Department of Economics and Finance

Chair of Mathematical Finance

MACHINE LEARNING AND THE GRADIENT DESCENT ALGORITHM

Supervisor:

Prof.ssa Sara Biagini

Candidate:

Andrea Zanetti

Student ID: 262471

ACADEMIC YEAR

2022/2023

Table of Contents

1.0 Machine Learning

1.1 Introduction

Today, machine learning is becoming an increasingly relevant topic, with estimations suggesting that it will influence everyone's life in the coming years. This technology allows machines to learn and evolve from their own experiences without the necessity of programming them. The roots of this "new'' technology can be found in the early days of computer science when academics attempted to design algorithms capable of autonomous decision-making resembling human intelligence. The advent of the digital age and the growing collection of data was one of the primary driving reasons behind the development of the first machines capable of autonomous learning. This evolution was crucial since traditional scientific-based approaches proved ineffective as data complexity and sophistication increased. Therefore, researchers realized that to acquire significant insights and generate accurate forecasts, they required a more flexible and data-driven strategy.

1.2 Background

Arthur Samuel, an American pioneer in video gaming and artificial intelligence, coined the term "machine learning" in 1959. He described it as the study of how machines can learn without manual programming. Samuel's research focused on developing systems, which could learn to play checkers and improve their performance over time with practice. This approach stands as an early instance of reinforcement learning, in which an agent interacts with an environment, receives feedback, and refines its behaviour in response. At that time, it was a relatively new concept, and Samuel's work was a breakthrough in demonstrating the potential of computers to learn and improve their performance through experience. However, real-world implementation of machine learning techniques encountered difficulties because of inadequate processing resources and scarcity of large, labelled data sets. It was not until the late 1990s and early 2000s that factors such as the increasing availability of information and the advancing abilities of computers brought back interest in machine learning. Therefore, a combination of the introduction of the internet and the

spread of digital technologies in the 21st century resulted in a rapid generation of data from sources such as social media, e-commerce, scientific studies, and sensor networks. With this groundwork, scientists had the opportunity to employ machine-learning models to extract valuable information, achieve reliable forecasts, and simplify decision-making processes. Nowadays this technology has entered various facets of our life. It powers recommendation systems, voice assistants, fraud detection, autonomous vehicles, medical diagnosis, and many other applications. Even after reaching significant improvements, machine learning still has issues such as bias in algorithms, interpretability, and ethical concerns related to data privacy and fairness continue to be areas of active research and development. However, with the advent of deep reinforcement learning and the integration of other emerging technologies like Internet of Things (IoT) and blockchain, the possibilities for further innovation and application seem limitless. Machine learning has undoubtedly transformed industries, drove innovation and shaping the way we interact with technology, and its potential continues to be an exciting area of discussion.

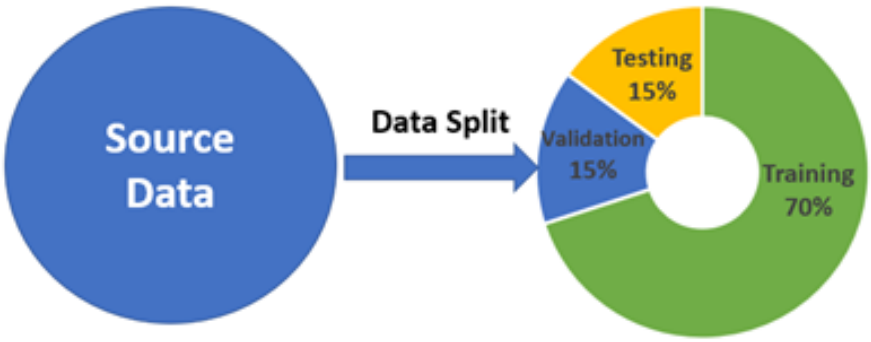
1.3 What is machine learning?

Machine learning is a branch of artificial intelligence, which focuses on the design and implementation of models, allowing computers to gain knowledge from data and develop strategies to make estimations. This system relies on the idea that machines can automatically learn and produce significant output from streams of data, to maximize their performance over time. This ability to learn and adapt based on experience distinguishes machine learning as one of the most transformative tools of our era. Moreover, the capacity to generalize beyond the training data enables models to make predictions also on new unseen examples. Now, since machine learning heavily relies on data, we need to understand the importance of the quality and quantity of the information we cluster. Data collection involves gathering relevant material from various sources, ensuring its accuracy, completeness, and relevance to the problem at hand. Indeed, raw data often requires pre-processing and manipulation to enhance its suitability for specific tasks. Two common

practices often employed by practitioners are data splitting and data cleaning. The former implies dividing a dataset into three distinct groups, usually training, validation, and testing sets. The separation enables the model to be trained on one set, validated on another, and tested on a different one. By employing this approach, we can assess the model's proficiency with unseen data. This serves as a safeguard against overfitting, a scenario in which the model memorizes the training data but falters when faced with new examples. The usual splitting ratios are:

- Training set: The largest subset (e.g., 60-80% of the data) used to train the model and learn the underlying patterns.
- Validation set: A smaller fraction (e.g., 10-20% of the data) used to adjust hyperparameters, pick the best model, and evaluate its generalization performance.
- Test set: A separate portion (e.g., 10-20% of the data) used to evaluate the final model's performance on completely unseen data.

Figure 1: Data Splitting



Data cleaning, commonly referred to as data scrubbing, is the second phase. It involves removing or fixing mistakes, inconsistencies, and useless data from the dataset. Clean data ensures that the model is exposed to accurate and trustworthy data, lowering the likelihood of biases or misleading patterns. Some typical issue that arises when clustering data are inconsistent recording, missing data, unwanted observations, and outliers. Let us explore some of them and find possible solutions:

1. Inconsistent Recording:

Inconsistent recording refers to inconsistencies or errors in data entry, formatting, or labelling. These inconsistencies can hinder data analysis and modelling. To address this problem, we can employ the following techniques:

- Standardization: Apply consistent formatting and labelling conventions to ensure uniformity across the dataset. This may involve converting variables to a common unit, correcting typos or misspellings, and aligning variable names.
- Data Validation: Conduct validation checks to identify and correct inconsistent or erroneous data.

2. Missing Data:

Missing data occurs when values are absent for certain observations or variables. Missing data can introduce bias or affect the representativeness of the dataset. Here are potential solutions for handling missing data:

- Deletion: If missing data is minimal and randomly distributed, it may be reasonable to remove the corresponding observations or variables.
- Imputation: Imputation techniques estimate missing values based on the available data.

3. Unwanted Observations:

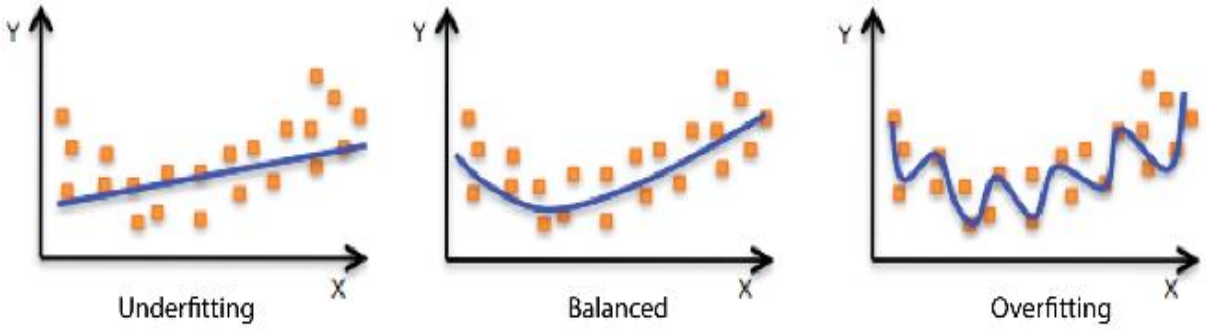
Unwanted observations refer to data points that are irrelevant, duplicate, or contain errors. It is important to identify and handle such observations appropriately:

- Duplicate Removal: Identify and remove duplicate observations to avoid redundancy and prevent skewed analysis results.
- Error Detection and Correction: Implement data validation checks or algorithms to identify and correct errors, such as incorrect measurements, outliers, or inconsistent values.

4. Outliers:

Extreme values, known as outliers, substantially diverge from the rest of the data. They have the potential to skew statistical assessments and exert an influence on the effectiveness of models.

Figure 2: Different type of model fitting



Source: datascience.eu

To gain a better understanding of why it is critical to handle data correctly during the algorithmic learning process, we can look at some real-world examples.

Assume your inbox contains a spam email filter. It may initially only have simple criteria set up to identify obvious spam emails based on keywords. These guidelines, however, become obsolete as new spam emails with different features appear. Here is where machine learning comes in. The automated learning system can uncover patterns, which distinguish spam from legitimate emails by examining a dataset of labelled emails (both spam and non-spam). It learns how to generalize these patterns and then uses them to determine whether incoming emails are spam or not. Another example is image recognition. Here, in the same way, machines are trained on datasets of images to recognize objects or identify specific features within images. For instance, a model can learn to differentiate between pictures of cats and dogs by exposing the system to a diverse range of cat and dog images. It learns to identify distinguishing characteristics, such as the shape of ears or the texture of fur. After training, even pictures that the model has never seen before can be correctly classified as either a cat or a dog. After going through these examples, we recognize the critical importance of gathering data and the necessity to apply every technique that will help the model to maximize its training.

1.4 Types of models

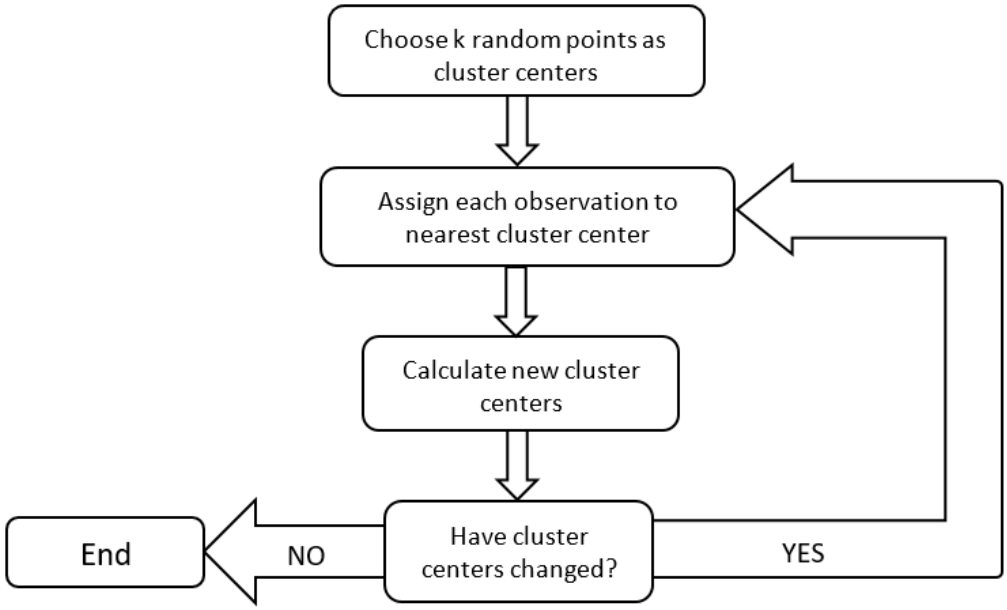
This section will examine many machine learning models, each of which has a unique set of features and applications. Let us explore the three main types: unsupervised learning, supervised learning, and reinforcement learning.

1.4.1 Unsupervised Learning

Unsupervised learning entails identifying patterns, without any specific target or output labels. Therefore, the goal is not to predict the value of a specific variable, but instead, it focuses to uncover hidden trends, structures, or connections within the data. Unsupervised learning algorithms are usually utilized to identify clusters, detect anomalies, or reduce the dimensionality of the data. These practices are very useful for businesses. For instance, banks use unsupervised learning to organize their customers based on some common features. A clustering algorithm is able to identify different groups of individuals based on their purchasing behaviour, and sort customers with similar buying patterns into different groups. As a result, the bank can provide targeted marketing strategies and improve their level of services. A widely used technique for clustering is the k-means algorithm, which aims to divide the data into k clusters. The algorithm operates through an iterative process that involves two main steps: the assignment step and the update step. Initially, it randomly places K cluster centroids, acting as the initial guesses for the cluster positions. During the assignment step, each data point is allocated to the closest centroid using the Euclidean distance metric, which is computed as, $\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$. In this step, the metric computes the direct distance between a data point and every cluster centre. Consequently, each data point is allocated to the cluster with the nearest centre. Moving on to the updating phase, the centroids undergo a re-evaluation through the computation of the mean of data points assigned to each cluster. This new mean value becomes the updated cluster centre, and the process continues with further iterations. These assignment and update procedures are carried out periodically until convergence, which happens when the centroids no longer vary significantly or until reaching a

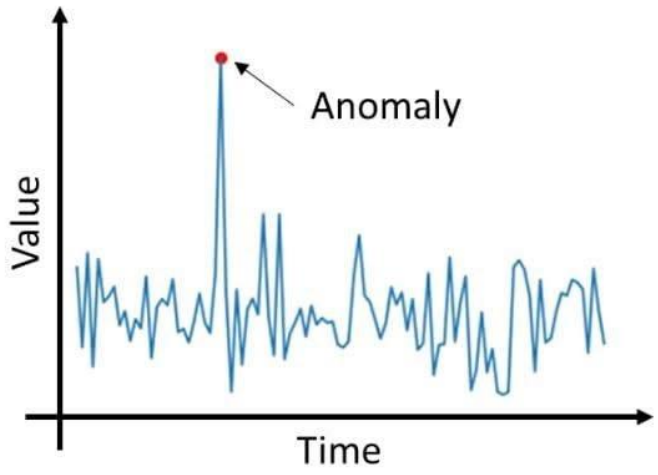
predetermined number of trials. At this stage, the algorithm has successfully determined k distinct clusters, and each data point belongs to one of these clusters.

Figure 3: K-means algorithm framework



Unsupervised learning is also used in algorithms for anomaly detection, which look for instances or data points that drastically vary from predicted behaviour. This can be helpful for fraud detection in financial transactions, where unusual or fraudulent activities are flagged as anomalies.

Figure 4: Anomaly detection



Source: Google

1.4.2 Supervised Learning

Supervised learning entails training a model using classified data, where each data point corresponds with a specific target. The model's aim is to establish a clear relationship between input features and their corresponding labels, enabling it to make precise predictions or classifications on unfamiliar data.

There are two widely recognized types of supervised learning:

a) Classification: This type aims to attribute a label or category to a given input, contingent on its distinctive features. Looking at a previous example, such a model can learn how to classify emails as either spam or non-spam based on various attributes such as the email's content, sender, or subject line.

b) Regression: It is a method of predicting a continuous numerical value from input features. The primary goal of regression is to evaluate the connection between independent and dependent variables, enabling us to predict outcomes or interpret the impact of the independent variables. For instance, a model can be trained to predict a house's price considering factors like size, location, and the number of bedrooms. The model learns the association between the features and the target variable in order accurately predict the price of unseen homes. Now we look at some different kinds of regression.

- Linear Regression: $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n + \varepsilon$

  In this method, it is assumed that there exists a linear correlation between the independent variables (x) and the dependent variable (y). Here, β_0 is the intercept term, βi are the coefficients, and ε represents the error term. The ordinary least squares (OLS) approach is used to estimate the coefficients (βi), which minimizes the sum of squared residuals between the expected and actual values.

- Ridge Regression:

  In order to deal with multicollinearity (high correlation) and avoid overfitting in linear regression, ridge regression is a regularization strategy. The penalty term, which is the sum of the squared coefficients times the regularization parameter λ, is added to the linear regression model. The objective function of ridge regression is:

  $j(b) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - a - b_1X_{i1} - \cdots - b_mX_{im})^2 + \lambda\sum_{j=1}^{m} b_j^2$

  The regularization term shrinks the coefficients, reducing their magnitudes. The parameter λ determines how much shrinkage occurs; higher values result in more severe shrinking.

- Lasso Regression:

  This method incorporates a penalty term into the linear regression model. By constraining some coefficients to absolutely zero, lasso regression creates sparsity by using the L1 norm of the coefficients as the penalty term. The objective function of lasso regression is:

  $$j(b) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - a - b_1X_{i1} - \cdots - b_mX_{im})^2 + \lambda\sum_{j=1}^{m}|b_j|$$

  Lasso regression performs automatic feature selection by effectively setting irrelevant features' coefficients to zero.

1.4.3 Reinforcement Learning

Reinforcement Learning entails directing an agent to engage with an environment, gaining knowledge through feedback, which manifests as either rewards or penalties. The agent understands how to operate in such settings to maximize the cumulative rewards over time. It investigates its surroundings, learns from the repercussions of its activities, and modifies its decision-making method accordingly. A classic demonstration of reinforcement learning involves training an autonomous entity to play in a gaming environment. The player engages in the game environment and earns rewards (e.g., points for winning) or punishments (e.g., losing a life) based on the effects

of those decisions. The agent discovers optimal techniques that lead to higher rewards through trial and error.

1.5 Applications

Machine learning, particularly neural networks, which we will explore in the following section, has seen extensive adoption and integration across different sectors, leading to a significant revolution in various industries, transforming decision-making processes. Its adaptability and capacity to extract valuable insights from vast amounts of data have opened doors to game-changing innovations, enhancing efficiency, accuracy, and overall performance in real-world applications. In healthcare, the impact of AI, particularly through neural networks, has been profound. Medical image processing is one area where these technologies excel, enabling neural networks to detect anomalies in X-rays, MRIs, and mammograms with accuracy. By training on massive volumes of labelled data, these networks empower radiologists with invaluable support in diagnosing diseases at an early stage, which, in turn, leads to better patient outcomes and potentially life-saving interventions. Additionally, AI-driven analysis of complex genetic data holds immense promise in forecasting disease effects, tailoring personalized medical treatments, and even aiding in the development of ground-breaking pharmaceuticals. The transformative potential of neural networks in healthcare extends beyond diagnostics and treatment, as they can also allow patient monitoring, optimize resource allocation in hospitals, and improve overall healthcare management. The banking industry has been another pioneer in the adoption of those technologies, particularly in the realm of fraud detection. Neural networks, with their ability to analyse vast and diverse financial datasets, have proven crucial in spotting patterns indicative of fraudulent transactions. This capability has revolutionized fraud prevention measures, empowering financial organizations to protect their clients' assets and maintain the integrity of their operations. Furthermore, AI-powered credit scoring has transformed the lending landscape, moving beyond traditional credit assessment methods to consider a wide range of data points and past patterns. As a result, financial institutions can offer

more accurate and personalized credit evaluations, expanding access to credit for deserving individuals and businesses. Beyond the mentioned industries, AI and machine learning have found applications in diverse fields, including agriculture, energy, education, and entertainment. In the field of healthcare, specialized algorithms can streamline patient care, treatment planning, and medical imaging analysis, ultimately leading to improved outcomes and more efficient healthcare delivery. The energy sector benefits from AI-driven predictive maintenance in power plants and smart grid optimization, resulting in greater energy efficiency and reduced environmental impact. In education, personalized learning platforms cater to individual students' needs, promoting a more effective and engaging educational experience. The entertainment industry leverages AI-generated content, virtual reality, and personalized content recommendations to enhance user experiences and capture audience attention.

2.0 Artificial Neural Networks

2.1 ANNs Overview

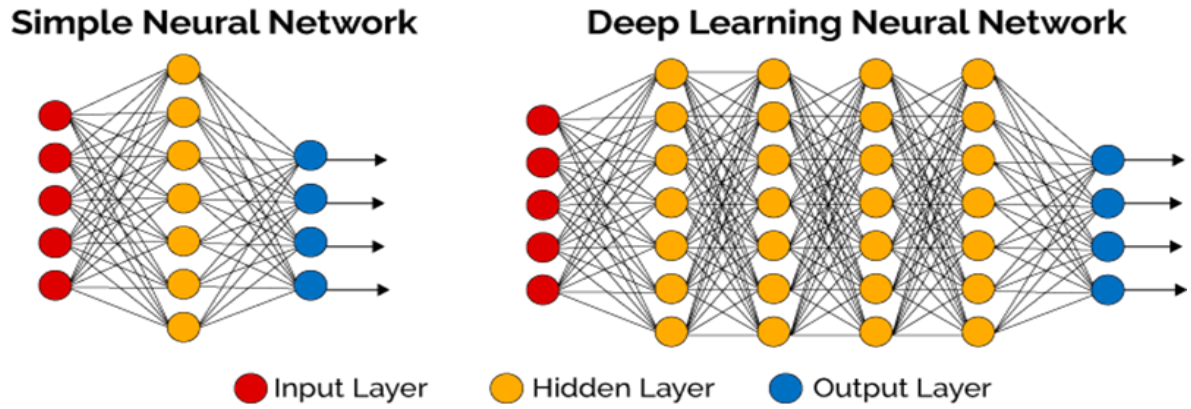
At the core of artificial intelligence and machine learning lie Artificial Neural Networks (ANNs), engineered to replicate the structure and operations of the human brain. ANNs were first proposed in the 1940s by neurophysiologist Warren McCulloch and mathematician Walter Pitts as a mathematical representation of artificial neurons known as McCulloch-Pitts neurons. Successively, Frank Rosenblatt invented the perceptron, an early type of neural network, in the late 1950s and early 1960s. The perceptron was devised to imitate the functioning of a single neuron and was designed to learn from training examples to make forecasts or decisions. However, only after the introduction of deep learning in the late 2000s, neural networks consolidated. Deep learning, often known as deep neural networks, is the development of neural networks having several hidden layers. The effectiveness of deep neural networks relies on their adeptness at discerning intricate patterns within unprocessed data. A standout advantage of deep learning lies in its capacity to overcome the limitations faced by shallow neural networks. Shallow networks typically consist of

only one or two hidden layers, making them less effective at capturing and representing complex patterns in data. On the other hand, deep neural networks possess the capability to iteratively acquire various levels of abstraction, enabling them to discern advanced features from unprocessed input.

2.2 Structure

Artificial Neural Networks are sophisticated structures comprising multiple elements, each playing a crucial role in the network's functionality and learning capabilities. Let us explore each part in detail to understand how they work. Neurons constitute the fundamental building blocks of ANNs. They are responsible for recognizing input signals, conducting computations, and generating output signals. Neurons are structured into layers, typically comprising one or more hidden layers, in addition to an input and an output layer. The input layer takes the raw data, and the hidden and output layers process it to provide predictions or classifications. Feedforward neural networks, also known as single-layer hidden ANNs, consist of an input layer, one hidden layer, and an output layer. On the other hand, multiple hidden layer ANNs, such as deep neural networks (DNNs), have more than one hidden layer. The additional hidden layers allow for more complex representations and hierarchical feature learning. Deep networks can capture and model intricate relationships in the data by progressively extracting abstract features from lower to higher layers. The difference between the single and multiple hidden layers underlies in their structure, as the former are simpler and easier to train but limited in their capacity to learn intricate patterns, whereas the latter can learn highly complex representations but are more challenging to train due to issues like vanishing gradients and overfitting.

Figure 5: Single vs. multiple layers



Source: ResearchGate

Weights are another key feature of ANNs since they represent the strength of the links between neurons. Each neuronal connection has a weight, which defines the impact of the input on the output. Weights change throughout training to improve the network's performance. The network improves to allocate higher weights to connections that contribute more to correct predictions while decreasing weights for less influential connections. Weights always combine with biases, which are values that represent the weighted sums of each neuron's inputs. Biases allow the network to evolve and model more complex connections than a simple linear translation. By incorporating them, the network improves its flexibility and ability to discover intricate patterns in data. Then, we have activations functions, mathematical functions, which are implemented to the weighted sum of inputs in each neuron. These functions set a threshold for connections to react and generate an output. They bring flexibility and non-linearity to the network, enabling it to capture and represent complex patterns in the input data. Activation functions have their own characteristics, influencing how the network learns and generalizes from the data. Here are a few illustrations:

- The sigmoid function: $g(x) = \frac{1}{1+e^{-x}}$

  It transforms input values into a range spanning from 0 to 1, generating a distinctive "S"-shaped curve. The sigmoid function is used as an activation function in the output layer for binary classification tasks, where it helps to interpret outputs as probabilities or make decisions based on a threshold.

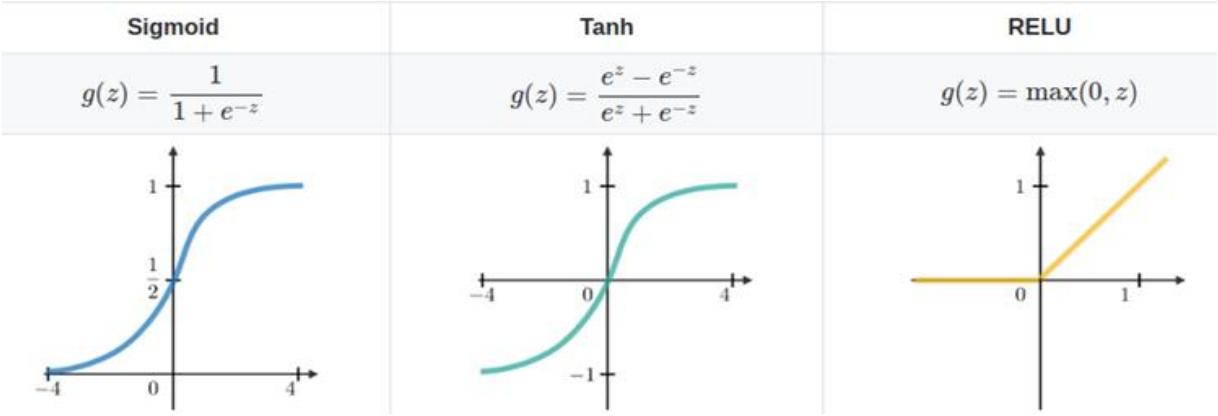
- Rectified Linear Unit (ReLU): $g(x) = \max(0, x)$

If the supplied value is positive, it returns the value, otherwise, it returns zero. The simplicity and effectiveness of ReLU in tackling the vanishing gradient problem have led to its wide acceptance and use. It speeds up training and improves the network's capacity to model complicated patterns.

- Hyperbolic Tangent (tanh): $g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

It maps input values to a range between -1 and 1, producing an "S"-shaped curve cantered at 0. Tanh enables the network to capture non-linear relationships in the data, making it suitable for hidden layers in ANNs.

Figure 6: Activation functions



Source: ResearchGate

2.3 ANNs training

2.3.1 Data Preparation and Setup

Artificial Neural Networks undergo a multi-stage training process that iteratively enhances their efficiency. It begins with the preparation of a dataset, establishing the groundwork for subsequent stages. At this outset, a raw dataset is collected and organized, often necessitating pre-processing steps like data cleaning, normalization, and feature scaling to ensure uniformity and enhance convergence during training. For instance, in a dataset containing housing prices, features like square footage and number of bedrooms might be normalized to a common scale to prevent one

feature from dominating others in the network's calculations. Thereafter the dataset is divided into a training set and a validation set. This partitioning facilitates the evaluation of the network's performance on unseen data and guards against overfitting.

2.3.2 Initialization and setup

During the training phase, input data is fed into the network's input layer, and computations are carried out within each neuron. For instance, in a feedforward neural network, let's consider a neuron with inputs, x1, x2, x3 and weights w1, w2, w3. The weighted sum z is calculated as z = w1x1+w2x2+w3x3. Biases b are then added, yielding z+b, which undergoes a non-linear activation function such as the sigmoid function. By employing this activation function, non-linearity is introduced, allowing the network to grasp intricate patterns within the data. By choosing appropriate initial weights and biases, like w1=0.2, w2=−0.3, w3=0.5, and b=0.1, the network is primed to better fit the data distribution. Proper initialization set the stage for forward and backward computations, ultimately leading to the network's ability to learn and generalize from the training data.

2.3.3 Error calculation (loss function)

Error calculation involves measuring the difference between the predictions made by the network and the real target values. This process guides the network towards refining its parameters for improved accuracy. The loss function, often denoted as L, serves as the mathematical measure of this discrepancy. For instance, in a regression task predicting house prices, the mean squared error (MSE) can be used as the loss function:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \widehat{y_i})^2$$

Where N is the number of examples, y_i is the actual house price for the ith example, and $\widehat{y_i}$ is the predicted price. The loss function assesses the mean squared difference between the anticipated and real values across all cases. During forward propagation, as inputs pass through the network, the

predicted outputs are generated. Afterward, the loss function is employed to contrast these forecasts against the genuine target values. This results in a singular metric indicating the network's overall effectiveness in relation to the provided instances. Throughout the training process, the objective is to systematically refine the network's parameters using optimization methods such as gradient descent to reduce the loss function. In essence, error calculation and the associated loss function provide a clear and quantifiable measure of the network's accuracy, enabling the optimization process to steer the network towards better predictions

2.3.4 Backpropagation

The backpropagation stage is a pivotal step in the training of Artificial Neural Networks (ANNs), responsible for refining the network's performance by adjusting its parameters based on the calculated error. This process involves attributing the error backwards through each layer of the network, determining how much each neuron contributed to the overall discrepancy between predictions and actual values. For instance, in a multi-layer feedforward network, the error attributed to a neuron j in a hidden layer is computed by considering its impact on the subsequent layer's errors:

$$\delta j = f'(z_j) \sum_k w_{jk}\, \delta_k$$

Here, $f'(z_j)$ represents the derivative of the activation function applied to the weighted sum *zj* of neuron *j, wjk* signifies the weight connecting neuron j to neuron k in the subsequent layer, and δk represents the error attributed to neuron k in the next layer. By considering how much each parameter contributed to the error, the network learns to update its parameters in a direction that reduces the overall discrepancy. Through this iterative process, the network refines its capacity to detect intricate patterns within the data. Backpropagation, coupled with gradient descent optimization, empowers ANNs to progressively improve their predictive accuracy over successive training iterations. As part of the backpropagation process, the error is allocated to neurons in the preceding layers, proportionate to their respective impact on the overall error. This assignment is

determined by the weights interconnecting the neurons. In parallel, the weights of the neurons undergo refinement to reduce the error. This weight adjustment procedure frequently employs optimization techniques, with the gradient descent algorithm being one of the most widely used methods. This weight adjustment procedure frequently employs optimization techniques, with the gradient descent algorithm being one of the most widely used methods.

2.3.5 Optimization

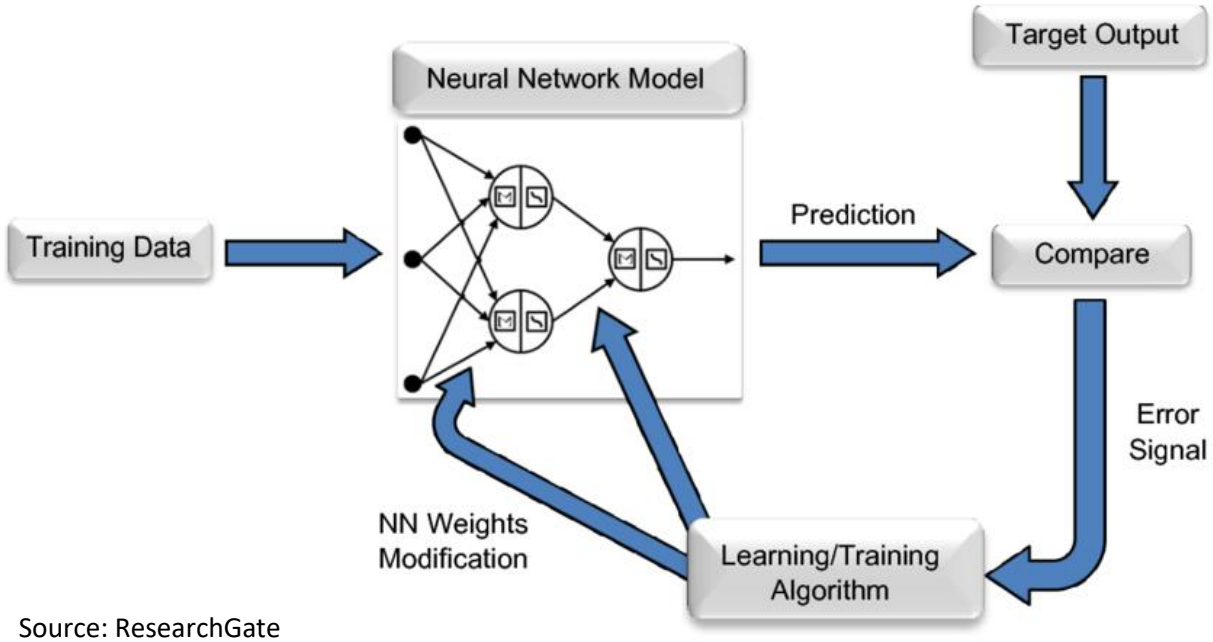
In the optimization process, adjustments are made to the network's parameters with the aim of minimizing the specified loss function. Various optimization algorithms are used to achieve this, each employing distinct strategies to adjust the parameters. These algorithms include not only gradient-based methods like gradient descent but also more advanced approaches such as stochastic gradient descent (SGD), mini-batch gradient descent. The fundamental principle behind these optimization methods remains the same: iteratively update the network's parameters in a direction that reduces the loss. The specific direction and step size are determined by the algorithm's mechanics. Instead of focusing solely on gradients, these methods consider factors like historical gradient information, adaptive learning rates, and momentum to navigate the parameter space effectively. The iterative optimization process continues until a desirable level of accuracy is reached.

2.3.6 Convergence and Accuracy

Convergence and accuracy mark the culmination of the intricate training process for Artificial Neural Networks (ANNs). Convergence denotes the point at which the network's parameters have been iteratively adjusted to minimize the loss function, resulting in the network's predictions approaching a stable state. This implies that the network has effectively acquired meaningful patterns and relationships within the training data. Simultaneously, accuracy refers to the network's ability to make precise predictions on both the training data and unseen examples. As the training advances, the loss diminishes, leading to enhanced accuracy hen making predictions on validation

or test datasets. Achieving convergence and high accuracy underscores the successful acquisition of the underlying data patterns by the network, ultimately enabling it to generalize and make accurate predictions on new and previously unseen data.

Figure 7: Neural network training process



Source: ResearchGate

3.0 The Gradient Descent Algorithm

3.1 Overview

Gradient descent serves as a fundamental optimization method extensively employed in machine learning. Its main objective is to systematically fine-tune a model's parameters with the aim of minimizing a cost function. In the realm of artificial neural network (ANN) training, this cost function quantifies the disparity between the network's predicted outputs and the true target values. Thus, this approach relies on the idea of iteratively adjusting model parameters by traveling in the direction of the loss function's steepest fall. This is accomplished by computing the loss function's gradient with respect to the parameters and updating the parameters in the gradient's opposite direction. In each iteration, the update's magnitude is governed by the learning rate, which determines the size of the step taken. Throughout this cyclical process, the algorithm settles on the collection of parameters which result in the least amount of loss. While gradient descent is a

commonly used optimization technique in practical applications, it may encounter challenges in finding the global minimum as it can sometimes get trapped in a local minimum or a saddle point. Moreover, it offers some beneficial aspects that make it an ideal candidate.

Scalability: In the context of complex models with numerous parameters, such as deep neural networks, gradient descent efficiently adjusts these parameters in a manner that is computationally feasible even for large datasets.

Non-Convex Optimization: Many real-world loss functions are non-convex, possessing multiple local minima. Gradient descent navigates this landscape by leveraging its iterative nature to find promising parameter configurations.

Generalization: Properly configured gradient descent helps prevent overfitting by guiding parameter updates towards regions of the parameter space that generalize well to unseen data.

3.2 Optimization Problems and Requirements

ANNs are composed of multiple interconnected layers, each containing numerous parameters (weights and biases) that influence the network's behaviour. The purpose of training is to identify the best parameter values that minimize a predetermined cost function. The cost function evaluates the difference between the network's forecasts and the real target values within the training dataset. Solving the optimization problem entails searching for parameter values that lead to the lowest possible cost, thereby ensuring accurate and meaningful predictions on unseen data. Optimizing ANNs poses several challenges due to the complex and non-convex nature of the parameter space.

High-Dimensional and Non-Convex Space: The vast number of parameters in ANNs results in a high-dimensional parameter space, often characterized by numerous local minima, maxima, and saddle points. This non-convex landscape complicates the optimization process, as traditional methods may get stuck in suboptimal solutions.

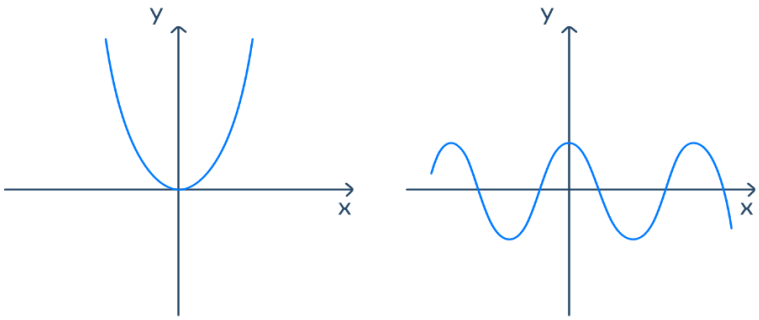
Vanishing and Exploding Gradients: During backpropagation, gradients can either vanish (become extremely small) or explode (become extremely large) as they propagate through multiple layers. This phenomenon can hinder or accelerate the learning process, respectively.

Slow Convergence: The parameter space's complex geometry may lead to slow convergence, where the optimization process takes a substantial number of iterations to reach an acceptable solution.

Gradient descent is applicable to functions that meet two key criteria: they must be both differentiable and convex.

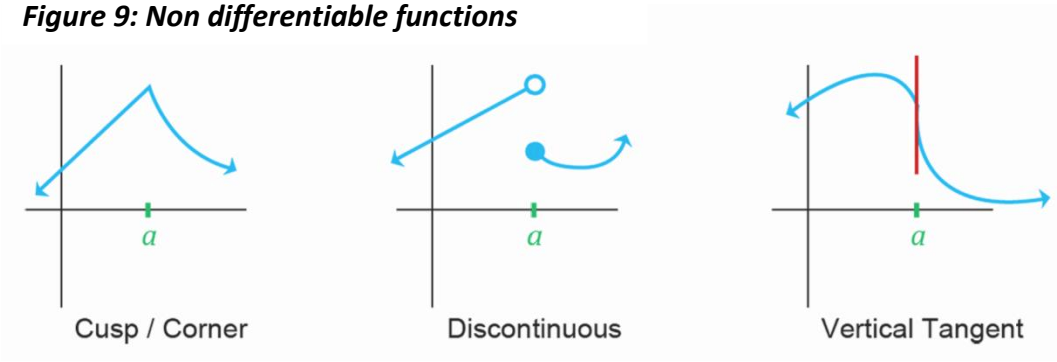
If a function is differentiable means, it has a well-defined derivative at every point within its domain, allowing us to analyse how small changes in inputs affect the output. In optimization, derivatives play a crucial role as they can signify critical points such as minima, maxima, or saddle points when they equal zero. For a function having this characteristic is crucial for gradient descent because it enables the calculation of the gradient, which provides the direction to update the parameters. Functions that are not differentiable at certain points might have sharp corners, cusps, or discontinuities, making the gradient undefined or misleading. In such cases, alternative optimization methods may need to be considered. Let's begin by examining examples of functions that satisfy these conditions.

Figure 8: Differentiable functions



While non-differentiable functions often exhibit features like steps, cusps, or discontinuities.

Figure 9: Non differentiable functions



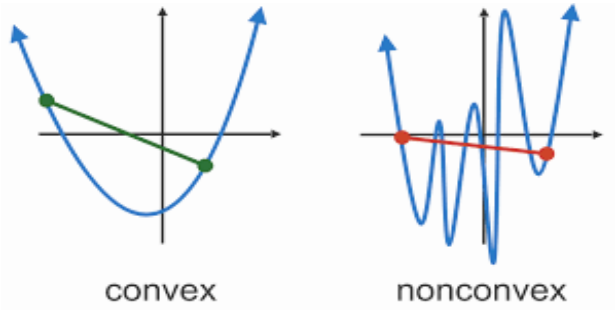
Now, let's move on to the next requirement, which is that the function must be convex. In the case of a univariate function, a function is considered convex when, for any two points within its domain, the straight line connecting those two points always remains above the graph of the function. In simpler terms, the function always lies beneath the secant line formed between any two points in its domain. Mathematically, a function $f(x)$ is convex if, for all x_1 and x_2 in its domain and for all λ in the range [0, 1], the following inequality holds:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$

Convex functions play a crucial role in gradient descent optimization due to their well-behaved nature. When dealing with a convex function, gradient descent is highly likely to converge to the global minimum rather than getting stuck in local minima or saddle points. The fact that the entire region around a point on a convex function lies above its tangent plane ensures that each step taken by the gradient descent algorithm moves it closer to the optimal solution. Convexity also guarantees that there is only one possible minimum, which simplifies the optimization process. This contrasts with non-convex functions, where multiple local minima might exist, making it challenging for gradient descent to find the best solution.

Below there are two functions with exemplary section lines.

Figure 10: Convex vs. non-convex function



Another mathematical approach to determine the convexity of a univariate function is by evaluating its second derivative and ensuring it remains greater than zero throughout its domain.

$$\frac{\partial^2 f(x)}{\partial x^2} > 0$$

Let's explore this concept with a simple quadratic function:

$$f(x) = x^2 + 3x - 1$$

Its first and second derivative are:

$$\frac{\partial f(x)}{\partial x} = 2x + 3, \qquad \frac{\partial^2 f(x)}{\partial x^2} = 2$$

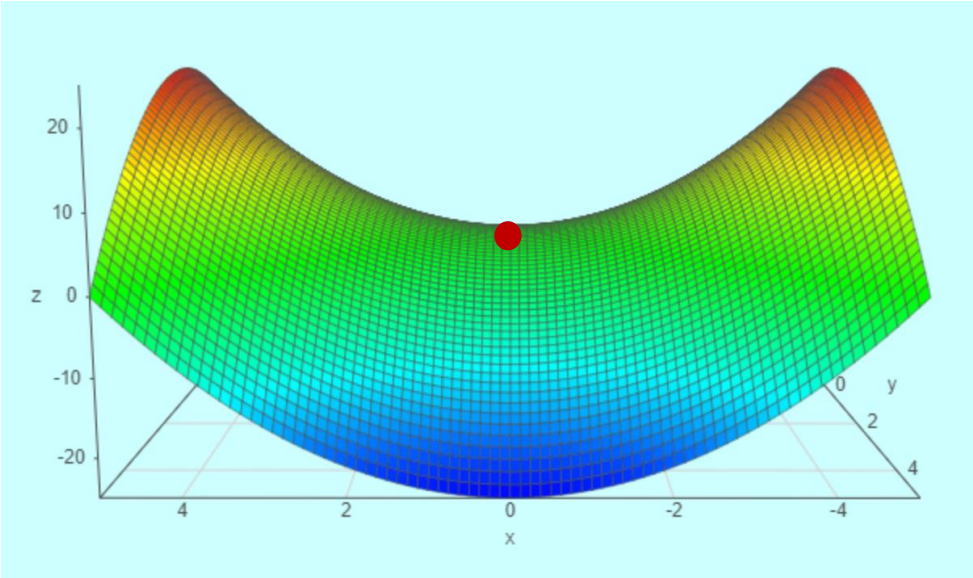
Because the second derivative is always bigger than 0, our function is strictly convex.

For multivariate functions, assessing whether a point is a saddle point involves a more intricate calculation, specifically the computation of a Hessian matrix.

Example of a saddle point in a bivariate function is show below.

$$z = x^2 - y^2$$

Figure 11: Saddle point of a bivariate function



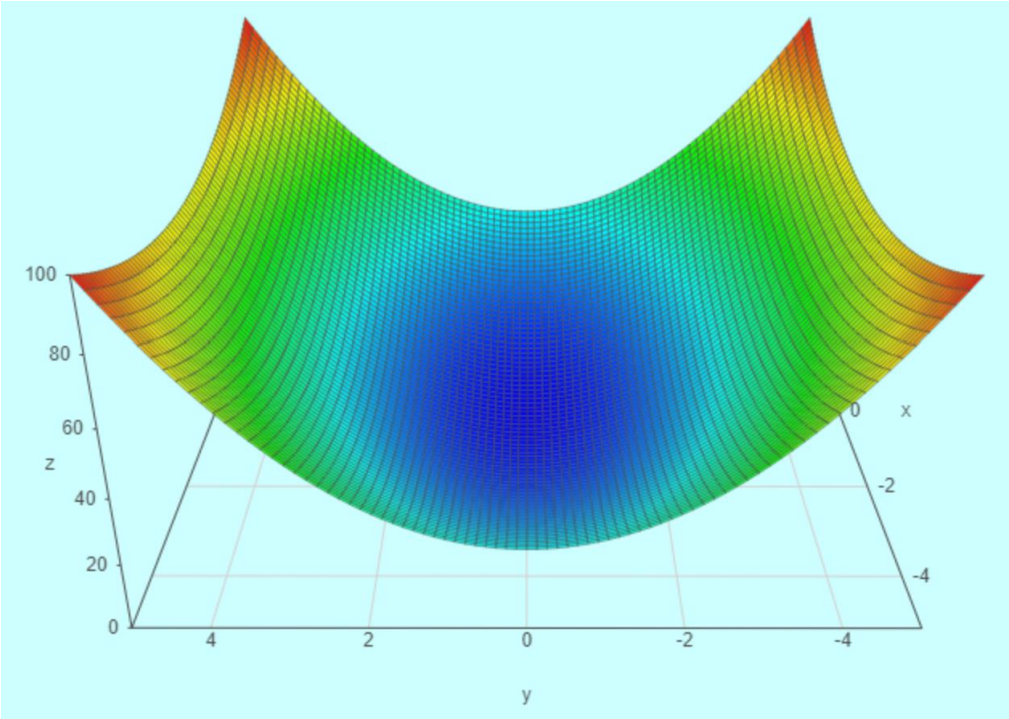
3.3 How does it work?

Intuitively, the gradient represents the rate of change of a function at a specific point, but it also considers a specified direction. In the context of a single-variable function, it's essentially the first derivative calculated at a chosen point. However, when dealing with multivariable functions, it becomes a vector that describes the derivatives along each of the coordinate axes. These individual derivatives are referred to as partial derivatives since they capture the slope along one axis while disregarding the others. The gradient of an n-dimensional function, denoted by $\nabla f(p)$, at a given point p is defined as follows:

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$$

To gain a clearer understanding of how to compute it, let's work through a manual calculation for a two-dimensional function provided below.

$$f(x) = x^2 + 3y^2$$



Let's assume we are interested in a gradient at point p (5,5):

$$\frac{\partial f(x,y)}{\partial x} = 2x, \frac{\partial f(x,y)}{\partial y} = 6y$$

so consequently:

$$\nabla f(x,y) = \begin{bmatrix} 2x \\ 6y \end{bmatrix}$$

$$\nabla f(10{,}10) = \begin{bmatrix} 10 \\ 30 \end{bmatrix}$$

By looking at these values we conclude that the slope is three times steeper along the y axis. The Gradient Descent Algorithm iteratively approximates the next point by utilizing the gradient at the current position, rescaling it using a designated learning rate, and then subtracting this value from the present position, effectively taking a step forward. Because we want to minimize the function, it subtracts the value. The formula for this process is:

$$p_{n+1} = p_n - \eta \nabla f(p_n)$$

There is a crucial parameter which adjusts the gradient and, thus, regulates the step size. It is known as the learning rate, and it holds a significant influence on performance outcomes.

In brief, the various steps of the Gradient Descent algorithm are:

1. Begin by selecting an initial position at random (initialization).
2. Define a loss function that needs to be minimized.
3. Calculate the gradient and take a step in the opposite direction, scaled accordingly (aiming for minimization).
4. Continue with steps 3 and 4 until one of the following conditions is met:
    - The step size falls below a specified tolerance due to scaling or a shallow gradient, and the maximum allowed iterations have been reached.
    - The optimizer reaches an absolute minimum.

3.4 Intuition Behind Gradient Descent

To gain a better understand of the gradient descent algorithm, picture a drone capturing footage as it descends from a high altitude or a paper airplane gliding down from a tall building. Throughout their descent, both the drone and the paper airplane are drawn in the direction of the steepest decline, consistently moving towards the lowest point.

The concept underlying gradient descent shares a similarity: it initiates from an arbitrarily chosen position represented by the point or vector $\mathbf{v} = (v_1, \ldots, v_r)$ and gradually shifts it towards the direction of the steepest decline in the cost function. As previously mentioned, this direction corresponds to the negative gradient vector, $-\nabla C$.

Upon establishing an initial point $\mathbf{v} = (v_1, \ldots, v_r)$, the update process begins, propelling it towards a new position following the path of the negative gradient: $\mathbf{v} \rightarrow \mathbf{v} - \eta \nabla C$, where η is a small positive value known as the learning rate. This learning rate essentially governs the size of the step taken

during the update, making it a crucial parameter. Its significance will become evident in the upcoming section.

Should η be exceedingly small, the algorithm may converge at a sluggish pace. Conversely, overly large η values can lead to convergence difficulties or even cause the algorithm to diverge.

3.5 Learning Rate

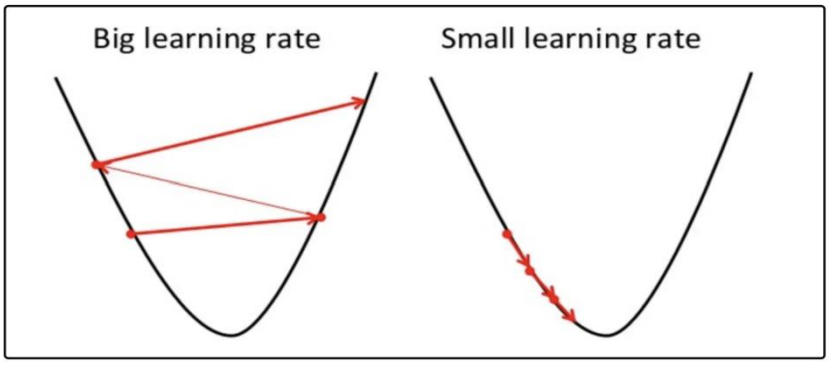
The learning rate, as a hyperparameter, governs the magnitude of adjustments made in the parameter space when updating the model's parameters in every cycle of the gradient descent algorithm. A carefully chosen learning rate is crucial for achieving efficient and stable convergence, while an inappropriate learning rate can hinder progress or even cause the algorithm to fail.

Effect of Learning Rate:

Large Learning Rate: When the learning rate is excessively large, there's a risk of the algorithm surpassing the optimal solution. This can lead to oscillations around the optimal point or even cause the algorithm to diverge entirely, preventing convergence. In such cases, the algorithm might keep bouncing between different points without getting closer to the minimum of the cost function.

Small Learning Rate: On the other hand, an extremely small learning rate slows down the convergence process. The algorithm takes tiny steps, which could make it get trapped in local minima or saddle points for a long time. This results in slow progress towards the optimal solution, increasing the time required to achieve convergence.

Figure 12: Big vs. Small learning rate



Choosing the right learning rate

The choice of learning rate plays a pivotal role in determining the effectiveness and efficiency of optimization algorithms, particularly in the context of machine learning and deep learning models. It directly impacts the convergence speed and stability of these algorithms, ultimately influencing the model's ability to find an optimal solution. A suboptimal learning rate can lead to several issues, such as slow convergence, overshooting, or even divergence, hampering the entire optimization process. Hence, selecting an appropriate learning rate is essential for achieving reliable and effective optimization outcomes.

Strategies for Determining the Optimal Learning Rate:

Learning Rate Schedules: Fixed learning rates might work in simple cases, but they can be limiting when dealing with complex optimization landscapes. Learning rate schedules offer a more flexible approach. By adjusting the learning rate dynamically as the optimization progresses, these schedules strike a balance between quick initial progress and fine-tuning as the algorithm gets closer to the optimal solution. This adaptive adjustment mitigates the risk of overshooting and enables a smoother convergence trajectory.

Adaptive Methods: Adaptive optimization methods have gained popularity due to their ability to autonomously modify the learning rate during training. Techniques like AdaGrad, RMSProp, and Adam maintain a history of past gradient magnitudes and adjust the learning rate accordingly. This approach ensures that each parameter receives an appropriate learning rate based on its individual behaviour during optimization. This adaptivity allows these methods to converge faster and more reliably across a wide range of problems.

Grid Search and Random Search: Hyperparameter tuning methods, such as grid search and random search, are valuable tools for identifying an optimal learning rate. By systematically or randomly exploring a predefined range of learning rates and observing their impact on convergence,

practitioners can gather insights into the learning rate's behavior within the context of their specific problem. This empirical approach guides the selection of a suitable learning rate, helping avoid the guesswork associated with a fixed value.

4.0 Conclusion

In conclusion, the remarkable efficacy of gradient descent solidifies its status as the heart of Artificial Neural Network training, orchestrating a dynamic force that shapes the landscape of optimization. Its iterative finesse stands as a pivotal bridge connecting the realms of prediction and reality, seamlessly navigating the intricate, nonlinear terrains that characterize modern machine learning challenges. Notably, it demonstrates a profound adaptability, effortlessly accommodating an array of architectures and loss functions.

Balancing on the fine line between convergence and divergence, gradient descent expertly guides the training process. It catalyses efficient parameter updates, a vital mechanism in the relentless pursuit of model refinement. In doing so, it deftly steers neural networks away from local minima, mitigating the challenges posed by gradients and, in the process, accelerates the path to convergence.

Beyond its mechanical function, gradient descent emerges as an embodiment of the art of optimization itself. It signifies a ceaseless quest for efficiency and accuracy in the expansive realm of machine learning, driving researchers and practitioners to continually seek innovative ways to harness its potential. In this light, the algorithm transcends its technical nature, representing a profound dedication to excellence that characterizes the ever-evolving landscape of artificial intelligence.

As we delve deeper into the intricate world of neural network training, gradient descent remains not only a powerful tool but also a testament to the unwavering commitment to pushing the boundaries

of what is possible. Its legacy continues to influence the trajectory of artificial intelligence, shaping the future of innovation and discovery in this dynamic field.

Bibliography

Mirko Stojiljković (2020), Stochastic Gradient Descent Algorithm with Python and NumPy.

Retrieved from: https://realpython.com/gradient-descent-algorithm-python/

Robert Kwiatkowski (2021), *Towards Data Science: Gradient Descent Algorithm - a deep dive.*

Retrieved from: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

Niklas Donges (2023), *Gradient Descent in Machine Learning: A Basic Introduction*

Retrieved from: https://builtin.com/data-science/gradient-descent

Joris Langeveld (2021), *Machine learning in quay wall design*, KTH Royal Institute of Technology

Retrieved from: https://www.diva-portal.org/smash/get/diva2:1598164/FULLTEXT01.pdf

Arwa E. Abulwafa (2022), Nile Journal of Communication & Computer Science, Volume 3, Number 1.

Retrieved from: https://njccs.journals.ekb.eg

Nijiati Abulizi (2023), *Exploring the Differences Among Linear Models: From Ordinary Least Squares to Polynomial Regression.*

Retrieved from: https://nijat.medium.com/exploring-the-differences-among-linear-models-from-ordinary-least-squares-to-polynomial-regression-687ed4a05d9e

Arun Kumar Pandey (2022), *Regression algorithms.*

Retrieved from: https://medium.com/@arunp77/regression-algorithms-29f112797724

Marcin Frąckiewicz (2023), *AI Data Partitioning: A Critical Step in the Machine Learning Pipeline.*

Retrieved from: https://ts2.space/en/ai-data-partitioning-a-critical-step-in-the-machine-learning-pipeline/

Tahera Firdose(2023), *Understanding Outliers: Impact, Detection, and Remedies*.

Retrieve from: https://tahera-firdose.medium.com/understanding-outliers-impact-detection-and-remedies-ea2192174477

Leonid Vadim (2023), *Neural Networks: How they Work and Why they Matter*.

Retrieved From: http://web.archive.org/web/20230601100519/https://manandtech.com/neural-networks/

John C Hull (2021), *Machine Learning in Business: An Introduction to the World of Data Science, third edition.*