, for a large sample size, the BIC favors the simpler model, while for a small n, it will pick the model with a similar

level of complexity as *AIC*. The model with the lowest *BIC* is considered to have the best balance from a Bayesian perspective.

10.7 Linear classifier

As already mentioned, the goal of classification is to assign the input vector x to one of the discrete classes y. Consider independent and identically distributed data $(x_1, y_1), \ldots, (x_n, y_n) \sim P$ where $x_i \in R$ are features and $y_i \in \{0, 1, \ldots, K-1\}$. The discriminant is a function $h: \chi \to \{0, 1, \ldots, K-1\}$ that takes the feature vector and assigns it to the one class. Since x can belong to one and only one class, the input space χ is disjointed into K class labelled decision regions. The border of each region is a decision boundary that represents the surface that separates different classes in classification models.

The **Linear Discriminant Analysis** (LDA) is a supervised algorithm used for both dimensionality reduction and classification. Let's consider the binary classification problem. In LDA, we assume that: discriminant functions are linear, features of each class follow a Gaussian distribution, and all classes have equal covariance matrices.

The simplest linear discriminant function is obtained by taking the linear function of the input vector as follows

$$y_i = w^T x_i + b$$

where w is the weight vector and b is the bias. The negative of the bias is also known as the threshold. A feature vector x belongs to a positive class if its discriminant function is $w^T x \geq -b$. Otherwise, it belongs to the negative class. Geometrically, the weight vector determines the orientation of the decision boundary, while the threshold b determines where along the decision boundary the split between classes occurs. The weight vector can be adjusted using least square methods, Fisher criterion, or perceptron.

The **least squares** classifier fits a linear model by minimizing the squared errors between the true class y and the predicted class $\hat{y}$ i.e. $\hat{w} = argmin_{\hat{w}}(y - w^T x)^2$. The squared error is a convex function and has a unique and simple closed-form solution. It guarantees to achieve a global minimum; however, this is not necessarily the best solution. For example, in the presence

of outliers, the least squares method tends to shift the decision boundary toward the outliers.

The Fisher linear discriminant represents the simplest linear discriminant function that projects the *D-dimensional* feature vector x down to one dimension (for binary classification) using $y(x) = w^T x$ (**Figure 74**).

The corresponding decision boundary is defined by the relation $y(x) == 0$ which corresponds to a *D-1* dimensional hyperplane (in 2D it is a line, in 3D it is a plane …) and each subspace represents a class ($+1$ or -1). So, an input vector x is assigned to the class +1 if $y(x) > 0$, and to -1 otherwise.

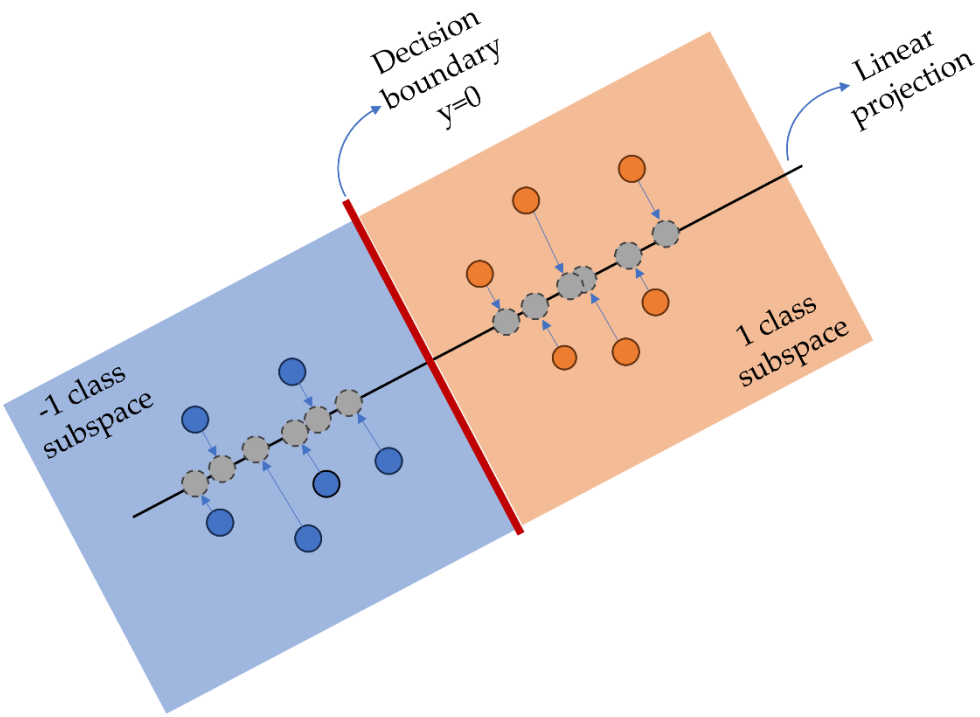


Figure 74 FLD finds a linear projection of data and classifies the projected values by checking against the threshold

The projection onto one dimension can lead to sustainable data loss, so classes that are well separated in the original *D* dimension can significantly overlap in one dimension. Therefore, we should select the weight vector that maximizes the class separation. Geometrically, the separation between classes is maximized if the distance between their centroids is larger and the scatter within classes is smaller. So, a hyperplane is created based on simultaneously

maximizing the between-class variance and minimizing the within-class scatter, i.e., by maximizing the Fisher criterion. The Fisher criterion is defined as

$$J(w) = \frac{(\mu_{-1} - \mu_1)^2}{s_{-1}^2 + s_1^2}$$

where $\mu_i = w^T \frac{1}{n} \sum_{i=1}^{n} x_n$ are the mean values of two classes after the projection along w i.e.

$y_i = w^T x_i$ and $s_k^2 = \sum_{i=1}^{n} (y_i - \mu_i)^2$ is the *kth* within-class variance.

So Fisher's criteria try to find the linear combination of parameters w that maximizes the between-class variance (S_B) relative to the within-class variance (S_w). The w is determined by setting the derivative of J to 0, i.e.

$$\frac{\partial J}{\partial w} = 0 \Rightarrow \frac{\partial \left(\frac{w^T S_B w}{w^T S_w w} \right)}{\partial w} = (w^T S_w w) S_B w - (w^T S_B w) S_w w = 0 \Rightarrow S_w^{-1} S_B w = J(w)$$

Therefore, the projection vector w is the eigenvector of $S_w^{-1} S_B$ so we need to choose the eigenvector that corresponds to the maximum eigenvalue to maximize class separability. Geometrically, in order to divide the feature space into k different classes, at most k-1 equations are needed. Due to that, the number of created components is equal to the number of classes - 1.

Similar to PCA, the Fisher Linear Discriminant can be used for dimensionality reduction. As already mentioned, the PCA finds the most accurate data representation in a lower-dimensional space (it projects the data in the direction of maximum variance). The direction of maximum variance may be useless for classification features. On the other hand, in LDA, features are reduced by projecting data onto directions that maximize class separability.

The linear classification can be used for multi-class classification problems. LDA is simple to implement, especially for binary classification, easy to interpret, and provides good accuracy in the classification of linearly separable data.

In contrast, there are a few drawbacks, such as: insufficient robustness against outliers and small sample size (high number of features and low number of samples), inapplicable for multi-model (more than one mode, i.e., more than

one distinct peak suggesting different subgroups within class) datasets, and the singularity of the within-class scatter matrix.

Algorithm Fisher Linear Discriminant

Input: dataset D with (x_i, y_i) elements $i = 1, \ldots, n$ and $y_i \in \{1,2\}$
Return: weight vector w, threshold $\boldsymbol{b}$, prediction $\hat{y}$
1. Compute the class mean as $\mu_1 = mean(x_i, y_i = 1)$ and $\mu_{-1} = mean(x_i, y_i = -1)$
2. Compute within-class variance

$$S_w = 0$$

for i in $range(0, len(D) - 1)$:
    if $y_i = 1$:
        $S_w += (x_i - \mu_1)(x_i - \mu_1)^T$
    else:
        $S_w += (x_i - \mu_2)(x_i - \mu_2)^T$
3. Compute projection vector $w = S_w^{-1} \cdot (\mu_1 - \mu_2)$
4. Normalize projection vector $w = \frac{w}{||w||}$
5. Compute threshold $b = \frac{w^T \mu_1 + w^T \mu_2}{2}$
6. Make a prediction
if $w^T x \geq b$:
  $\hat{y} = 1$
else:
    $\hat{y} = -1$

10.8 Logistic regression

Logistic regression represents the baseline supervised ML tool for classification and the foundation of neural networks. The classification problem is similar to the regression model but the output is a discrete value. The linear regression can be generalized to the classification problem by defining a different family of probability distributions. Let's consider a binary classification. The goal of binary classification is to train a classifier that can make a binary decision on new input data.

Consider a single input observation x represented by N independent variables $x = \{x_1, \ldots, x_N\}$ and a predicting outcome is categorical variable y that can be 1 (positive class) or 0 (negative class). We want to know the probability

$p(y = 1|x)$ that x is a member of the class. In the case of binary classification, the probability of one class defines the probability of the second class since their sum must be equal to 1. Therefore, if p is the probability of the positive class, then *1-p* is the probability of the negative class, i.e.

$$p(y|x;\beta) = \begin{cases} p(x) & if\ y = 1 \\ 1 - p(x) & otherwise \end{cases}$$

The ration $p/(1-p)$ is known as odds, and logit is the logarithm of odds, i.e.

$$y = logitp = ln\frac{p}{1-p}$$

We assume that the relationship between input and output is linear, i.e.

$$y = \beta_0 + \sum_{i=1}^{N} \beta_i x_i = \beta_0 + \beta^T x$$

where $\beta = \{\beta_1, \ldots, \beta_N\}$ represents regression coefficients that reflect the strength of the relationship between independent variables and outcome. However, in contrast to normal distribution in linear regression, which is parameterised by mean, the distribution of binary variables is binomial (the output must be between 0 and 1). The linear function is unbounded, and nothing forces y to be between 0 and 1. This is solved by passing y through a logistic sigmoid function that squashes the output of the linear function into the interval 0 to 1 and interprets these values as a probability $p(y = 1|x;\beta) = \sigma(y)$ i.e.

$$p(y = 1) = \sigma(y) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \ldots + \beta_N x_N)}}$$

The sigmoid function is also called the logistic function, and therefore, this regression is also known as logistic regression. This creates the logit

$$ln\frac{p(x)}{1-p(x)} = \beta_0 + \sum_{i=1}^{N} \beta_i x_i$$

The regression coefficients are estimated by maximum likelihood. For one instance $(x_i,\ y_i)$ the probability can be written as

$$p(y|x;\beta) = p(x)^y (1 - p(x))^{1-y}.$$

For n training data points $\{(x_i, y_i)\}_{i=1}^{n}$ that were generated independently, the likelihood is defined as

$$L(\beta) = \prod_{i=1}^{n} p(y_i|x_i;\beta) = \prod_{i=1}^{n} p(x_i)^{y_i}\big(1 - p(x_i)\big)^{1-y_i}$$

The aim is to select a coefficient that will predict a high probability of positive class samples and a low probability for negative class samples. We can define the loss function by taking the negative logarithm of the likelihood, which gives the cross-entropy loss function given as

$$l(\beta) = -logL(\beta) = -\sum_{i=1}^{n}\big[y_i logp(x_i) + (1 - y_i)log\big(1 - p(x_i)\big)\big]$$

The gradient of the loss function with respect to β_j is given by

$$\boldsymbol{\nabla}_{\beta} l(\beta) = \frac{\partial l}{\partial \beta_j} = \sum_{i=1}^{n}\big(y_i - p(x_i)\big)x_{ij}$$

The updates will be given by $\beta_{t+1} := \beta_t - \eta \boldsymbol{\nabla}_{\beta} l(\beta_t)$

Another way to solve the maximum likelihood equation is by using the Newton-Raphson approach. So we want to maximize the log-likelihood $l(\beta)$. The maximum occurs when the gradient is zero $\boldsymbol{\nabla}_{\beta} l(\beta) = 0$. The Newton-Raphson approaches solve this equation by iteratively updating β using both the gradient and the Hessian, i.e.

$$\beta_{t+1} = \beta_t - H(\beta_t)^{-1}\boldsymbol{\nabla}_{\beta} l(\beta_t)$$

where H is Hessian. The Newton-Raphson approach automatically rescales the gradient by the curvature, so it does not require manual tuning of the learning rate, enabling faster convergence. It is a standard procedure used in logistic regression.

So the outcome of the logistic regression model is a probability, and we want to classify new points as 1 or 0 by checking which of those classes has a higher probability. If $p(y = 1|x) \geq p(y = 0|x)$ then the new sample is classified as 1, otherwise it is classified as 0. This is the same as using the prediction threshold $t > 0.5\ i.e. h_{\beta} > 0.5$. Mathematically, the probability threshold 0.5 corresponds to $\frac{1}{1+e^{-z}} = 0.5 \Rightarrow z = 0$ i.e., input to sigmoid is 0.

Therefore, if the input to the sigmoid function is negative, logistic regression predicts a negative class; otherwise, it predicts a positive class. However, the classification thresholds should be adjusted in case of imbalanced data sets or cost-sensitive classification. If the positive class is rare, the lower threshold may increase sensitivity. Consequently, if the cost of predicting a false positive class is much higher than predicting a false negative class, a threshold greater than 0.5 should be used to reduce the number of false positives.

Logistic regression can be seen as the simplest form of a feedforward neural network that consists of one neuron and a sigmoid activation function. More complex neural networks extend this concept by stacking multiple layers of neurons to model nonlinear relationships.

Despite the name, logistic regression is a simple and effective classification method. It achieves high accuracy if the classes are linearly separable. Logistic regression is useful for identifying the most discriminative variables in a dataset where there are many variables to consider. It is less robust than more sophisticated models such as ANN, but it is easier to interpret the outputs and understand how decisions are made.

10.8.1 Multi-class logistic regression

Multi-class logistic regression, also known as multinomial logistic, is a generalization of standard logistic regression for classification that involves more than two classes. So the output variable *y* is still discrete, but now it can take the *K* different values (classes), so $y \in \{1, \ldots, K\}$. The classes are represented as a one-hot encoder vector. The multinomial problem is parameterized by the *K-1* parameter by fixing one class to the referent, and the model estimates the logits of each other class relative to the referent. So for each observation *x* we will output a *K*-dimensional vector representing the estimated probabilities for each class, i.e. $p(x) = (p_1(x), \ldots, p_K(x))$ with $p_k(x) = p(y = k|x)$ and $\sum_{k=1}^{K} p_x(x) = 1$

In multinomial regression, the sigmoid logistic function is replaced by the softmax function, which is defined as

$$p_k(x_i) = softmax(z_{ik}) = \frac{e^{z_{ik}}}{\sum_{j=1}^{K} e^{z_{ij}}}$$

where $z_{ik} = \beta_k^T x_i + b_k$ is a linear score for sample *i* and class *k*, x_i is the feature vector for data point *i*, and b_k is the offset for class *a*. The softmax regression algorithm (**Figure 75**) applies binary logistic regression to multiple classes at once. It uses a score to compute the probability that the training sample x_i belongs to class *k*.

To determine the regression parameter vector β_k we use the maximum likelihood. The likelihood function is given by

$$L(\beta) = \prod_{i=1}^{n} \prod_{k=1}^{K} p(x_{ik})^{y_{ik}}$$

We define the cost function by taking the negative logarithm, which gives

$$l(\beta, b) = -\sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} log p_k(x_i)$$

which is known as the cross-entropy error function for the multi-class classification problem (also known as categorical cross-entropy). In order to determine the model parameters, the gradient of the cost function with respect to all of the parameter vector β . The derivative of the softmax function for class *k* is given by

$$\nabla_w L(\beta_k, b) = \frac{\partial l}{\partial \beta_k} = \sum_{i=1}^{n} \big(y_{ik} - p_k(x_i)\big) x_i$$

where β_k is the vector of regression coefficients of *x* for the *kth* class of *y*. The gradient descent is used to update the weight with a learning rate η i.e.

$w_k := w_k - \eta \nabla_w L(\beta, b)$ for each class $j = 1, \ldots, K$.

The softmax function will output the estimated probability for each class per sample (p_{ik}) and the class with the highest probability is assigned to the sample, i.e.

$$y_i = argmax_k \, p_{ik}$$

Multinomial logistic regression is harder to interpret because there are several regression coefficients associated with each independent variable. It does not consider statistical independence between features, and it is not suitable for a very large number of classes for learning.

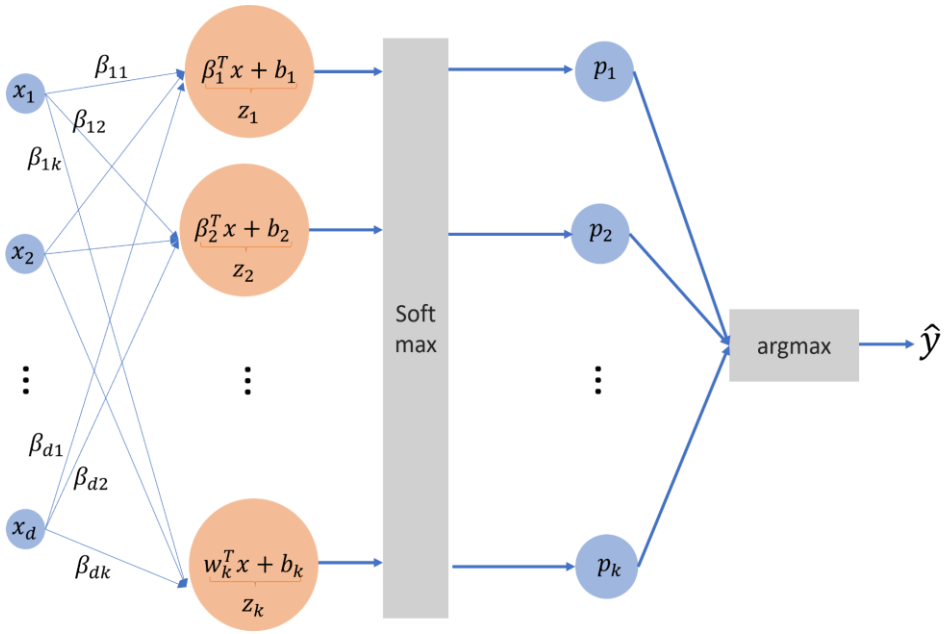


Figure 75 Softmax regression for K classes

11 PROBABILITY BASICS FOR MACHINE LEARNING

Probability theory is one of the central foundations in ML, since the algorithms often rely on probabilistic assumptions of the data. The probability theory represents the mathematical study of uncertainty. The uncertainties rise from both the noise of measurement and the finite size of the dataset.

The probability space is defined by the triple (Ω, F, P) where:

- Ω is a sample space of all possible outcomes. Those possible outcomes need to be distinguishable from each other. They are mutually exclusive, i.e., either one happens or other happens, but not both, and they are collectively exhaustive (no matter what happens, the result will be an element of the sample space). Consider rolling a six-sided die. The sample space represents $\Omega = \{1, 2, 3, 4, 5, 6\}$.
- Event space $F \subseteq 2^{\Omega}$ is the subset of a sample space that represents the collection of all allowed events. For example, we want to get a number greater than 3 in our rolling die experiment. There are 3 numbers greater than 3, so the event space is $F = \{4, 5, 6\}$. And,
- p is the probability assigned to a subset of the sample space (what we believe is likely to happen or not likely to happen). The probability that the event $E \in F$ to a real value between 0 and 1, i.e. $p: F \rightarrow [0, 1]$. The probability characteristics are:
    - It is not negative $p(X) \geq 0$,
    - The probability of the overall sample space is equal to 1 $p(\Omega) = 1$, and
    - If we have two events (two subsets) X and Y that are disjoint, the probability that one or another happens is equal to the sum of their individual probabilities $p(X \cup Y) = p(X) + p(Y)$

Random variables are actually functions that map the outcomes in outcome space to real values. The random variables allow us to provide more uniform treatment of probability theory. The probability of a random variable X taking on the value of x_i is donated by $p(X = x_i)$ or more compact $p(X)$. If the sample

space consists of n possible outcomes, which are equally likely, then the probability of any event X is given by

$$p(X) = \frac{number\ of\ elements\ of\ X}{n}$$

Then, in the above example, the probability of rolling greater than 3 is $\frac{3}{6} = 0.5$.

Consider two random variables X that can take any of the values x_i where $i = 1, \ldots, M$ and Y can take the values y_j where $j = 1, \ldots, L$.

Let N be the total number of trials in which we sample both variables X and Y, and let the number of trials in which $X = x_i$ and $Y = y_j$ is n_{ij}. Moreover, let the number of trials in which $X = x_i$ irrespective of the value of Y is donated by c_i and $Y = y_j$ is r_j. The information of multiple discrete random variables is summarised in the contingency table.

The contingency table (**Table 16**) where n_{ij} represents the number of points in the corresponding cell of the array, and the sum of column i corresponds to $X = x_i$ regardless of X, and the sum of row j represents $Y = y_j$ regardless of X.

Table 16 Contingency table

X/Y	y_1	…	y_L	Row sum r_j
x_1	n_{11}	…	n_{1L}	$r_1 = \sum_{j=1}^{L} n_{1j}$
…	…	…	…	…
x_M	n_{M1}	…	n_{ML}	$r_M = \sum_{j=1}^{L} n_{Mj}$
Col sum c_i	$c_1 = \sum_{i=1}^{M} n_{i1}$	…	$c_L = \sum_{i=1}^{M} n_{iL}$	$N = \sum_{i=1}^{M}\sum_{j=1}^{L} n_{ij}$

The joint probability that $X = x_i$ and $Y = y_j$ happen at the same time is given by $p(X = x_i, Y = y_j) = p(X, Y) = \frac{n_{ij}}{N}$.

Similarly, the probability that X takes the value x_i is given by $p(X) = \frac{c_i}{N}$.

The $p(X)$ can be obtained by marginalizing the $p(X, Y)$ over all possible Y i.e.

$$p(X) = \sum_{j=1}^{L} p(X, Y)$$

which is the sum rule of probability.

Conditional distributions are the crucial tools in probability theory for reasoning about uncertainty since they allow us to update the probability in the face of new events. The conditional probability of $Y = y_j$ given $X = x_i$ is the probability that Y occurs given that X has already occurred. This means that X becomes a new sample space, so the probability that the event Y, X occurs is equal to the probability of $p(Y, X)$ relative to the $p(X)$. The condition distribution is written as $p(Y = y_j | X = x_i)$ or shorter $p(Y|X)$ and it is given by

$$p(Y|X) = \frac{p(Y \cap X)}{p(X)} = \frac{p(Y,X)}{p(X)} = \frac{n_{ij}}{c_i}.$$

Condition distribution is defined when $p(X) > 0$.

Let X be the event that a traveler visited Milano (M), let Y be a traveler that visited Paris (P). We want to determine the probability that users will visit Paris, given that they visited Milano. Let's say we observe the 40 travelers ($N = 40$). The total number of travelers who visited Milano is 6, and the total number of travelers who visited Paris is 9, while 3 travelers visited both cities. So the marginal distribution is $p(M) = \frac{6}{40} = 0.15$ and $p(P) = \frac{9}{40} = 0.225$. Given that the visitor already visited Milan, the conditional probability that he will visit Paris is a subset of the visitors who visited both Paris and Milan; therefore $p(P|M) = \frac{p(P \cap M)}{p(M)} = \frac{3/40}{6/40} = 0.5$. On the other hand, the probability that a visitor will go to Milan, given that they visited Paris, is $p(M|P) = \frac{3}{9} = 0.3333$

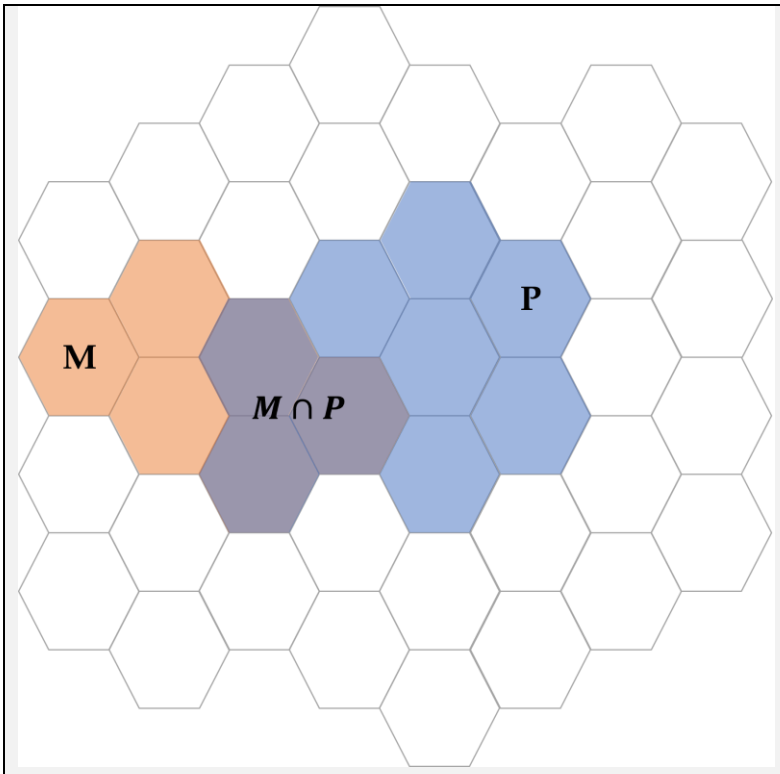


Figure 26 Geometrical representation of conditional probability. Each hexagon represents a visitor.

The definition of conditional probability can be rewritten as

$$p(X,Y) = \frac{n_{ij}}{N} = \frac{n_{ij}}{c_i} \cdot \frac{c_i}{N} = p(Y|X) \cdot p(X)$$

which is known as the chain rule of probability. More generally, for events $X_1, X_2, \ldots, X_n$ the chain rule can be written

$$p(X_1, X_2, \ldots, X_n) = p(X_1)p(X_2|X_1)p(X_3|X_1, X_2) \ldots p(X_n|X_1, X_2 \ldots X_{n-1})$$

The chain rule is used to evaluate the joint probability of some random variables, especially when there is (conditional) independence across variables

From the chain rule (also known as the product rule) of probability and the symmetry property of the joint probability $p(X,Y) = p(Y,X)$ the following relationship can be obtained

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

where $p(Y|X)$ is the posterior probability, $p(X|Y)$ is likelihood, $p(Y)$ is the prior probability and $p(X)$ is the marginal probability of X. This relationship is also known as Bayes' theorem, which plays a crucial role in ML. Intuitively, we can observe Bayes' theorem as updating our prior belief given evidence.

Let's consider a classification of satellite images into water (W) and forest (F) areas, based on the NDWI index. Based on available maps, we know that 20 % of the study area is covered by water, i.e.

$p(W) = 0.2$ and $p(F) = 0.8$

This is our prior knowledge about land cover, independent of the observed pixel. Next, we consider the NDWI value for each class. Water areas typically have a high value of NDWI, while the forest typically has a low NDWI value. So if pixel has NDWI=0.7, we can calculate probability based on the training data

$$p(NDWI = 0.7|F) = 0.15$$

$$p(NDWI = 0.7|W) = 0.75$$

This is the likelihood that measures how probable the observed NDWI data is, given that the pixel belongs to the forest class.

Based on Bayes' theorem, we can update our village about pixel class

$p(W|NDWI) = \frac{p(NDWI|W)\cdot p(W)}{p(NDWI)}$ i.e. $p(W|0.7) = 0.2 \cdot 0.75 = 0.15$

$$p(F|0.7) = 0.8 \cdot 0.15 = 0.12$$

After normalization, we will have

$p(W|0.7) = \frac{0.15}{0.15+0.12} = 0.55$ and $p(F|0.7) = \frac{0.12}{0.15+0.12} = 0.45$

So, even though the prior probability that the pixel represents the water class was 20%, after observing that the NDWI=0.7, the posterior probability of the pixel bearing water is increased to 55%.

The two events X and Y are independent if

$$p(X, Y) = p(X) \cdot p(Y)$$

If two events are independent, then

$$p(X|Y) = p(X)$$

meaning that X and Y are independent if knowledge that Y occur does not affect the probability that X occurs. Therefore, the occurrence of X is independent whether or not Y occurs.

11.1 Probability density

The probability for discrete sets of events can be extended to the probabilities with respect to continuous variables. By a continuous variable, we consider a random variable whose sample space is infinite. This is tricky since if each variable has a non-zero probability, the total sum will add up to infinity, which violates the requirement that the total probability must sum up to 1. The X is a continuous random variable if there exists a nonnegative function $p(x)$defined for all real $x \in (-\infty, \infty)$ having the property that for any set B of real numbers

$$p\{X \in B\} = \int_B p(x)dx$$

The function $p(x)$ is called the probability density function of the random variable X. This means that the probability that X will be in B can be calculated by integrating the probability density function over the set I.

Therefore, the probability that X will lie in an interval $[a, b]$ is given by

$$p(a \leq X \leq b) = \int_a^b p(X)dX$$

11.2 Probability distributions

The most common continuous distributions (probability density) are Bernoulli, Poisson, Normal distribution, etc.

The Bernoulli distribution is a simple discrete distribution in which the random variable can take exactly two possible values $x \in \{0, 1\}$. One example of a Bernoulli random variable is the outcome of a coin toss, where possible outcomes are heads or tails. It is specified by a single parameter p, i.e., $p(X = 1) = p$ and $p(X = 0) = 1 - p$ where $0 < p < 1$. Bernoulli distribution can be written as

$$p(X) = p^x(1-p)^{1-x}$$

The **Binomial distribution** models the outcome of performing multiple independent Bernoulli trials, each with the same probability p. Instead of just success and failure, it gives the probability of observing exactly k successes out of n trials. It is defined as follows

$$p(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Let X be the number of successes (value 1) that occur in the n trials, then X is said to have a binomial distribution with parameters (n, p) denoted as $X \sim Bin(n, p)$. However, for large n and small p values, the binomial is hard to compute (such as $X \sim Bin(10^4, 10^{-6})$) and it can be approximated using the Poisson distribution.

The Poisson distribution is, in fact, a limiting case of the binomial distribution. It gives the probability when the chance of an event p is small, but the total number of trials n is large. It is often used if an event occurs independently and randomly over a fixed interval of time, and the mean rate of occurrence is constant over time, then the number of occurrences in a fixed time period follows the Poisson distribution. It is a discrete distribution with a probability mass function of a random variable X is defined as follows

$$p(X = k;\ \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$$

where $k = 0, 1, \ldots$ is the number of events observed and $\lambda = n \cdot p$ is called the average arrival rate. The mean value of a Poisson random variable is λ, and its variance is also λ.

Gaussian distribution, also known as **normal distribution**, is one of the most used probability distributions for continuous variables. It appears in different contexts. According to the Central Limit Theorem, when the number of trials n becomes larger, the distribution of the Binomial variable can be approximated by the Gaussian (especially if p is close to 0 or 1). Moreover, the Poisson distribution begins to resemble a Normal distribution when λ is large (**Figure 76**).

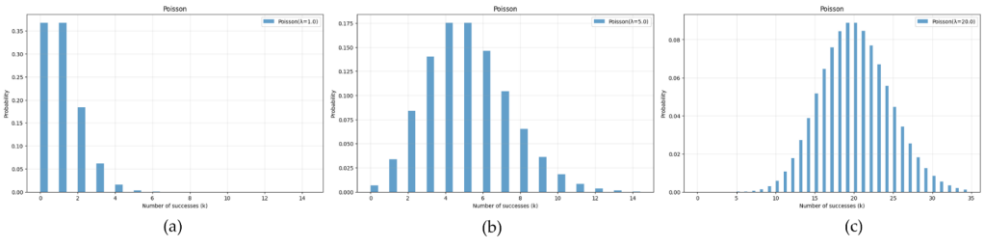


Figure 76 Plots of the Poisson probability function for various values of λ (a) λ=1, (b) λ=5, (c) λ=20

The Gaussian distribution is defined by two parameters: the mean μ and variance σ^2 (**Figure 77**). This means that we can compute any probability of interest given only the mean and standard deviation. For a single real-value variable x, it is given by

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where x can take any value $-\infty < x < \infty$. The argument of the exponential function is the quadratic function of the variable x. Since the coefficient is negative, the parabola points downwards. The coefficient $\frac{1}{\sqrt{2\pi}\sigma}$ is a constant since it does not depend on x. The random variable X that follows a normal distribution is denoted as $X \sim N(\mu, \sigma^2)$.

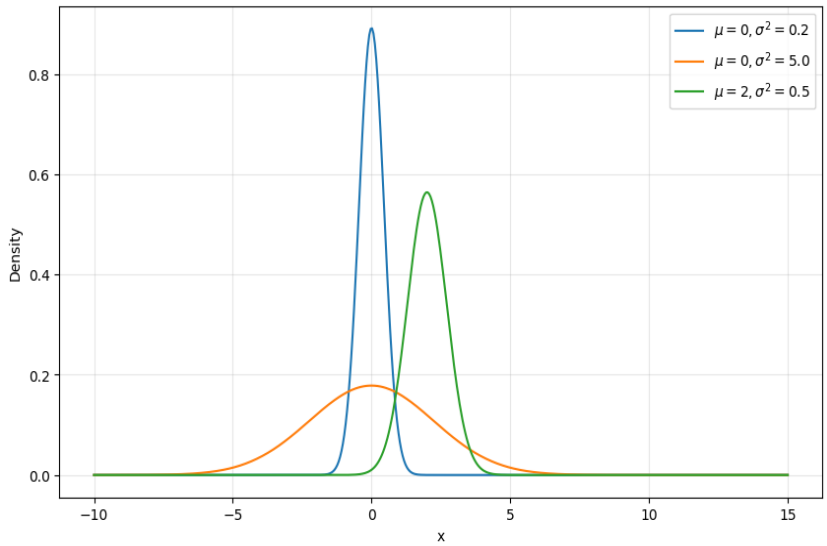


Figure 77 Gaussian distribution for different parameters. μ controls the location of the center of density, σ² controls how spread out the density is.

It is a bell-shaped curve (**Figure 77**) and it is assumed that during any measurement, values will follow a normal distribution with an equal number of measurements above and below the mean values. If the distribution of measurement is normal, then their mean (average of all values), median (mid-point of distribution), and mode (the most frequent value observed during the experiment) are the same. Moreover, the level of confidence can be expressed based on the mean and standard deviation, i.e. $\mu \pm \sigma$ contains 68.2% of all values, $\mu \pm 2 \cdot \sigma$ contains 95.5% of all values and $\mu \pm 3 \cdot \sigma$ contains 99.7% of all values.

The standard Normal distribution is a Normal distribution with a mean $\mu = 0$ and $\sigma^2 = 1$. The random variable that follows a standard normal distribution is often denoted by $Z \sim N(0,1)$.

For a multivariate Gaussian distribution (**Figure 78**), the probability density is defined over a vector of inputs as follows

$$f(x) = \frac{1}{\sqrt{2\pi^n |\Sigma|}} e^{-\left(\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)}$$

where n is the number of variables, μ is an nx1 vector of means, Σ is the covariance matrix nxn. The covariance matrix needs to be symmetric, positive semidefinite, and it can be factored as $\Sigma = AA^T$. The argument of the exponential function is a quadratic form in the vector variable x. Since Σ is positively defined (and therefor it inverse is also positively defined) than for any $x \neq \mu \ -\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu) < 0$ and therefore a quadratic bowl is downward open.

The computing multivariate Gaussian for large n (number of parameters grows quadratically with n) can be computationally demanding. It can be reduced by assuming that the covariances are zero; therefore, the determinant $|\Sigma|$ will be a product of the variance, and the inverse can be computed as the inverse of the diagonal elements.

The n-dimensional multivariate Gaussian with a diagonal covariance matrix $\Sigma = diag(\sigma_1^2, \ldots, \sigma_n^2)$, can be viewed as a collection of d independent Gaussian-distributed random variables with mean μ_i and variance σ_i^2.

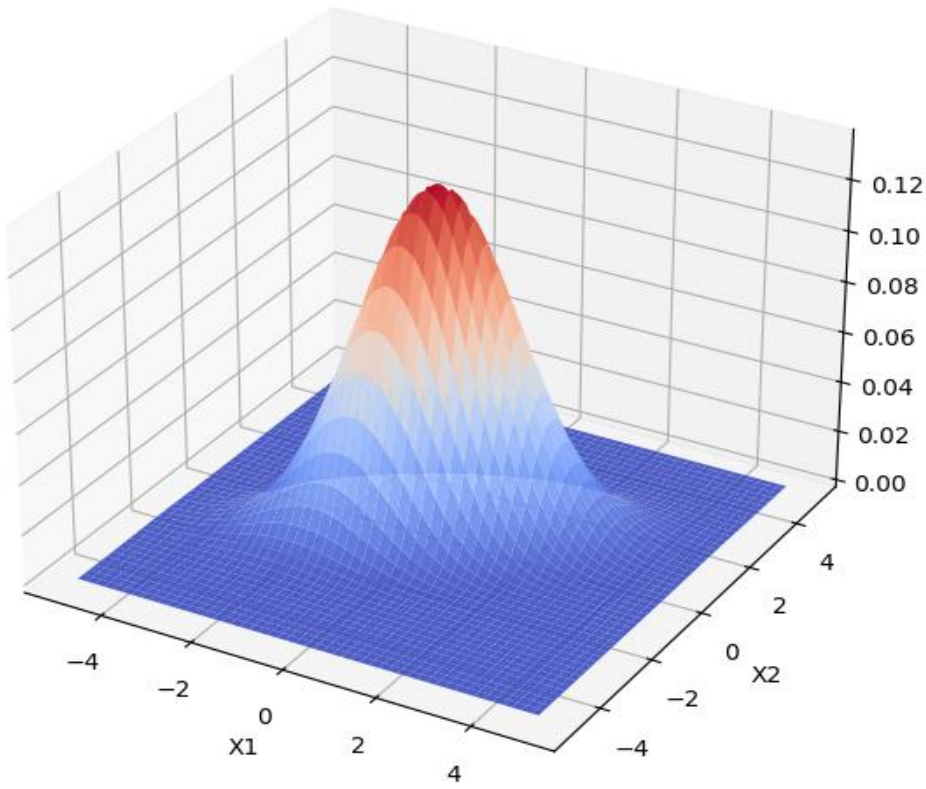


Figure 78 Multi-variate Gaussian

Moreover, if $Z \sim N(0,1)$, Z can be defined as a collection of d independent standard normal variables. Feature more, if $Z = A^{-1}(X - \mu)$ then from algebra $X = AZ + \mu$. Therefore, any random variable X with a multivariate Gaussian distribution can be interpreted as the result of applying a linear transformation to some collection of n independent standard normal variables.

The analysis of contour lines provides a better understanding of the multivariate Gaussian. The contour lines represent the region of equal probability density. Due to the quadratic form in the Gaussian equation, these contour lines are ellipses (**Figure 79** (a)). The orientation and shape of the ellipse are determined by the covariation matrix. If Σ is diagonal, the ellipses are axis aligned (since variables have different variance, the ellipses will be stretched horizontally or vertically), if Σ has off-diagonal elements, the ellipses are rotated (if two variables are positively correlated, the major axis is

along a line with positive slope) (**Figure 79** (c) and (d)). Moreover, the independent variables have the same variance and therefore circular contours (**Figure 79** (b)).

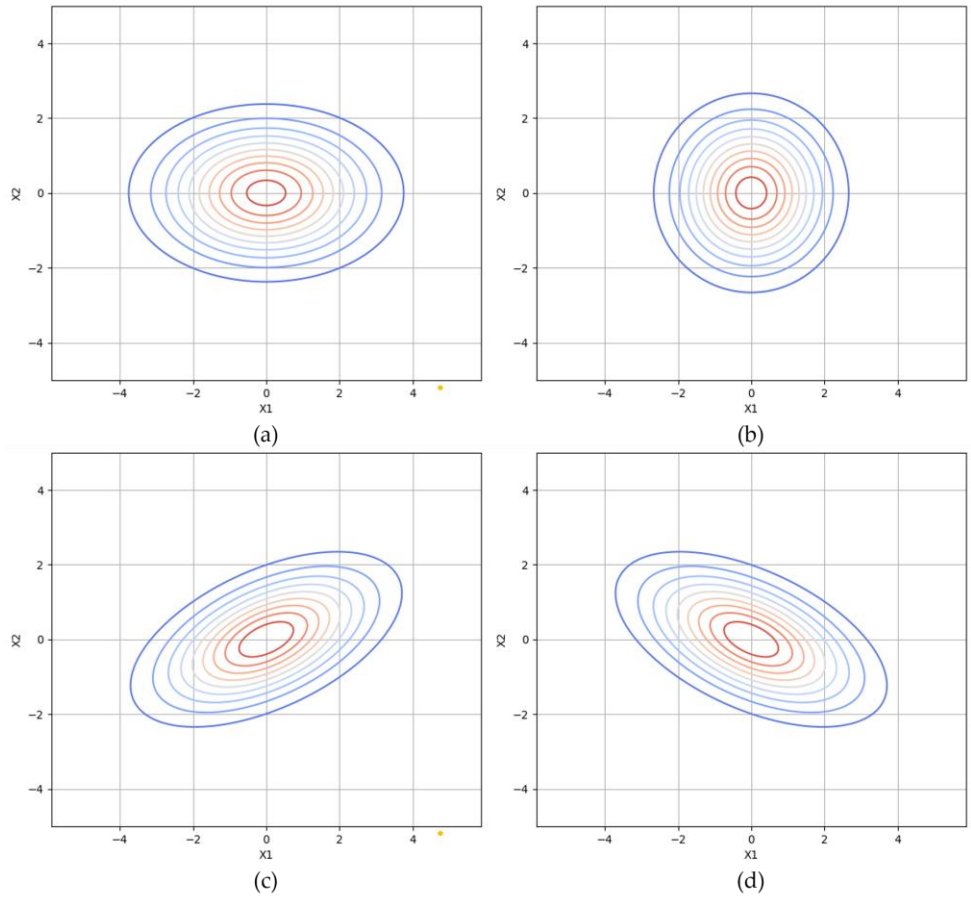


Figure 79 Contour lines of multi-variate Gaussian (a) diagonal covariance matrix, (b) independent variables, (c) positively correlated variables and (d) negatively correlated variables

11.3 Expectations and Variance

The expectation of a random variable, also known as the mean, first moment, or expected value, is denoted by $E(X)$ for the discrete distribution is given by

$$E(X) = \sum_{x_i \in X}^{\cdot} x_i p(X = x_i)$$

Therefore, the expected value of X is the weighted average of the possible values that X can take on weighted by the probability that X assumes that value.

Let's find the $E(X)$ where X is the outcome of rolling the dice. Since $p(1) = p(2) = p(3) = p(4) = p(5) = p(6)$ the expected value is

$$E(X) = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} = \frac{7}{2}$$

Calculate the $E(X)$ where X is a Bernoulli random variable with parameter p. Since $p(0) = 1 - p$ and $p(1) = p$ than $E(X) = 0 \cdot (1 - p) + 1 \cdot p = p$

In the case of continuous variables, expectations are expressed in terms of an integration with respect to the corresponding probability density, i.e.

$$E(X) = \int_{-\infty}^{\infty} xp(x)dx$$

Calculate the $E(X)$ of a random variable uniformly distributed over an interval (a, b)

$$E(X) = \int_a^b \frac{x}{b-a} dx = \frac{b^2 - a^2}{2(b-a)} = \frac{a+b}{2}$$

Therefore, if random variables are uniformly distributed over an interval (a, b) then the expected value is the middle point of the interval.

Calculate the $E(X)$ when X is normally distributed with parameters μ and σ^2

$$E(X) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{\infty} xe^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$$

If we express $x = (x - \mu) + \mu$ transform

$$E(X) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx + \mu \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$$

letting $y = x - \mu$ leads to

$$E(X) = \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty} ye^{-\frac{y^2}{2\sigma^2}}dy + \mu\int_{-\infty}^{\infty} f(x)dx$$

where $f(x)$ is the normal density. By symmetry, the first integral must be 0. Due to that,

$E(X) = \mu\int_{-\infty}^{\infty} f(x)dx = \mu$

Therefore, the expected value of a random variable that follows a normal distribution is the mean value.

The variance of the distribution, also known as the second moment, is defined as follows

$$Var(X) = E\big((X - E(X))^2\big)$$

It measures the expected square of the deviation of X from its expected values. It is often denoted by σ^2. The variance of a random variable X is not a linear function of a random variable. The standard deviation, denoted as σ, is given by $\sigma = \sqrt{Var(X)}$.

If random variables X and Y are independent than

$$Var(X + Y) = Var(X)Var(Y)$$

The covariance of two random variables measures how closely related the two random variables are. It is given by

$$Cov(X, Y) = E((X - E(X))(Y - E(Y))$$

Let X be normally distributed with parameter μ and σ^2. Find $Var(X)$

$Var(X)$=E$((X - \mu)^2) = \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty}(x - \mu)^2 e^{-\frac{(x-\mu)^2}{2\sigma^2}}dx$

by introducing the substitution $y = (x - \mu)/\sigma$ we have

$$Var(X) = \frac{\sigma^2}{\sqrt{2\pi}}\int_{-\infty}^{\infty} y^2 e^{-\frac{y^2}{2}}dy$$

Integrating by parts $u = y$ and $dv = ye^{-\frac{y^2}{2}}dy$ gives

$$Var(X) = \frac{\sigma^2}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{y^2}{2}} dy = \sigma^2$$

11.4 Bayesian classification

Generally, there are two approaches used in classification: generative and discriminative. In a discriminative approach (such as logistic regression), classifiers learn what features from the input dataset are most useful to discriminate between different classes. On another hand, generative approaches, such as Bayesian classifiers, are based on obtaining a distribution over some input data. A Bayesian classifier is a probabilistic approach based on the Bayesian theorem that addresses classification problems by modeling the distribution of the input class. Therefore, it returns the class most likely to generate the observation.

Let us consider the binary classification where samples belong to y_1 or y_2. We assume the prior probabilities $P(y_1)$ and $P(y_2)$ are known or it can be assumed that the classes are equally liked or calculated from training samples. If n is the total number of available training samples and n_1, n_2 of them belong to y_1 and y_2, respectively, then the prior probability is given by

$$P(y_1) \approx \frac{n_1}{n} \text{ and } P(y_2) \approx \frac{n_2}{n}$$

Once we observe a feature vector x, we estimate the conditional probability density distribution $p(x|y_i)$, where $i = 1, 2$, describing the distribution of feature vectors $x = (x_1, \ldots, x_k)$ in each class. If the feature vector can only take discrete values density function $p(x|y_i)$ becomes a probability denoted by $P(x|y_i)$. Applying the Bayes' theorem will lead to the posterior probability given by:

$$P(y_i|x) = \frac{p(x|y_i)P(y_i)}{p(x)}$$

where $p(x)$ is the input data probability distribution, and for which we have

$$p(x) = \sum_{i=1}^{2} p(x|y_i)P(y_i)$$

The Bayes classification rule is based on minimization of probability error i.e., maximization of a posterior probability (MAP), i.e. $\hat{y} = arg\ max_{y\in\{0,1\}} P(y|x)$. Since $p(x)$ is the constant across all classes, it does not affect the argmax, so

$\hat{y} = argmax_{y\in\{0,1\}} \hat{p}(x|y_i)\hat{P}(y_i)$.

Therefore, we can determine the class of a sample by considering the inequality between $p(x|y_1)P(y_1)$ and $p(x|y_2)P(y_2)$.

The classification can be given as:

if $p(x|y_1)P(y_1) < p(x|y_2)P(y_2)$, x belongs to the class y_2,
if $p(x|y_1)P(y_1) > p(x|y_2)P(y_2)$, x belongs to the class y_1, and
if $p(x|y_1)P(y_1) = p(x|y_2)P(y_2)$, x can be assigned to any of the classes.
Therefore, the decision boundary is the set of all x where posterior probabilities are equal.

So, the classifier balances the likelihood (i.e., how well x fits each class) with prior knowledge. Since the prior is a single fixed number, the likelihood grows exponentially with the number of samples, and the influence of the prior on the posterior fades away (**Figure 80**).

If $P(y_1) = P(y_2) = 1/2$ than priors cancel out and classification depends only on the likelihood i.e. the decision boundary is determined by considering the inequality between $p(x|y_1)$ and $p(x|y_2)$.

In multi-class classification Bayes classifier generalizes directly $\hat{y} = arg\ max_{y_m} P(y_m|x)$ where m represents the number of classes. The MLE or MAP can be used to estimate the distribution parameters and then calculate an *argmax* decision rule over m classes for classification. However, if the number of features is large, estimating the probability of every possible combination would require a huge number of parameters and a large training dataset.

This can be addressed by introducing Naive Bayes. The Naive Bayes assumes that each feature of X is conditionally independent of the others, given Y, i.e.

$$p(x_1, \ldots, x_k|y) = \prod_{j=1}^{k} p(x_j|y)$$

where k is the number of features.

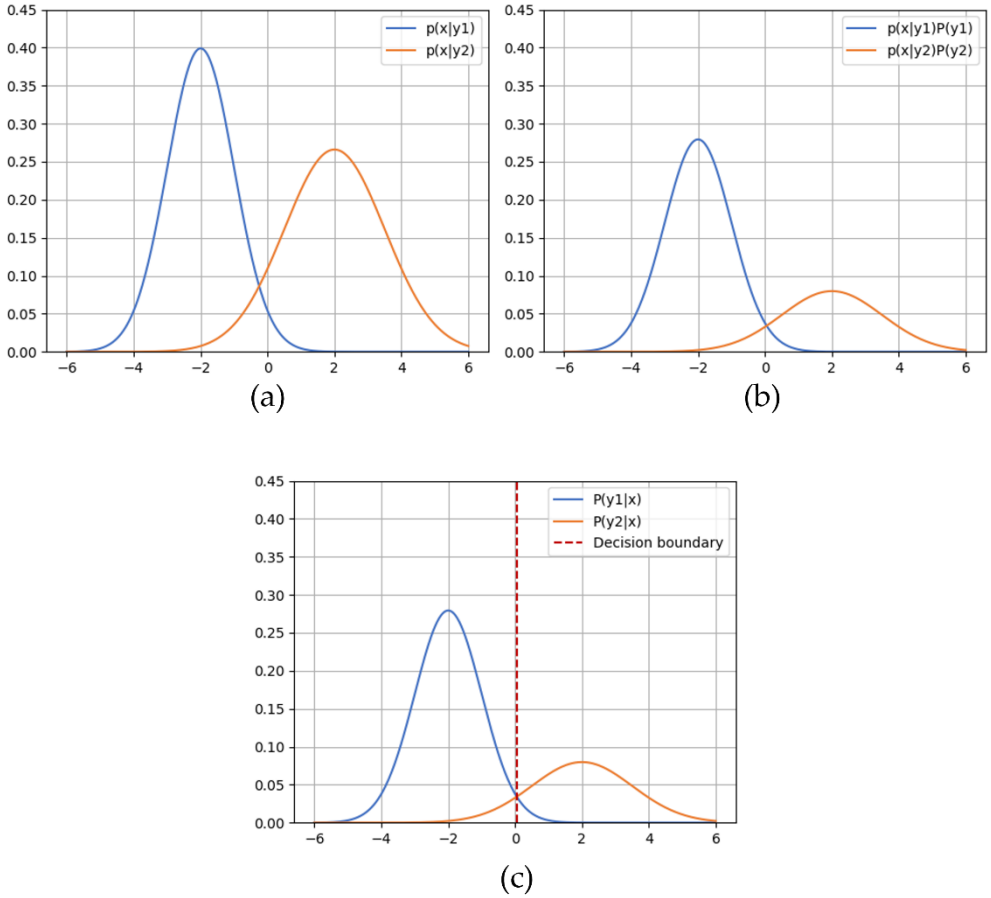


Figure 80 Steps in Bayes classification (a) compute the class conditional probability, (b) multiply by the class prior probability, (c) obtain the posterior probability, and find the decision boundary

This assumption is often wrong in the real world, but it significantly reduces the computational complexity, allowing us to make predictions using space and data, which is linear with respect to the size of the features. Therefore, it enables training and making predictions for a huge feature space. The prediction algorithm can be presented as

$$\hat{y} = arg \max_{y \in \{0,1\}} \hat{p}(x|y)\hat{P}(y) = arg \max_{y \in \{0,1\}} \hat{P}(y) \prod_{i=1}^{m} \hat{p}(x_i|y)$$

$$= arg \max_{y \in \{0,1\}} log\hat{P}(y) + \sum_{i=1}^{k} \hat{p}(x_j|y)$$

11.5 Bayesian error

In Bayesian classification, the feature space is partitioned into two decision regions R_1 and R_2 associated with y_1 and y_2 respectively. The boundary between regions is the decision boundary. Therefore, for all values of x in R_1 the classifier decides y_1 and for all samples in R_2 the classifier decides y_2. However, the errors are unavoidable if class distributions overlap (**Figure 81**). Due to that, there is a finite probability that observation x is located in the R_2 region, in fact, belongs to the class y_1. Then our decision is in error. The total probability, P_e, of committing a decision error for the case of two classes is given by

$$P_e = \int_{-\infty}^{x_0} p(x|y_1)P(y_1)dx + \int_{x_0}^{\infty} p(x|y_1)P(y_1)dx$$

As already mentioned, the P_e is minimized if each x is assigned to the class having the largest posterior probability. The P_e is equal to the total shaded area under the curves.

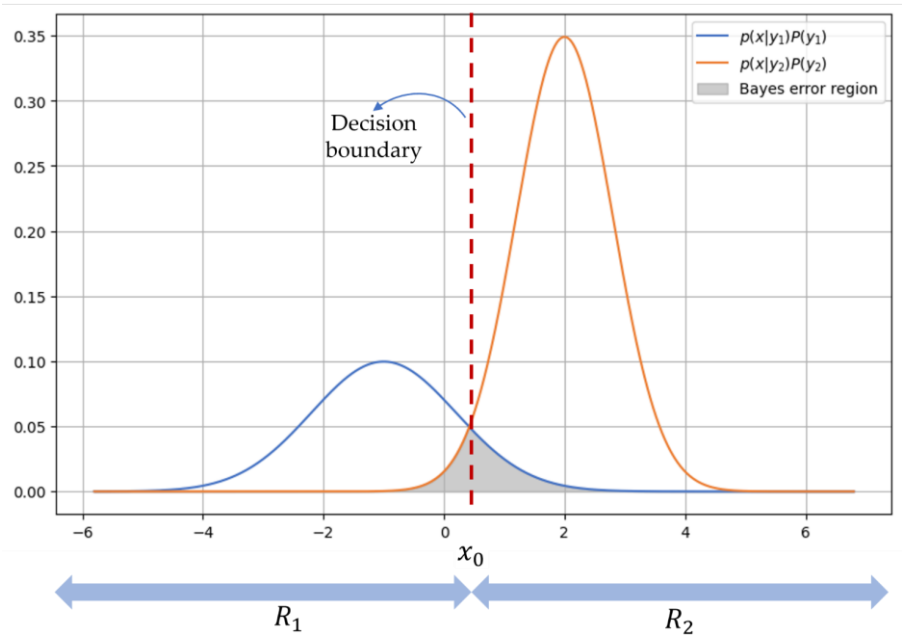


Figure 81 Bayes' error corresponds to the overlap between the class distributions. The optimal decision boundary is where the curves cross (x0).

Bayesian error is a very important concept in ML. It is the minimum error you could obtain using any kind of classifier. The classifier that achieves this error is an optimal classifier. It is a theoretical value that is hard to obtain in real-world problems, since it is hard to estimate the exact data distribution of various classes from a finite dataset. Moreover, the loss function is discontinuous and not convex, and therefore hard to optimize.

12 SUPPORT VECTOR MACHINE

Support Vector Machine (SVM) was proposed by [10], is a supervised ML algorithm used for both classification and regression. It is one of the best off-the-shelf ML algorithms that have been extensively used in various tasks. SVM is a discriminative approach, so it does not provide posterior probability.

As we already mentioned, in binary logistic regression, the decision boundary is linear. The SVM also uses the linear separator by employing the optimal margin principle.

Let us consider a binary classification problem with labels y and a feature vector x_i. Class

$y = 1$ is a positive class and $y = -1$ is the negative class, respectively. We assume that positive and negative classes are linearly separable. The goal is to find a decision boundary given as

$$g(x) = w^T x + b = 0$$

that classifies data correctly. The decision boundary separates the feature space into two subspaces. If $w^T x + b \geq 0$ than $y = 1$, otherwise $y = -1$. However, such a decision boundary is not unique (**Figure 82**).

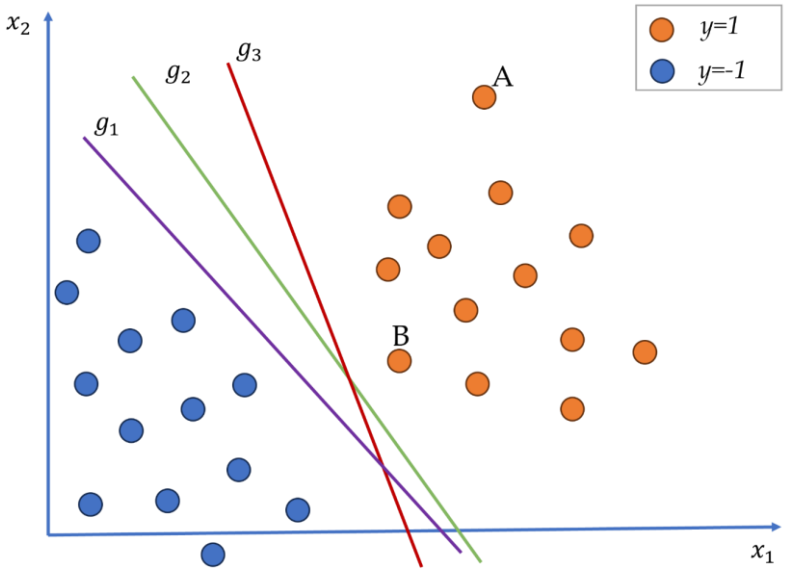


Figure 82 Binary classification in a 2D feature space. Multiple decision boundaries can provide correct classification

The question is which decision boundary we should select. Let's consider point A, which is located far away from the decision boundary, meaning that we are very confident that point A belongs to the positive class. On the other hand, point B is close to the decision boundary, and small changes to the decision boundary can result in classifying point B as a negative class. Although point B is correctly classified, we are not very confident in the prediction at point B. So we wish to find a decision boundary that will provide correct and confident classification on training examples. Taking that into account, we would most likely select a green line because it provides the largest gap between classes. Due to that, the data points can move more freely without causing errors. Therefore, the generalization ability of the classifier is higher. So if there are multiple decision boundaries that provide accurate classification, we should select one that provides the best generalization (i.e., it provides stable performance on the unseen data).

The SVM ensures the generalization of maximizing the margins. Margin is defined as the smallest perpendicular distance between the decision boundary and all data samples. To quantify the margin that a decision boundary leaves from both classes, we start with the assumption that the w are b are constrained so that the output of the linear model is always larger than 1 or smaller than -1, i.e.

$$\begin{cases} w^T x + b \geq 1 & if\ y_i = 1 \\ w^T x + b \leq -1 & if\ y_i = -1 \end{cases}$$

Thus, if $y_i = 1$ than our prediction if $w^T x + b > 0$ is correct, while it is correct and confident if $w^T x + b$ is a large positive number.

Consider two points x_1 and x_2 that both lie on the hyperplane, i.e. $g(x_1) = g(x_2) = 0$ and therefore $w^T(x_1 - x_2) = 0$. So any difference vector that lies inside the hyperplane is orthogonal to w. Thus w points in the direction normal to the hyperplane and determine its orientation.

To compute the margin, let's analyses the training set $s = \{(x_i, y_i);\ i = 1, \ldots, n\}$. For a candidate hyperplane (w, b), the functional margin of the training point (x_i, y_i) is given by

$$\widehat{\gamma_i} = y_i(w^T x_i + b)$$

This quantity is positive if the point is correctly classified, and its magnitude reflects how confidently the classifier assigns the labels. The true geometrical distance from any point to a hyperplane is given by

$$\gamma_i = \frac{\widehat{\gamma_i}}{||w||} = \frac{y_i(w^T x_i + b)}{||w||}$$

therefore geometrical margin is a scaled version of the functional margin (i.e., if $||w||$=1 than the functional margin is the same as the geometrical margin). It is invariant to rescaling the parameters, so if we replace w with $2w$ and b with $2b$ the geometric margin does not change.

Let's analyse the closest point x_3 of the positive class to the boundary. The margin of x_3 can be computed by projecting the line segment vector $x_3 - x_1$ over the orientation vector w i.e.

$$\frac{1}{2}\rho = \frac{w^T(x_3 - x_1)}{||w||} = \frac{(w^T x_3 + b) - (w^T x_1 + b)}{||w||} = \frac{1}{||w||}$$

where $||\cdot||$ is the norm (magnitude) of a vector. Hence, the point closest to the decision boundary is at a distance $\frac{1}{||w||}$. The magnitude of w does not change orientation, it only scales the margin width. Therefore, to maximize the margin $\rho = 2/||w||$ we need to minimize the norm of the weight vector w. Taking that into account, we need to find a decision boundary with parameters w and b so that

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

subject to constraint $y_i(w^T x_i + b) \geq 1 \quad \forall i = 1, \ldots, n$ i.e., all training points are classified correctly and lie outside the margin.

Finally, for a given training set S, the geometrical margin of the classifier is defined as the smallest geometrical margin among all training examples

$$\gamma = \min_{i=1,\ldots,n} \gamma_i$$

Since the objective function is a quadratic function and all constraints are linear inequality constraints, this is a convex problem, so any local solution is also a global optimum. This is an important property of SVM. The SVM

optimization problem has 1 objective and n corresponding constraints, where n represents the number of training samples.

In order to solve this constrained problem, for each x_i we introduced Lagrange multipliers $\lambda_i \geq 0$. The overall Lagrangian function is given as

$$L(w, b, \lambda) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \lambda_i (y_i (w^T x_i + b) - 1)$$

where $\lambda = (\lambda_1, \ldots, \lambda_n)^T$. The first term is our objective, while the second term penalizes violation of the constraint (loss function). The minus sign in front of the Lagrange multiplier term is because we are minimizing with respect to w and b and maximizing with respect to λ. From the Karush-Kuhn-Tucker condition, at the optimum, we require stationarity w.r.t. w and b. By setting the derivatives of $L(w, b, \lambda)$ with respect to w and b equal to zero, we obtain the following two conditions

$$w = \sum_{i=1}^{n} \lambda_i y_i x_i$$

Returning w back into L, the dual optimization problem becomes

$$max_\lambda \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (x_i \cdot x_j)$$

subject to $\lambda_i \geq 0 \; \forall i = 1, \ldots, n$

$$\sum_{i=1}^{n} \lambda_i y_i = 0$$

If $\lambda_i = 0$ the constraint for point i is not active, i.e., points lie outside the margin and do not influence the solution. Therefore, Lagrangian multiplier vector is a sparse vector and only vectors that lie on the margin hyperplanes $w^T x_i + b = \pm 1$ will have positive λ_i. Thus, the vector parameter w of optimal solution is a linear combination of the feature vector $n_s < n$ with $\lambda_i \geq 0$ i.e.

$$w = \sum_{i=1}^{n_s} \lambda_i y_i x_i$$

The feature vectors that contribute to w are called support vectors, and the optimum decision boundary is known as an SVM. Geometrically, the hyperplane bisects the closest points of opposite classes, and only supported vectors are used to determine the optimal solution, leading to robustness and good generalization (**Figure 83**).

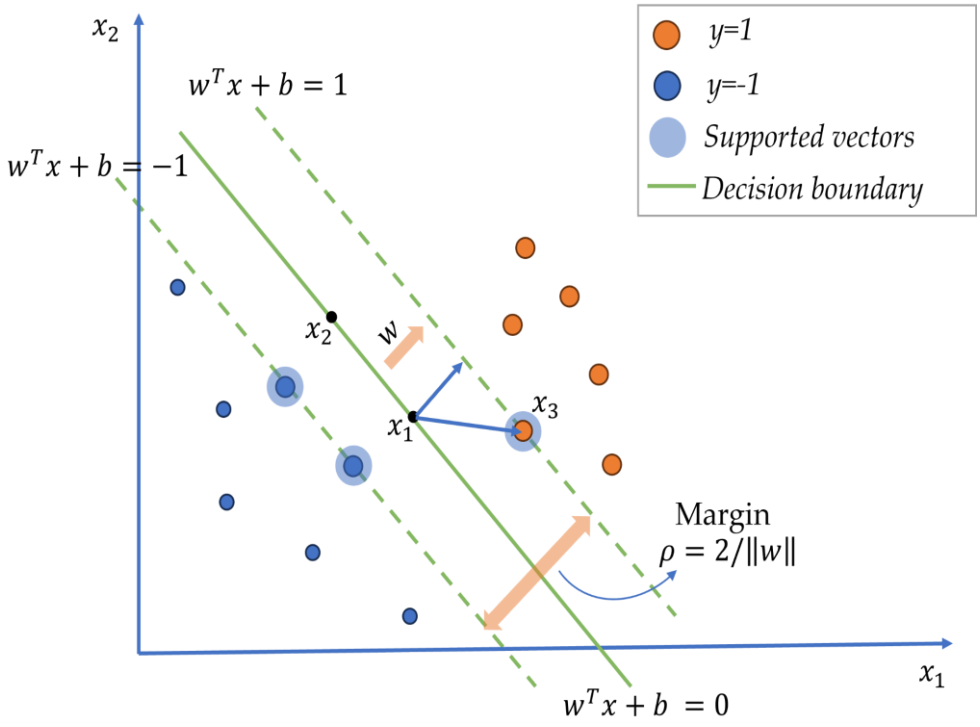


Figure 83 Support vector optimization

To predict a new point x_{new}

$$w^T \cdot x_{new} + b = \sum_{i=1}^{n} \lambda_i y_i (x_i \cdot x_{new}) + b$$

If the quantity is bigger than 0, the new point belongs to the positive class. Thus, the prediction on a new point depends only on the inner product between x and the supported vectors.

There are several important characteristics of SVM:

- The optimal decision boundary of SVM is unique. As already mentioned, the optimization is a strict convex function and thus it guarantees that any local minima is also a global minimum.
- Uses just a subset of training data (supported vectors) to determine decision boundaries. It is computationally effective,
- It is robust to outliers,
- It is little affected by data distribution and density, and
- It works well in high-dimensional space, especially where the number of features is higher than the number of samples.

On the other hand, it is computationally expensive and it performs badly when classes overlap.

12.1.1 Soft margins

So far, we assumed that classes are linearly separable. When classes are linearly separable, the loss function results in infinite error if a point is misclassified and zero error if it is correctly classified, and then optimized model parameters are used to maximize the margin. However, in practice, data are usually not perfectly separable (class overlap, mislabeling, noise, etc). Consider the cases presented in the **Figure 84**.

Although classes' conditional probability slightly overlap, points can still be linearly separable, but due to the presence of outliers, the decision boundary makes a swing, resulting in a narrow margin (i.e., low generalization ability). On the other hand, we can maximize margin, but the constraint is violated (some points are misclassified). The question is which of those cases is better? In general, there is a trade-off between the margin maximization and the number of mistakes on the training data. Therefore, if classes slightly overlap, we can allow some of the training points to be misclassified in order to increase the margin.

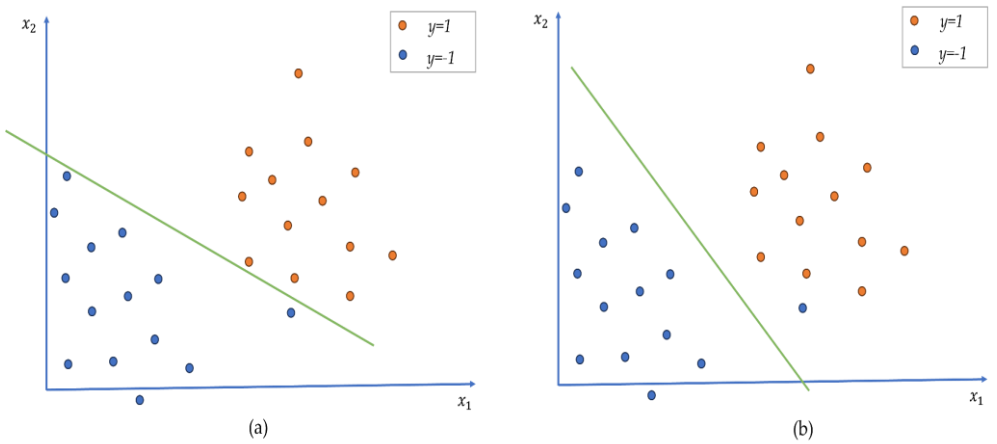


Figure 84 Linear non-separable classes (a) point samples are correctly classified but the classifier has a much smaller margin, (b) the margin is maximized, but there is a misclassified point

Consider the non-linear separable classes presented in the **Figure 85**. There are three possible cases: the vector is located outside the margin and it is correctly classified, the vector flies inside the margin and it is correctly classified, or the vector is misclassified.

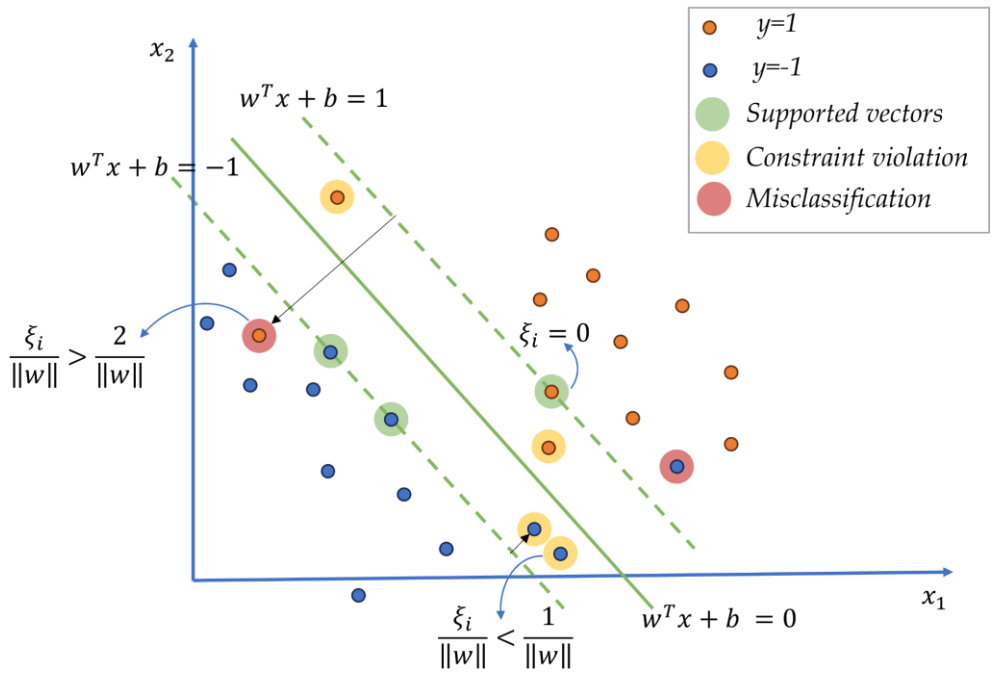


Figure 85 Soft margin SVM

To make the algorithm work for a non-linearly separable dataset and reduce the sensitivity to outliers, we can modify the approach so that data points can be on the wrong side of the boundary, but with a penalty that increases with distance from the boundary. To do so, we introduced the slack variables ξ_i where $i = 1, \ldots, n$..The misclassification is penalized as a linear function of the distance from the point to distance boundary by using a single type of constraint

$$y_i(w^T x + b) \geq 1 - \xi_i$$

where $\xi_i \geq 0$. Vectors for which $\xi_i = 0$ are correctly classified, if $0 < \xi_i < 1$ vector lies inside the margin and it is correctly classified (this is a margin violation), and for $\xi_i > 1$ vector is misclassified. This relaxation that allows vectors to be misclassified is known as a soft margin.

The goal is to maximize the margin while minimizing the number of vectors with ξ_i. Therefore, we need to minimize

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

s.t. $y_i(w^T x_i + b) \geq 1 - \xi_i \quad \forall i = 1, \ldots, n$

$$\xi_i \geq 0 \quad \forall i = 1, \ldots, n$$

where $C > 0$ is a constant that controls the trade-off between training error and margin width. The large C leads to small ξ_i and strong penalization of margin violations, while small C allows more tolerance for misclassification but enables better generalization. Misclassified points can be farther away from the decision boundary, and the model won't change the boundary to fit them.

12.1.2 Kernel

Soft margin allows the SVM to deal with slight distribution overlapping. However, if classes are not linearly separable. Consider the following case (**Figure 86**) where the positive and negative class is arranged into two concentric circles. In the original feature space, there is no straight line that can separate these classes.

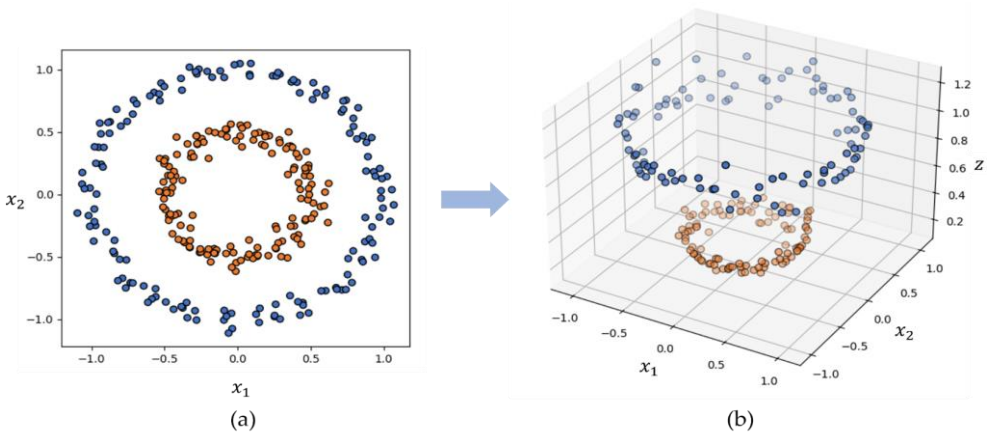


Figure 86 (a) classes in the original feature space, (b) classes embedded in a higher-dimensional space

In such cases, the SVM relies on the so-called kernel trick. The basic idea is to map data in a much higher-dimensional space where they can be separated. If we go back to our example, we can project data into 3D space by adding a square radius $z = x_1^2 + x_2^2$ as a new feature, i.e. (x_1, x_2, z). In higher-dimensional space, the two circles lie on parallel planes, and we can create a linear hyperplane to separate them.

The kernel function computes an inner product between two data points in the high-dimensional feature space without performing the explicit mapping.

As already mentioned, the dual optimization problem only depends on the inner product. Thus, if data are not linearly separable, we can map them into a higher-dimensional feature space $\phi(x)$ so that

$$\left(x_i \cdot x_j\right) \mapsto K\left(x_i, x_j\right) = \phi(x_i)^T \cdot \phi\left(x_j\right)$$

where K is the kernel function. Therefore, the dual problem with the kernel is given by:

$$max_\lambda \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j K\left(x_i, x_j\right)$$

subject to $\sum_{i=1}^{n} \lambda_i y_i = 0,\ 0 < \lambda_i \leq C\ \forall i = 1, \ldots, n$

Now the algorithm learns by using features ϕ.

Once the optimization is solved, the classification is given by

$$g(x) = sign\left(\sum_{i=1}^{n} \lambda_i y_i K(x_i, x) + b\right)$$

The SVM dual only needs dot product; by replacing it with a kernel, SVM is able to perform nonlinear classification without explicitly computing high-dimensionality (i.e., the kernel function directly gives the inner product). Due to that, it is computationally efficient and scalable, even when the feature space is infinite-dimensional. The most commonly used kernels are: the linear kernel, the polynomial, and the radial basis function (RBF) kernel.

The **linear kernel**, given as $K(x, z) = x \cdot z$, is the simplest kernel used in SVM. It does not perform any explicit mapping, i.e., it operates in the original feature space. It is used when classes are approximately linearly separable.

The **polynomial kernel** allows modeling the nonlinear relationship between features. It is given by $K(x, z) = (x^T \cdot z + c)^d$ where d is the degree of the polynomial and c is a constant that controls the trade-off between higher-order versus lower-order terms (usually 0 or 1). Consider n training samples, each having two $m = 2$ features, and we want to use a degree $d = 2$ polynomial expansion. Degree 2 polynomial expansions include all monomials up to degree 2, i.e. $\phi(x) = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2]$ so points are mapped from a 2D original feature space to a 6D feature space.

By combining the dual optimization and kernel trick, we can compute the value of the kernel without explicitly writing the blown-up representation, i.e., the K values are computed by performing operations in the original space, and everything works as if we had mapped to higher higher-dimensional space, so $\phi(x)^T \phi(z) = (x^T \cdot z + 1)^2$. This allows SVM to fit more complex classifiers without significantly increasing computational cost. However, fitting increasingly complex models to the training set of the same size without regularization can lead to overfitting.

The **Radial Basis Function** is a function whose values depend on the distance to a center in the input space (usually Euclidean). The most commonly used RBF is the Gaussian RBF, and is defined as

$$K(x, z) = exp\left(\frac{||x - z||^2}{2\sigma^2}\right)$$

where σ^2 is known as bendwidth, and it is directly related to the width parameter $\gamma = 1/2\sigma^2$. The width parameter controls the influence of each training point. The RBF Kernel measures the similarity between two vectors. If it is close to 1, x and z are close, while near 0 values imply that x and z are far apart. So if γ is large, the Gaussian is very narrow and only points close to each other have a high similarity $K(x, z) \approx 1$. The decision boundary fits data points tightly, leading to complex boundaries and possible overfitting (**Figure 87**(c)). On another hand, for small γ, the Gaussian is wide and the decision boundary becomes smooth and more general. However, it can lead to underfitting (**Figure 87**(a)). Therefore, selecting an appropriate γ value is crucial for SVM performance. RBF Kernel implicitly maps the input data into an infinite-dimensional feature space, enabling separation of very complex patterns.

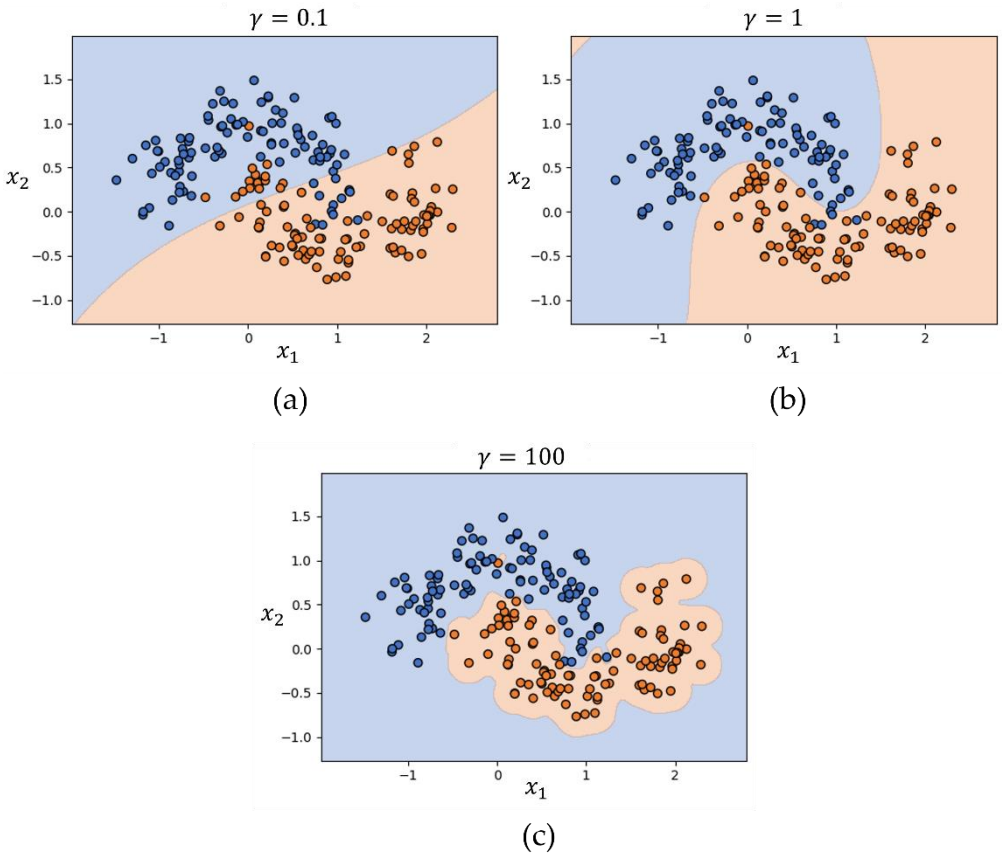


Figure 87 Decision boundary by using different γ values (a) underfitting, (b) just right, and (c) overfitting

12.1.3 Multi-class SVM

Although SVM is fundamentally designed for binary classification, it can be adapted to perform multiclass classification. This is accomplished by applying the same principles after breaking down the multi-class problem into multiple binary classification problems by using one-vs-one (OvO) or one-vs-rest (OvR) approaches (**Figure 88**).

Suppose our data has *K* different classes $y \in \{1, \ldots, K\}$. The main idea behind the one-vs-one approach is to simply train $\frac{K(K-1)}{2}$ different binary classifiers for every possible pair of classes. To make predictions for new data points, each classifier will result in one possible class label, and the point is assigned to the label with the higher number of votes.

OvR approach constructs only *K* classifiers, and k^{th} mode is trained by using data from the *K* class as positive, and all of the remaining data is considered negative. In an ideal scenario, when a new point is presented to the classifiers, exactly one of the *K* classifiers will label the point as positive, and all the remaining will classify it as negative. In reality, all classifiers can assign data points to negative classes, or more than one classifier can assign them to positive classes. A new point is fed into all *K* classifiers and each outputs a decision value (distance from its separation hyperplane). In reality, all classifiers can assign data points to negative classes or more than one classifier can assign it to positive classes.

There is no definitive rule for choosing between one-vs-one (OvO) and one-vs-rest (OvR) SVMs. OvO requires more classifiers ($\frac{K(K-1)}{2}$ compared to *K* in one-vs-rest), but each is trained just on a subset of data, reducing the computational cost per model. On the other hand, OvR classifiers are always trained on an imbalanced dataset. Let's say that we have a perfectly balanced training dataset classified in five classes, the individual classifiers in the one-vs-rest approach will be trained on 20% of positive and 80% of negative samples, and the symmetry of the original problem is lost.

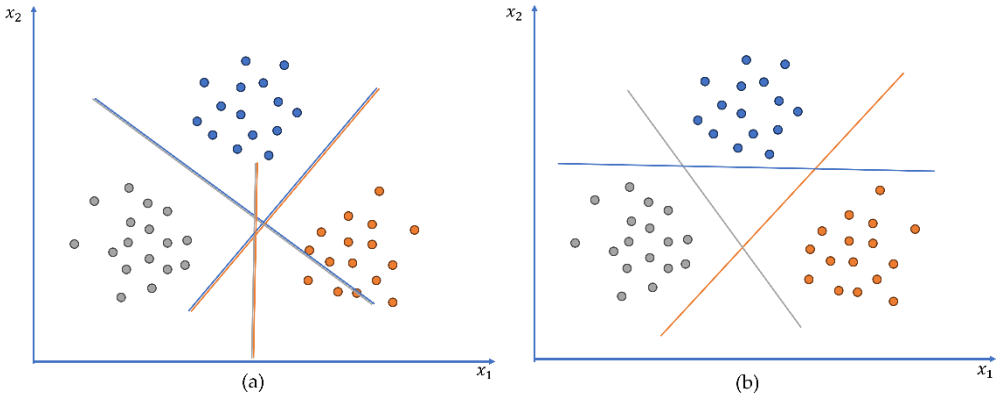


Figure 88 (a) OvO approach, (b) OvR

12.1.4 Supported Vector Regression

Although the SVM has been primarily introduced for binary classification, its extension to regression is also known as Support Vector Regression (SVR). The main idea is to find a function

$$g(x) = w^T x + b$$

that approximates all training set (X, y) with a margin of tolerance ϵ and at the same time as flat as possible (minimizing function complexity). Due to that, SVR uses the ϵ insensitive loss given by

$$L_\epsilon = \{0, \quad if\,|y_i - (w \cdot x_i + b)| \leq \epsilon\; |y_i - (w \cdot x_i + b)| - \epsilon, \quad otherwise$$

This defines the tolerance tube with radius ϵ. If the predicted value is within the tube, the loss is equal to zero. If the prediction lies outside the tube, the loss grows linearly with the distance from the boundary. As ϵ increases, the function is allowed to move away from the data points, the number of supported vectors decreases, and the fit decreases. However, sometimes functions that approximate all pairs (x_i, y_i) with ϵ precision does not exist. In that case, we want to allow some errors by which predictions exceed the allowed tolerance.

Analogy to soft margins in SVM, we can account for the errors outside the tolerance tube by introducing two slack variables ξ_i^+ and ξ_i^-. They define a positive and negative derivation outside the tolerance area. The optimization problem is given by:

$$min\ \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}(\xi_i^+ + \xi_i^-)$$

subject to

$$y_i - w^T x_i - b \leq \epsilon + \xi_i^+$$

$$w^T x_i + b - y_i \leq \epsilon + \xi_i^-$$

$$\xi_i^+, \xi_i^- \geq 0$$

The constant $C > 0$ determines the trade-off between the complexity of the function f and the tolerance of derivations outside the ϵ tube.

Just as in SVM, kernels can be applied in SVR to handle non-linear relationships, allowing the regression function to capture complex patterns in data.

13 DECISION TREES

Decision trees have been extensively used for both classification and regression tasks. It recursively splits the feature space into homogeneous regions based on the values of different features until a stopping criterion is met.

Consider dataset $D \subset (X, y)$ that contains n data samples, each of them being d-dimensional and classified into K classes ($y = \{y_1, \ldots, y_K\}$). A decision tree divides the feature space $(x_1, \ldots, x_d)$ into K distinct regions $R_1, \ldots, R_K$ to correspond to K classes. For every data sample that falls into the region R_K algorithms make the same prediction. The condition is that regions don't intersect with each other, and the union of all regions covers an entire feature space.

The root node represents the entire feature space. The algorithm aims to identify the feature and threshold that lead to the best split based on a specific criterion $\theta \in R$ i.e. $[x_j \geq \theta]$ where $j \in \{1, \ldots, d\}$. To determine "best", we measure the impurity of a node. Each split results in two child nodes, and each split must generate subsets that are more class homogeneous. This process continues sequentially, with each region being split on a specific feature. If the node is not feature-splitted, then we assign the prediction y to the region corresponding to this node. This node, where the tree ends, is also known as a leaf node, and the number of leaf nodes is equal to the number of distinct classes within the dataset. The complexity of the tree is measured by the number of splits in the tree.

This mapping results in a tree-like structure where we follow the right branch of the node if the condition is met and the left otherwise. The process continues until the stopping criterion is reached, such as: if the maximum value of decrease in the node impurity over all possible splits is less than threshold, the predefined depth of the tree is reached, the number of data samples in the terminal node is less than the minimum number, nodes are homogeneous, or there is no further improvement in purity.

Splitting criteria are used to determine where a tree should split. The main aim is to use a criterion that quantifies node impurity (which quantifies the homogeneity of the labels at a given node) and splits the node so that the

overall impurity of the child node decreases compared to the parent node. Different splitting criteria, such as gain index, information gain, or information gain ratio, can be used.

The choice of splitting criterion directly influences the tree's structure and algorithm performance.

The gain index (Gini) quantifies the purity of a specific class after splitting using a particular feature. It is given by

$$Gini(D) = 1 - \sum_{i=1}^{K} {p_i}^2$$

where p_i is the proportion of the training samples in the set that belongs to class i. K is the total number of unique classes in the dataset. *Gini* represents the probability of incorrectly classifying a sample from the set D if we randomly label it based on the class distribution in the node. If the dataset is split on feature F into two subsets D_1 and D_2 with size N_1 and N_2 respectively, $Gini$ is given by

$$Gini_F(D) = \frac{N_1}{N} Gini(D_1) + \frac{N_2}{N} Gini(D_2)$$

The value of $Gini$ ranges from 0 (perfect purity - node contains only one class) and 1 (maximum impurity). It is highly efficient for standard classification tasks, but misclassifying samples always results in the same amount of loss regardless of the distance in the original scale between the observed and the predicted class.

Information Gain (IG) is based on the entropy that measures the impurity or randomness in a data set. The aim is to find a split that decreases uncertainty before and after the split. It determines the effectiveness of a feature in splitting the training data into homogenous sets. The entropy (H) of set D is given as follows

$$H(D) = -\sum_{i=1}^{K} p_i log_2(p_i)$$

where negative signs ensure that information is positive or zero, p_i is the proportion of instances in D that belongs to class i, i.e., $p_i = \frac{|D_i|}{|D|}$ For a node

with one class $p_i = 1$ and therefore entropy is equal to 0. If mixing is higher, the entropy will also be higher. Consider a feature (F) with possible values $\{\delta_1, \ldots, \delta_m\}$. Then, splitting D on F will result in $D_1, \ldots, D_m$subsets where each subset D_j contains the instances in D such that $F = \delta_j$. Then the entropy of D_j is given by

$$H(D_j) = -\sum_{i=1}^{K} p_{ij} log_2(p_{ij})$$

The information gain of splitting on F is calculated as the difference between the entropy of the parent node $H(D)$ and weighted entropy after the split $H(D|F)$ and it is given by

$$IG = H(D) - H(D|F) = H(D) - \sum_{j=1}^{m} \frac{|D_j|}{|D|} H(D_j)$$

where m are the different values that feature F can take, and D_i is the subset of D for which feature F has the value m. IG measures the reduction in entropy from the original data set D to the set D_j created after the split. Based on that, it determines the usefulness of a feature f at classification. A high IG indicates a more effective feature for splitting the data, and therefore, it results in a more homogeneous subset. Although IG is a popular splitting criterion, it tends to favor the attributes with many distinct values.

Information gain ratio (IGR) is designed to address the limitations of IG by considering the number and size of branches when choosing a feature. The IGR normalizes the IG by considering the intrinsic information (also known as split information). This normalization reduces bias toward the multi-value features, resulting in a more balanced and effective decision tree. It is given by

$$IGR(D, F) = \frac{IG(D, F)}{SplitInfo(D, F)}$$

where the $SplitInfo(D, F) = -\sum_{j=1}^{m} \frac{|D_j|}{|D|} log_2\left(\frac{|D_j|}{|D|}\right)$ is the entropy of the subset D_j.

To fit the model, algorithms minimize the loss (impurity) function in the child compared to the parent node. Due to the discrete structure of the decision tree,

the loss function is not smooth, and therefore, a greedy algorithm is used to fit data to the model.

Algorithm Build tree

Input: dataset D with (x_i, y_i) elements, *k*-number classes, *f*-features

Return: decision tree

function BuildTree(D):

if stopping criterion is met **than:**

return leaf node with the class label

best_gain=[]

for *j*=1 to *d*: *#loop over all features*

for m in possible_values(F_j): *#loop over possible values (split points) of feature*

$D_{Left}, D_{Right} \leftarrow split(D, F, \theta)$ *#temporarily partition the data into left and right subsets*

$gain = H(D) - \frac{|D_{Left}|}{|D|} H(D_{Left}) + \frac{|D_{Right}|}{|D|} H(D_{Right})$ *# test all possible combinations*

if $gain > best\_gain$:

$best\_gain = gain$

$F_{best} = F_j$

$\theta_{best} = m$

$D_{LeftBest} = D_{Left}$

$D_{RightBest} = D_{Right}$

left_child$\leftarrow BuildTree(D_{LeftBest})$

right_child$\leftarrow BuildTree(D_{RightBest})$

return Node(F_{best}, θ_{best}, left_child, right_child)

Greedy means that at each step, the algorithm chooses the best split locally (the largest reduction in the loss right now), without considering the long-term effect deeper in the tree. Therefore, once a split is made, it is never revisited. This enables efficient training, but also leads to suboptimal trees. Due to that, techniques such as pouring or ensembling methods are used to improve performance.

The performance of decision trees is highly influenced by the quality of training data, tree depth (tree size), splitting criteria, and tree pruning methods. Moreover, feature selection and feature engineering can improve the efficiency and accuracy of models.

The decision tree algorithms are intuitive and easy to interpret, making them valuable where understanding of the decision-making process is crucial, can handle both categorical and numerical data, don't demand feature scaling, are robust to outliers, are not affected by non-linear relationships between parameters, and are efficient with small to medium-sized datasets. Moreover, they implicitly perform feature selection and feature importance analysis by using the feature selection measure. However, without proper pruning (limiting tree growth), they tend to overfit, leading to poor prediction accuracy. Also, small changes in the training data result in very different trees due to the hierarchical nature of tree classifiers.

13.1.1 Pruning

Given a greedy strategy for building the tree, the remaining question is when to stop adding nodes to reduce the possibility of overfitting. There are two main approaches: pre-pruning (early stopping) and post-pruning. Pre-pruning reduces the possibility of overfitting by reducing the size of the tree. To do so, different criteria such as maximum depth, minimum number of samples in a node, the information gain is below a certain threshold, etc., can be used. The main advantage is that the tree remains small, and it is computationally efficient. Pre-pruning relies on the threshold (such as information gain) to decide whether to continue or stop splitting. Algorithms evaluate each split locally without looking ahead. Due to that, it is possible that none of the current splits don't reduce the error significantly, while after several steps a significant error reduction is obtained. So if the applied threshold is too aggressive, it can lead to underfitting. Due to that, common

practice is to grow a large tree, using the number of samples in leaves as a stopping criterion. and prune back the resulting tree.

Post-pruning is usually done by trimming down decision tree paths that do not provide significant improvement in predicting accuracy. Therefore, post-pruning is based on a criterion that balances the trade-off between residual error and model complexity to improve generalization ability. The pruning starts from the leaves and for each node analyzes if the increased performance associated with it is worth the extra model size. If not, the node is removed by merging it back into a tree. Therefore, the subtrees are removed if they lead to a small reduction in error relative to their size.

Consider a grown tree T_0 with leaves indexed as $L = 1, \ldots, |t|$ with leaf node t representing the region R_t. The $|t|$ denote the total number of leaf nodes in T. We want to create a tree $T \subset T_0$ to be a subset of a tree T_o.

Usually, advanced methods such as cost-complexity are used. The minimal cost-complexity pruning is given by

$$R_\lambda(T) = R(T) + \alpha|t|$$

where $R(T)$ is a total misclassification error of the whole tree. α is a tuning parameter that controls the trade-off between model complexity and accuracy. If $\alpha = 0$ than pruning is not performed. Larger α means a stronger penalty for model complexity, resulting in a simpler tree. Minimal cost-complexity pruning finds the subtree of T_0 that minimizes the $R_\alpha(T)$.. The process is continued until only the root remains. The result is a sequence of subtrees $T_0 \supset T_1 \supset \ldots T_m$ where T_m is the root-only tree. Each subtree is evaluated using cross-validation error.

13.1.2 Ensemble methods

Using a single tree can be challenging due to its high sensitivity to small changes in the data. Therefore, another way to deal with overfitting is by averaging the predictions over multiple samples. Ensemble methods combine several single trees to produce a one, more robust predictive model. There are several types of ensemble methods, such as:

- Bagging (also known as bootstrap aggregation) - The basic idea is to create B bootstrap samples ($X_1, \ldots, X_B$) by uniformly sampling from the

original dataset X with replacement (i.e., the same training sample can be selected multiple times). Usually, ⅔ of bootstrap samples represent the original training data, while the remaining ⅓ is a replacement. Each bootstrap sample is used to train the model. The final prediction is made by average (for regression) or majority vote (for classification). A classic example is Random Forest.

- Pasting - creates a subset of the data without replacement, therefore each subset contains only unique samples from the dataset.
- Boosting - combines multiple weak learners to reduce bias by sequentially training base models. Each new model is trained to correct the errors made by the previous models. The final prediction is the weighted sum of the base model predictions. Typical representatives are AdaBoost, Gradient Boosting, and XGBoost.

It is applicable for both regression and classification. In classification, plurality voting is used to decide the overall ensemble classification. Ensemble methods provide higher accuracy and better generalization ability compared to single models.

13.1.3 Out-of-Bag error estimation

The out-of-bag error estimation enables the calculation of the generalization error of bagged models without the need for an external validation set. As already mentioned in bagging, bootstrap samples are created by randomly sampling with replacement. Due to that, some data samples are included multiple times in a bootstrap sample, while some are excluded. The data points that are excluded are also known as out-of-bag (OOB) samples. These OOB samples can be used to estimate the performance of the trained model as follows:

- Generate B bootstrap samples,
- For each sample X_B identify the OOB samples X_B^{OOB},
- Train the B-th base model h_B,
- For each data sample $(x_i, y_i) \in D$, collect the prediction from all base models for which was (x_i, y_i) an OOB sample by $\hat{y}_i^{OOB} = \frac{1}{|M_i|}\sum_{B \in M_i} h_B(x_i)$, and

- Calculate the OOB error by comparing the OOB prediction with the true values y_i.

The OOB approach is a valuable method for evaluating the ensemble methods. It provides efficient use of training data, internal estimation of model performance, eliminating the need for a separate validation set, and unbiased estimation of true error.

13.2 Random forest

Random forest is based on constructing a huge number of decision trees, each of which is trained using a unique part of the training data. Each tree within the forest makes a prediction for an output given an input, and the final prediction is formed by collecting the majority vote of all trees in the forest (**Figure 89**). Growing trees on different data subsets prevents decision trees from being overly specialized to the training data, reducing overfitting. Random forest also enables feature importance analysis. The random forest algorithm uses two main techniques to reduce overfitting and improve accuracy:

- Bagging - create B different bootstrap samples by random sampling with replacement. By using replacement, we don't split the training dataset into subsets and train each tree on a different subset. Consider training set D with n samples; we will still feed each tree a training set of size n. But instead of the original training data, we randomly sample a size n with a certain level of data representation. For example, if our training data were [a, b, c, d, e, f, g], then we might give to one of our trees the following list [a, b, d, d, f, g, g]. This means that the same data point can be randomly sampled more than once. For each training dataset X_i a tree T_i $(i = 1, \ldots, B)$ is constructed.
  Therefore, by sampling the data with replacement, the algorithm generates multiple training sets that are slightly different from each other. This type of sampling ensures reduced variance and prevents overfitting.

- Random feature selection - consider that dataset D contains one feature that is highly correlated with the output. In decision trees, most

of the trees will use this feature in the top split, and all trees will look similar. Prediction will be highly correlated, and the average variance will be higher than the average of uncorrelated predictions. Random forest reduces tree similarity and prediction correlation by choosing only from a random subset of features. This ensures more variation among the trees (reduces correlation) and reduces the chance of selecting the same best feature for every tree.

Each individual tree has high variance, but low bias, and averaging these trees reduces the variance and breaks the bias-variance trade-off.

Algorithm Random Forest

Input: dataset D with (x_i, y_i) elements where $i = 1, \ldots, n$, k - number classes, d - number of features, T- number of trees

for t=1 to T:

Randomly select m instances from D with replacement

Randomly select f features from total d features (where $f << d$)

Built a decision tree h_t based on the sampled instances and features

end

#To make prediction for new instance x

if classification task **then**:

$f(x) = argmax_c \frac{1}{T} \sum_{t=1}^{T} I\{h_t(x) = c\}$ *#majority vote across trees*

else if regression task **then**:

$f(x) = \frac{1}{T} \sum_{t=1}^{T} h_t(x)$ *#average of tree prediction*

end

end

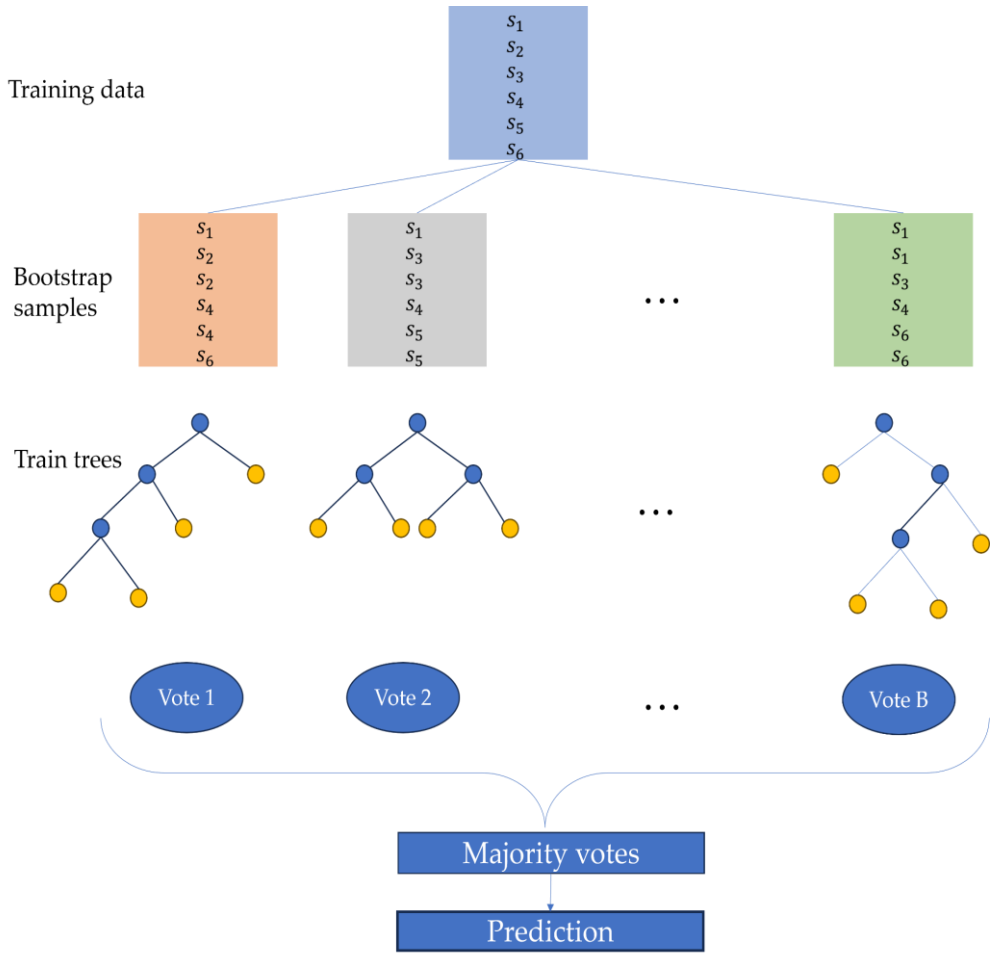


Figure 89 Random Forest algorithm

13.2.1 Decision tree for regression

For regression tasks, a commonly used criterion is the Sum of Squared Errors (SSE) to select the optimal feature f and optimal threshold t. SSE measures the variance within a node, and the aim is to detect feature-threshold pairs that minimize SSE after the split. This algorithm for building the regression tree is given below.

Algorithm Build tree regression

Input: dataset D with (x_i, y_i) elements, k - number classes, f-features

Return: decision tree

function BuildTreeRegression(D):

if stopping criterion is met **than**

return leaf node with the class label

best_gain=[]

for j=1 to d: *# loop over all features*

for m in possible_values(F_j): *#loop over possible values (split points) of feature*

$SSE(D) = \sum_{i=1}^{n} \left(y_i - \underline{y}\right)^2$ *#calculate SSE for the parent node before split*

$D_{Left}, D_{Right} \leftarrow split(D, F, \theta)$ *#temporarily partition the data into left and right subsets*

$SSE(D_{left}) = \sum_{i \in D_{left}} \left(y_i - \underline{y}_{left}\right)^2$ *#calculate SSE for each node*

$$SSE(D_{right}) = \sum_{i \in D_{right}} \left(y_i - \underline{y}_{right}\right)^2$$

$gain = \frac{N_{left}}{N} SSE(D_{left}) + \frac{N_{right}}{N} SSE(D_{right})$ *#test all possible combinations*

if $gain > best\_gain$:

$best\_gain = gain$

$F_{best} = F_j$

$\theta_{best} = m$

$D_{LeftBest} = D_{Left}$

$D_{RightBest} = D_{Right}$

left_child← $BuildTree(D_{LeftBest})$

right_child← $BuildTree(D_{RightBest})$

return Node(F_{best}, θ_{best}, left_child, right_child)

14 NEURAL NETWORK

A Neural Network (NN) represents a system that utilizes a network of functions to identify underlying patterns and learn relationships in the data, and apply acquired knowledge to map input to output. It consists of layers of interconnected simple computational elements called neurons.

14.1 Perceptron

The perceptron, developed by Rosenblatt, is the simplest model of an artificial neuron. It is a building block of NNs. The perceptron (**Error! Reference source not found.**) takes input values $x_i \;\; i \in \{1, \ldots, d\}$, creates a linear combination of the n input variables and corresponding weights w_i , and sums them up, i.e.,

$$z = f(x) = \sum_{i=1}^{n} w_i x_i + b$$

Weights represent the importance of the input to the output; the larger the weights, the greater the influence of the input feature on the output. After that, the results are then passed through a differentiable, nonlinear activation function to produce the output. Therefore, perceptron computes a linear combination of inputs and then applies the nonlinear function.

As already mentioned in the Section 9.10, the choice of activation function primarily depends on the assumed data distribution and the type of the problem. In the original Rosenblatt [4] perceptron, the activation was a step function, so the binary output is determined by

$$h(z) = \begin{cases} 1 & if \sum_j w_j x_j > threshold \\ 0 & if \sum_j w_j x_j \leq threshold \end{cases}$$

Then the output is given by

$$y = h(f(x))$$

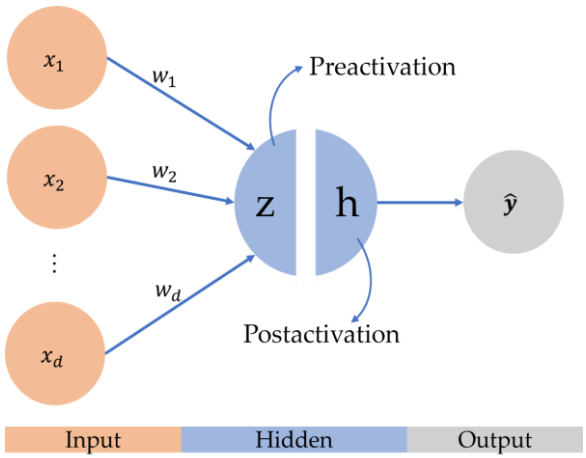


Figure 90 Perceptron

For example, you are deciding whether to go hiking. There are three factors that influence your decision:

- Weather x_1 were $x_1 = 1$ if the weather is good, or $x_1 = 0$ if it rains.
- A company x_2 where $x_2 = 1$ if your friend wants to go, and $x_2 = 0$ if your friend doesn't want to go.
- Shoes x_3 were $x_3 = 1$ if you have proper shoes, and $x_3 = 0$ if you don't have proper shoes.

You can use perceptron to model this decision. If all three factors are equally important to you, then the weights are equal $w_1 = w_2 = w_3$.

However, let's say that there is no way you would go hiking without proper shoes. Then you can choose a weight $w_3 = 0.7$ for shoes, and $w_1 = 0.4$, $w_3 = 0.25$ for other factors. Finally, you choose a threshold of 0.5 for a perceptron. Let's say that $x_1 = 0$, $x_2 = 0$, and $x_3 = 1$ then $0 \cdot 0.25 + 0 \cdot 0.3 + 1 \cdot 0.7 = 0.7$ since $0.7 > 0.5$ the decision is to go hiking.

The larger value of w_3 indicates that having the right shoes matters a lot to you, much more than weather or company. Therefore, the perceptron will output 1 if you have proper shoes and 0 otherwise. It makes no difference whether your friends want to go or not, regardless of the weather.

The lower the threshold is, the easier it is for the perceptron to output a 1.

The weights and threshold values are parameters of the neuron, and by adjusting these values, we can obtain different models. To simplify perceptron, the sum is changed by the dot product, and the threshold is moved to the other side of the inequality, also referred to as bias b ($b - threshold$). Taking that into account, the perceptron can be rewritten as

$$y = \begin{cases} 1 & if\ w^T x + b > 0 \\ 0 & if\ w^T x + b \leq 0 \end{cases}$$

However, threshold logic is rigorous. For example, let's consider the classification of satellite images, where we aim to detect forests solely based on NDVI. The input to the model is the pixel value. If the threshold is 0.7 and the weight is 1, the pixel with an NDVI value of 0.71 would be classified as forest. However, the pixel with an NDVI value of 0.69 would be classified as non-forest. This behavior is not the result of the pixel nature or the chosen perceptron parameters; rather, it is due to the characteristics of the used activation function. For most real-world applications, small changes in the input value don't produce sudden changes. Taking that into account, we want small changes in weights and bias to cause only small changes in the network result (i.e., we want a smooth decision function that gradually changes from 0 to 1). To overcome this limitation of perceptron, the sigmoid neuron is introduced (**Figure 91**).

The output of a sigmoid neuron is given by

$$y = \frac{1}{1 + e^{-(b + \sum_{i=1}^{n} w_i x_i)}}$$

Introducing the sigmoid activation function, there are no sudden changes around the threshold, and the output is a real value between 0 and 1, which can be interpreted as a probability. In contrast to the step function, which is not smooth, not continuous, and not differentiable, the sigmoid function is smooth, continuous, and differentiable everywhere. This is extremely important since it enables the use of gradient descent, allowing the network to learn effectively.

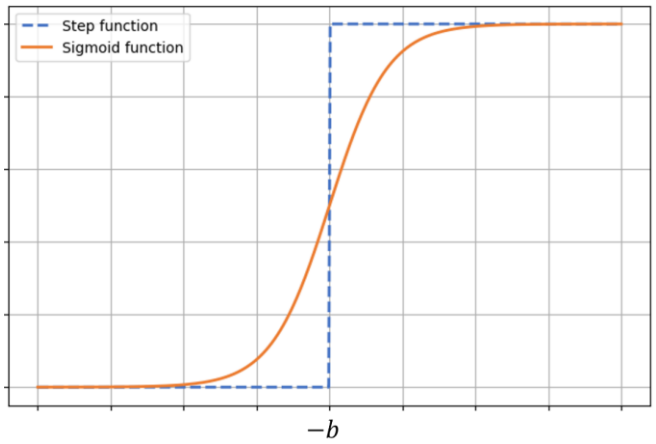


Figure 91 Sigmoid vs step activation function

14.2 Network architecture

A layer, created by composing multiple neurons, represents the fundamental data structure in NNs. Generally, the network contains three types of layers (**Figure 92** (d)):

- Input layer - a set of neurons that directly receives the information from the training dataset,
- Hidden layer - allows the network to learn non-linear patterns between input and output data by transforming input data into a high-dimensional space. neurons that transform training data to extract patterns captured in weights, and
- Output layer - neurons that make predictions based on the input data. Predicted value can be categorical, binary, or continuous, which is controlled by the activation function applied on the output layer

The layers are connected as an acyclic graph (i.e., there are no cycles or closed loops) where the output of one layer represents the input to the next layer. Such networks are also known as feedforward networks since information in the network flows in one direction from the input layer, through the hidden layers used to define f, and finally to the output y. Therefore, there are no feedback connections in which the model's outputs are fed back into itself.

A feedforward NN can be viewed as a composite of many different nonlinear functions that takes input variables x_i and transforms them to produce the

output variables y_k. Each training data point x contains a label $y \approx f^*(x)$. During training, the value of parameters will be adjusted to provide the best approximation of the real function f^* that maps input to output. For example, in classification, we aim to approximate a function that maps a surface reflectance vector x to a class y.

Training data samples directly specify what output layer should produce (i.e., values close to y). Since the behavior of other layers is not directly specified by the training data, they are referred to as hidden layers. The number of hidden layers determines the depth of the model. Each layer consists of neurons that work in parallel, and each neuron receives input from many other neurons and computes its own activation value.

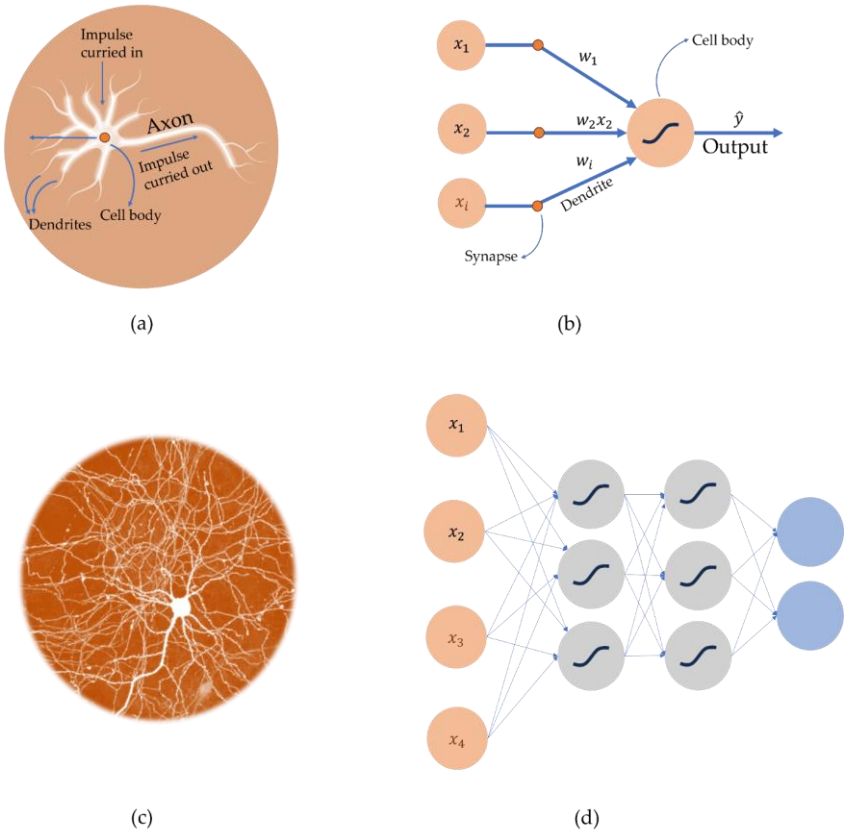


Figure 92 (a) single biological neuron, (b) single artificial neuron, (c) human brain, and (d) feedforward NN (input layer - orange neurons, hidden layer-grey neurons, and output layer - blue neurons)

The architecture of NNs is defined by the number of layers, connections between those layers, and the number and type of neurons per layer. The design of the input and output layers is straightforward. The number of neurons in the input layer is defined by the number of explanatory variables in the training data, while the type of target features specifies the number of neurons in the output layer. For example, in image classification, the input layer matches the shape of the image (height, width, and number of bands) while the output layer matches the number of classes. For regular networks, the most common layer type is the fully connected layer, where each neuron is connected to every neuron in the neighborhood. Neurons within the same layer are not connected. The most challenging task in designing NNs is to determine the number of hidden layers (i.e., depth of network) and the number of neurons per layer (i.e., width of layer).

The universal approximation theorem [43] states that a NN with at least one hidden layer and enough neurons can approximate any continuous function to any desired precision. Taking that into account, the first approach would be to start with two neurons and continue adding more until a satisfactory function approximation is reached. The logical question would be, why do we use more hidden layers?

Although the single-layer network is sufficient to represent any function, the number of neurons may be infeasibly large (an exponential number of units in a shallow network). However, in practical application, using a deeper model can reduce both the number of parameters and the generalization error.

For example, let's consider a rectifier network (using ReLu as the activation function) that defines a linear boundary, splitting the input space into two regions. Imagine the input space is a piece of paper, and the linear boundary is obtained by folding it so that regions coincide and are mapped to the same output. Each hidden unit adds a new fold of activation space on top of the previous layer. By reusing and composing these folding operations from layer to layer, it is possible to obtain an exponentially large number of regions (**Figure 93**). Montufar [44] showed that a deep rectifier can divide the input space into exponentially more regions than a shallow network with the same number of units. This is a crucial property that enables deep networks to compute very complex functions with relatively few parameters.

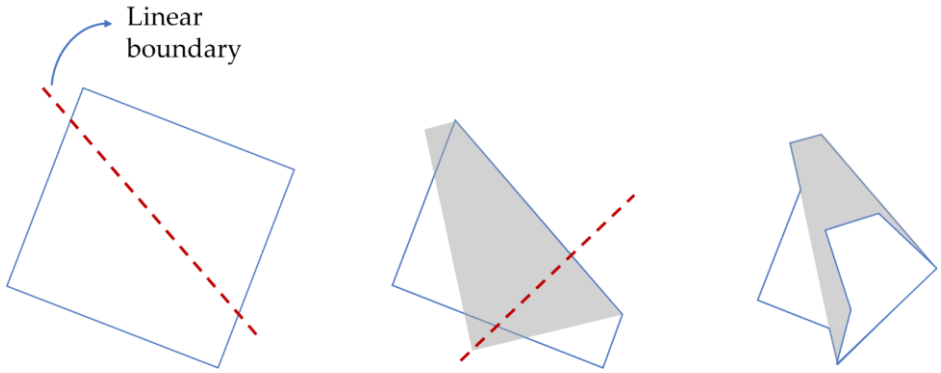


Figure 93 Folding input space. Each hidden unit adds a new fold on top of the previous.

The term Deep Neural Networks (DNNs) refers to a network with multiple layers. Since each layer is a function, a deep network is a composite of many functions

$$f(x) = f_L\left(f_{L-1}\left(\dots\left(f_2\left(f_1(x)\right)\right)\right)\right)$$

where L is the number of layers.

The size of an NN is usually expressed by the number of parameters or the number of neurons. For example, the network presented in **Figure 92** (d) has $3 + 3 + 2 = 8$ neurons (the input neurons are not included). On the other hand, the same network has $4 \cdot 3 + 3 \cdot 3 + 3 \cdot 2 = 12 + 9 + 6 = 27$ weights and $3 + 3 + 2 = 8$ biases, so a total of 33 learnable parameters. With the increase in the number of layers and parameters, the network's capacity also increases. High capacity means that the network can express more complicated patterns and relationships. However, it is easier to overfit the data.

Designing the hidden layer is challenging since there is no clear guidance on how to do so. Usually, the architecture is determined via a trial-and-error approach. Processes typically begin with a relatively small number of layers and units, after which the network is trained and its performance is evaluated on a validation set. We increase the model size gradually if the validation loss is decreasing.

Feedforward NNs represent a vital concept since they form the basis for many more complex architectures. They represent the basis for Recurrent Neural Networks (RNN), which have been extensively used in natural language

applications. Moreover, Convolutional Neural Networks (CNNs), used for object detection and recognition, are a specialized type of feedforward network.

14.2.1 Unit type

There are three types of units: input, output, and hidden units. They can be further differentiated based on the activation function that they use. In addition to structure selection, important aspects in designing NNs represent the selection of unit type.

14.2.1.1 Input units

Each input neuron corresponds to a feature in the input data. NNs use tensors, high-dimensional arrays, as a data structure. Tensor is defined by: rank (that represents the number of axes, for example, a matrix has two axes, a 3D matrix has three axes), shape (the dimensionality of each axis), and data type (it is almost always numerical values, for example, float32, uint8, etc.)

For example, the image is a 3D tensor shape (height, width, number of bands). Pixel values are real numbers, so the data type is float 64. A batch of images is a 4D tensor (number of samples, height, width, number of channels). For example, if the batch size is 64 and we have a satellite image with six channels, with each image consisting of 256 x 256 pixels, then the tensor shape will be (64, 256, 256, 6). The point cloud is also modeled as an n-dimensional tensor depending on the number of features.

14.2.1.2 Hidden units

Each hidden unit computes two functions, one that performs a linear transformation and an activation function that performs the nonlinear transformation. A crucial aspect of designing the NN is the selection of the type of hidden unit. Similar to selecting the number of parameters, the process began by selecting the hidden unit based on intuition, then training the network with that unit, and evaluating performance on the validation set.

Although various types of hidden units are available, piecewise linear ReLu units are the most commonly used default choice. ReLus are typically used on top of a linear transformation. Since the ReLu is inactive for negative values, it is recommended to set all biases to a small positive value, allowing the

derivatives to pass through. Additionally, the generalization of ReLu functions can be utilized, such as LReLu, PReLu, ELu, SELu, or maxout. ReLu (see Section 9.10) and all its generalizations are easier to optimize if the behavior is closer to linear.

Prior to ReLu, most units used sigmoid or tanh activation functions. As already mentioned in 9.10, the tanh is preferable over the sigmoid when the output can be both positive and negative. As already mentioned, the sigmoid function is prone to saturation for both large and low values, making gradient propagation very difficult. Therefore, it is not recommended for hidden units in feedforward networks.

Additionally, softmax units, RBF, softplus (smooth version of ReLu), or hard tanh (bounded tanh) units can be used.

14.2.1.3 Output unit

The hidden units provide a set of hidden features. The output unit also transforms these features to complete tasks for which the networks are designed. The choice of output unit is tightly related to the choice of loss function. The probabilistic interpretation of the network output can provide insight into both selecting the output unit type and the loss function.

For a linear regression model with a Gaussian noise distribution, the error function corresponds to the negative log likelihood.

Consider the input vector x and the linear predictor given as $\hat{y}_i = w^T x_i$. Gaussian noise distribution with variance σ^2 is given as $p(y|x,w) = N(y|\hat{y},\sigma^2)$. To extend the linear models to represent a nonlinear function, we transform the input $\phi(x)$ by a nonlinear transformation $\hat{y}_i = w^T\phi(x_i)$. The log-likelihood for one point n is given by $ln\, p(y|w) = -\frac{1}{2}ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y_n - \hat{y}_n)^2$. Therefore, the negative log-likelihood represents the error contribution from point n, and it is given by (constant $\frac{1}{2}ln(2\pi\sigma^2)$ does not depend on w so it's ignored):

$$E_n(w) = \frac{1}{2\sigma^2}(y_n - \hat{y}_n)^2 = \frac{1}{2\sigma^2}\left(y_n - w^T\phi(x_n)\right)^2$$

If we take the derivative with respect to the parameter vector w of the contribution to the error function from point n:

$$\nabla_w E_n = \frac{1}{2\sigma^2} \cdot 2 \cdot \left(y_n - w^T \phi(x_n)\right)(-x_n) = \frac{1}{\sigma^2}(\hat{y}_n - y_n)\phi(x_n)$$

Therefore, the gradient takes the form of the error multiplied by a feature vector.

The same form will be obtained for a combination of the logistic sigmoid activation function and the cross-entropy loss function, as well as for the softmax activation function with the categorical cross-entropy loss function.

The NN can be viewed as a generalized linear model with a nonlinear activation function. In NNs, usually cross-entropy is used as a loss function. The type of cross-entropy used is defined by the representation of the output. There is a natural link between the type of problem, the error function, and the output unit activation function.

If the conditional probability of the input is a Bernoulli distribution, a sigmoid output unit is combined with the cross-entropy error function. On the other hand, for multiclass classification problems where the output is one of K mutually exclusive and exhaustive values, the softmax unit and categorical cross-entropy loss function are used (see Section 9.9.2). Softmax can be observed as a generalization of the sigmoid for a multinomial output distribution. Like the sigmoid, softmax can saturate when the difference between input values becomes extreme. Consequently, many loss functions based on softmax also saturate, leading to a vanishing gradient, especially when the input is extremely negative. However, if loss functions use log (such as cross-entropy), it undoes the exponential of the softmax, preventing gradient vanishing, making training stable and effective.

14.2.2 Chain rule

Let w be the input of the graph, and we use the same function $f: R^n \to R^n$ at every step of the chain $x = f(w)$, $y = f(x)$ and $z = f(y)$. The derivative $\frac{\partial z}{\partial w}$ measure the rate of change between variables w and z. Consequently if w

change by infestation a smaller value Δw then z will change by approximately $\Delta w \frac{\partial y}{\partial z}$. To compute $\frac{\partial z}{\partial w}$ we apply the univariate chain rule expression, i.e.,

$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} = f'(f(f(w)))f'(f(w))f'(w)$$

where $f(w)$ is computed each time it is needed, leading to exponential redundancy, making a naive implementation of the chain rule infeasible.

Example: Consider a feed-forward network, each unit computes a weighted sum of its input,

$z = wx + b$. The sum is transformed by a nonlinear sigmoid activation given as

$$\hat{y} = \sigma(z) = \left(\frac{1}{1+e^{-z}}\right)$$

Let's use the squared error loss function

$$E_n = E + \frac{\lambda}{2}w^2$$

In order to perform gradient descent, we want to find the derivative of E with respect to w, and E with respect to b.

$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \frac{\partial E_n}{\partial w_{ji}^{(l)}} = \frac{\partial E_n}{\partial z_j^{(l)}}\frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l)}}$$

$$\frac{\partial E_n}{\partial w} = \frac{\partial}{\partial w}\left[\frac{1}{2}(\hat{y}-y)^2\right] = \frac{\partial}{\partial w}\left[\frac{1}{2}(\sigma(wx+b)-y)^2\right] =$$

$$= \frac{1}{2}\frac{\partial}{\partial w}(\sigma(wx+b)-y)^2 = (\sigma(wx+b)-y)\sigma(wx+b)(1-\sigma(wx+b))x_i$$

where $\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) = \sigma(z)(1-\sigma(z))$ for sigmoid activation function.

Similarly,

$$\frac{\partial E_n}{\partial b} = \frac{\partial}{\partial b}\left[\frac{1}{2}(\sigma(wx+b)-y)^2\right] = \frac{1}{2}\frac{\partial}{\partial b}(\sigma(wx+b)-y)^2 =$$

$$=(\sigma(wx+b)-y)\sigma(wx+b)(1-\sigma(wx+b))$$

However, this naive implementation of the chain rule has several disadvantages. As you can see, the above derivations contain many repeated terms. For example, there are six copies of sigma (w x + b), meaning there is a lot of redundant work, even for a simple example.

In deep networks with millions of parameters, there can be an exponentially large number of these redundant computations. We can apply a multivariate chain rule that generalizes the univariate chain rule if the function depends on multiple variables. To enable a computation of derivatives, the value of $f(w)$ is computed once and stored in the variable x and so on. There is no redundant computation, and if the memory requirements to store the value are lower, this approach is preferable. This is the approach that backpropagation uses.

The derivation can be expressed compactly as a Jacobian-gradient product, making the training computationally feasible, especially for large models. For a vector input $x = \{x_1, \ldots, x_n\}$, parameter matrix w, and vector output $y = \{y_1, \ldots, y_k\}$ Jacobian matrix J_{ki} organize all partial derivations into an $k \times n$ matrix (see Section 9.1.2)

$$J_{ki} = \frac{\partial y_k}{\partial w_i} = \left(\frac{\partial y_k}{\partial w_1}, \ldots, \frac{\partial y_k}{\partial w_n}\right)$$

We want to minimize an error function E with respect to the parameter w. The derivative of the error function is given by

$$\frac{\partial E}{\partial w} = \sum_{k,j} \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_j} \frac{\partial z_j}{\partial w}$$

where $\frac{\partial E}{\partial y_k}$ is the gradient of the error with respect to the *k-th* output, $\frac{\partial y_k}{\partial z_j}$ is the Jacobian of layer activation, $\frac{\partial z_j}{\partial w}$ is the Jacobian of z with respect to each parameter.

In a compact matrix form, it is given as

$$\frac{\partial E}{\partial w} = \left(\frac{\partial E}{\partial y}\right) J_y(z) J_z(w)$$

14.3 Backpropagation

As already mentioned, in feedforward networks, each sample in the input data x propagates through hidden units at each layer and finally produces $\hat{y}$. During training, the scalar loss E is produced. This process is called forward propagation.

A NN can be represented as a directed graph where nodes represent the variables and edges represent the learnable parameters. The variables in the node are computed as a function of the variable of the inflowing edge and the learnable parameters associated with it. The exception is the input layer that has fixed values.

Let's consider the NN with L hidden layers. Each neuron in the network computes the weighted sum of its inputs and adds bias, i.e.,

$$z_j^{(l)} = \sum_i w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)}$$

where $a_i^{(l-1)}$ is the activation of the previous layer or x_i for the first layer, $w_{ji}^{(l)}$ is the weight connecting the neuron i in $l-1$ to neuron j in l layer, $b_j^{(l)}$ is the bias of the neuron j in layer l, and $z_j^{(l)}$ is the pre-activation of a neuron j in l layer.

The pre-activation is passed through a non-linear activation function $h(\cdot)$ (such as sigmoid, tanh, ReLU, etc.), and post-activation output is given as

$$a_i^{(l)} = h\left(z_j^{(l)}\right)$$

The forward propagation continues from layer to layer until the network outputs $\hat{y}_k = a^{(L)}$.

For a single training point, the loss function is denoted by E_n. If the training dataset consists of N independent and identically distributed samples, then the total error (E) represents the sum over each data sample in the training set, i.e.,

$$E(w) = \sum_{n=1}^{N} E_n(w)$$

where w represents model parameters, and $E_n(w)$ is the error from the n-th training sample.

To optimize an NN, we need to compute the gradient of the loss function with respect to the parameter w_{ji} . Since the E_n depend on the weight w_{ji} only via summed input z_j the gradient with respect to w_{ji} can be determined by applying the chain rule (see Section 9.1.2)

$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \frac{\partial E_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l)}}$$

The second term is equal to

$$\frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l)}} = \frac{\partial \sum_k w_{jk}^{(l)} a_k^{(l-1)}}{\partial w_{ji}^{(l)}} = a_i^{(l-1)}$$

Taking that into account

$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \frac{\partial E_n}{\partial z_j^{(l)}} a_i^{(l-1)} = \delta_j^{(l)} a_i^{(l-1)}$$

where $\delta_j^{(l)} = \frac{\partial E_n}{\partial z_j^{(l)}}$ is called the delta term for a neuron in a layer l. Based on the above expression, it is evident that the gradient of the loss function with respect to the parameters w_{ji} can be determined simply by multiplying the δ (also known as errors) for the unit at the output end of the weight (layer l) and activation for the unit at the input end of the weight (layer $l-1$).

To determine the delta term, we apply the chain rule of derivatives. Since the loss function depends on the weighted sum of neuron j only via the weighted inputs of all the neurons that are connected to the layer $l+1$

$$\delta_j^{(l)} = \sum_i \left(\frac{\partial E_n}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial z_j^{(l)}} \right) = \sum_i \left(\frac{\partial E_n}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \right)$$

The first derivative is just $\delta_j^{(l+1)}$, while the second derivative

$$\frac{\partial z_i^{(l+1)}}{\partial a_j^{(l)}} = \frac{\partial \left(\sum_k w_{ik}^{(l+1)} a_k^{(l)} \right)}{\partial a_j^{(l)}} = w_{ij}^{(l+1)}$$

Therefore, we get

$$\delta_j^{(l)} = \sum_i \left(\delta_i^{(l+1)} w_{ij}^{(l+1)} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \right) = \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \sum_i \left(\delta_i^{(l+1)} w_{ij}^{(l+1)} \right)$$

since $a_j^{(l)} = h\left(z_j^{(l)}\right)$ the partial derivation outside of the sum is just the partial derivation of the activation function

$$\delta_j^{(l)} = h'\left(z_j^{(l)}\right) \sum_i \left(\delta_i^{(l+1)} w_{ij}^{(l+1)} \right)$$

The δ the term for the output layer needs to be computed first, and it is given as

$$\delta_j^{(L)} = \frac{\partial E_n}{\partial z_j^{(L)}} = \frac{\partial E_n}{\partial a_j^{(L)}} h'\left(z_j^{(L)}\right)$$

For MSE $E_n = \frac{1}{2} \sum_j \left(y_j - a_j^{(L)} \right)^2$

$$\frac{\partial E_n}{\partial a_j^{(L)}} = a_j^{(L)} - y_j$$

therefore

$$\delta_j^{(L)} = \left(a_j^{(L)} - y_j \right) h'\left(z_j^{(L)}\right)$$

The new weights are then updated via gradient descent

$$w_{ji}^{(l)} = w_{ji}^{(l)} - \eta \cdot \delta_j^{(l)} a_i^{(l-1)}$$

where η denote the learning rate.

Algorithm Backpropagation

Input: Training dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$, a multilayer neural network with L layers, parameters w_{ij}^{l}, and activation function h, loss function $E(\hat{y}_j, y_j): R^N \rightarrow R$, learning rate $0 < \eta < 1$, number of epochs

#initialize parameters

for $w_{ji}^{(l)}$ in the network **do**

$w_{ji}^{(l)} \leftarrow a\ small\ random\ number$

#Forward propagation to compute outputs

for i =1 to epochs **do**

for each $(x, y) \in D$ **do**

for $l = 1$ to L **do**

if $l = 1$ **do**

for each neuron j in the input layer **do**

$a_j^0 \leftarrow x_j$

else

for each neuron j in the layer **do**

$z_j^{(l)} \leftarrow \sum_i w_{ji}^{(l)} a_i^{(l-1)}$

$a_j^{(l)} \leftarrow h\left(z_j^{(l)}\right)$

#Backward propagation of the delta term from the input to the output

for each neuron j in the output layer **do**

$\delta_j^{(L)} \leftarrow \frac{\partial E(\hat{y}_j, y_j)}{\partial y_i} h'\left(z_j^{(L)}\right)$

for $l = L - 1$ to 1 **do**

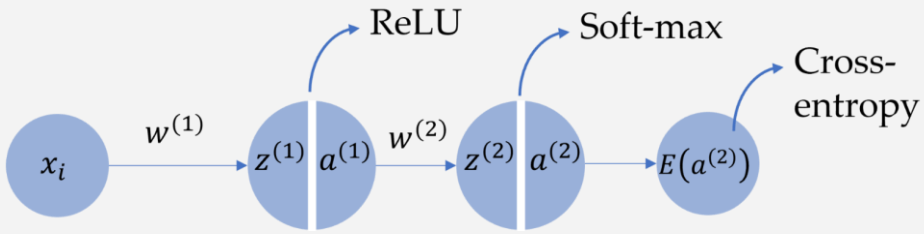
$\delta_j^{(l)} \leftarrow h'\left(z_j^{(l)}\right) \sum_i \left(w_{ij}^{(l+1)} \delta_i^{(l+1)}\right)$

#Update the weights using delta

for each $w_{ji}^{(l)}$ in the network **do**

$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \delta_j^{(l)} a_i^{(l-1)}$

Example: Let's consider the classification of a satellite image, where the input vector $x = [B, G, R, NIR]$, and we want to classify it into four classes. We use the ReLU activation function for the hidden layer and the softmax activation function for the output layer, cross-entropy as the loss function, and 0.1 as the learning rate. The computational graph is given in the image



Let's consider one pixel with $x = [0.12, 0.35, 0.28, 0.58]$, that belong to class 2 $y = [0, 0, 1, 0]$ $w^{(1)} = [0.020, -0.010, -0.009, 0.016]$ and $b^{(1)} = 0$

$w^{(2)} = [0.066, -0.05, -0.075, 0.02]$ and $b^{(2)} = [0, 0, 0, 0]$. Due to simplicity, we will show a network that has one layer and one hidden unit.

$z^{(1)} = \sum w^{(1)} x$=0.0053

$$a^{(1)} = ReLU\left(z^{(1)}\right) = ReLU(0.0053) = 0.0053$$

$$z^{(2)} = w^{(2)} a^{(1)} = [0.0004, -0.0003, 0.0004, 0.0001]$$

$$a^{(2)} = softmax\left(z^{(2)}\right) = [0.2499, 0.2499, 0.2500, 0.2499]$$

$$E = -\sum_{i=1}^{k} y_i log a_i^{(2)} = -log(0.2499) \approx 1.386$$

Backpropagation

$$\delta^{(2)} = a^{(2)} - y = [0.2501, 0.2498, -0.7499, 0.2499]$$

$$\frac{\partial E}{\partial w^{(2)}} = \delta^{(2)} \left(a^{(1)}\right)^T$$

$$\frac{\partial E}{\partial b^{(2)}} = \delta^{(2)}$$

$$w_{new}^{(2)} = w^{(2)} - \eta \frac{\partial E}{\partial w^{(2)}} = [0.066, -0.050, 0.075, 0.019]$$

$$b_{new}^{(2)} = b^{(2)} - \eta \frac{\partial E}{\partial b^{(2)}} = [-0.0249, -0.0249, 0.0749, -0.0249]$$

Backpropagation of error through $w^{(2)}$

$$\delta^{(1)} \equiv \frac{\partial E}{dz^{(1)}} = u \odot g'\left(z^{(1)}\right) = \left(w^{(2)}\right)^T \delta^{(2)} \odot 1_{z^{(1)}>0} = -0.0042$$

where $\odot$ represents the Hadamarov product, i.e. $\delta^{(1)}$ by each input is given as,

$$\frac{\partial E}{\partial w^{(1)}} = \delta^{(1)} x^T = [-0.0056, -0.0165, -0.0132, -0.0273]$$

$$\frac{\partial E}{\partial b^{(1)}} = \delta^{(1)}$$

$$w_{new}^{(1)} = w^{(1)} - \eta \frac{\partial E}{\partial w^{(1)}} = [0.0208, -0.0086, -0.0076, 0.0183]$$

$$b_{new}^{(1)} = b^{(1)} - \eta \frac{\partial E}{\partial b^{(1)}} = 0.0047$$

14.4 Training of a neural network

Designing and training a NN is similar to training any other machine learning model with gradient descent. Optimization (training) is a process of finding weights and biases to get the best approximation of $y(x)$ for all training inputs x.

The first step represents defining the problem that we want to solve (classification, regression, etc). This is crucial information, as it is directly related to the voice of the output layer, the loss function, and performance metrics. Training of the network requires data preprocessing, network architecture design, selection of the optimizer, loss function, and type of output unit.

14.4.1 Preprocessing

Training NNs requires preprocessing of the raw data before it is fed to the model. Preprocessing steps are domain-specific (specific to image or point

clouds). The most commonly used steps include data normalization, feature engineering, and handling missing values, among others.

Normalization is a preprocessing step that rescales all input data to a consistent range. It enables more efficient learning, as large values can dominate the gradient update, causing very slow convergence (see Section 9.11 for more details).

A commonly used technique is min-max normalization, which scales values into the interval [0, 1]. It is mathematically defined as

$$x_i^n = \frac{x_i - min(x)}{max(x) - min(x)}$$

where $min(x)$ and $min(x)$ denotes the maximum and minimum values, respectively, in each feature vector.

Additionally, z-score normalization, also known as standardization, can be used. It transforms each feature to have zero mean and a standard deviation of 1. The z-score is calculated as

$$z = \frac{x_i - \mu}{\sigma}$$

Feature engineering is the process of transforming input data before feeding it to the model, making the data better suited to the specific task. It includes methods for feature selection (choosing a set of features that improves model performance), feature synthesis (creating a new feature set from raw data), and feature extraction (transforming features to obtain more representative features for the specific task). For example, we need to train models that classify point clouds into multiple classes. If we feed X, Y, Z coordinates and intensity into a NN model can struggle due to high class overlap (power lines and tree canopies). Adding new features such as surface normal or multi-scale features can significantly improve classification accuracy.

Before deep learning, feature engineering was a crucial step, as ML algorithms (such as RF or SVM) are not able to detect useful features on their own. By introducing DNNs that can automatically transform raw data into useful features, the influence of feature extraction on model performance is significantly reduced. However, the ability of deep learning models to learn features independently relies on having a substantial amount of training data

available. Due to this, feature engineering can still be used to solve tasks with fewer resources or with far less data. If only a small training dataset is available, then feature engineering becomes crucial.

14.4.2Initialization

Training a NN is similar to training any other machine learning model. However, the nonlinearity of a network function causes the loss function to become nonconvex. Due to nonconvexity, deep learning models are typically trained using an iterative, gradient-based optimizer (see Section 9.1) that requires the initialization of model parameters. In contrast to convex loss functions that guarantee convergence from any initial point, nonconvex loss functions are highly sensitive to the initial parameters. The initial point has a strong influence on how optimizers navigate the loss landscape, determining whether the algorithm converges at all, the convergence rate, and the quality of the final solution (whether the algorithm converges to a local or global minimum). Therefore, parameter initialization plays a significant role in NN learning, and it can be crucial for maintaining numerical stability and achieving a high convergence rate. The selection of the initialization scheme is closely related to the choice of activation function.

In a **Fully Connected Network (FCN)**, all neurons have the same input, and their response will be identical regardless of their position. Therefore, permutation of neurons in the hidden layer will not change the network function but yield a different point in parameter space. This property of the network is known as the symmetry of the NN.

Consider a FCN with a $d - 1$ hidden layer, and each layer contains n neurons. In each layer, we can permit the neurons in $n!$ different ways so the total number of permutations is $(n!)^{d-1}$. Due to that, the loss landscape will have $(n!)^{d-1}$ identical minima, making it highly redundant. While symmetry is not necessarily harmful, it creates a challenge for optimisation.

Parameter initialization schemes need to break symmetry. If we initialize weights to have the same constant value, all neurons in the hidden layer will have the same response. However, if all neurons compute the same output, then they will also compute the same gradient during backpropagation, causing each neuron to have the same update.

To break a symmetry, hidden units must have different initial parameters, ensuring that neurons compute diverse functions.

Usually, weights are initialized as small random numbers sampled from a multi-dimensional Gaussian or uniform distribution. The choice of scale is crucial, since it directly influences training effectiveness. Although large initial values will lead to higher redundant units, they will avoid losing signal during forward or back propagation. However, large weights result in large outputs of matrix manipulation, causing a vanishing gradient for sigmoid or tanh region (since they saturate for large values) or an exploding gradient if the ReLu activation function is used. Therefore, weights should be large enough to propagate information through the network without vanishing, but small enough to avoid saturation.

It is common practice to initialize weight to be very close to zero, but not zero (for example, a Gaussian distribution with zero mean and a small standard deviation, such as 0.01). In this way, we ensure that all neurons are random and unique at the beginning, allowing them to compute distinct updates during training and enabling them to learn diverse features. However, this initialization is not sensitive to the number of inputs to a specific neuron, i.e., the variance of the output of randomly initialized neurons increases linearly with the number of inputs. Authors [45] and [46]proposed an initialization scheme that maintains the variance of activation constant across all layers, thereby preventing vanishing or exploding gradients. This is achieved by scaling the variance of weights according to the number of inputs. the weighted variance is set to $Var(w) = \frac{2}{n_{in}+n_{out}}$ for tanh/sigmoid [45] and to $Var(w) = \frac{2}{n_{in}}$ for the ReLu activation function [46].

However, initial scaling of the weight can lead to very small individual weights when the layers become large. To address those limitations, Martens [47] introduced a sparse initialization scheme. This scheme limits the number of non-zero weights to each neuron, ensuring a high difference between them and preventing saturation. However, setting most weights to zero can slow down the learning process, and it slows down gradient descent by initializing weights that are too large for certain neurons. Bias is always initialized to zero or small positive values.

14.4.3 Optimization

The loss function evaluates the network's approximation ability by comparing the predicted and true values during training. Those values are averaged to provide a single numerical value. An optimization finds a set of parameters that will minimize a loss. The performance metrics are used to evaluate the model's performance after training, with the aim of providing deeper insights into model performance, assessing the model's generalization ability, and comparing different models. The most commonly used loss functions and performance metrics are presented in Section 9.9. The last-layer activation function should be selected based on the type of problem. Most commonly used combinations are shown in the **Table 17**.

Table 17 Selection of lost function and last-layer activation based on the problem type

Problem	**Loss function**	**Last-layer activation**
Binary classification	BCE	sigmoid
Multi-class classification	CCE	softmax
Regression	MSE	linear

One of the fundamental issues in training machine learning is the trade-off between optimization and generalization. As already mentioned, generalization refers to the model's ability to perform well on unseen data. Therefore, we want our model to have good generalization, but we cannot directly control generalization; we can only control optimization. Due to that, deep learning models tend to perform well on training data, but this really changes when it comes to fitting the test dataset. So correlation between optimization and generalization is extremely important. At the beginning of the training process, the loss on training and test data is correlated. The network does not model all relevant patterns in the training data, and there is still room to improve, so the model is underfitting (see Section 9.13).

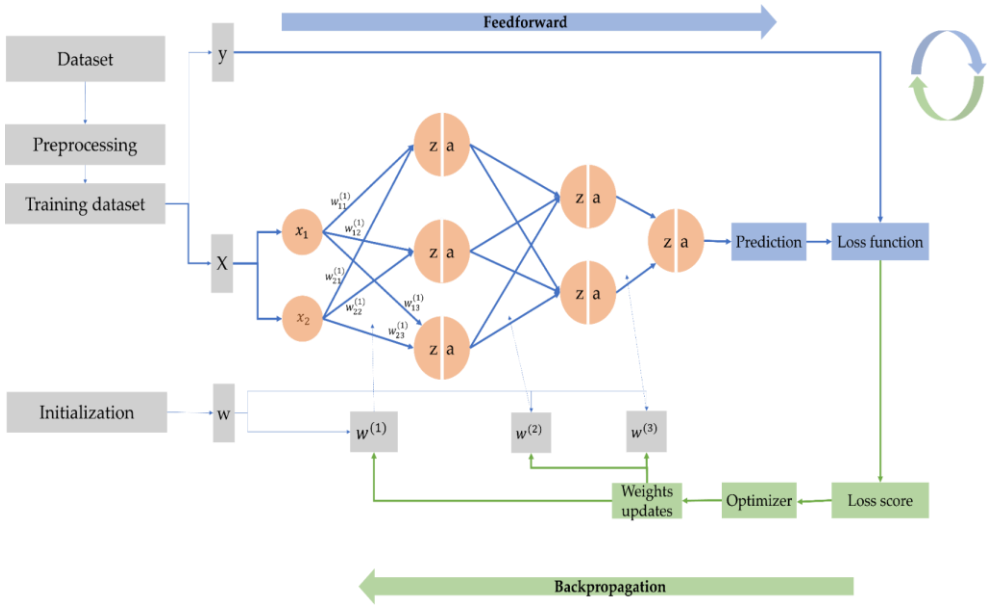


Figure 94 Training a neural network

However, after a certain number of iterations, models start to overfit (i.e., they learns irrelevant patterns). The best solution for overfitting is to enlarge the training dataset. However, often this is not possible and we use regularization (see Section 9.14) such as L1/L2 regularization, dropout, early stopping, etc.

Once the model design and regularization technique are chosen, we train the network using backpropagation to calculate the contribution of each parameter to the error (**Figure 94**). The optimizer utilizes the gradient updates from backpropagation to adjust network parameters, thereby minimizing a loss function. The commonly used optimizers are: SGD, Adam, and RMSProp (please see Section 9.5-9.7).

Example: Estimate the Water Quality Parameters (WQP) concentration from satellite images and machine learning algorithms.

Monitoring of WQP in inland water bodies - such as Turbidity, Total Suspended Solids (TSS), Total Nitrogen (TN), Total Phosphorus (TP)-is performed by modeling the relationship between satellite-derived surface reflectance and corresponding in-situ water quality observations employing deep learning algorithms. This example is published in [48].

Preprocessing: Field campaigns conducted between 2013-2017 by the Agency for Environment protection of Serbia were used as in-situ data, while the time series of Landsat 8 and Sentinel 2 are accessed through Google Earth Engine.

In-situ data contain the coordinates of monitoring stations along with corresponding WQP concentrations. The locations of the monitoring stations were visually inspected using Landsat 8 satellite images. In total, 23 monitoring stations were selected for this study, resulting in 313 and 408 data samples for Landsat 8 and Sentinel 2, respectively.

For both Sentinel 2 and Landsat 8, surface reflectance values were extracted for the B, G, R, NIR, SWIR1 and SWIR 2 bands, along with the image acquisition dates for each monitoring station. In addition to spectral bands, several spectral indices were calculated, including G/R, NIR/R, NIR/B, R/G, R/B, NDVI, NDWI, and Normalized Difference Turbidity Index (NDTI) are calculated. The maximum allowed time difference between water sampling and satellite overpass was set to three days, and only matching pairs within this interval were retained. The input data were normalized to a [0,1] range by using min-max normalization. The data was split into 70 % for training, 10 % for validation and 20 % for testing.

Training: ANN and SVM algorithms are used to model the relationship between spectral features and in-situ WQP concentration.

ANN: The number of the input neurons was selected to be equal to the selected input bands that had a strong correlation with WQPs, and the number of output neurons was selected to be one. The trial-and-error approach was used for the selection of a proper number of hidden neurons i.e. the number of hidden neurons was modified in order to minimize RMSE at the training phase. In order to reduce overfitting the performance is monitored on validation dataset and early stopping is used. The hidden layer used the Tanh activation function to capture the nonlinear relationship between input and output variables. The learning rate and decay rate were determined through grid search (Learning rate: [0.0001, 0.001, 0.01, 0.1]; Weight decay: [0.000001, 0.00001, 0.0001]). To avoid

overfitting, early stopping was used. The weights assigned to each connection were randomly generated before training and updated using a backpropagation algorithm. The final model configuration corresponding to the lowest validation error was selected.

SVM: For comparison, a SVM regression model was developed using Radial Basis Function (RBF) kernel to handle nonlinear relationship. The model parameters the kernel width (γ) and the regularization parameter (C)- were optimized throught a grid search combined with cross-validation on the validation dataset. The optimal configuration (C = 100 and γ=0.5) provided a robust model minimizing overfitting.

The algorithm performance was evaluated using NRMSE. The results of accuracy assessment are presented bellow

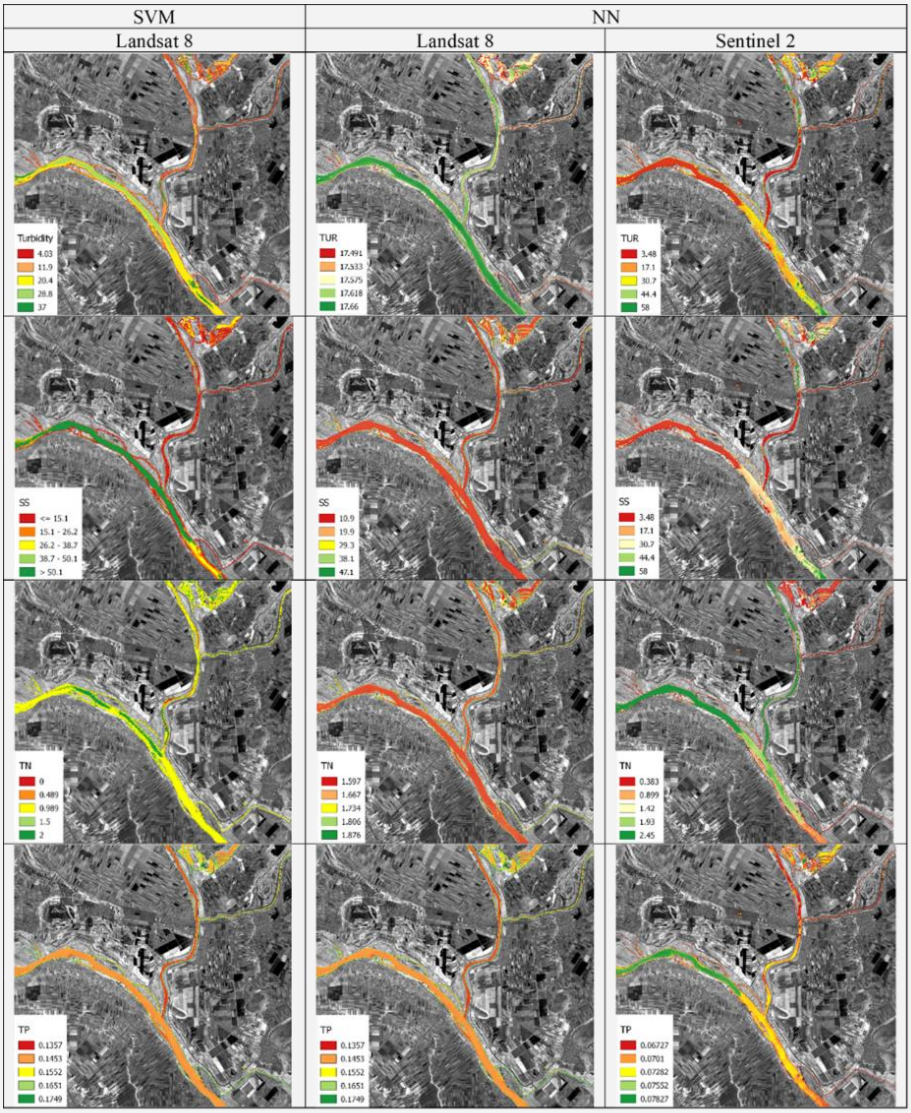
Parameter	Landsat 8		Sentinel 2	
	ANN	SVM	ANN	SVM
	NRMSE	NRMSE	NRMSE	NRMSE
Turbidity	6.85%	5.04%	3.18%	7.28%
TSS	10.81%	6.65%	3.72%	6.88%
TN	12.56%	6.88%	12.82%	7.38%
TP	12.76%	9.72%	10.27%	6.33%

The SVM model demonstrate greater robustness to small data samples and mixed pixels which likely explains its higher accuracy compared to the ANN. However, the results for Turbidity and TSS indicated that the ANN more accurately predicted parameters with a wider dynamic range. Furthermore, the accuracy of the ANN was observed to increase with the larger training datasets, confirming its sensitivity to data volume.

Due to higher spatial and temporal resolution, Sentinel 2 represents a more suitable alternative for water quality monitoring. It provides higher accuracy and 25 % larger data set in 50 % less acquisition time compared

with Landsat 8. On the other hand, the Landsat imagery enables the analysis of long-term trends, seasonal variations and historical changes in surface water quality.

Prediction: The trained ANN and SVM models are applied to water bodies extracted from Landsat 8 and Sentinel 2 imagery to estimate the concentration and spatial distribution of selected WQP. The spatial variation of WQP across the study area is presented in the maps below.



15 CONVOLUTION NEURAL NETWORK

Convolution Neural Networks (CNN) are introduced as alternatives to regular NN. Although FCN has been extensively used in early stages of deep learning, dealing with high-dimensional data is challenging. Consider a satellite image of size 256 x 256 pixels and 6 bands corresponding to B, G, R, NIR, SWIR1, and SWIR2. In a feedforward NN, each pixel corresponds to the node in the input layer, resulting in 256 x 256 x 6 = 393 216 nodes. Since each node of one layer is connected to each unit in the next layer, computation of just one neuron in the first layer will require 393 216 weights. The high number of connections is computationally expensive, and it requires a huge amount of training data to avoid overfitting.

To address these challenges, we can utilize any existing structural knowledge about how inputs should map to outputs, even before we see any data. This concept is based on a prior probability distribution (see Section 11). Since input neurons can connect to any output neuron and the model must learn all dependencies from scratch, the FCN has a very weak prior about the structure of the function. On the contrary, CNN can be viewed as a specialized FCN with an infinitely strong prior on its weights. The priors leverage the invariant properties of the structured data by effectively assigning zero probabilities to connections that violate locality and translation invariance constraints. As a result, CNN is well-suited for processing grid-like structured data such as images or time series. For example, an image is a 2D grid of pixels, audio can be represented as a special form of time series data (1D sequence where values are sampled at regular intervals in time), while video has a 3D grid structure (height x width x time plus channels).

The CNN has completely revolutionized image classification, object detection, and pattern recognition, leading to accuracy that was unattainable with traditional machine learning algorithms.

15.1 CNN architecture

Similar to NN, CNN has an input layer, an output layer, and many hidden layers in between. The hidden layer of a CNN consists of three different types of layers: a convolutional layer, a pooling layer, and a fully connected layer (also known as a dense layer).

The convolution layer applies multiple learnable filters in parallel to produce a set of linear activations. A nonlinear activation function, such as ReLU, is then applied to each linear activation, followed by a pooling layer that reduces the spatial dimensionality of the output feature maps. Those operations are performed over a large number of layers, where each layer detects different features, gradually transforming a raw input into a more abstract representation. Finally, once the size of feature maps becomes reasonably small, fully connected layers combine all the extracted features from the previous layer with all the features in the next layers into a higher representation used for the final prediction.

The key aspects of designing CNN architecture are the selection of network depth (number of layers), and width (number of filters in the convolution layer and number of neurons in the FC layer). The network size and number of associated parameters directly influence its performance and complexity.

The depth and width of the network are functions of several factors, including task complexity, size of training dataset, and computational resources. Each convolution layer increases the number of feature maps that compensate for the amount of information lost due to the pooling layer. Deeper networks can capture more complex patterns but demand a large training dataset. Moreover, input data resolution should be taken into consideration, as a high-resolution image requires more depth to progressively downsample and extract features at multiple scales. In most CNNs, the width increases with depth. A common rule is to start with 32-64 filters and double them every few layers.

As in NN, there is no clear guidance on how to design an optimal CNN architecture (**Figure 95**). It is usually determined based on expert opinion and a trial-and-error approach.

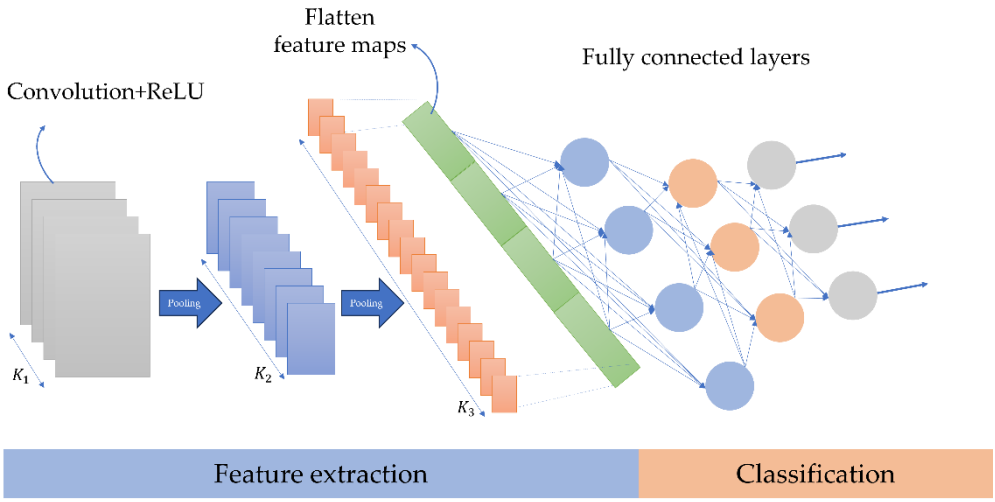


Figure 95 CNN Architecture. K1,K2,K3 denote the number of kernels in each convolution layer

15.1.1 Convolution operation

Convolution is a mathematical operation that describes a rule for combining two functions to form a third one. It is crucial in signal and image processing. Convolution operates on two signals, where one is considered the input and the other is a filter (or kernel) applied to the input signal, producing a third function as output, also known as a *feature map*.

Let's consider the 1D convolution. For example, tracking the location of a car by using GNSS. A sensor will provide a single output $x(t)$ that represents the position of the car at time t. Let's say that GNSS sensors provide a measurement at regular intervals, once per second. If a signal gets noisy due to environmental or sensor-induced errors, we can obtain a more precise car position by averaging several measurements (**Figure 96**). Since more recent measurements are more relevant, we need to compute a weighted average that gives higher importance to recent measurements by using a weighted function $w(a)$. By applying the weighted average operation at every moment, we obtain a third function s that provides a smoothed estimation of the car position. This operation is called convolution, and it is given by

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da$$

where a is the age of measurement.

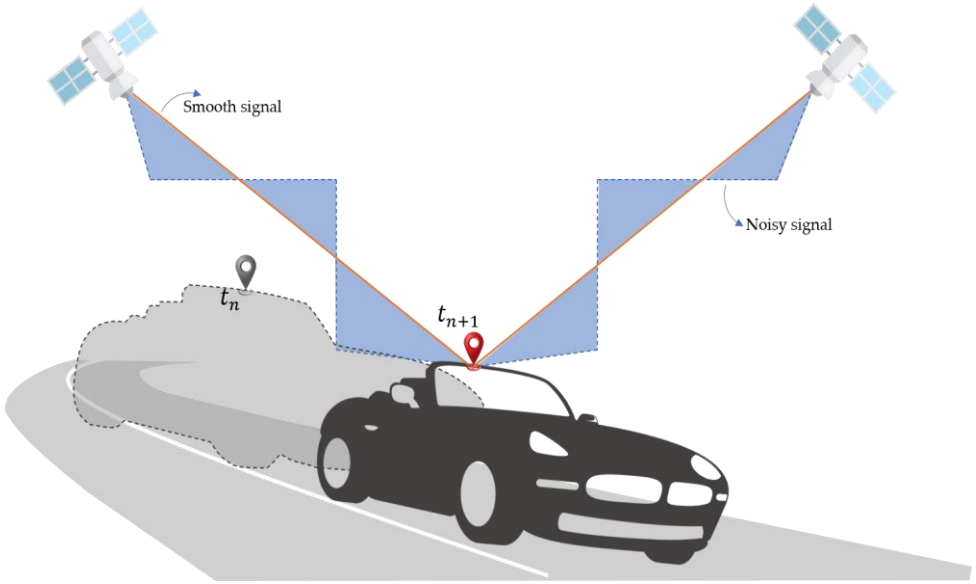


Figure 96 Tracking the car position by GNSS

In deep learning, inputs are usually a multidimensional array (tensor) of data, and the filter is usually a multidimensional array of parameters that are adapted by the learning algorithm. For example, if input is a 2D image denoted by I and we use a 2D kernel K, then the convolution is given by:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

When designing and training a NN for image classification, there are several key structural knowledge that convolution layers use to improve the accuracy. Firstly, nearby pixels are more strongly correlated than more distant pixels. Convolution layers exploit this property by extracting local patterns within a small 2D window of the input. Therefore, only a local neighborhood is affected by neurons from the previous layer. The spatial extent of the neighborhood is also known as the receptive field or filter size. The size of the filter, denoted as f, is typically set to 3x3 or 5x5 pixels. The filter size must be the same within a layer, but it may vary between layers. Although each neuron is connected only to the local region, the extent of the filter along the depth axis is always equal to the depth of the input volume. Focusing on a small receptive field enables convolution to detect simple patterns such as edges, corners, and textures.

For example, consider an input image with a size of 256x256x6. If the filter size is 3x3, each neuron in the convolution layer will have a total of 3x3x6 = 54 connections, plus one bias parameter.

Secondly, the visual world is translation invariant, meaning that the same pattern can appear anywhere in the image. For example, after learning a certain pattern in the upper right corner of an image, convolution can recognize it anywhere, such as in the lower left corner. In contrast to fully connected networks, where each weight is used only once, convolution layers leverage parameter sharing, where the same filter is applied across the entire input. Parameter sharing ensures that learned parameters in one region can be reused to recognize the same patterns in another, rather than requiring a separate set of parameters for every location (**Figure 97**). This significantly reduces the number of parameters and training samples required to learn representations, thereby improving the network's generalization ability.

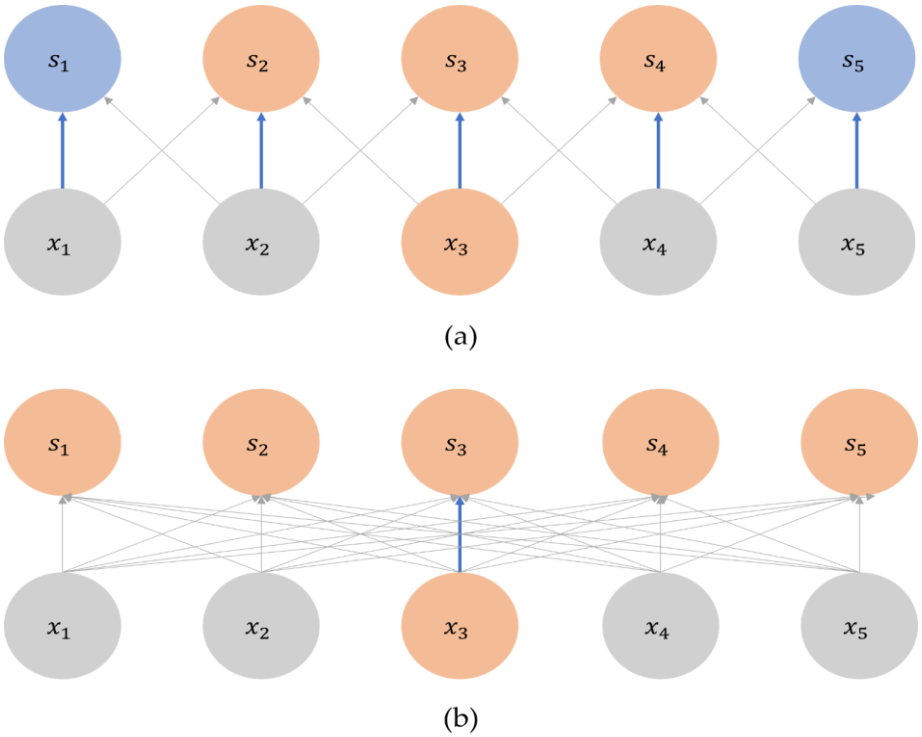


Figure 97 Parameter sharing (a) convolution - the blue arrow represents the center of the 3-element kernel. The same parameter is used at all input locations. The input neuron x_3 affects only three outputs. (b) fully connected - there is no parameter sharing, and the blue arrow is used only once. The input neuron x_3 influence all outputs.

Finally, the images are fundamentally spatially hierarchical, where low-level features combine to form higher-level representations. The lower convolutional layer will learn small local patterns, such as edges. A second convolutional layer will learn larger patterns composed of features from the first layer, and so on (**Figure 98**). This allows convolution to efficiently learn increasingly complex and abstract visual concepts.

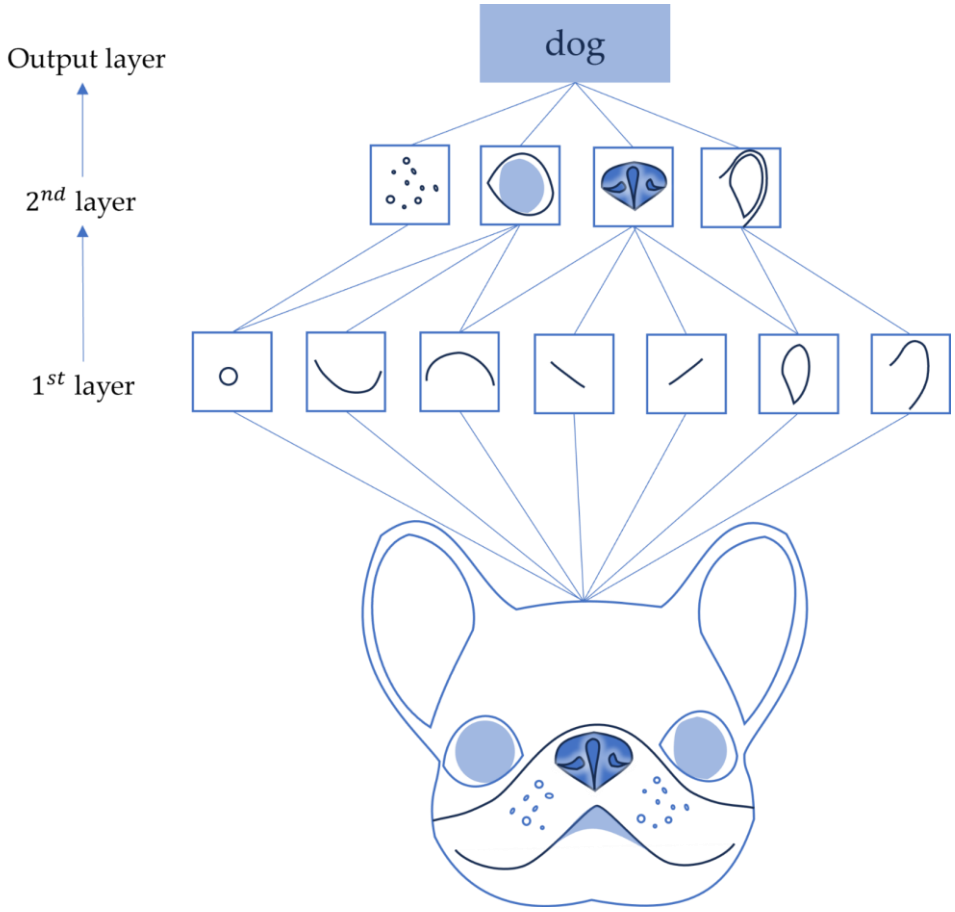


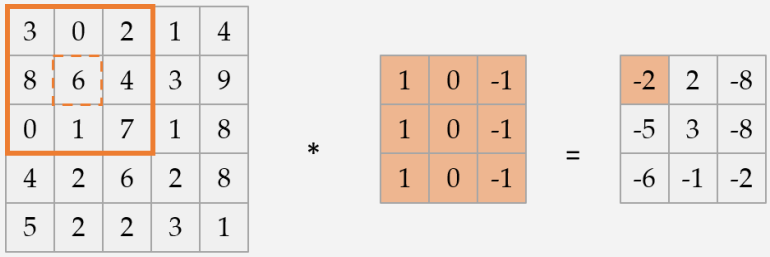
Figure 98 Image hierarchy. Edges combine into corners, corners combine into more specific features such as eyes or ears, which combines into complex concepts such as dogs

Convolution works by sliding small windows over the 3D input feature maps. At each position, the kernel is centered on a pixel of interest, extracting the 3D patch of the local neighborhood (**Figure 100**). Each such patch is transformed by an element-wise multiplication with the learned weights of the convolution filter and summed up to a single scalar value. All of these outputs are then spatially reassembled into a 3D output feature map.

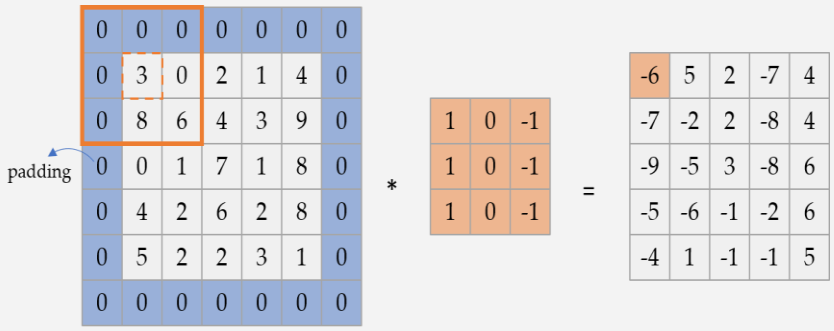
Each spatial location in the output feature map corresponds to the same location in the input feature map. The size of the output feature map is controlled by three hyperparameters: depth, padding, and stride. Padding and stride control the spatial extent of output (width and height), and depth controls the number of channels.

The convolution can be computed only in positions where all components of the filters have corresponding input components. This is not the case with the input boundaries. Padding controls the spatial dimensionality of output by adding additional pixels, usually zeros, around the border of the input map before applying the convolution operation. Therefore, if no padding is used, the output size gradually shrinks after each layer, leading to a loss of information around edges.

Consider applying the kernel size 3x3 to the 5x5 input feature map. Since the kernel can only be centred where a full 3x3 window fits inside the input, output will be reduced to 3x3.



By adding a one-pixel-wise border of zeros (padding=1) enables application of the kernel at the edges as well and the output size remains 5x5.



Moreover, in images the pixel values change gradually, which means that it is not always necessary to apply a filter at every location. Instead certain blocks can be skipped. Stride parameter controls the number of steps the kernel moves each time it slides across feature maps (**Figure 99**). The stride one means that the kernel shifts by one pixel at a time, covering every possible location. If stride increases to 2, the kernel skips one pixel between positions, so the output feature map becomes smaller because fewer local neighborhoods are visited. The larger stride reduces spatial dimension and computational cost but also leads to loss of fine-gradient details.

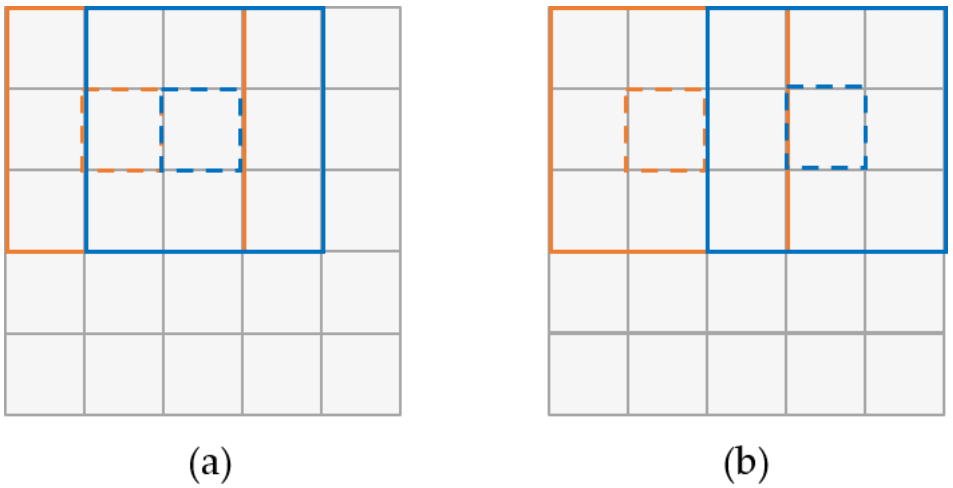


Figure 99 (a) stride=1, (b) stride=2

For example, applying a 3x3 kernel with strid 1 on 7x7 input gives the output size of 5x5, while using stride 2 reduces it further to 3x3.

The depth of output represents the number of filters used by the convolution. If we use *k* filters, then the result is *k* new images. Increasing the number of filters enhances the network's capacity to learn various types of features at the same spatial locations.

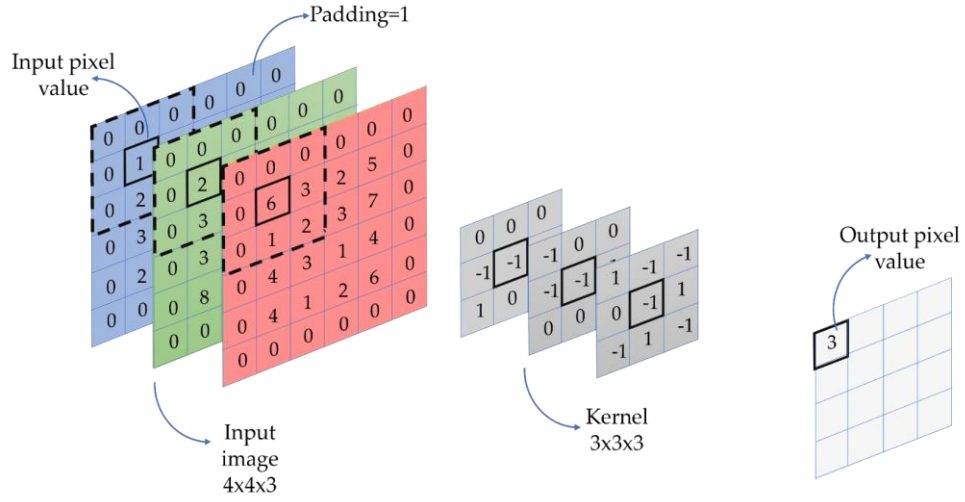


Figure 100 Convolution operation using a 3x3x3 Kernel

15.1.2 Pooling layer

The output of the convolution layer is the input to the pooling layer. The pooling layer reduces the spatial dimensionality while preserving the most important information. It works by sliding a small receptive field across the input and computing a single output for each region. Different approaches, such as max pooling and average pooling, can be used. Typically, a max polling (**Figure 101**), which returns the maximum value of the elements within the receptive field, is used. So, the pooling layer indicates that the feature has been detected somewhere within the receptive field, ensuring the translation invariance. Therefore, from a probability perspective, it can be observed as a prior assumption that output is insensitive to small translations of the corresponding regions in the input.

To ensure a balance between dimensionality reduction and information preservation, the 2x2 max pooling with a stride of 2 has been the standard approach. While a 2x2 pooling operation with a stride of 2 reduces both the height and width of the feature map by half, the depth remains unchanged, allowing the network to maintain all feature representations learned by previous convolutional layers.

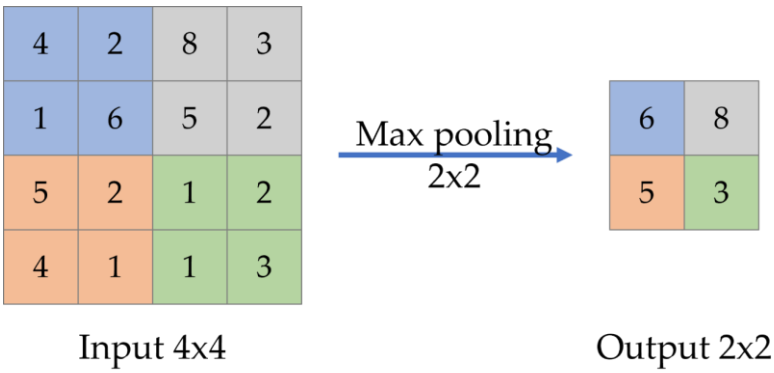


Figure 101 2x2 Max pooling

Contrary to the convolution layer, the pooling layer does not have any learnable parameters. Pooling reduces both the number of parameters and computations in the network, which helps to control overfitting.

Moreover, the pooling layer helps the network to learn a global representation of an object. Early convolution layers detect local features such as edges, corners, or textures. The pooling layer gradually aggregates this local information, producing the feature maps with coarser spatial resolution. By repeatedly applying pooling, the network essentially compresses information from the entire input into a smaller spatial map where each activation is sensitive to a larger receptive field.

For example, in an image of a dog (**Figure 98**), the early layers will detect the edges of the eye or ear. After several convolution and pooling operations, output feature maps encode information from many of these local patterns together, allowing the network to create a representation of the dog as a single object.

15.1.3 Fully connected layer

A fully connected layer is typically used as the final layer in a CNN architecture. Each neuron in this layer shares its weights with all other neurons of its preceding layer, enabling the learning of global relationships between features. Once convolution and pooling layers significantly reduce the size of the feature map, the fully connected layer integrates high-level features into a global representation and uses this representation to produce

the final prediction. In classification tasks, the fully connected layer is typically followed by a softmax activation function.

15.2 Training convnet

15.2.1 Preprocessing

Images are usually very large with tens or even hundreds of megapixels. Since the convolution operation is performed across the whole image, the computational and memory requirements grow quadratically with image size. This can make training very slow or even impossible on standard GPUs. To address this, the standard practice is to split images into small patches, usually 256x256 pixels, depending on available GPU memory.

15.2.2 Backpropagation

As already mentioned, CNNs are deep feed-forward networks where an input, such as an image, is passed through a convolutional layer to extract local features, an activation function to introduce non-linearity, a pooling layer that reduces dimensionality, and a fully connected layer that combines the extracted features to produce an output. This output is compared with the ground truth data using a loss function to calculate an error. The entire network is trained through error minimization using backpropagation to calculate the gradient of the error function. In CNN, the gradient is computed with respect to both the input and the filter.

In the fully connected layers, the gradient is propagated backward in the same way as in standard neural networks. The propagation of the error signal through the convolution layer involves a slight modification of the usual backpropagation algorithm to ensure that the shared weights constraints are satisfied. For each convolution filter, the gradient is represented as a convolution operation between the error signal from the next layer with the corresponding region of the input feature map. Since a filter is applied across many positions, its weight updates are obtained by summing the contributions from all locations where the filter was applied. This ensures that each kernel learns features that are useful across the entire input space. Unlike the filters that affect all outputs, each input influences one or more of them.

The gradients with respect to the input are also computed by convolving the flipped filter weights (flipping first vertically and then horizontally) with the error signal, allowing the backward flow of information through the network. Each input pixel receives gradient contributions only from outputs whose respective fields. In order to ensure that the spatial dimensionality of the propagated gradients matches those of the input feature map and that every element receives an appropriate gradient update, the gradient of the loss with respect to the output is zero-padded.

Moreover, the pooling layer requires special treatment during backpropagation. In max pooling, the gradient is passed back only to the location of the maximum value selected during the forward pass, while all other positions receive zero gradient. In average pooling, the gradient is distributed equally among all inputs in the pooling region. In this manner, backpropagation ensures that this is the case.

The final step is the parameter update, where the gradients are used to update the weights and biases of the network through an optimization algorithm such as SGD or Adam. This process is repeated over many epochs and batches of training data until the model converges.

15.3 Regularization of CNN

One of the most important characteristics of deep learning is that it can automatically detect important features, eliminating the need for manual feature engineering. This can be done only if a large number of training samples are available. As already mentioned, the deeper the network is, the more complex patterns it can learn; however, it requires more training data. In training CNN networks, overfitting is the main challenge. Hypothetically, if we had an infinite data set, the model would never overfit. However, it is relatively rare to have a dataset of sufficient size, since labelling data is a time-consuming and financially demanding process.

15.3.1 Data augmentation

To expand the dataset, we can utilize data augmentation. Data augmentation is used to enlarge the training dataset by augmenting the real samples via a number of random transformations. The aim is to diversify the training

dataset without altering image semantics, thereby enabling the model to learn various aspects of the object and enhance its generalization ability.

In remote sensing, images are acquired with different sensors under varying climate conditions, resulting in differences in spectral reflectance and spatial resolution that affect the object's appearance in the image. Different data augmentation techniques, such as geometric transformation (rotation, translation, flipping, scaling, cropping), change of brightness and contrast of image, adding noise (salt and pepper, additive Gaussian, additive Poisson), as well as simulation of clouds, smoke, and fog, have been successfully used to diversify the dataset.

Data augmentation provides better generalization, helps solve class imbalance, and preserves network capacity. However, the inputs remain highly intercorrelated because they originate from a small number of original images. Consequently, it is often combined with dropout to enhance the generalization ability (see Section 9.14.3).

15.3.2 Batch Normalization

One of the main challenges during deep network training is internal covariate shift [49], where the distribution of each input feature map changes due to parameter updates during training. As networks become deeper, the small changes to the network parameters amplify even further. Batch normalization (BN) [39] stabilizes the training process by normalizing sub-networks and layers to maintain a constant variance (see also Section 9.11). It is given by

$$BN(x) = \gamma \odot \frac{x - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

where B is minibatch $x \in B$, $\hat{\mu}_B$ is the sample mean, $\hat{\sigma}_B$ is the sample standard deviation, γ is the scale parameter, and β is the shift parameter. Both parameters are learned during training. The BN layer is incorporated in the network architecture after convolution and before the nonlinear activation function. The BN significantly speeds up the training process, stabilizes training, and reduces both the vanishing and exploding gradient issues. It is less sensitive to changes in learning rate and weight initialization, and often eliminates the need for dropout. Moreover, each mini-batch will be normalized by using different $\hat{\mu}_B$ and $\hat{\sigma}_B$ each time, introducing additional

noise into the training process and leading to the better generalization ability of the model.

15.4 Transfer learning

As already mentioned, CNNs can learn local, translation-invariant features, making them extremely efficient for perceptual problems, and can be easily repurposed for different tasks. In reality, we often have a limited dataset for a new task. Therefore, it is relatively rare to train models from scratch. Instead, transfer learning, which utilizes a pre-trained model's knowledge on another learning task, has been extensively used.

A pre-trained model is a model that has been previously trained on a large dataset. So, if the original dataset is large enough and general enough, then the pre-trained network has a small number of task-specific features, while it shares a common low-dimensional representation across different tasks. Due to that, learned features can be useful in many different problems, even though these new problems may be completely different from the original task.

For instance, you might train a network on ImageNet databases that contain 1.4 million labeled images and 1000 classes (mostly animals and everyday objects), and then reuse it for detecting objects in high-resolution orthophotography.

There are two primary ways to utilize pre-trained models: feature extraction and fine-tuning.

Feature extraction involves utilizing the representation learned by a previous network as a fixed feature extractor for new samples. These features are then run through a new classifier, which is trained from scratch. So we use the convolution base of previously trained networks, running the new data through it, and training a new classifier on top of the output. This is due to the fact that convolutions detect the generic concepts over a picture, which are useful regardless of the computer-vision problem at hand. The generality of features extracted by a specific convolution layer depends on the depth of the layers. Shallow layers extract local, highly generic feature maps (edge, colors, texture) while deeper layers extract more abstract concepts (ears, eyes, etc).

So, if the new dataset is significantly different from the dataset on which the original model was trained, only the first few layers of the model should be used. Feature extraction is computationally efficient and reusable

The second strategy is fine-tuning. Fine-tuning enables slight adjustments of pretrained models to fit the target task, refining the parameters to make them more relevant for specific problems. Typically, a few top layers are replaced to match the number of classes in a new task; the weights of those layers are randomly initialized and then fine-tuned to fit specific tasks. Different strategies, such as full fine-tuning (where all layers of the pre-trained model are adjusted) or partially fine-tuning (where only the high convolutional layer and classifier are fine-tuned). The more parameters you train on the small dataset, the higher the risk of overfitting. Thus, in this situation, it is a good strategy to train only the top few layers. Typically, during fine-tuning, an optimizer with a very low learning rate is used to limit the magnitude of the modification.

There are several parameters that should be considered when deciding on a type of transfer learning. The most important are the size of the new dataset and its similarity to the original dataset:

- If the target dataset is small and similar to source datasets, the feature extraction may be the most effective approach.
- If the target dataset is large and similar to the original dataset, we can fine-tune through the full network, allowing all layers to adapt to the new task.
- If the target dataset is small but very different from the original dataset, it is recommended to train just the top layers of the network or consider traditional machine learning algorithms.
- If the target dataset is large and very different from the original dataset, we can initialize the network with weights from a pretrained model and fine-tune it throughout the entire network.

Transfer learning substantially reduces the size of the training dataset and speeds up training.

15.5 CNN architectures

Modern CNN architectures employ a modular design, facilitating easier design, training, and optimization. The network consists of predefined structural blocks of layers, and stacking those blocks creates a deep network capable of modeling spatial and temporal correlations. Regarding architecture, research efforts have focused on designing new building blocks.

15.5.1 LeNet

LeNet-5 [50] is one of the earliest CNNs, designed for handwritten digit recognition of the MNIST dataset. It consists of input layers, two convolutional layers followed by two pooling layers, three fully connected layers, and finishes with an output layer with a softmax activation function (**Figure 102**). Although it is designed for small-scale problems, LeNet-5 introduced several key concepts, including convolution, non-linearity, and pooling units, as well as end-to-end learning via backpropagation in a complex, dynamic architecture. It was the first to demonstrate practically the capabilities of CNN in automatic learning hierarchical features and became a foundation for all subsequent DL models.

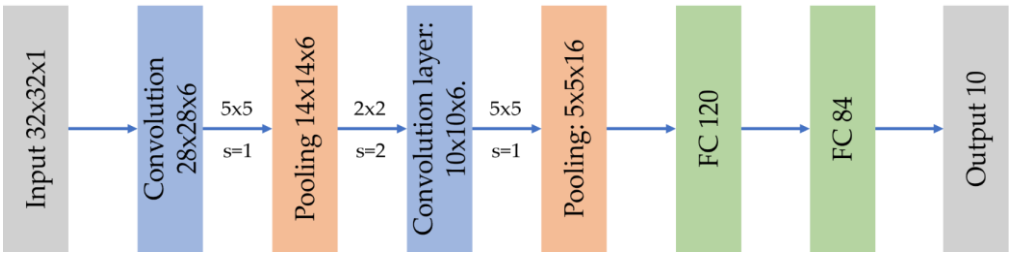


Figure 102 LeNet 5 architecture

15.5.2 AlexNet

AlexNet [15] extended LeNet architecture to a larger and deeper model, demonstrating that these principles could be successfully scaled to large and complex visual tasks when combined with sufficient data. They won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) with a 15.4% error rate. It consists of five convolution layers followed by a max pooling layer, and three fully connected layers (**Figure 103**). It implements several new features to improve performance. It utilizes a ReLU as an activation function

to accelerate convergence, employs GPU parallelization to facilitate the training of large models, and incorporates data augmentation and dropout to mitigate overfitting. The model is trained by using batch SGD with momentum and weight decay. They emphasize the importance of depth in achieving high classification accuracy.

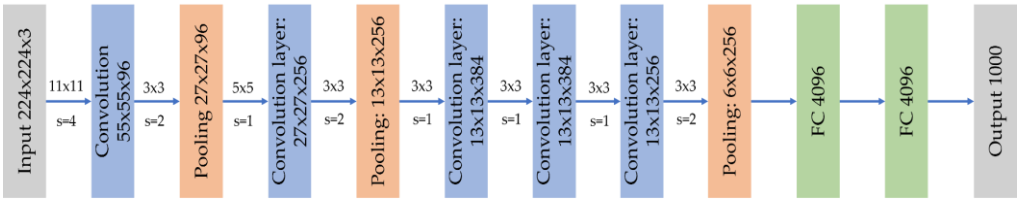


Figure 103 AlexNet Architecture

15.5.3 VGGNet

VGGNet [51] is built on the foundation of AlexNet. VGG network uses small convolution filters of size 3x3 and padding 1 to maintain a spatial resolution, followed by ReLU, and stacks them in multiple consecutive layers (**Figure 104**). The max pooling layer, 2x2 with a stride of 2, is used after each stack of convolution layers (rather than after every single convolution layer, as in AlexNet). The core idea behind using a small filter size is to increase the network depth while maintaining computational efficiency. Stacking multiple 3x3 convolutions effectively enlarges a receptive field (for example, two tacked 3x3 convolutions are equal to a receptive field of a single 5x5), introduces more non-linearity, and increases network capacity while preserving the number of parameters.

Following the convolution part of the network, the three fully connected layers are included. The VGG architecture varies between VGG11 and VGG19, differing only in depth: VGG11 contains 8 convolutional and 3 fully connected layers, while VGG19 includes 16 convolutional and 3 fully connected layers. Moreover, the number of channels increases by a factor of 2 after each max pooling layer. The VGG demonstrated that the increasing network depth significantly improves classification accuracy.

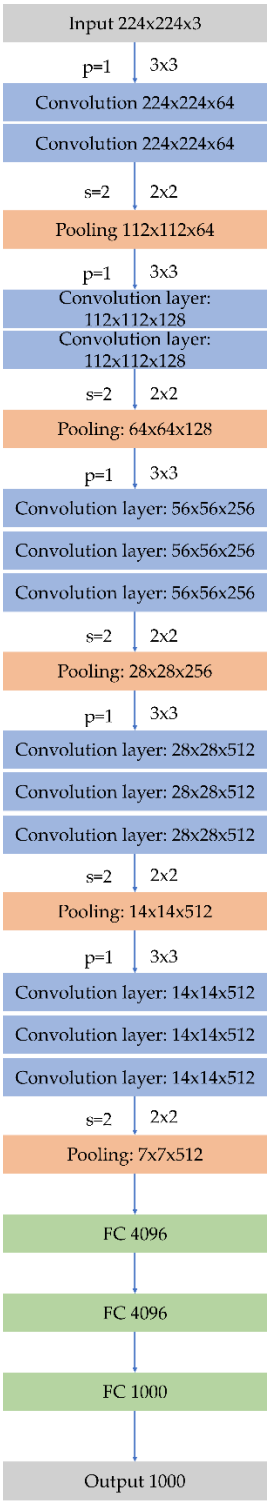


Figure 104 VGGNet 16 architecture

15.5.4ResNet

Although VGG confirmed that network depth is of crucial importance for classification accuracy. However, even after batch normalization, the deeper network leads to lower accuracy for both the training and validation datasets. meaning that degradation in classification accuracy is not caused by overfitting, and that adding more layers to a suitable deep model results in higher training error even if added layers are identity mapping (returning the input without changing it). It often makes training harder due to vanishing or exploding gradients.

The Residual Network (ResNet) [46] addresses this problem by changing the learning objective from fitting a full transformation $\big(H(x)\big)$ to each few stacked layers to fit only residual mapping. This is done through a simple operation

$$F(x) = H(x) - x \text{ therefore}$$

$$y = F(x, w) + x$$

where y is the output vector, $F(x, w)$ represents the learned residual mapping and x is the input vector to those layers. The residual block consists of two 3x3 convolutional layers, followed by a batch normalization layer and a ReLU activation function. The skip connection bypasses two convolutions, ensuring that if the residual path learns nothing (i.e., the gradient is zero), the network still behaves as an identity function (**Figure 105**). Moreover, the skip connection does not add any parameters to the model.

The ResNet is built by stacking residual blocks. The network ends with a global average pooling layer and a fully connected layer with softmax activation. The ResNet 34, ResNet 50, ResNet 101, and ResNet 152 are the most commonly used models. In deeper networks (from ResNet 50 and beyond), the bottleneck block is introduced to reduce memory and computational complexity by using two 1x1 convolutions (**Figure 105**). The first 1x1 convolution reduces the number of feature maps, while the second 1x1 convolution layer restores the dimensionality.

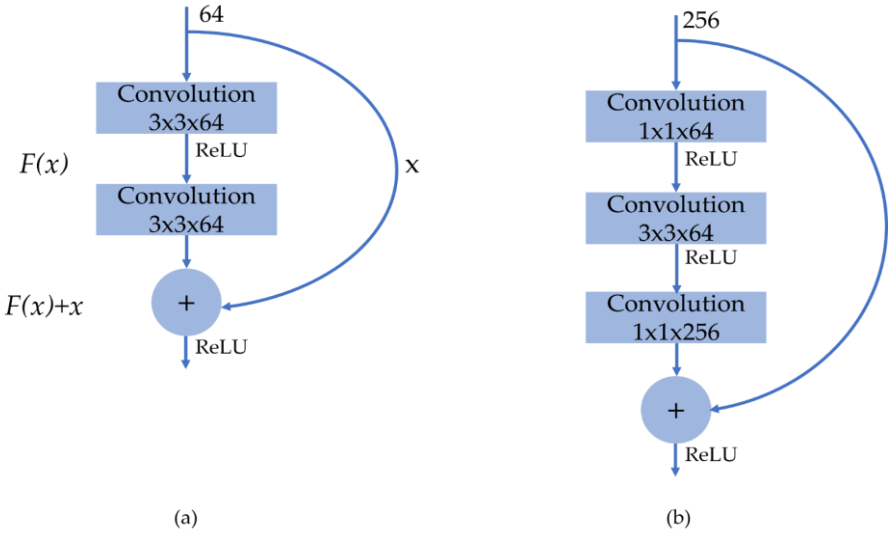


Figure 105 (a) Building block ResNet 34, (b) bottleneck building block for ResNet 50 and deeper

To provide better insights into the effect of skip connection, Lee et al. [52] visualise the loss landscape. It shows that in traditional models that don't use skip connections, as the depth of the network increases, the loss landscape transforms from nearly convex to extremely irregular, containing many local minima and saddle points (**Figure 106** (a)). This makes gradient-based optimization.

By introducing skip connections, the loss landscape becomes smoother and flatter (**Figure 106** (b)), which facilitates stable and efficient training. The flatness represents the wider region around the minima where training loss remains low, ensuring that small changes in parameters cause smaller changes in loss, which is crucial for both training stability and better generalization

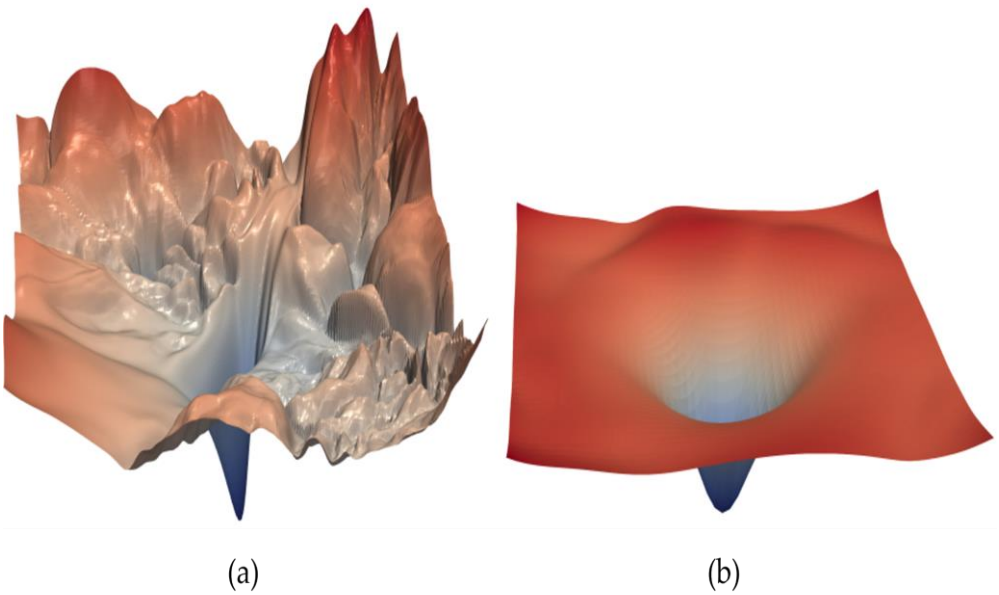


Figure 106 Loss landscape of ResNet 56 (a) without skip connection, (b) with skip connection (courtesy of Le et al. [54])

The ResNet fundamentally changed deep learning. It has several benefits:

- mitigates the vanishing gradient problem, enabling networks to be trained with hundreds of layers,
- improves optimization, allowing the model to converge faster,
- introduces a modular design, i.e. residual block, which can be easily stacked to build deeper architecture in a systematic way, and
- provide better generalization, achieving state-of-the-art accuracy in many computer vision tasks.

15.5.5 UNet

UNet architecture, introduced by Ronneberger et al. [18], is one of the most important CNN models for semantic segmentation. Semantic segmentation is a challenging task since the output is an image that contains the prediction for each pixel. If we have an input 224x224 that is passed through ResNet, the output will be 7x7 convolution activations. In the end, we need to have a 224x224 pixel segmentation mask. However, the 7x7 grid does not contain enough information to fully regenerate every pixel in the output.

The primary concept behind UNet is to capture both the context and precise localization of features within an image. The architecture consists of a symmetrical encoder and a decoder part, as well as a skip connection that combines them (**Figure 107**).

The encoder type follows a typical CNN design consisting of two 3x3 convolutions, followed by ReLU and a 2x2 max pooling operation with a stride of 2. Pooling gradually reduces spatial resolution while increasing the number of channels, enabling the network to learn abstract and context-rich representations.

The decoder part is responsible for reconstructing spatial detail and producing a segmentation map of the same resolution as the input. Each step in the decoder consists of a transposed convolution that doubles the spatial information, followed by concatenation with the corresponding feature maps from the encoder. These skip connections preserve spatial information, allowing the decoder to recover fine-grained details. After concatenation, two 3x3 convolutions and ReLU activation are applied to fuse the features. After the final layer, a 1x1 convolution maps the resulting feature representation to the predefined number of output classes.

Although primarily designed for medical imaging, it has been widely adopted in remote sensing communities for semantic image segmentation. The main benefits of UNet:

- skip connection and symmetrical design allow precise localization,
- it works well with a limited amount of training data and
- computationally efficient and flexible.

Due to its simplicity and modularity, it can utilize pre-trained models, such as ResNet, as an encoder, combining the strong representation power of ResNet with the precise localization ability of UNet (**Figure 107**).

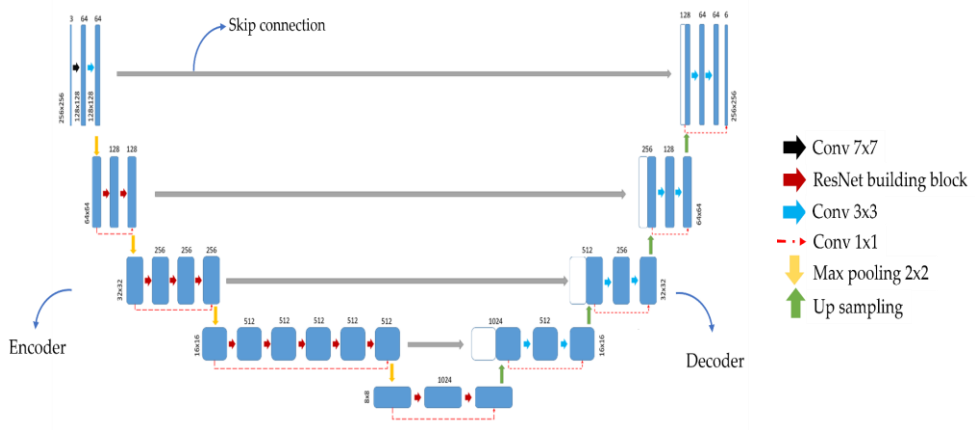


Figure 107 ResUNet architecture

Example: Extraction of floating plastic pieces from very high resolution remote sensing images based on deep learning algorithms. This research is published in [53].

Semantic segmentation of floating plastic is highly challenging due to several limitations: low amount of training data, highly imbalanced data sets, limited accuracy of ground truth data, and frequent scene changes due to constant plastic movement.

The proposed workflow consists of three main steps: preprocessing, classification and accuracy assessment.

The study area is the confluence of the Crna Rijeka and the Vrbas Rivers, near Mrkonjic Grad, Bosnia and Herzegovina. A net for collecting floating garbage was installed to mitigate pollution in the area. Floating waste in this area is primarily caused by the disposal of the garbage in illegal landfills and picnic sites along the river or directly in the river. In order to detect and map the plastic a UAV survey was conducted, using a DJI Mavic pro equipped with an RGB camera. The flight height was set to 90 m.

Preprocessing: The acquired images, together with the SfM algorithm were used to generate a high-resolution orthophoto with a spatial resolution of 3 cm. The creation of training data was both challenging and time consuming, due to the small size of the object, variations in color and spectral

signatures, different levels of submersion, and the constant moving of the floating plastic items.

To reduce the errors caused by the manual delineation,, the multiresolution segmentation algorithm was applied. This algorithm merges pixels into meaningful non-overlapping segments/polygons. Each segment was then manually labeled using QGIS software, based on a visual inspection of the orthophoto. Plastic waste was categorized into two groups: plastic and maybe plastic. The maybe plastic class was introduced to reduce the spectral confusion in the plastic class, and it was assigned to the segments where the operators were not able to state whether it was plastic by visual inspection and by analyzing the spectral signature.

Classification: An end-to-end semantic segmentation model for a floating plastic segmentation was developed based on ResUNet 50 architecture. This architecture is well suited for application with limited training data and provides precise segmentation results.

The performance of deep neural networks is highly limited by the low number of training data. As already mentioned, the size of the dataset needed for network training is a function of the size of the network (width and depth) and the complexity of the problem. To reduce overfitting the data augmentation techniques were applied including: rotating, horizontal and vertical flipping, zooming and brightness variation . Although the produced images are intercorrelated they are not the same, contributing to a better generalization capability. In addition to reducing overfitting, data augmentation also enhances performance in cases of class imbalance within the dataset.

Additionally, transfer learning was employed through fine-tuning of ResNet-50 model pretrained on the ImageNet dataset. The weights of the upper layers of the pretrained network were unfrozen and updated during the training phase to adapt to the specific characteristics of the datasets, while the lower layers remained frozen.

Implementation: Due to the limited processing power, the original images were decomposed to 256 × 256 px patches. The ResUNet model, which uses 3x3 convolution layers with padding of 1 and stride of 1, was fine-tuned on

the Crna Rijeka dataset. The dataset, consisting of 1846 images, was split into 80% of the data for training and 20% for validation. The batch size was limited by the GPU and it was chosen as large as possible.

Different loss functions, such as cross entropy, cross entropy weighted, and focal loss were tested. Since the highest accuracy was obtained using cross entropy, it was adopted for the final model. The models were implemented in Python 3 using popular deep learning libraries such as PyTorch, TensorFlow, Keras, and Matplotlib. Training was performed on the publicly available cloud platform Colaboratory (Google Colab), which is based on Jupyter Notebooks.

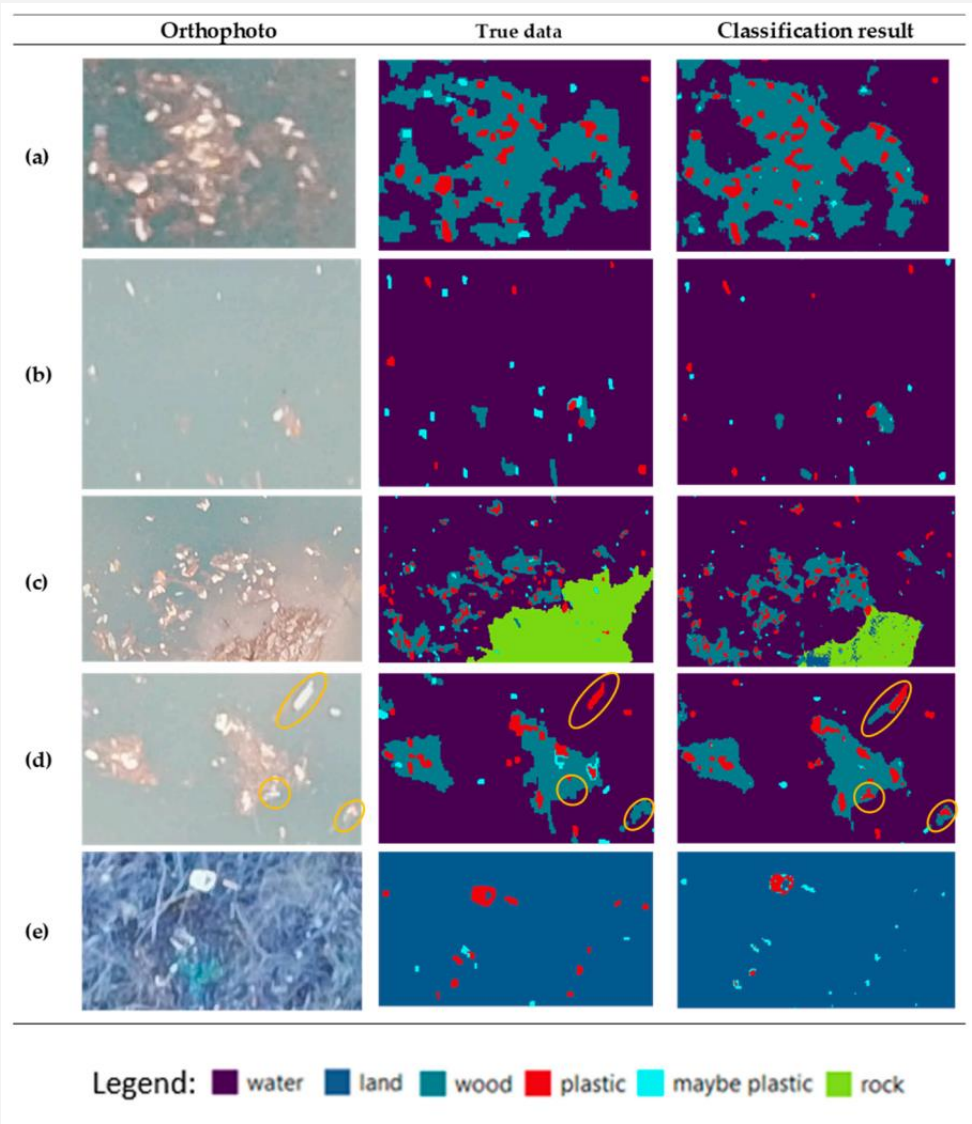
Accuracy assessment: To test the accuracy of the classification results three standard parameters were calculated: precision, recall, and F1-score. The results of the accuracy assessment are presented below

Class	Precision	Recall	F1
Plastic	0.82	0.75	0.78
Maybe plastic	0.62	0.34	0.43

The ResUNet-50 model demonstrated a stable performance in classifying plastic. The highest confusion was obtained for the "maybe plastic" class, which was occasionally misclassified as water or plastic. For this class the precision was high, while recall was low, indicating the underestimation of the area covered by the maybe plastic class. Although metrics such as precision, recall, and F1 score provide a deeper insight into the performance of the algorithm, the area and volume of the detected plastics are more relevant for stakeholders. In the Crna Rijeka case study, the algorithm only underestimated the plastic area by 3.4%, highlighting its strong potential for optimizing cleaning campaigns.

Visual inspection shows that the locations of the plastic pieces were accurately detected, although some pixels along edges of plastic items were misclassified as the surrounding class. No differences were observed in the performance of the model between grouped (a) or single plastic items (b). Interestly, the algorithm detected plastic accurately in shallow water (c)

which is typically challenging because the presence of the river bed increases water reflectance (same as plastic does). In this study case, the algorithm accurately extracted the plastic pieces that were omitted from the training data (d), showing good generalization abilities. Moreover, the model showed its potential for plastic detection not just in water but also on land, with lower accuracy compared with the floating plastics (e).



16 FUTURE OF GEOAI

Over the past few years, GeoAI has revolutionized many geospatial tasks, rapidly becoming a growing subfield of spatial data science across various domains (**Table 18**).

Table 18 Overview of GeoAI application across different domains and research topics

Geospatial domain	Research topic	Description	Commonly used architectures
Remote sensing	Scene classification	assigning the entire RS image to a predefined category	CNN, ResNet
	Semantic segmentation	assigning each pixel to predefined classes	UNet, ResUNet
	Object detection	extracting the boundary box of the detected geospatial object	R-CNN, Fast CNN
	Instance segmentation	object-level classification	Recurrent Neural Network (RNN), Transformers
	Super-resolution and data fusion	increasing spatial resolution or enabling fusion of different sources such as optical+SAR+LiDAR	CNN, GAN,
	Change detection	monitoring spatial changes across time series of satellite images	Siemens network, temporal CNN
Cartography	Automatic	feature extraction	CNN, UNet

	map feature extraction	from geospatial images	
	Automatic map generation	automatic generation of vector maps from satellite images, and generation of semantic maps for autonomous driving	GAN
	Cartographic generalization	altering the map visualisation when the scale changes	CNN, Graph Neural Network (GNN), GAN
Environment	Monitoring climate changes	Track snow and ice melting, modeling climate parameters	CNN, Conv Long Short Term Memory (convLSTM), GNN
	Disaster events and early warning systems	Detect hazards, provide information for disaster response and resilience	CNN
	Agriculture and food security	crop type mapping, health monitoring,	CNN, ANN
	Water availability	water body mapping, water quality monitoring, wetland monitoring	CNN, ANN, RNN

While classical machine learning and deep learning models significantly advanced geospatial analysis, they are often task-specific, requiring large labeled datasets and retraining for each new sensor, region, or application. Moreover, most models are trained on geography-specific datasets, encoding location explicitly instead of considering it as another attribute. This limits

their scalability and generalization across geographic regions with different characteristics, making them unsuitable for global, dynamic Earth observation. In addition, many traditional models struggle to capture non-stationary and multi-scale dynamics that characterize complex real-world systems, such as climate and hydrological ecosystems.

16.1 Foundation models

In recent years, the launch of ChatGPT marked a major turning point, drawing high attention in Large Language Modeling (LLMs). The LLM models, such as Generative Pre-trained Transformers (GPT), are designed to encode a sophisticated understanding of the syntax, semantics, and contextual relationships within human language. By leveraging enormous amounts of data during training, these models develop a generalized representation that can be applied to a wide range of applications, including question answering, summarization, and code generation, demonstrating their ability to transfer knowledge across domains. Taking into account their fundamental roles in completing various domain-specific tasks, LLMs and other large-scale models are also referred to as foundation models (FMs) [54].

FMs [54], are pre-trained on large-scale data in a task-agnostic manner, and can be easily adapted to a wide range of downstream tasks across domains by fine-tuning.

With the introduction of AlexNet and ResNet, the GeoAI frameworks have shifted rapidly from custom-built, task-specific models trained from scratch to frameworks that leverage pre-trained models as feature extractors. This form of transfer learning dramatically reduces training time, improving model accuracy by exploiting learned rich and more abstract patterns that only need to be adapted to the specified task. While transfer learning enables the creation of foundation models, it is the scale of training that makes them powerful. The scale is determined by three main factors: access to massive and diverse datasets, improvements in computational hardware (such as GPUs and TPUs), and advances in Transformer-based architectures.

Transfer learning with annotated datasets has been standard practice for the last decades. However, the costs of annotation are often substantial, which limits practical scalability and benefits from pretraining. In contrast, self-

supervised learning performs pretraining tasks automatically using unlabeled data, enabling models to leverage massive datasets at an Internet scale. This approach provides more scalability to adaptively handle large quantities of high-dimensional data.

The transformer network [55] is built on a self-attention mechanism, which allows it to compare all elements across the input simultaneously and to capture long-range dependencies. This enables parallelized computation, offering a more flexible alternative to the rigid, fixed-weight computation of MLP and CNN.

One of the main advantages of attention over prior architectures is its generality: it is not strongly tied to a specific task or domain, unlike the local receptive fields of convolution. This general-purpose nature of attention and transformers contributes to their broad applicability.

16.2 Geospatial Foundation Models

The foundation models can generalize effectively only within the scope of their training data. While they have been trained on vast amounts of text, tabular data, images, and other forms of web-accessible content, they have had limited exposure to geospatial vector data, multi-spectral images, spatiotemporal datasets, or point clouds. Because these data types were largely absent during pretraining, general-purpose FMs are not expected to perform reliable geospatial reasoning or prediction tasks.

Moreover, many geospatial tasks are highly specialized and require types of reasoning beyond the capabilities of current general-purpose FMs, including:

- Data modality diversity - Geospatial applications often involve heterogeneous input (multispectral, radar images, LiDAR) and output formats. General-purpose FM is mostly trained on RGB imagery. Moreover, aerial or satellite images are characterized by different geometries, spectral properties, and object scales compared to a street-level view.
- Topological reasoning - understanding spatial relationships such as connectivity, adjacency, or containment between geospatial features

(such as buildings, parcel borders, roads, etc) is crucial for spatial analysis but remains difficult for existing FMs [56], and

- Spatial and temporal variability - Geospatial processes operate across large, continuous areas over time, requiring models to handle both local context and long-term temporal dependencies, which most FMs currently do not explicitly model.

These challenges underscore the need for Geospatial foundation models (GeoFMs) specifically designed to handle the unique properties of spatial data. Emerging GeoFMs aim to address these limitations by: employing LLMs as agents that can synthesize geospatial workflow using existing geospatial toolset and APIs (such as geopandas, rasterion, Google Earth Engine or remote sensing APIs), leveraging large-scale geographic knowledge graphs to encode spatial relationships and domain knowledge, and integrating the LLMs with knowledge graphs to enhance spatial reasoning and contextual understanding [57], [58].

To achieve state-of-the-art accuracy across diverse remote sensing tasks, GeoFM must incorporate the unique characteristics of remote sensing data. In addition to being task agnostic, it is desired to be:

- Sensor agnostic - capable of reasoning seamlessly across data collected from different sensors with varying spatial, spectral, and temporal resolution,
- Spatiotemporally aware - able to handle the spatiotemporal data of imagery while performing geospatial reasoning for tasks such as image geolocalization, change detection, and object tracking,
- Environmentally invariant - able to distinguish the intrinsic spectral properties of the objects of interest regardless of seasonal, atmospheric, or environmental conditions.

Another critical challenge in developing FMs for GeoAI is the complexity of vector data, which differs fundamentally from structured text or imagery data typically used in NLP. The vector data exhibit more complex data structures such as points, polylines, polygones, triangulated irregular network (TINs), 3D building models, point clouds, etc., which require a flexible NN architecture capable of learning from graphs, meshes, and irregular topologies. In other data modalities, such as geo-tagged videos or images,

spatial social network posts, and sensor data, contribute further to the multimodal nature of GeoAI. To effectively process and integrate these sources, GeoFM must develop advanced topological and semantic reasoning capabilities.

The multimodality of geospatial data thus represents one of the greatest challenges in developing GeoFMs. As a result, future research in geospatial data sciences and GeoAI must focus on designing models that can integrate and reason across these modalities in a coherent manner.

Recent examples demonstrate the potential of GeoFMs models. Prithvie-EO-2.0 is a multi-temporal transformer-based GeoFM pretrained on 4.2 million globally sampled time-series tiles from the NASA Harmonized Landsat-Sentinel 2 dataset at 30 m resolution and incorporates both temporal and location embedding to improve performance across three main tasks: disaster response, land cover mapping, and ecosystem dynamic monitoring [59]. SatClip learns an implicit representation of geographic locations by contrastively matching visual patterns from satellite imagers with their spatial coordinates, significantly improving generalizations across diverse location-based tasks [60]. These developments illustrate the ongoing shift in GeoAI toward multimodal, scale-aware, and geographically grounded models, making a new era of robust, task-agnostic tools for Earth observation and environmental analytics.

Although GeoFM is a recent development, emerging only in the last one to two years as a convergence of geospatial science and large foundation model research, its progress is accelerating rapidly. Currently, GeoFMs are mainly focused on raster images, primarily leveraging optical and radar satellite imagery. The next major step is expected to be the expansion across data modalities, integrating raster, vector, and point cloud data into a unified embedding space. In parallel, research will increasingly emphasize temporal transformations and change-aware embedding that capture the dynamics of processed and forecasting modalities. This evolution will transform GeoFM from a static mapping system into a dynamic Earth reasoning system, capable of understanding and predicting processes such as deforestation or urbanization.

In summary, future efforts should prioritize the development of sensor-agnostic, spatiotemporally aware, and environmentally invariant GeoFMs that leverage remote sensing to address urgent environmental and climate-related challenges. Such models have the potential to transform our understanding of complex Earth system dynamics at a global scale and to enable evidence-based decision-making. Beyond achieving technical accuracy, it is essential that GeoFM development also maximizes long-term social and environmental benefits. Without such a shift, there is a risk that advances in GeoAI may fail to translate into meaningful real-world impact or support comprehensive monitoring of SDG indicators.

17 REFERENCES

[1] A. M. Turing, "Computing Machinery and Intelligence," University of Maryland, 1950.

[2] J. McCarty, J. Minsky, N. Rochester and C. E. Shannon, "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence," 1955.

[3] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biology,* vol. 5, pp. 115-133, 1943.

[4] F. Rosenblatt, "THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN," *Psychologival Review,* vol. 65, no. 5, pp. 386-408, 1958.

[5] T. Cover and P. Hart, "Nearest Neighbor Pattern classification," *IEEE Transactions on Information Theory ,* vol. 13, no. 1, pp. 21-27, 1967.

[6] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors," *Nature,* vol. 323, pp. 553-536, 1986.

[7] A. E. Bryson and Y. C. Ho, Applied Optimal Control: Optimization, Estimation, and Control., Waltham: Blaisdell Publishing, Waltham., 1696.

[8] J. Pearl, Probabilistic Reasoning in Intelligent Systems,, 1988.

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation,* vol. 1, no. 4, pp. 541-551, 1989.

[10] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning,* vol. 20, pp. 273-297, 1995.

[11] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[12] J. Deng, W. Dong, R. Socher, L. Li, K. Li and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, USA, 2009.

[13] R. Raina, A. Madhavan and A. Y. Ng, "Large-scale Deep Unsupervised Learning using Graphics Processors," in *ICML*, 2009.

[14] Google, "Google Earth Engine," [Online]. Available: https://blog.google/outreach-initiatives/sustainability/introducing-google-earth-engine. [Accessed 22 2 2025].

[15] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, 2012.

[16] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Nets," in *arXiv:1406.2661*, 2014.

[17] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *arXiv:1512.03385*, 2015.

[18] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *MICCAI 2015*, 2015.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention Is All You Need," *31st Conference on Neural Information Processing Systems,* Vols. 1-15, 2017.

[20] M. Cazzaniga, F. Jaumotte, L. Li, G. Mellna, A. J. Panton, C. Pizzinelli, E. Rockall and M. Tavares, "Gen-AI: Artificial Intelligence and the Future of Work," International Monetary Found, Washington, DC., 2024.

[21] A. R. Huete, "A soil-adjusted vegetation index (SAVI)," *Remote Sensing of Environment,* vol. 25, pp. 295-309, 1988.

[22] A. Huete, K. Didan, T. Miur, E. P. Rodriguez, X. Gao and L. G. Ferreira, "Overview of the radiometric and biophysical performance of the MODIS vegetation indices," *Remote Sensing of Environment,* vol. 83, no. 1-2, pp. 195-213, 2002.

[23] B. Gao, "NDWI—A normalized difference water index for remote sensing of vegetation liquid water from space," *Remote Sensing of Environment,* vol. 58, no. 3, pp. 257-266, 1996.

[24] H. Xu, "Modification of Normalised Difference Water Index (NDWI) to Enhance Open Water Features in Remotely Sensed Imagery," *International Journal of Remote Sensing,* vol. 27, no. 14, pp. 3025-3033, 2006.

[25] USGS, "Normalized Burn Ratio," [Online]. Available: https://www.un-spider.org/advisory-support/recommended-practices/recommended-practice-burn-severity/. [Accessed 12 3 2025].

[26] L. Roth, A. Hund and H. Aasen, "PhenoFly Planning Tool: Flight planning for high-resolution optical remote sensing with unmanned areal systems," *Plant Methods,* vol. 14, pp. 1-21, 2018.

[27] ASPER, "ASPRS Accuracy Standards for Digital Geospatial Data," 2023. [Online]. Available: https://www.asprs.org/a/society/divisions/pad/Accuracy/Draft_ASPRS_Accuracy_Standards_for_Digital_Geospatial_Data_PE&RS.pdf. [Accessed 26 3 2025].

[28] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision* , vol. 60, no. 2, pp. 91-110, 2004.

[29] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM,* vol. 24, no. 6, pp. 381-395, 1981.

[30] J. Su and E. Bork, "Influence of Vegetation, Slope, and Lidar Sampling Angle on DEM Accuracy," *Photogrammetric Engineering & Remote Sensing,* vol. 72, no. 11, pp. 1265-1275, 2006.

[31] M. Wulder, J. White, R. Nelson, E. Naesset, H. Ole Orka, N. Coops, T. Hilker, C. Bater and T. Gobakken, "Lidar sampling for large-area forest characterization: A review," *Remote Sensing of Environment,* vol. 121, pp. 196-209, 2012.

[32] ITRF, "ITRF2020-u2023," [Online]. Available: https://itrf.ign.fr/en/solutions/ITRF2020-u2023?utm_source=chatgpt.com,. [Accessed 18 02 2025].

[33] U. N. G.-I. Agency, "Recent Update to WGS 84 Reference Frame and NGA Transition to IGS ANTEX," US National Geospatial-Intelligence Agency, 2025.

[34] A. F. Agarap, "Deep learning using Rectified Linear Units (ReLu)," in *arXiv:1803.08375v2*, 2019.

[35] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville and Y. Bengio, "Maxout Networks," in *arXiv:1302.4389v4*, 2013.

[36] D. A. Clevert, T. Unterthiner and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," in *arXiv:1511.07289* , 2015.

[37] G. Klambauer, T. Unterthiner, A. Mayr and S. Hochreiter, "Self-Normalizing Neural Networks," in *In Advances in Neural Information Processing Systems (NIPS)*, 2017.

[38] P. Ramachandran, B. Zoph and Q. V. Le, "Swish: A Self-gated activation function," in *arXiv:1710.05941v1*, 2017.

[39] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *arXiv:1502.03167v3*, 2015.

[40] H. Zhang, M. Cisse, Y. N. Dauphin and D. Lopez-Paz, "mixup: Beyond Empirical Risk Minimization," in *arXiv:1710.09412* , 2018.

[41] N. Srivastava, G. Hinton, A. Krizhevsky and a. et, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* vol. 15, pp. 1929-1958, 2014.

[42] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement,* vol. 20, no. 1, 1960.

[43] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and System,* vol. 2, 1989.

[44] G. Montufar, R. Pascanu, K. Cho and Y. Bengion, "On the Number of Linear Regions of Deep Neural Networks," in *arXiv:1402.1869*, 2014.

[45] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *13th International Conference on Artificial Intelligence and Statistics* , 2010.

[46] K. He, X. Zhang, S. Ren and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *arXiv:1502.01852v1*, 2015.

[47] J. Martens, "Deep learning via Hessian-free optimization," in *27th International Conference on Machine Learning*, Haifa, Israel, 2010.

[48] M. Govedarica and G. Jakovljevic, "Monitoring spatial and temporal variation of water quality parameters using time series of open multispectral data.," in *Seventh International Conference on Remote Sensing and Geoinformation of the Environment*, Paphos, 2019.

[49] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of Statistical Planning and Inference,* vol. 90, pp. 227-244, 2000.

[50] L. LeCun, L. Bottou, I. Bengio and P. Haffner, "Gradient Based Learning Applied to Document Recognition," in *Proceedings of the IEEE*, 1998.

[51] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *arXiv:1409.1556*, 2014.

[52] H. Li, Z. Xu, G. Taylor, C. Studer and T. Goldstein, "Visualizing the Loss Landscape of Neural Nets," in *arXiv:1712.09913v3*, 2018.

[53] G. Jakovljevic, M. Govedarica and F. Alvarez-Taboada, "A Deep Learning Model for Automatic Plastic Mapping Using Unmanned Aerial Vehicle (UAV) Data," *Remote Sensing*, 2020.

[54] R. Bommasani, D. A. Hudson, E. Adeli and a. et, "On the opportunities and risks of foundation models," in *arXiv:2108.07258*, 2021.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention Is All You Need," in *arXiv:1706.03762*, 2017.

[56] Y. Ji, S. Gao, Y. Nie, I. Majic and K. Janowicz, "Foundation Models for Geospatial Reasoning: Assessing the Capabilities of Large Language Models in Understanding Geometries and Topological Spatial Relations," in *arXiv:2505.17136v1*, 2025.

[57] G. Mai, W. Huang, J. Sun and a. et, "On the Opportunities and Challenges of Foundation Models for GeoAI (Vision Paper)," *ACMTrans. Spatial Algorithms Syst.,* vol. 10, no. 2, 2024.

[58] K. Janowicz, G. Mai, W. Huang, R. Zhu, N. Lao and L. Cai, "GeoFM: how will geo-foundation models reshape spatial data science and GeoAI?," *International Journal of Geographical Information Science,* vol. 39, no. 9, pp. 1849-1865, 2025.

[59] D. Szwarcman, S. Roy, P. Fraccaro and a. et, "Prithvi-EO-2.0: A Versatile Multi-Temporal Foundation Model for Earth Observation Applications," in *arXiv:2412.02732v2*, 2025.

[60] K. Klemmer, E. Rolf, C. Robinson, L. Mackey and M. Rubwurm, "SatCLIP: Global, General-Purpose Location Embeddings with Satellite Imagery," in *arXiv:2311.17179v3*, 2024.

Excerpt from review

From a pedagogical approach, the writing is exceptionally clear, logical, and well-structured. In each chapter the text moves from broad to specific to highly specific, with perfect clarity, using figures and tables to provide visual and comparative aids. Moreover, it adjusts to a logical motivation: It doesn't just state facts, it explains why. The clear examples and the "problem-solution" narrative is a highly effective teaching method. In addition, the book immediately organizes the vast field of AI into clear categories, so this systematic approach helps the reader build a mental map of the field.

In my view, the book is not just a technical manual; it's a mature academic text that understands the context and implications of the technology. Includes data and ethical limitations upfront (in the introduction), which is a sign of academic integrity.

Prof. Flor Álvarez Taboada

The publication “Introduction to Geospatial AI” by Gordana Jakovljević, Miro Govedarica, and Maria Antonia Brovelli is a timely and well-structured contribution to the growing body of literature at the intersection of artificial intelligence and geospatial technologies. It offers a clear, concise, yet remarkably focused overview of key AI concepts and their applications in geoinformatics, making it a valuable resource for both practitioners and researchers entering or deepening their understanding of this field.

Prof. Milan Rapaić

About authors

Gordana Jakovljević, Ph.D. is an assistant professor at the University of Banja Luka, Bosnia and Herzegovina. Her practical and theoretical research interest lies in the field of remote sensing, deep learning, and environment protection, especially in water management. The primary aim of her research is to develop a standardized, clearly defined methodology for the automated processing of remote sensing data in real or near-real time in order to increase the usability of remote sensing data in environmental management and decision-making. Her experties includes GeoAI, photogrammetry, laser scanning, GIS, webGIS, 3D visualisation. She is a member of FIG commission 4 and FIG 4.3 working group (Mapping plastic)

Prof. Miro Govedarica, Ph.D., is a Full Professor at the Faculty of Technical Sciences, University of Novi Sad, Serbia. His scientific and professional work is focused on geoinformatics, combining geospatial software engineering, data science, and artificial intelligence to address complex spatial problems. He has published numerous papers in international journals and conference proceedings and has led and participated in many national and international research projects.

A significant portion of his work is dedicated to GeoAI, encompassing the application of machine learning and deep learning techniques to spatial and spatiotemporal data, remote sensing, and Earth observation, as well as automated feature extraction and intelligent decision-support systems. His expertise also includes geospatial databases, spatial big data, photogrammetry, laser scanning, GNSS, and GPR.

Maria Antonia Brovelli is a distinguished academic and researcher with a background in Physics and a Ph.D. in Geodesy and Cartography. She is a Professor of GIS, Earth Observation and The Copernicus Green Revolution for Sustainable Development at Politecnico di Milano (PoliMI). Having dedicated her entire career to PoliMI, she began as a researcher and later became a Full Professor, also serving as the Vice-Rector of PoliMI for the Como Campus. Her contributions extend beyond academia and currently she holds key positions in various international organizations. She serves as the Vice President of the International Society for Digital Earth (ISDE) and the ISPRS Technical Commission on Spatial Information Science, and former chair and current member of the Advisory Board of the UN-GGIM Academic Network. Additionally, she has been involved in ESA's Advisory Committee of Earth Observation (ACEO).

Her research in geomatics has covered diverse areas, including geodesy, radar-altimetry, GIS, webGIS, VGI, Citizen Science, Big Geo Data, and GEOAI. Brovelli is a global leader in Open-Source GIS. She has an impressive publication record and has been involved in numerous national and international research projects. Recognizing her contributions, she has received awards such as the ISPRS President's Honorary Citation in 2020 and the Sol Katz Award from OSGeo in 2015. She also holds editorial roles in reputable journals, emphasizing her influence and leadership in the field.

CIP - Каталогизација у публикацији
Народна и универзитетска библиотека
Републике Српске, Бања Лука

528.8.04:004.8(0.034.2)

JAKOVLJEVIĆ, Gordana, 1991-
Introduction to Geospatial Artificial Intelligence [Elektronski izvor] / Gordana Jakovljević, Miro Govedarica, Maria Antonia Brovelli. - El. knjiga. - Banja Luka : Faculty of Architecture, Civil Engineering and Geodesy, 2025

Način pristupa (URL): Način pristupa (URL): https://blupress.unibl.org/aggf/index. - Опис извора дана 24. 12. 2025. - Насл. са насл. екрана. - Ел. публикација у PDF формату опсега [367] стр. - About the author: стр. [366-367]. - Напомене и библиографске референце уз текст. - Библиографија: стр. [359-364].

ISBN 978-99976-82-18-5

COBISS.RS-ID 143742721

УНИВЕРЗИТЕТ У БАЊОЈ ЛУЦИ
UNIVERSITY OF BANJA LUKA

АРХИТЕКТОНСКО-ГРАЂЕВИНСКО-ГЕОДЕТСКИ ФАКУЛТЕТ
FACULTY OF ARCHITECTURE, CIVIL ENGINEERING AND GEODESY

ISBN 978-99976-82-18-5

9 789997 682185