AI PUBLISHING

Deep Learning Fundamentals for Beginners

Learning AI and
Data Science easily

Our Books are designed
to teach beginners
Data Science and AI

DEEP LEARNING
FUNDAMENTALS
FOR BEGINNERS

AI PUBLISHING

First Printing, 2019

Edited by AI Publishing
Ebook Converted and Cover by Gazler Studio
Published by AI Publishing LLC

ISBN-13: 978-1-7330426-6-6
ISBN-10: 1-7330426-6-6

About the Publisher

At AI Publishing Company, we have established an international learning platform specifically for young students, beginners, small enterprises, startups, and managers who are new to data sciences and artificial intelligence.

Through our interactive, coherent, and practical books and courses, we help beginners learn skills that are crucial to developing AI and data science projects.

Our courses and books range from basic intro courses to language programming and data sciences to advanced courses for machine learning, deep learning, computer vision, big data, and much more, using programming languages like Python, R, and some data science and AI software.

AI Publishing's core focus is to enable our learners to create and try proactive solutions for digital problems by leveraging the power of AI and data sciences to the maximum extent.

Moreover, we offer specialized assistance in the form of our free online content and ebooks, providing up-to-date and useful insight into AI practices and data-science subjects, along with eliminating the doubts and misconceptions about AI and programming.

Our experts have cautiously developed our online courses and kept them concise, short, and comprehensive so that you can understand everything clearly and effectively and start practicing the applications right away.

We also offer consultancy and corporate training in AI and data sciences for enterprises, so that their staff can navigate through the workflow efficiently.

With AI Publishing, you can always stay closer to the innovative world of AI and data sciences.

If you are also eager to learn the A to Z of AI and data sciences but have no clue where to start, AI Publishing is the finest place to go.

Please contact us by email at: contact@aispublishing.net.

AI Publishing is searching for author like you

If you're interested in becoming an author for AI Publishing, please contact us at authors@aispublishing.net.

We are working with developers and AI tech professionals, just like you, to help them share their insight with the global AI and Data Science lovers. You can share all subject about hot topic in AI and Data Science.

Download the color images

We also provide a PDF file that has color images
of the screenshots/diagrams used in this book.
You can download it here:

https://www.aispublishing.net/book-dl-fundamentals

Get in touch with us

Feedback from our readers is always welcome.

For general feedback please send us an Email at
contact@aipublishing.net
and mention the book title in the subject of your message.

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you could report this to us as soon as you can.

If you are interested in becoming an AI Publishing author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please send us a Email at
author@aipublishing.net

How to contact us

If you have any feedback, please let us know by sending an email to contact@aispublishing.net.

This feedback is highly valued, and we look forward to hearing from you. It will be very helpful for us to improve the quality of our books.

Free Introductory ebook in Neural Networks

As a way of saying thank you for your purchase, AI Publishing is offering you a free book in Neural Networks.

Inside this book you will learn some of the most cutting-edge neural networks algorithms and techniques and through this book you will gain an immense amount of valuable examples.

So, what are you waiting for?

Click to the link below to get your free ebook

https://www.aispublishing.net/introduction_ann

Table of Contents

Preface

§ Who should read this book?

This book is written for beginners and novices who want to develop fundamental data science skills and learn how to build models that learn useful information from data. This book will prepare the learner for a career or further learning that involves more advanced topics. It contains an introduction and fundamental concepts used in data science and deep learning. The learner does not need to have any prior knowledge of machine learning or deep learning, but some basic understanding of mathematics is required.

§ Why this book?

This book contains a quick introduction and implementation of data science concepts. The working of each algorithm is traced back to its origin in probability, statistics, or linear algebra, which helps the learner understand the topics better. The concepts of probability and statistics are defined and explained at a rudimentary level to make things simple and easy to comprehend. For intuitive understanding, algorithms have been explained through proper visualizations and various examples. New notations or symbols are introduced and

described inside relevant topics only to create a flow in the content. In short, the material has been designed and written in a way as to not overwhelm a novice reader.

The book explains theoretical concepts behind the working of machine learning and deep learning algorithms. Each chapter begins with an explanation of the chapter's content relevant to data science.

§ Book Approach

This book assumes that you know nothing about deep learning. Its goal is to give you the concepts, the intuitions, and the tools you need to actually learn how deep learning works and how and when to use it.

§ What are data science and deep learning?

Today, we are bombarded with information that is generated through machines in all corners of the world. From surveillance cameras, GPS trackers, satellites, and search engines to our mobile phones and smart appliances in the kitchen, all these entities generate some kind of data. Usually, it contains information about users: their routines, their likes and dislikes, their choices, or even their work hours.

The most important reason for the growth of machine learning in recent years is the exponential growth of available data and computing power. Surveillance cameras, GPS trackers, satellites, social media, and millions of other such entities generate data. Data about users' habits, routines, likes, and dislikes is collected through various apps and during web surfing.

So out of all this data, we need to extract useful and relevant information, and this is what data science is all about. Data science is actually *making sense of the data*.

Today, the research is more focused on making sense of this data and extracting useful information from it. By collecting and analyzing large-scale data, not only can we develop useful applications, but we can also tailor the application for personalized use as per each user's needs. Statistics and probability provide the basis to carry out data analysis in data science. These play a crucial role and are the most important requirements to learn about.

§ Data science applications

Data science has been applied to a vast range of domains, like finance, education, business, and healthcare. Data science is a powerful tool in fighting cancer, diabetes, and various heart diseases. Machine learning algorithms are being employed to recognize specific patterns for symptoms of these conditions. Some machine learning models can even predict the chance of having a heart attack within a particular time frame. Cancer researchers are using deep learning models to detect cancer cells. Research is being conducted at UCLA to identify cancer cells using deep learning.

Deep learning models have been built that accurately detect and recognize faces in real time. Through such models, social media applications like Facebook and Twitter can quickly recognize the faces in uploaded images and can automatically tag them. Such applications are also being used for security purposes.

Speech recognition is another major success and a dynamic area of research. The machine learns to recognize the voice of a person, can also convert the spoken words to text, and can understand the meaning of those words to get the command.

One of the hottest research areas is self-driving cars. The car learns to drive as it interacts with its environment, using data from the camera and various sensors. Those cars use deep learning and learn to recognize and understand a stop sign, differentiate between a pedestrian and a lampost, and avoid collision with other vehicles.

1

Fundamentals of Probability

1.1. Why Is Probability Important for Data Science?

Data science uses inferential statistics to analyze trends from data, while inferential statistics makes use of probability distributions of data to base its results. As machine learning and deep learning deal with non-deterministic random and stochastic processes, probability helps to deal with uncertainties. We use probability to estimate and predict unseen data by learning the underlying patterns and distributions of available data. Probability is also used in building machine learning classification models, e.g., the probability that an image contains a cat given some certain features extracted from the image.

1.2. Basic Definitions

1.2.1. Experiment

Experiments are situations that have uncertain outcomes. For example, for a dice roll experiment, there can be six possible

outcomes (assuming the use of one six-faced die), and we are uncertain about which face will come up.

1.2.2. Event

An event is one or more outcomes of the experiment. An event is represented as a set. For example, in a dice roll experiment, an event of getting a three face up will be:

$$E = \{3\}$$

1.2.3. Certain Event

An event which is certain to occur, e.g.,

$$E = The\ outcome\ of\ a\ coin\ flip\ will\ either\ be\ heads\ or\ tails$$

1.2.4. Impossible Event

An event which cannot occur or is impossible, e.g.,

$$E = The\ outcome\ of\ a\ dice\ roll\ is\ 7$$

1.2.5. Sample Space

Sample space is the set of all possible outcomes of an experiment. For example, the sample space of a coin flip is:

$$S = \{Head, Tail\}$$

The sample space of a dice roll is:

$$S = \{1, 2, 3, 4, 5, 6\}$$

1.2.6. Random Variable

The random variable maps the set of possible outcomes of an experiment to measurable space (numerical outcomes). In

other words, it is just a way to represent the outcomes of an experiment.

For example, the outcomes of a coin flip experiment can be represented using a random variable:

Random Variable (X)	Experiment Outcome
0	Heads
1	Tails

A random variable can be of two categories:

- Discrete
- Continuous

A discrete random variable can only take certain fixed values or finite values within a range. For example, a fair coin flip can only have two outcomes: heads and tails. It cannot have any other outcome. Similarly, in a dice roll, the top face can only have a value from one to six. It cannot have any value outside this range. It cannot even be 3.5 or 5.5 or any other decimal value.

A continuous random variable can have an infinite number of possible values, e.g., heights of randomly selected trees or temperature in a day. Here, the height of a tree can be any value, like 20 feet, 20.4 feet, or 26.555 feet.

1.3. Probability in Practice

Probability is the likelihood of the occurrence of an event in an experiment. It is the chance that a specific event will occur. For example, there is a 50 percent chance of getting heads in a coin flip and a 50 percent chance of getting tails. There is one out of six chance that we will get a six in a dice roll.

The mathematical expression of probability for an event to occur is given by:

$$P = \frac{Number\ of\ ways\ an\ event\ can\ occur}{Total\ number\ of\ outcomes\ of\ experiment}$$

The probability value of an event is always between zero and one. An impossible event has a probability value of zero, and an event that will undoubtedly occur has a probability value of one.

Probability Example 1:

Consider the experiment of rolling a die. Our sample space would be:

$$S = \{1, 2, 3, 4, 5, 6\}$$

To find the probability of getting a three in a dice roll, our event will be:

$$E = \{3\}$$

So, our probability will be:

$$P = \frac{n(E)}{n(S)} = \frac{1}{6}$$

Where n(E) means the number of elements in an event set, and n(S) stands for the number of elements in the sample space.

To find the probability of getting an odd number in the dice roll, our event will be:

$$E = \{1, 3, 5\}$$

Because these are the odd numbers from our sample space (possible outcomes of the experiment). So our probability will be:

$$P = \frac{n(E)}{n(S)} = \frac{3}{6} = \frac{1}{2}$$

Probability Example 2:

Consider another example of a bag containing six red balls and four blue balls.

If we randomly draw a ball from the bag,

The probability of picking a red ball is: $\frac{6}{10} = \frac{3}{5}$

The probability of picking a blue ball is: $\frac{4}{10} = \frac{2}{5}$

1.3.1. Conditional Probability

Conditional probability is the probability for an event to occur, given that another event has already occurred. The probability that event A will occur given that event B has occurred is given by:

$$P(A \mid B) = \frac{P(A \text{ and } B)}{P(A)}$$

To understand the reason to introduce conditional probability, we need to go through a few concepts that will explain the need for it.

1.3.2. Equally Likely Events

If two events have the same probability of occurrence, the two events are said to be equally likely. For example, the following two events are equally likely (50-50 chance) in an experiment of coin flip.

$$E_1 = \{Heads\}, E_2 = \{Tails\}$$

1.3.3. Mutually Exclusive Events

Two events are said to be mutually exclusive if they both cannot occur at the same time. In other words, the two events do not share outcomes.

To elaborate, consider an experiment of a dice roll and two following outcome events:

$$E_1 = Getting\ an\ odd\ number\ on\ dice\ top\ face = \{1, 3, 5\}$$

$$E_2 = Getting\ an\ even\ number\ on\ dice\ top\ face = \{2, 4, 6\}$$

These two events are mutually exclusive, as there is no overlap in their elements. So, the probability that both events will occur is zero.

$$P\ (A\ and\ B) = 0$$

Now consider these two events:

$$E_1 = Getting\ a\ three\ on\ dice\ top\ face = \{3\}$$

$$E_2 = Getting\ an\ odd\ number\ on\ dice\ top\ face = \{1, 3, 5\}$$

In this case, the two events are not mutually exclusive, as the number three is common to both events, hence:

$$P\ (A\ and\ B) = \frac{1}{6}$$

1.3.4. Independent Events

Two events (A and B) are independent when the outcome of one event does not affect the outcome of the other event. For example, if we roll two dice, the probability of getting a six on the first dice roll (event A) does not affect the probability of getting a six on the second dice roll (event B). So, both dice have independent 1/6 probabilities of getting a six outcome.

If two events are independent, then the probability that both events will occur:

$$P\ (A\ and\ B)\ =\ P(A)\ .P(B)$$

1.3.5. Dependent Events

Two events are dependent when the outcome of one event affects the outcome of the other event. For example, if we draw two balls without replacement (draw the first ball and not replace it back in the bag) from a bag containing six red and four blue balls, the probability of the second ball being red or blue depends on the probability of the first ball being blue or red. To elaborate, consider the probability for the color of the first ball:

$$P(red)\ = \frac{6}{10}, P(blue)\ = \frac{4}{10}$$

Now let's suppose that the first draw was red. Then, we will be left with five red balls and four blue balls. That means, for the second ball:

$$P(red)\ = \frac{5}{9}, P(blue)\ = \frac{4}{9}$$

If the first draw were blue, we would be left with six red balls and three blue balls. That means, for the second ball:

$$P(red)\ = \frac{6}{9}, P(blue)\ = \frac{3}{9}$$

The probability for two dependent events to occur is given by:

$$P\ (A\ and\ B)\ =\ P(A)\ .P(B\mid A)$$

1.3.6. Bayes' Theorem

Bayes' theorem, or Bayes' rule, calculates the conditional probability of occurrence of an event given that some other event has already occurred. Bayes' theorem is used in the

Bayesian inference method, where it is used to update the probability of a hypothesis as new evidence is provided. The probability of a hypothesis $\boldsymbol{H}$ given some evidence $\boldsymbol{E}$ is given by:

$$P\,(H|E)\;=\;\frac{P\,(E\mid H)\;\;P(H)}{P(E)}$$

Bayes' theorem has several applications in machine learning. It is used in spam email filtering, determining the accuracy of medical test results, or even determining the financial risk of investing in a company based on various factors.

Bayes' Theorem Example:

Consider the example of a spam email filter. Now, we want to consider emails as spams based on the words or sentences included in the email content (some examples are "lottery," "double your earnings per week," and "monthly trial"). So, to find the probability for an email to be spam given the content of the email is:

$$P\,(spam\mid content)\;=\;\frac{P\,(content\mid spam)\;\;P(spam)}{P(content)}$$

As this model gets more emails, it learns and gets more accurate in classifying spam emails.

Consider another example of two children. Given that one of the children is a girl, what is the probability that both children are girls?

Let us suppose O to be the event that one child is girl and T to be the event that both children are girls. So, we need to find:

$$P\,(T\mid O)\;=\;\frac{P\,(O\mid T)\;\;P(T)}{P(O)}$$

We know that:

$$P\,(O \mid T) \;=\; 1$$

Sample Space:

$$S \;=\; \{BB, BG, GB, GG\}$$

So,

$$P\,(T) \;= \frac{1}{4}$$

$$P\,(O) \;= \frac{3}{4}$$

$$\Rightarrow P\,(T \mid O) \;=\; \frac{1\,\frac{1}{4}}{\frac{3}{4}} = \frac{1}{3}$$

We won't go any further with this, as it would be out of the scope of this book.

1.4. Probability Distributions

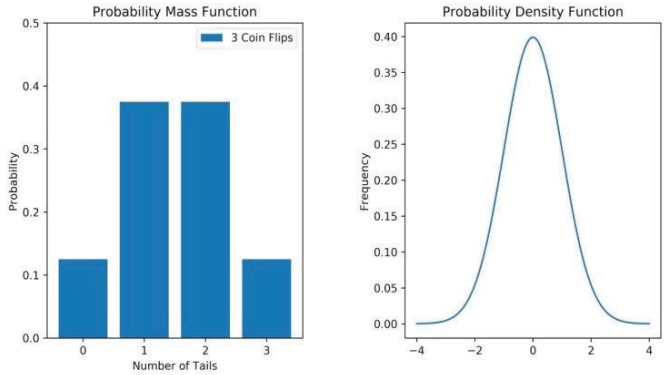
The probability distribution describes how probabilities are distributed to all possible outcomes of an experiment. For example, consider an experiment of a coin tossed three times. Our sample space would be:

$$S \;=\; \{HHH, HHT, HTH, HTT, THH, THT, TTH, TTT\}$$

Here, $\boldsymbol{H}$ represents heads, and $\boldsymbol{T}$ represents tails. So, the number of tails obtained can be zero (tail never comes up in three tosses), one (tail comes up only once), two (we get tail in any two tosses), or three (tail in all cases). So the probability of each case, given the sample space would be:

Number of Tails (X)	0	1	2	3
Probability	1/8	3/8	3/8	1/8

Since this random variable is discrete, this probability distribution is a **probability mass function**.



We can find cumulative probabilities from a probability distribution. For example, the probability of getting two or fewer heads in the experiment of three coin flips above is:

$$P(X \leq 2) = P(X = 0) + P(X = 1) + P(X = 2)$$

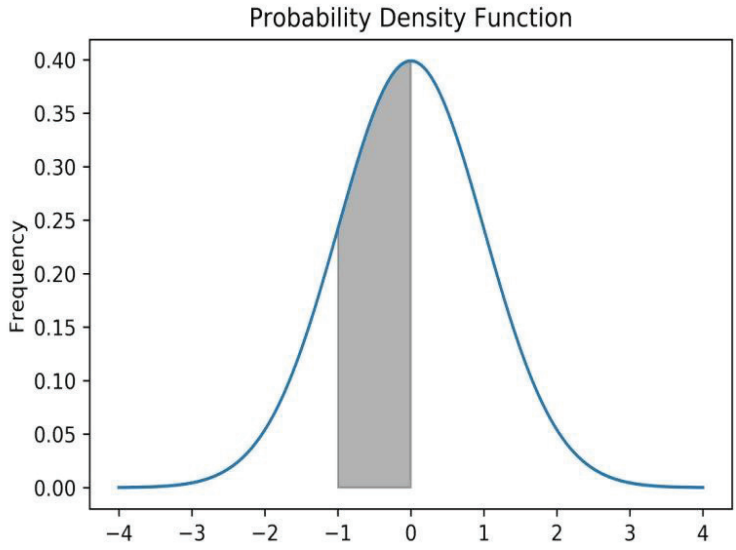
$$P(X \leq 2) = \frac{1}{8} + \frac{3}{8} + \frac{3}{8} = \frac{7}{8}$$

For a continuous random variable, the probability distribution is defined by a **probability density function**. For probability density function, there is an infinite number of values between any two data points, and the total area under the curve is equal to 1. One important thing to notice is that the output of PDF (probability density function) is not the probability. To find probability, we need to find the area under the curve

using integration from calculus, which is essentially cumulative probability. As a result, the probability of a continuous random variable at a particular value will be zero. For example, to find the probability that the outcome is between -1 and 0:

$$\int_{-1}^{0} f(x)\,dx = P(-1 < x < 0)$$

The area is shown in the figure below:



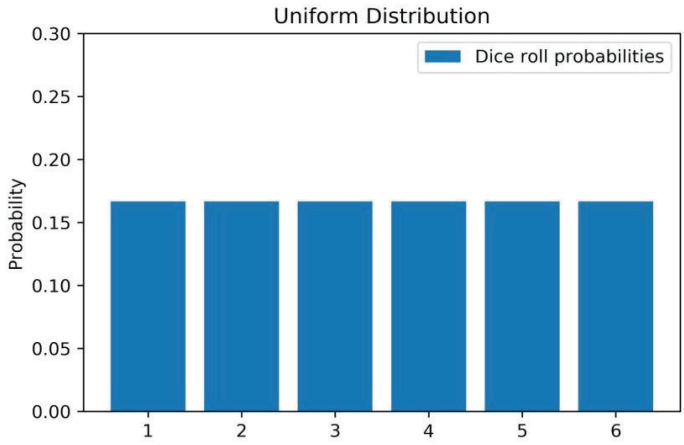
There are many distributions that show the patterns in data. Only a few are explained here.

1.4.1. Uniform Distribution

It is the simplest distribution when all the outcomes of an experiment have an equal probability of occurrence. In other words, when all the outcomes of an experiment are equally likely to occur.

For example, the outcomes of a dice roll:

Outcome	1	2	3	4	5	6
Probability	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$



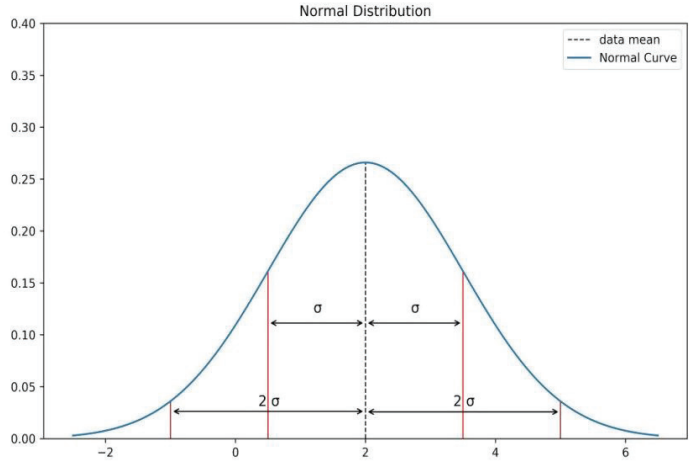
1.4.2. Gaussian Distribution

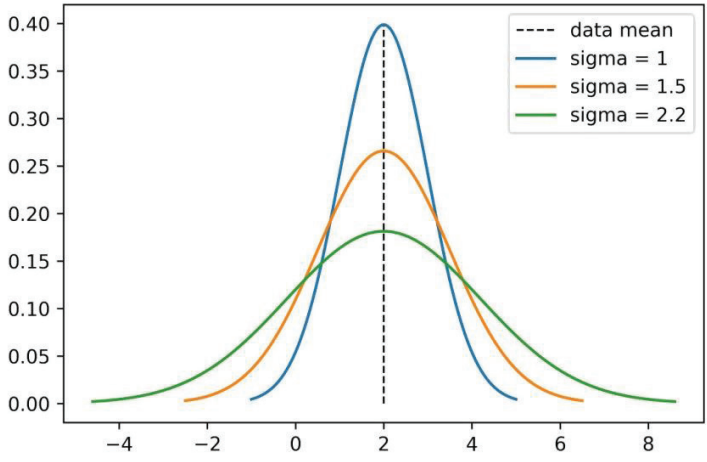
Gaussian distribution is also known as a normal distribution that forms a bell curve graph and is symmetric about the mean. This means that the data near the mean value is more frequent than the data distant from the mean. Another property is that the mean, median, and mode are equal for this distribution. The distribution function of a normal random variable is given by:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2}$$

Here, σ is a standard deviation, and μ is the mean of data.

The distribution of a normal random variable has the shape of a normal curve. A normal distribution is shown in the figure below. A normal curve of continuous data is also shown in the figure. Three sigma rules apply on a normal distribution, which means that 68 percent of data lies within one sigma or specifically, in the range $(\mu - \delta, \mu + \delta)$, 95 percent of data falls within two standard deviations $(\mu - 2\delta, \mu + 2\delta)$, and 99.7 percent falls within three standard deviations $(\mu - 3\delta, \mu + 3\delta)$.





σ (standard deviation) tells how dispersed the data is. In the figure above, with sigma = 1, we have a tightened curve along the y-axis as data that is much closer to the mean. As we increase the value of sigma, the curve tends to flatten along the x-axis.

Z Scores

Z score is the distance of an observed value from the mean of data in terms of standard deviation. A positive z score means that the observed value is 'z' above the mean, and a negative z score means that the observed value is 'z' below the mean.

$$z = \frac{x - \mu}{\sigma}$$

Here, $\boldsymbol{x}$ is observed value, μ is the mean of data, $\boldsymbol{z}$ is z score value, and σ is the standard deviation of data.

1.4.3. Binomial Distribution

In binomial distribution, we find the probability that the experiment will result in some fixed number of successes. The experiment consists of n repeated trials, and each trial has two possible outcomes: success and failure. The probability of success and failure remains the same for all trials of the experiment. Consider an experiment of flipping a coin three times. A single coin flip is a trial, and 'heads' is our success while 'tails' is a failure. For each trial, the probability of success is the same (i.e., 50 percent or $\frac{1}{2}$)

So, the probability that we will get $\boldsymbol{x}$ successes (each success has a fixed probability $\boldsymbol{p}$) in $\boldsymbol{n}$ trials is given by:

$$b(x;\, n, p) = {C_x}^n\, (p)^x (1 - p)^{n-x}$$

Binomial Distribution Example:

Suppose a die is tossed five times, and we want to find the probability of getting four exactly two times. We know:

Our success in getting a four: $p = \frac{1}{6}$

Total Trials: $n = 5$

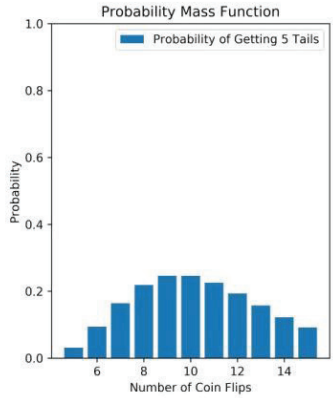
Successes: $x = 2$

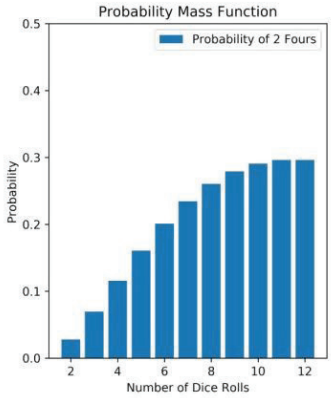
So, using the binomial formula, we get the probability of getting four two times in five trials:

$$b(2;\ 5, \frac{1}{6}) = {C_2}^5 (\frac{1}{6})^2 (\frac{5}{6})^3 = \quad 0.161$$

The following figure shows the results from two binomial experiments. In a coin flip experiment, the probability of success and failure is the same. So, the probability of getting five tails increases with the number of experiments, reaches max, and then goes down because the probability of failure also increases. In a dice roll experiment, the graph is skewed to the right direction because the probability of success is quite low compared to failure probability.

This means that binomial distribution approaches normal (Gaussian distribution) when the probability is equal to 0.5. If the probability is less than 0.5, the distribution graph will be skewed to the right, and if the probability is greater than 0.5, the graph will be skewed to the left.





1.4.4. Poisson Distribution

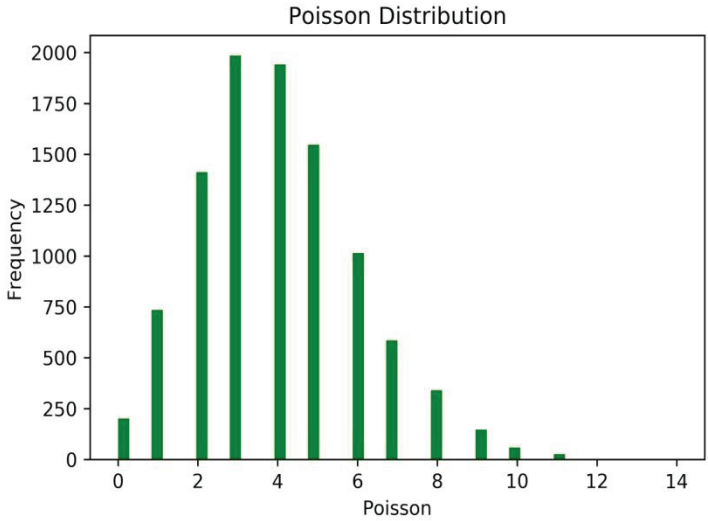
Poisson distribution is somewhat similar to the binomial distribution. Poisson distribution counts discrete occurrences in the continuous domain, while binomial distribution counts discrete occurrences among discrete trials. Poisson distribution is an approximation of binomial distribution for very large n (number of trials). Binomial distribution approximate to Poisson distribution is shown in the figure below.

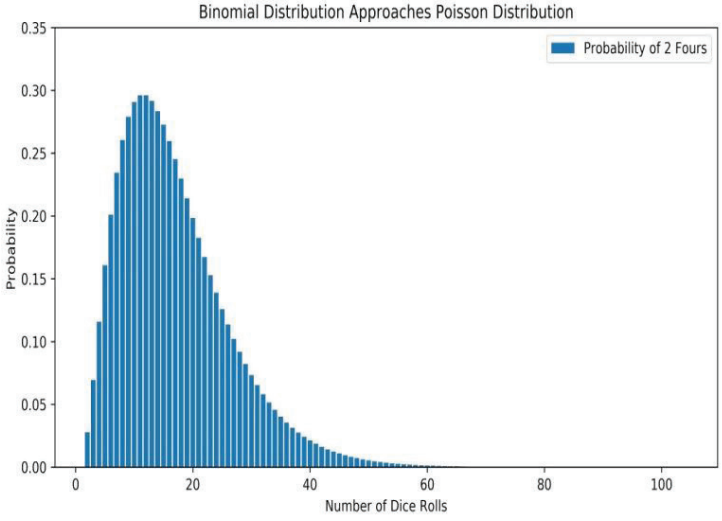
The Poisson distribution is the outcome of a Poisson experiment. In the Poisson experiment, there are infinite infinitesimally small trials, and each trial has one success. The probability of success is proportional to the size of the region under the curve. This region represents continuous values like length, area, height, temperature, etc.

The probability that $\boldsymbol{x}$ successes will occur with the average number of successes μ in a given region is:

$$P(x;\ \mu) = \frac{1}{x!}e^{-\mu^x}$$

An example Poisson distribution plot is shown in the figure below.





Poisson Distribution Example:

The average number of cars sold by a company is five cars a day. What is the probability that exactly seven cars will be sold the next day?

We know the average success (number of cars sold): $\mu = 5$

Our required success (number of cars sold in the next day): $x = 7$

So, our probability that seven cars will be sold is given by:

$$P(x;\ \mu) = \frac{1}{x!}e^{-\mu^x}$$

$$P(7;\ 5) = \frac{1}{7!}e^{-5^7}$$

$$= 0.104$$

2

Fundamentals of Statistics

2.1. Why Is Statistics Important for Data Science?

Some of the algorithms in machine learning have been borrowed from statistics. So, we need some basic knowledge of statistics to understand how to extract useful information from data, and how to build estimation models for data prediction based on hypothesis and assumptions. For example, linear regression is widely used in several machine learning problems. In statistics, it is used for fitting a line to data values while in machine learning, it is more like learning weights (constant values in line equation) through examples.

Some uses of statistics in machine learning are:

- **Data Understanding:** Understanding the distribution of data variables and their relationships, so we can design a model predictor best suited for data. For this, we need to understand what data distribution is, what relationships our variables can have, and how to understand those relationships.
- **Data Cleaning:** We cannot simply get raw data and feed it to machine learning models for our task because

there are certain complexities within. For example, the data might be corrupted, erroneous, or just missing. So, we need to fix those issues; e.g., we can fill the missing data following the same data distribution. We can also identify outliers and abnormal data distributions to eliminate corruption or errors.

- **Data Preparation**: Sometimes, our data features are not all on the same scale, which leads to some issues in model training. (It is further explained in training models part.) Sometimes, the data is textual, and we need to encode it in the numeric form to make it compatible with our model. Hence, we need data scaling, sampling, and encoding from statistics.
- **Model Configuration and Selection:** Hyperparameters of a machine learning model control the learning method, which can lead to different results from the model. Using statistical hypothesis testing techniques from statistics, we can compare the results of different hyperparameters. Similarly, we need such statistical techniques to select a model by comparing the results of models and their properties.
- **Model Evaluation:** To evaluate a model, we need statistical methods for data sampling and resampling. We also need metrics to properly evaluate the model and quantify the variability in predictions using estimation statistics.

2.2. Data Types:

Data in statistics are classified in the following types and subtypes:

1. Qualitative Data
    a. Nominal/Categorical
    b. Ordinal
2. Quantitative Data
    a. Discrete
    b. Continuous
        i. Interval
        ii. Ratio

2.2.1. Qualitative Data

Qualitative data is non-numerical and categorical data. Categorical data can be counted, grouped, and ranked in order of importance. Such data is grouped to bring order or make sense of the data.

Nominal:

Nominal scales represent data that does not have quantitative values or any numerical significance. This scale is used for classification or categorization of the variables. These variables are simply labels of data without any specific order. Such a scale is often used in surveys and questionnaires.

Of these three colors, which one do you like the most?

1. Red
2. Green
3. Blue

For example, in the above question, color is a categorical variable, but there is no specific way to order these colors from low to high or high to low.

So, for a question in a survey:

Which pizza crust do you like?

1. Thin crust
2. Stuffed crust
3. Cheese stuffed crust
4. Hand-tossed crust

Only the types of crust are significant for analysis, and their order does not matter. The results of such questions are analyzed to provide the most common answer, which describes the customers' preferences.

Ordinal:

An ordinal scale represents data that has some specific order. That order relates elements of data to each other in a ranked fashion. Numbers can be assigned as labels and are not mathematically measured as scales.

How satisfied are you with your test scores?

1. Extremely unsatisfied
2. Unsatisfied
3. Neutral
4. Satisfied
5. Extremely satisfied

Here, these options are assigned numbers. That is, a ranking of five is better than a ranking of three, but we cannot quantify the difference between these two rankings (i.e., how good ranking five is compared to ranking three).

2.2.2. Quantitative Data

Data that is numerical or can be measured is called quantitative data. For example, heights of 12-year-old boys, GPA scores of a class, distance of stars from the Earth, temperature of the day at different times, and so on. Numerical data have two sub-data types:

1. Discrete
2. Continuous

Discrete:

Discrete data is numerical data that cannot be subdivided into smaller parts. For example, the number of people in a room, the number of apples in a basket, the number of planets in the solar system, etc.

Continuous:

Continuous data can be broken down into smaller parts. For example, temperature, distance, the weight of a person, etc. Continuous data can be further categorized into two types:

1. Interval
2. Ratio

Interval:

The interval scale represents data that has a specific order, and that order has mathematical significance, which means that unlike nominal and ordinal scales, the interval scale quantifies the difference between the order of variables. This scale is quite effective because we can apply statistical analysis on such data. The only drawback of this scale is that we cannot compute ratios because it does not have a true zero value (a

starting point for values). Hence, a zero value does not mean the *absence of value* and is, therefore, not meaningful.

For example, 70 degrees Celcius is less than 90 degrees Celcius, and their difference is a measurable 20 degrees Celcius as is the difference between 90 degrees Celcius and 110 degrees Celcius. Also, 40 degrees Celcius is not twice as hot as 20 degrees Celcius. A value of 0 degrees Celcius is arbitrary as it does not mean *no temperature*; we know that temperature can be expressed by a negative value.

Ratio:

The ratio scale has all the properties from the interval scale and also defines a true zero value. A meaningful zero in ratio scale means an *absence of value*. Because of the existence of a true zero value, variables of ratio scale do not have negative values, and it allows us to measure the ratio between two variables. This scale allows applying techniques of inferential and descriptive statistics to variables. Some examples of ratio variables are height, weight, money, age, and time.

For example, what is your weight in pounds?

- < 100 lbs
- 100 lbs – 120 lbs
- 121 lbs – 140 lbs
- 141 lbs – 160 lbs
- > 160 lbs

Similarly, there is no such thing as age 0, because that essentially means you don't exist. Because of this, we can state that the age of 24 is twice the age of 12 and that we cannot have a negative age value.

2.3. Statistics in Practice

Statistics is defined as the collection, analysis, and interpretation of data. It transforms raw data into useful information and helps us understand our data required to train our machine learning models and interpret their results. The field of statistics has two major divisions: Descriptive statistics and Inferential statistics. Both of these divisions combined are powerful tools for data understanding, description, and data prediction. Descriptive statistics describes and interprets data and helps us understand how two variables or processes are related, while inferential statistics is used to reason from available data.

2.4. Descriptive Statistics

Descriptive statistics describes features of data by summarizing it. For example, we have test scores of 100 students from a class. Descriptive statistics gives detailed information about those scores; e.g., how spread the scores are if there are outliers (scores way above or lower than the average score), how scores are distributed (how many students with a low score, high score, or average score), and many other similar stats of those results.

2.4.1. Basic Definitions

Population

A population is something under observation for study. It is a set of observations. It also describes the subject of a particular study. For example, if we are studying the weights of men, the population would be the set of weights of all men in the world.

Population Parameters

Population parameters are characteristics and stats about population such as mean and standard deviation.

Sample

A sample is a randomly taken small part of the population or just a subset of the population. The observations and conclusions made on sample data represent the properties or attributes of the entire population. For example, consider a study where we want to know how many hours, on average, do teenagers spend doing physical exercise. Surveying all the teenagers of the world or even a country is impractical because of time and resource constraints. So, we take a sample from the population that represents the entire population.

Descriptive statistics is further divided into measures of center and measures of dispersion.

1. Measures of Center
    a. Mean
    b. Median
    c. Mode
2. Measures of Dispersion
    a. Range
    b. Percentiles/Quartiles
    c. Interquartile range
    d. Variance
    e. Standard deviation
    f. Kurtosis
    g. Skewness

The measure of Center/Central Tendency

Central tendency describes a dataset with a single value, which is the central position within that dataset. This central position provides a summary of the whole data.

Three measures of central tendency are:

- Mean
- Median
- Mode

Mean

It refers to the mean or average of data and is the sum of all values of the data divided by the number of values. It is represented by letter the μ or $\underline{\boldsymbol{x}}$ and is given by:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Some properties of mean make it very useful for measuring.

For example, consider the following array of numbers and their mean calculated:

$$Given\ Array\colon \{-5, 6, 3, 0, 6\}$$

$$\mu = \frac{(-5 + 6 + 3 + 0 + 6)}{5} = \frac{10}{5}$$

$$\mu = 2.0$$

A disadvantage of using statistical mean is that it can be biased because it is affected by outliers in the data. Consider the above example again and see how the existence of an outlier will affect the mean value:

$$Given\ Array: \{-5, 6, 3, 0, 6, 110\}$$

$$\mu = \frac{(-5 + 6 + 3 + 0 + 6 + 110)}{6} = \frac{120}{6}$$

$$\mu = 20.0$$

Median

Median is another measure of central tendency and is the middle value of ascendingly sorted data. If the number of values in a dataset is even, the median is the mean value of two middle values in a sorted array.

$$Given\ Array\ \{-5, 6, 3, 0, 6\}$$

$$Sorted\ in\ Ascending\ Order\ \{-5, 0, 3, 6, 6\}$$

$$Median\ is\ our\ middle\ value\ in\ sorted\ array = 3$$

One important property of the median is that it is not affected by outliers. Consider the example of statistical mean with outlier:

$$Given\ Array\ \{-5, 6, 3, 0, 6, 110\}$$

$$Sorted\ in\ Ascending\ Order\ \{-5, 0, 3, 6, 6, 110\}$$

$$Median = \frac{(3 + 6)}{2} = \frac{9}{2}$$

$$Median = 4.5$$

So, the median should be used when data is skewed (not symmetric).

Mode

A mode is a most frequently occurring value in the data. Some datasets can be multimodal (having more than one mode), and some may not have any mode at all. For example,

$$Given\ Array$$

$$\{1, 6, 0, 9, 3, -2, 4, -1, -1, 5, 1, 5, 3, 1, 7, -3, -4, 7, 8, -1\}$$

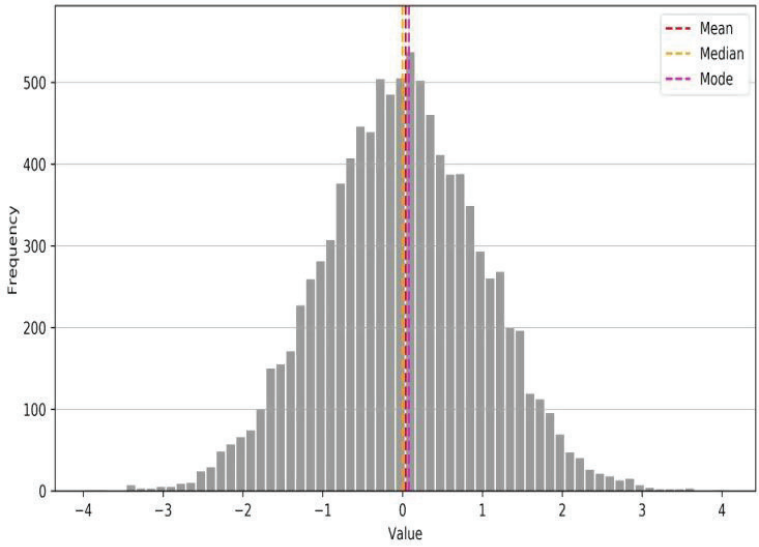
$$Modes:\ -1, 1, 3, 5, 7$$

$$Given\ Array: \{-1,7,3,15,2\}$$

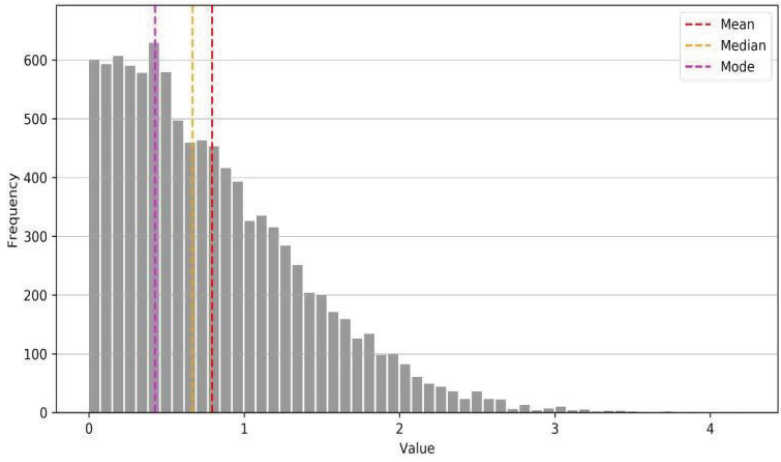
$$This\ data\ does\ not\ have\ any\ mode.$$

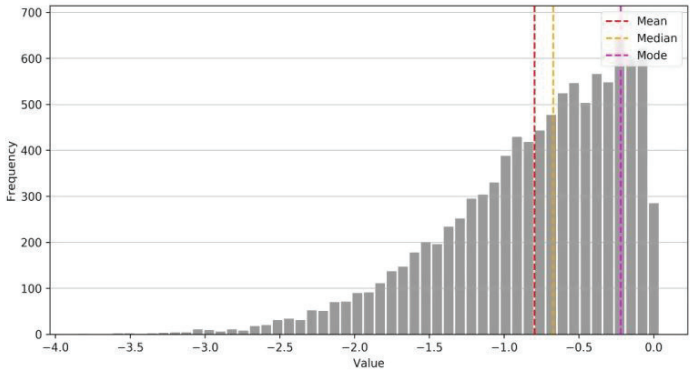
The mode is rarely used as a central measure of tendency. One problem with the mode is that there can be multiple modes in data, and they can be spread out.

The following three histograms with different symmetric and skewed distributions show three measures of center. In the case of symmetric (normal) distribution, mean, median, and mode are all the same. We can use any of these three measures, but mean is usually preferred because it considers all the values of data in the calculation, which is not the case for median and mode.



Now consider when the data is right skewed. Here, the mean is not a good representation of the center of the dataset, and so it is better to choose the median measure.





Measures of Dispersion

The measure of dispersion tells how much the data distribution is stretched out or how variable the data is. Sometimes, the measure of central tendency is not enough to grasp the distribution of data. For example, two data distributions can have the same mean, but one distribution can be more spread out than the other.

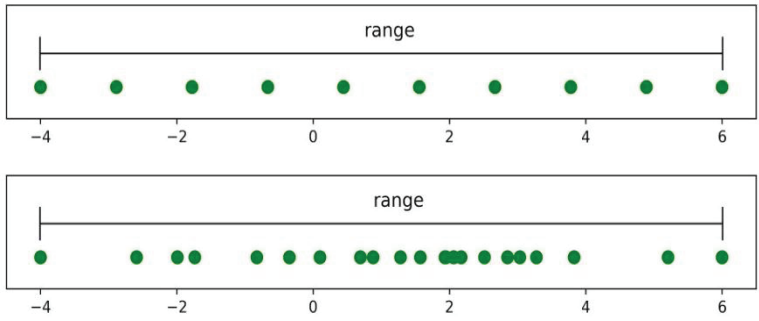
Range

The range is the simplest measure of dispersion. It is the difference between the maximum value and the minimum value of the data.

$$Given\ Array: \{-4, 7, 1, 7, -5, 6, 3, 0\}$$

$$Range: 7 - (-5) = 12$$

A range provides us with a quick way to get a rough idea of the spread of the distribution, but it does not give much detail about the data. Two datasets can have the same range, but their values can vary significantly. Also, a range is sensitive to outliers or extreme values.



Percentile

A percentile represents a value below which a given percentage of data falls. In other words, a percentile is the relative position of a value in a sorted dataset. A student scored 89 out of 100 in a test. This figure alone does not have significant meaning unless we know what his/her position in the class was. It is possible that a score of 89 falls in the 30th percentile, which means that it is better than 30 percent of the class, but it can also be in the 70th percentile (70 percent of students have scored less than 89 on the test).

Index of a number at $\boldsymbol{Pth}$ percentile in an ordered list of values is given by:

$$Index = \frac{P}{100} * N$$

For example,

$$Given\ Sorted\ Values: \{-1,5,12,13,30,50\}$$

For example, for a list of numbers given above,

$$15th\ percentile: P_{15} = \frac{15}{100} * 6 = 0.9$$
$$\approx 1\ (1st\ number\ in\ the\ list)$$

$$30th\ percentile:\ P_{30} = \frac{30}{100} * 6 = 1.8$$
$$\approx 2\ (2nd\ number\ in\ the\ list)$$

$$60th\ percentile:\ P_{60} = \frac{60}{100} * 6 = 3.6$$
$$\approx 4\ (4th\ number\ in\ the\ list)$$

$$99th\ percentile:\ P_{99} = \frac{99}{100} * 6 = 5.9$$
$$\approx 6\ (6th\ number\ in\ the\ list)$$

Quartiles

Quartiles are three values that divide a dataset into four parts, where each part contains 25 percent of the total data. First, second, and third quartiles are actually 25th, 50th, and 75th percentiles, respectively. The first quartile divides the data into 1:3 parts and has 25 percent of the data on the left side (or below it). The second quartile divides the data in 1:1 part while the third quartile divides the data into 3:1 parts (75 percent of the data on the left side). So, the second quartile is the middle value (or median) of the dataset, and it divides the data into two equal parts. Then, the first quartile is the middle value (or median) of the first part, and the third quartile is the middle value (or median) of the second part.

For example,

Given array:

$$[\ 10, -9, -7, 10, 3, -10, -2, -3, 14, -2, -6, -9, 14, 16, 0]$$

Sorted array:

$$[-10, -9, -9, -7, -6, -3, -2, -2,\ \ 0,\ \ 3, 10, 10, 14, 14, 16]$$

$$Q_1 = -2$$

Interquartile Range

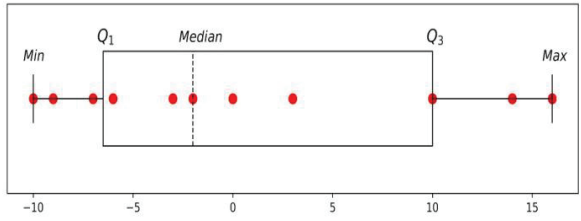
Interquartile range (IQR) is the difference between the third quartile and the first quartile. An important property of IQR is that it is not affected by outlier values, which makes it preferable over the range. It focuses on the middle 50 percent values of data.

Sorted array:

$$[-10, -9, -9, -7, -6, -3, -2, -2, \ 0, \ 3, 10, 10, 14, 14, 16]$$

$$IQR: 16.5$$

Visual representation of some of these measures is shown in the following boxplot.



Variance

Variance is a measure of the spread of data. It quantifies how far, on average, the data is from the mean value.

$$Var(X) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

Standard Deviation

Standard deviation is just the square root of variance. The difference between variance and the standard deviation is that the variance value is on a large scale, while the standard

deviation has the same scale as other dataset values. It is represented by the Greek letter σ and is given by:

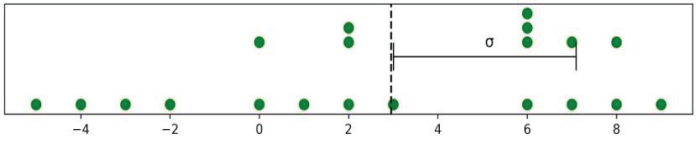
$$\sigma = \sqrt{Var(X)}$$

For example, consider the following values. Their mean and standard deviation has been calculated below:

$$\{2, 8, 0, 2, -2, -4, 7, 1, 7, -5, 6, 3, 0, 6, 8, 6, 2, 6, -3, 9\}$$

$$\mu = 2.95$$

$$\sigma = 4.2$$



Skewness

Skewness is a measure of data asymmetry. A symmetric distribution means that the data values are equally distributed around the mean value, as in the case of a normal distribution. In the case of asymmetry, data is distributed unevenly, or the distribution is not symmetrical about the mean. In positively skewed data distribution, values are concentrated to the left, and in negatively skewed data distribution, values are concentrated to the right, as shown in the figure below. If data skewness is not close to zero, the data is not normally distributed.

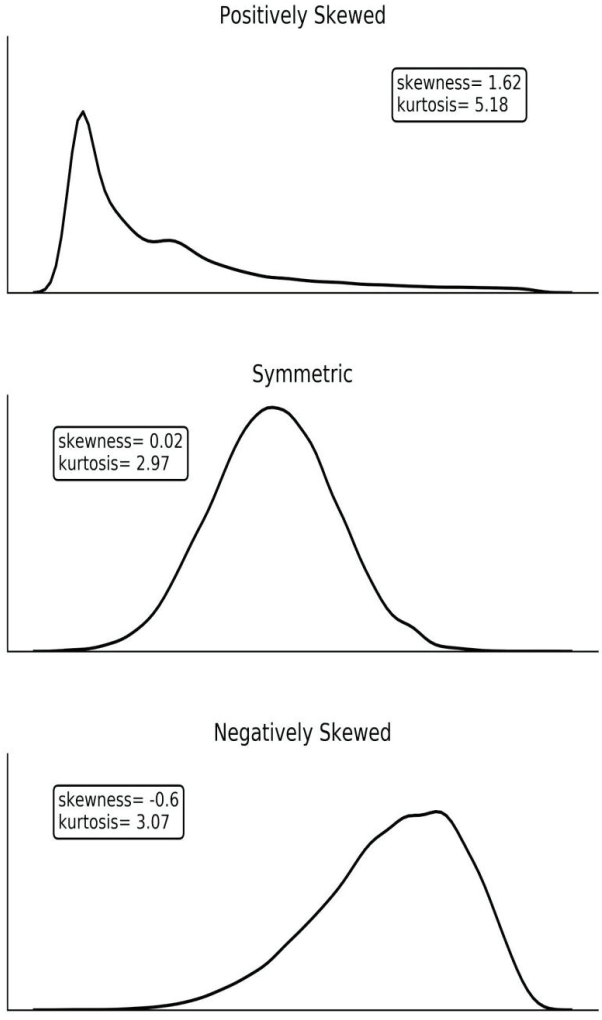
$$Skewness = \frac{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^3}{\sigma^3}$$

Kurtosis

Kurtosis describes the tail of distribution with reference to a normal distribution. A normal distribution has a kurtosis equal to 3 and is called mesokurtic. A distribution with shorter and thinner tails and broader and lower peaks than a normal distribution is called platykurtic and has a kurtosis of less than 3. A distribution with longer tails and higher and sharper peaks than a normal distribution is called leptokurtic and has a kurtosis greater than 3.

$$Kurtosis = \frac{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^4}{\sigma^4}$$

Where N is the number of values, μ is the mean of values, and σ is a standard deviation.



2.5. Inferential Statistics

Descriptive statistics measures are limited because we can only summarize the observed data and cannot generalize those statistical measures to unseen data. For example, given the test scores of a person for the last two years, descriptive statistics can only provide a summary of the tests in those two years and cannot predict the score of the next test. But

we can achieve this using inferential statistics. Inferential statistics learns the pattern of seen data and predicts unseen data based on it. More precisely, we draw inferences about a population based on a sample from the same population.

2.5.1. Basic Definitions

Hypothesis

A hypothesis is an explanation of an observation or a phenomenon. Some describe hypothesis as an educated guess, but actually, it is more informed than a guess because it is based on existing knowledge and experience. A hypothesis is falsifiable and testable. In statistics, a hypothesis is an assumption or educated guess about a population parameter.

Sample Statistic

As the population parameter is a characteristic of a population, sample statistics is a characteristic of a sample. In inferential statistics, we use sample statistics of a randomly drawn sample from a population to infer or make an educated guess about that same population parameter.

Sampling Distribution

Rather than using the whole population to learn its characteristics, researchers use small samples of the population to draw inferences about the population itself. For that, they use sampling distribution, which is a probability distribution. Consider a population of N values. The histogram of this population will show the distribution of those N values. We select a sample of size n from this population and calculate a sample statistic (for example, its mean value). If we repeat

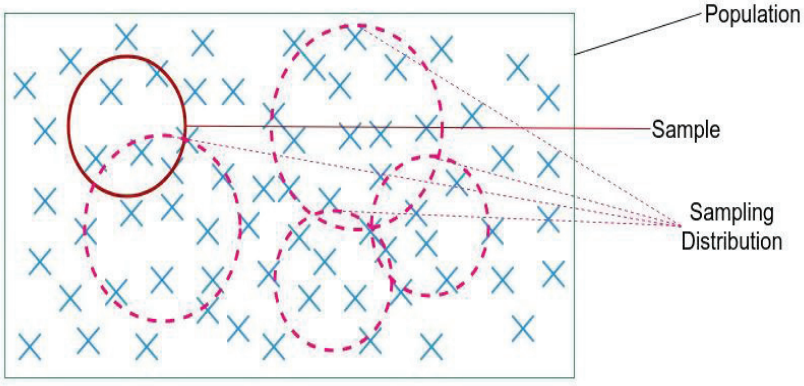
this drawing procedure indefinitely, it will generate a new population of those sample statistic values. The histogram of this new population will be the sampling distribution of the sample statistics. For example, we have a population of 1000 people. We randomly select a sample of 10 people and compute the mean value of their heights. Next, we again select a sample of 10 people and compute the mean value. If this process is repeated indefinitely, the distribution of those infinite mean values is called sampling distribution.

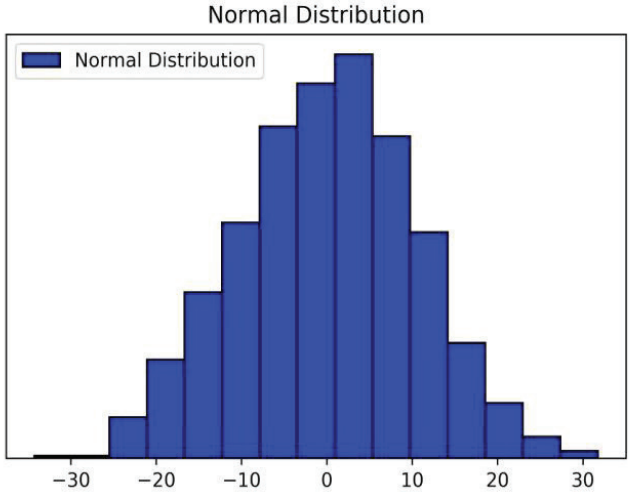
Central Limit Theorem

This theorem says that if we infinitely draw random independent samples of sufficiently large size from a population with mean μ and standard deviation σ and calculate the mean of each sample, then all these mean values will form a normal distribution. Irrespective of the fact that the population distribution is normal or not, the sampling distribution will approach normal distribution as the sample size increases.

Let's consider this in more detail. For example, we can draw a random sample of size k $(x_0, x_1, x_2, \ldots, x_k)$ from a population and compute its mean $\underline{\boldsymbol{x}}$. If we infinitely repeat this process of calculating means of randomly drawn samples, we will get a Gaussian distribution of those mean values.

The following figure shows randomly drawn samples in dotted circles, and the histogram shows sampling distribution.





Statistical inference methods can be of two types.

1. Estimating population parameters
2. Testing the hypothesis of population parameters

Estimation

There are two ways to estimate the population parameter using sample statistics.

- Point estimate
- Confidence interval

Point Estimate

This single value represents the drawn sample, e.g., mean, median, deviation, etc. A point estimate is not perfect and can vary from sample to sample. This means our sample statistic is close to the population parameter, but it is not exactly equal. For example, in a population of 100,000 values, for every 100 randomly selected values, we may get a different mean (but close enough values) every time. This mean value is called a point estimate.

Confidence Interval

Estimating population parameters using sample statistics is very hard because we are uncertain of how well the sample statistics represent the population distribution. Confidence interval tackles this issue and provides a range of values that population parameters can take. This interval value is a measure of precision in our estimate. A large value indicates low precision in estimated value, and a small value indicates high precision.

Hypothesis Testing

Hypothesis testing is a systematic way to test the ideas or claims of a hypothesis. Rather than examining the entire population to accept or reject a hypothesis, hypothesis testing is concerned with using a random sample to make statistical decisions to accept or reject the hypothesis. Two types of hypothesis are used in testing:

- Null Hypothesis
- Alternative Hypothesis

Null Hypothesis:

A null hypothesis H_0 is the status quo or default belief about a population. It is believed to be true unless there is significant evidence to disprove it.

Alternative Hypothesis:

Contrary to the null hypothesis, there is the alternative hypothesis H_A, which is the inverse of the null hypothesis and is accepted only if the null hypothesis has been rejected.

For hypothesis testing, we begin by assuming the null hypothesis to be true and find conditional probability (p-value) of observed outcome based on the condition that H_0 is true. We set a certain probability threshold and accept the null hypothesis if the p-value is greater than the threshold. Otherwise, we reject the null hypothesis and accept the alternative hypothesis.

3

Fundamentals of Linear Algebra

3.1. Why Is Linear Algebra Important for Data Science?

In linear algebra, we represent real-world problems in linear equations and linear functions, and try to solve them. Linear Algebra is the most important skill in data science, as its concepts are widely applied in machine learning. Consider the following points to understand how widespread the use of linear algebra in data science is:

- Data is represented as a matrix or vector
- Solving a machine learning problem is sometimes just solving a system of linear equations
- Convolution is nothing more but a small matrix being multiplied to a big matrix
- Finding optimal model parameters is a matter of finding function gradients
- Many algorithms use eigenvectors and eigenvalues in deep learning; e.g., Principal Component Analysis and data compression

By using linear algebra, we can better understand how algorithms work.

3.2. Notations and Definitions

Scalar: A single value. It can be an integer or a float.

Vector: A 1-D array of scalars

Matrix: A 2-D array of scalars

Tensor: An n-d array of scalars

3.3. Vector and Matrix

Row Vector: A vector of shape (*1* x *n*), i.e., containing only one row

$$\begin{bmatrix} -4 & 0 & 0 & 11 & -14 & 0 & 15 \end{bmatrix}$$

Column Vector: A vector of shape (*n* x *1*), i.e., containing only one column

$$\begin{bmatrix} -4 \\ 0 \\ 0 \\ 11 \\ -14 \\ 0 \\ 15 \end{bmatrix}$$

Square matrix: A matrix whose rows and columns are equal.

$$A = \begin{bmatrix} -3 & 0 & 1 \\ 0 & 0 & 7 \\ -9 & 0 & 6 \end{bmatrix} \quad B = \begin{bmatrix} -3 & 0 & 1 & 9 \\ 0 & 0 & 7 & -11 \\ -9 & 0 & 6 & 4 \\ 1 & 0 & 0 & 13 \end{bmatrix}$$

Rectangular Matrix: A matrix whose rows and columns are not equal.

$$A = \begin{bmatrix} 5 & -1 & 0 & 5 \\ -16 & 8 & -1 & -1 \\ 0 & 13 & 14 & 1 \end{bmatrix} \quad B = \begin{bmatrix} -3 & 15 \\ 0 & 8 \\ -9 & -7 \\ 1 & 5 \end{bmatrix}$$

3.3.1. Matrix Addition

Matrix addition involves the element-wise addition of two matrices. Two matrices can only be added if both the matrices have the same dimensions. Matrix addition is commutative, distributive, and associative.

3.3.2. Matrix Subtraction

It involves the element-wise subtraction of two matrices. Two matrices can be subtracted only if both have the same dimensions. Matrix subtraction is commutative, distributive, and associative.

$$A = \begin{bmatrix} 5 & -1 & 0 & 5 \\ -16 & 8 & -1 & -1 \\ 0 & 13 & 14 & 1 \end{bmatrix} \quad B = \begin{bmatrix} -9 & -1 & 0 & 5 \\ 6 & 11 & 0 & 0 \\ 43 & 13 & 0 & 1 \end{bmatrix}$$

$$A + B = \begin{bmatrix} -4 & -2 & 0 & 10 \\ -10 & 19 & -1 & -1 \\ 43 & 26 & 14 & 2 \end{bmatrix}$$

$$A - B = \begin{bmatrix} 14 & 0 & 0 & 0 \\ -22 & -3 & -1 & -1 \\ -43 & 0 & 14 & 0 \end{bmatrix}$$

3.3.3. Matrix Multiplication

Each row in the first matrix is multiplied element-wise to all columns in the second matrix. Therefore, the number of columns in the first matrix should match the number of rows in the second matrix. For the first matrix of shape *n x m*, and the second matrix of shape *m x k*, the resultant multiplication matrix will have a *n x k* shape. Matrix multiplication is not commutative.

For example, consider matrix A of shape *3 x 2* (three rows and two columns), and B of shape *2 x 3* (two rows and three columns), the resultant matrix is of shape *3 x 3.*

$$A = \begin{bmatrix} 5 & -1 \\ -1 & 8 \\ 11 & 0 \end{bmatrix} \quad B = \begin{bmatrix} -9 & 2 & -3 \\ 0 & 0 & 6 \end{bmatrix}$$

$$A \times B = \begin{bmatrix} -45 & 10 & -21 \\ 9 & -2 & 51 \\ -99 & 22 & -33 \end{bmatrix}$$

3.3.4. Scalar Value and Matrix Multiplication

A scalar multiplied to a matrix will be multiplied to all elements of the matrix.

$$A = \begin{bmatrix} -3 & 0 & 1 \\ 0 & 0 & 7 \\ -9 & 0 & 6 \end{bmatrix}$$

$$6A = 6 * \begin{bmatrix} -3 & 0 & 1 \\ 0 & 0 & 7 \\ -9 & 0 & 6 \end{bmatrix} = \begin{bmatrix} -18 & 0 & 6 \\ 0 & 0 & 42 \\ -54 & 0 & 36 \end{bmatrix}$$

$$\frac{1}{2}A = \frac{1}{2} * \begin{bmatrix} -3 & 0 & 1 \\ 0 & 0 & 7 \\ -9 & 0 & 6 \end{bmatrix} = \begin{bmatrix} -1.5 & 0 & 0.5 \\ 0 & 0 & 3.5 \\ -4.5 & 0 & 3 \end{bmatrix}$$

3.3.5. Matrix Transposition

The transposition of a matrix is the creation of a new matrix whose rows and columns have been switched. In other words, its columns are the rows of the original matrix or the rows are the columns of the original matrix. Transposition of a matrix is represented as A^T or A'.

$$A = \begin{bmatrix} -3 & 0 & 1 \\ 0 & 0 & 7 \\ -9 & 0 & 6 \end{bmatrix}$$

$$A^T = \begin{bmatrix} -3 & 0 & -9 \\ 0 & 0 & 0 \\ 1 & 7 & 6 \end{bmatrix}$$

Identity Matrix

An n x n matrix (square matrix) with ones on its diagonal and zeros elsewhere. A *3 x 3* and *4 x 4* identity matrices are given below:

$$I_{3\times3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad I_{4\times4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Diagonal Matrix

A matrix with non-zero elements on the main diagonal and zeros elsewhere.

$$D_1 = \begin{bmatrix} -11 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad D_2 = \begin{bmatrix} 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 13 & 0 \\ 0 & 0 & 0 & -5 \end{bmatrix}$$

Inverse Matrix

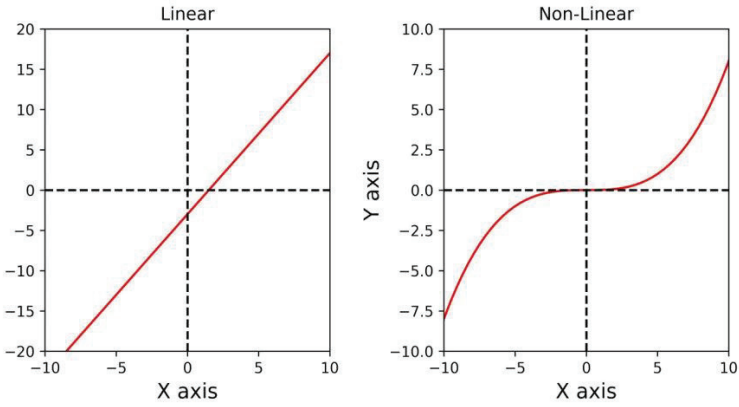
An inverse of a matrix is another matrix, which, when multiplied with the original matrix, results in an identity matrix. It is possible that the inverse of a matrix does not exist.

$$AA^{-1} = A^{-1}A = I_{n\times n}$$

$$A = \begin{bmatrix} 3 & 0 & 2 \\ 2 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix} \quad A^{-1} = \begin{bmatrix} 0.2 & 0.2 & 0 \\ -0.2 & 0.3 & 1 \\ 0.2 & -0.3 & 0 \end{bmatrix}$$

3.4. Function

A function represents a relationship between two or more variables. It maps or associates input values to output values. Functions can be linear (for a linear relationship between variables) and nonlinear (mapping a non-linear relationship between variables), as shown in the figure below.



3.4.1. Linear Relationship

A linear relationship between two variables can be represented using the equation:

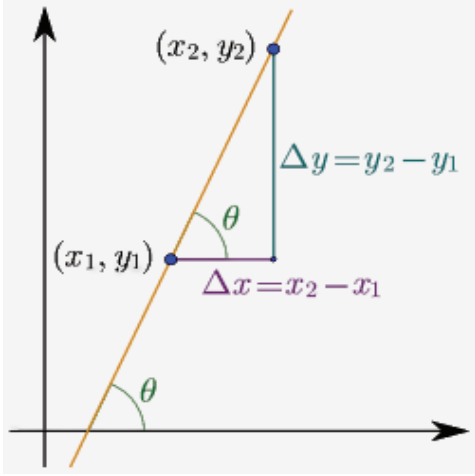
$$y = mx + c$$

Since two variables are involved in this equation with x being a dependent variable and y as an independent variable, we can visualize the line in the 2D cartesian coordinate system. M is the **slope** of the line, and c is the **y-intercept** (the point where the line intersects y coordinates). The figure below shows two linear functions with different *m* and *c* values.

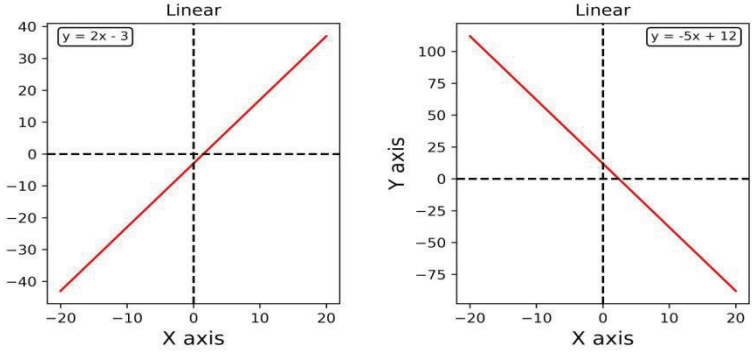
Slope

The slope of a line describes the steepness and direction of the line. For example, in the figure below, the line in the left figure has a positive slope, and the line in the right figure has a negative slope. The magnitude of the slope value tells how strongly the function is increasing or decreasing. In other words, it shows us how steep the function is. For any two points $(x_1, y_1), (x_2, y_2)$ on the line, the slope is given by the formula:

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$$



A horizontal line has a zero slope, and the slope of a vertical line does not exist.

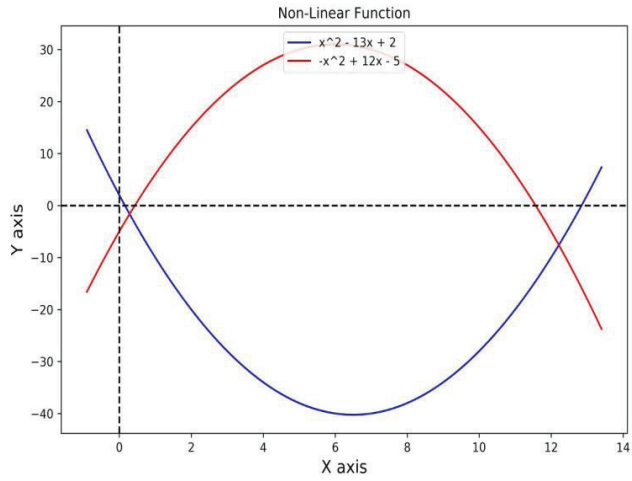


3.4.2. Nonlinear Relationship

Nonlinear relationship between two variables can be represented by the equation:

$$y = a\,x^2 + b\,x + c$$

Two variables are involved in this equation, with x being an independent variable and y as a dependent variable. The difference is that this is not a line equation, and hence, it does not correspond to a line in the 2D cartesian coordinate system. It is actually a curve (more precisely, a parabola). The figure below shows two curves with different values of constants (a, b, c).



3.5. Derivative

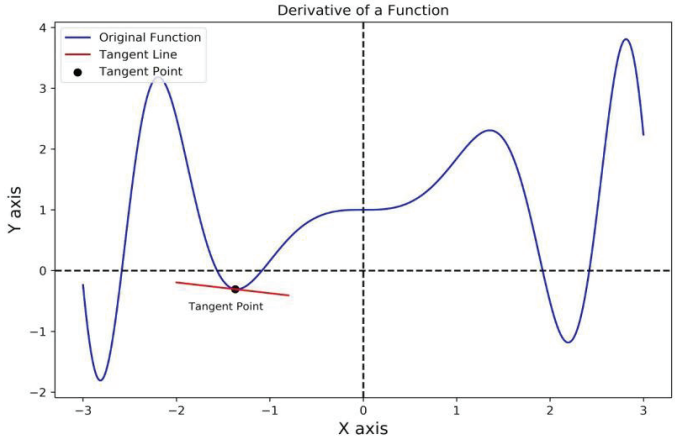
3.5.1. Derivative with One Variable

A derivative is the rate of change in one variable with respect to another variable. In the case of a 2D cartesian function, the derivative is the rate of change in y with respect to the rate of change in x and is denoted by $\frac{dy}{dx}$ or $f'(x)$ for a function of x.

The derivative of a function at a point (x_0,y_0) is the slope of the tangent line at that point.

Tangent Line

A tangent line to a function at a point is a straight line that touches the function or graph at that point. A function (blue) and a tangent line (red) are shown in the figure below.



Given the function: $f(x) = x\,sin(x^2) + 1$

The derivative of a function is given by:

$$\frac{dy}{dx} f(x) = \frac{dy}{dx}(xsin(x^2) + 1)$$

$$f'(x) = sin(x^2) + 2x^2 cos(x^2)$$

For the slope of the tangent line at $x = -1.4$, we evaluate the function derivative at $x = -1.4$

$$m = |\, f\,'(x)\,|_{x=-1.4} = -0.5622$$

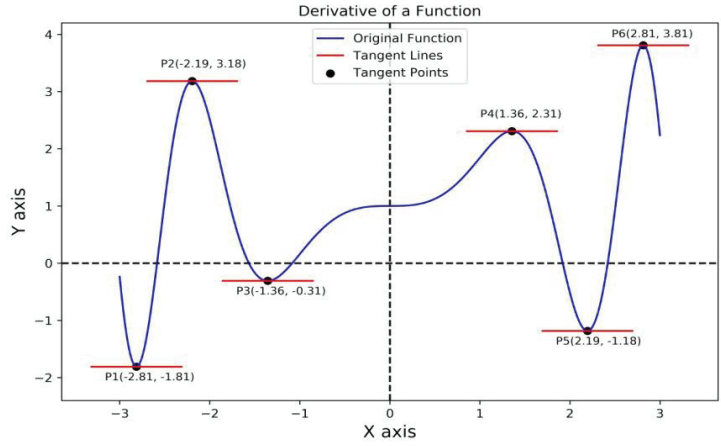
As we can see, the slope of the tangent line is negative, and it can also be visualized from the figure. Also, the slope is close to zero as the tangent line is almost horizontal (slope will be zero for horizontal lines).

Function Minimas and Maximas

One important thing to notice here is the function of extreme points (maximas and minimas).

Maximas are the points for which the function has maximum value in a local neighborhood. Consider point P2 in the figure below. It has maximum value in its close neighborhood. Similarly, P4 and P6 are maximas of the function. So, in the range shown in the figure, there are three maximas, but an important thing to notice is that only P6 has the highest value of all. Therefore, P6 is a global maxima, and P2 and P4 are local maximas in this range. Maximas have the property that slope is positive on the left side, and it is negative on the right side.

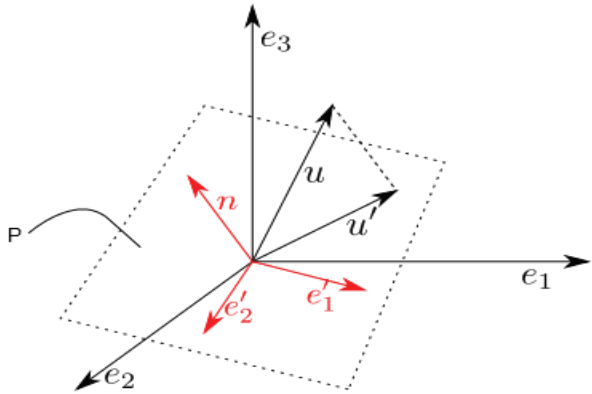
Minimas are the points for which the function has minimum value in the local neighborhood. Consider point P1 in the figure below. It has the minimum value in its neighborhood. Similarly, P3 and P5 are minimas of the function. In this case, P1 is global minima, and P3 and P5 are local minimas. Minimas have the property that slope is negative on the left side of the point, and it is positive on the right side of the point.



Another important thing to notice is that the derivative function (slope of tangent line) at those extreme points is zero. This means that a point is minima or maxima if the first derivative of the function is zero at that point. Those extremes can directly be extracted by solving the derivative function for zero.

3.5.2. Derivatives with Multiple Variables

As a linear function with two variables corresponds to a line in 2D space, a linear function with three variables will correspond to planes in 3D space.



For equations with more than three variables, we cannot visualize the resulting high dimensional space. The good news is that we can still use the same derivative concepts in higher dimensions with a little modification. As we know, the equation of a line in two dimensions is:

$$y = mx + c$$

and its derivative with respect to x:

$$\frac{dy}{dx} = m$$

Now, consider the equation of a 3D plane with three variables (x, y, z):

$$z = ax + by + c$$

This equation has two independent variables: (x, y). So, to find the derivative of this function, we need to consider the change in each independent variable separately. This means that first, we will calculate the derivative of z with respect to x and then, its derivative with respect to y. While calculating the derivative for one variable, all other variables are considered as constants.

Following are the derivatives of $\boldsymbol{z}$ with respect to $\boldsymbol{x}$ and $\boldsymbol{y}$ (from the equation above):

$$\frac{\partial z}{\partial x} = a$$

$$\frac{\partial z}{\partial y} = b$$

Here ∂ is the notation for partial derivative of a function with multiple variables.

Even though we cannot visualize the functions and their extremas for dimensions > three, we can take derivative concepts from two- and three-dimensional computations of equations and use them to find extremas. We will do that in the machine learning and deep learning part where the function contains a number of variables, and we have to find minimas of the function.

3.6. Chain Rule

The chain rule is a way to calculate the derivative of a function that consists of a composite.

Some complicated expressions are composed of multiple functions. For example,

$$f = (x + y)\, z$$

This function can be broken down as

$$g = x + y$$

$$\Rightarrow f = g * z$$

Let's suppose that we are given this last equation, and we know that $\boldsymbol{g}$ is a function of x and y, and we need to calculate the derivative of the function with respect to x. We cannot directly compute the derivative unless we insert the function equation of $\boldsymbol{g}$. It is simple in this case but can be quite complex if $\boldsymbol{g}$ consists of more than one complex function. In such cases, we need some other way to calculate derivatives.

When we know the equations separately, we can find the derivatives of each equation with respect to their variables.

$$f = g * z$$

$$\Rightarrow \frac{\partial f}{\partial g} = z, \frac{\partial f}{\partial z} = g$$

$$g = x + y$$

$$\Rightarrow \frac{\partial g}{\partial x} = 1, \frac{\partial g}{\partial y} = 1$$

The chain rule is a way to combine those expressions to find the derivatives of variables of composed functions. It states:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}, \frac{\partial f}{\partial y} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial y}$$

3.6.1. Line Fitting

Consider two points in the 2D cartesian coordinate system, as shown in the figure below. We need to fit a line to these points, i.e., a line that passes through both points.

As we know that the line equation is:

$$y = mx + c$$

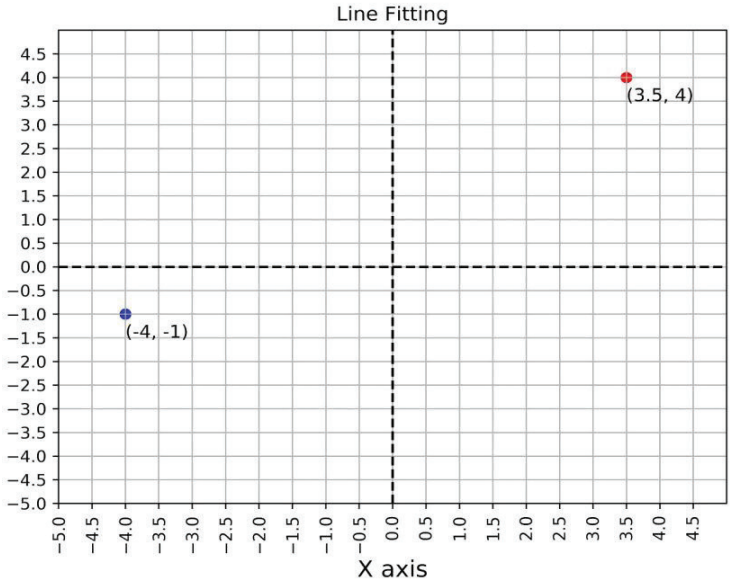
$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$$

Using the given two points, we find the slope of the line.

$$m = \frac{5}{7.5} = 0.66$$

So, our line equation becomes:

$$y = \frac{5}{7.5} x + c$$



If a point lies on the line, it must satisfy its equation. In other words, as the point (-4, -1) lies on the line, we must have:

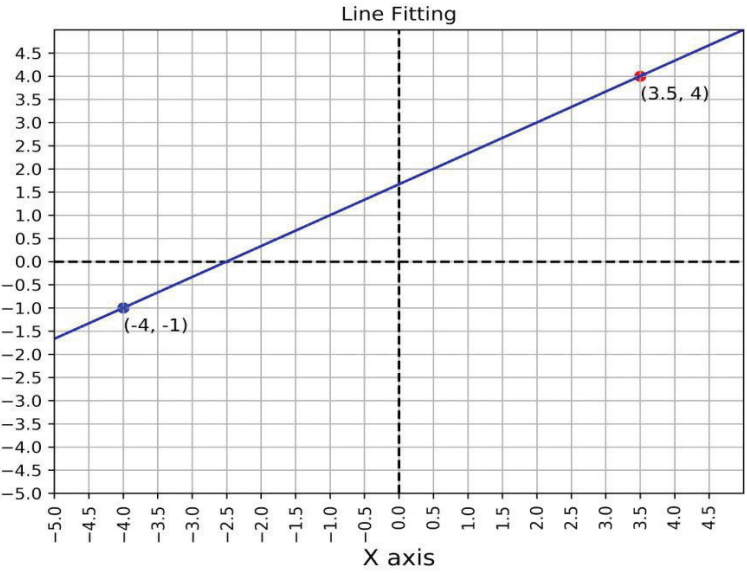
$$-1 = \frac{5}{7.5}(-4) + c$$

$$\Rightarrow c = \frac{5}{3}$$

This is our y-intercept of the line. So, our final equation of the line becomes:

$$y = \frac{5}{7.5}x + \frac{5}{3}$$

This line is shown in blue in the figure below.



Now consider another point (shown in black) in between those two points (blue and red) such that these three points are collinear (in the same line) so we can fit a single line.

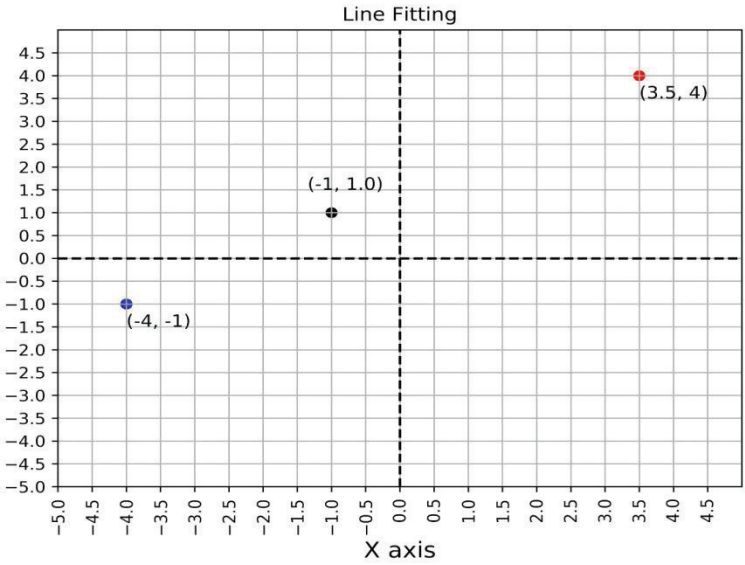
We can follow the same method to find our line of the equation. We can select any two points to find the slope and a third point to find the y-intercept in the equation.

Using the two points (-1, 1) and (-4, -1), we find the slope of the line.

$$m = \frac{-1 - 1}{-4 - (-1)} = \frac{2}{3}$$

So, our line equation becomes:

$$y = \frac{2}{3} x + c$$



Now, as the third point lies on the line, it must satisfy its equation:

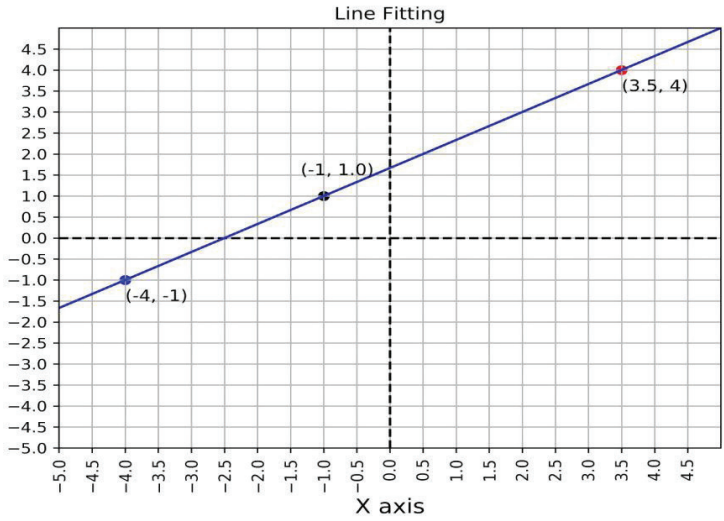
$$4 = \frac{2}{3} (3.5) + c$$

$$\Rightarrow c = \frac{5}{3}$$

This is our y-intercept of the line. So, our final equation of the line becomes:

$$y = \frac{2}{3} x + \frac{5}{3}$$

This line is shown in blue in the figure below.



Method 1: Ordinary Least Squares

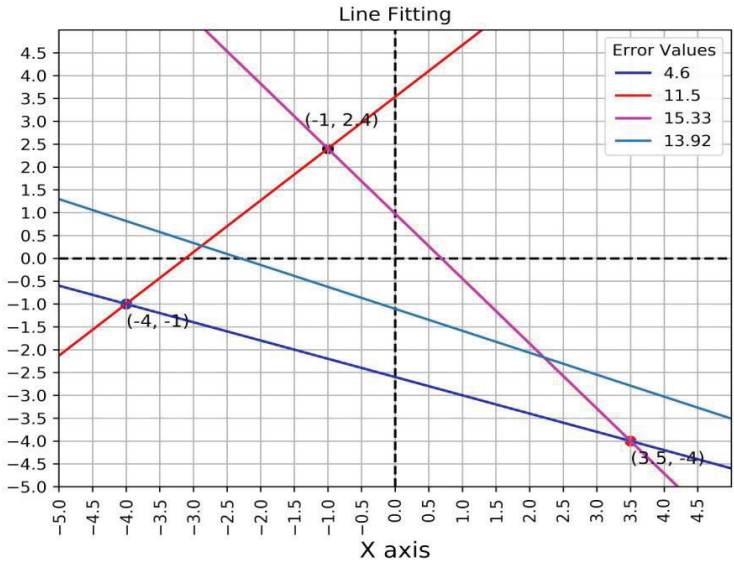
We have considered the case where all the three points are in line. Now, we will consider a case when the three points are not collinear. In that case, no line can pass through all three points. So, we try to find a line which is as close as possible to all the three points. We can define the metrics for a line to be close to all the points in many ways. We are defining this metric as an error term. Lower the error, more close the line to the points will be.

To explain, consider three non-collinear points, and a few lines fit through them. For different lines, we get different error scores, and the lowest error value (4.6) is achieved with the blue line passing through the points (-4, -1) and (3.5, -4).

Error metric used here is squared error of all the points:

$$L = \sum_{i=0}^{n} (y_i - pred_i)^2$$

Here y_i is the y value of the ith coordinate point and $pred_i$ is the y output using the equation of the line. In short, this error term squares the vertical distance between the original point and the line. This method is called the ordinary least squares regression method. To minimize this function, we need to take the first derivative of this equation, equate it to zero and find expressions for C_0 and C_1. The algorithm below shows the extracted expressions for each parameter.



In general, for a given set of points $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots, (x^{(n)}, y^{(n)})\}$, we want to find a single line that can accurately predict the output values. The equation of a line is:

$$y = c_0 + c_1 x$$

We want to find the values of C_0 and C_1 so that the $y_i \approx y_{pred}$. Using the ordinary least square regression method, we can

derive the coefficients directly, but we can only compute parameters for datasets with one independent variable.

Algorithm 1 Linear Regression

1: $\bar{x} \leftarrow \frac{\sum x^{(i)}}{m}$, $\bar{y} \leftarrow \frac{\sum y^{(i)}}{m}$

2: $SP_{xy} \leftarrow \sum_{i=0}^{m-1}(x^{(i)} - \bar{x})(y^{(i)} - \bar{y})$

3: $SP_{xx} \leftarrow \sum_{i=0}^{m-1}(x^{(i)} - \bar{x})^2$

4: $c_1 \leftarrow \frac{SP_{xy}}{SPxx}$

5: $c_0 = \bar{y} - c_1\bar{x}$

Method 2: Normal Equation

For the same problem of line fitting as above, we can use matrix manipulation to get our equation parameters. Using a normal equation for matrices, we can directly compute the parameters of the line equation.

$$C = (X^T X)^{-1} X^T Y$$

$$X = \begin{bmatrix} 1 & x^{(0)} \\ 1 & x^{(1)} \\ \vdots & \vdots \\ 1 & x^{(m)} \end{bmatrix}, Y = \begin{bmatrix} y^{(0)} \\ y^{(1)} \\ \vdots \\ (m) \end{bmatrix}$$

Here, $\boldsymbol{C}$ is a vector of coefficients $\boldsymbol{C_0}$ and $\boldsymbol{C_1}$

Method 3: Gradient Descent

Both of the above methods are not feasible for a very large number of points or for multivariate regression. So, we use a third method, called gradient descent.

As we know from method 1, the cost function tells us how bad the model is performing by calculating an error metric. To fit a line to several points, we need to find the parameter

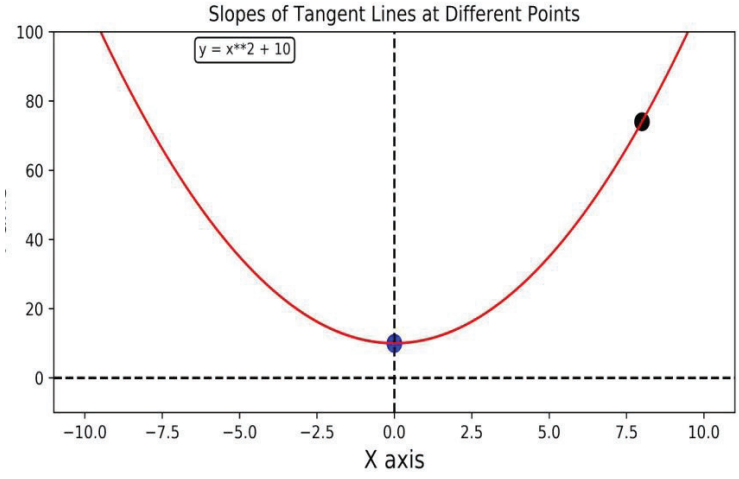
values that will minimize this error or cost function. Essentially, we need to find a point where the first derivative of the cost function is zero, and gradient descent allows us to do that. Gradient descent is a way to iteratively find minima of a cost function. We randomly start from a parameter value, calculate the output of cost function at that value, and update our parameters as:

$$c_k = c_k - \alpha \frac{\partial J}{\partial c_k}$$

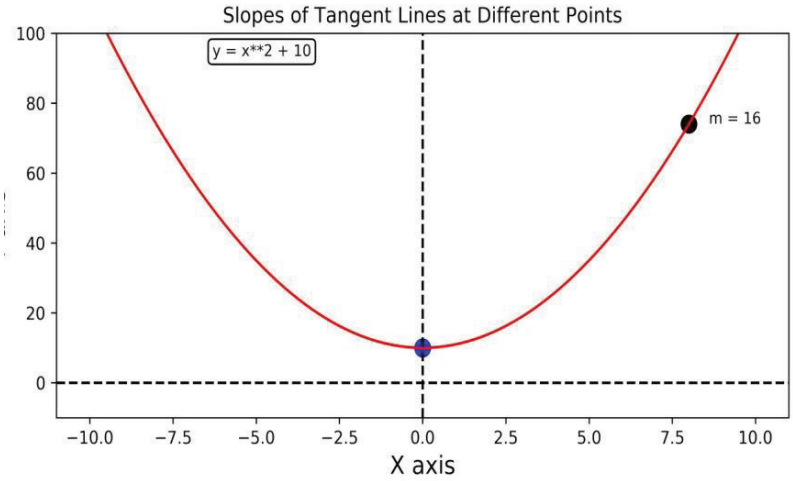
where C_K is the K^{th} parameter of our model, α is the learning rate, and $\frac{\partial J}{\partial c_k}$ is the derivative of a cost function with respect of parameter C_K.

To understand gradient descent, consider the following graph of a function:

$$y = x^2 + 10$$

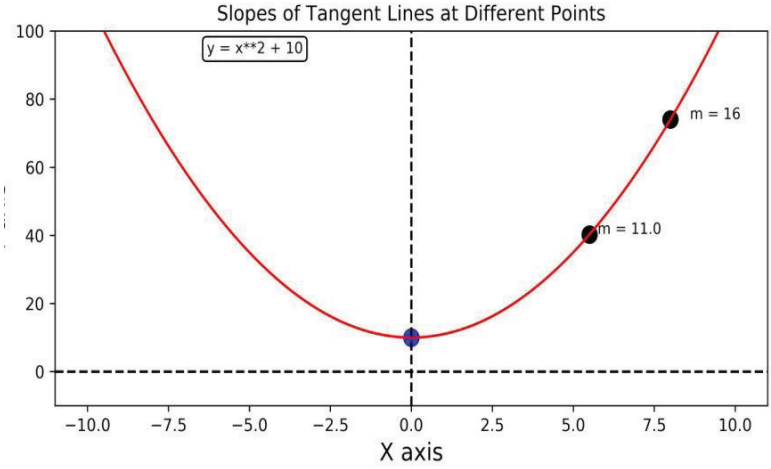


How can we move from black dot to blue dot (minima of function), given that we know only the x and y value of black dot (x = 8, y = 64)? Here, our knowledge of derivatives and extreme values comes handy. Consider the calculating slope of the function at the black point:



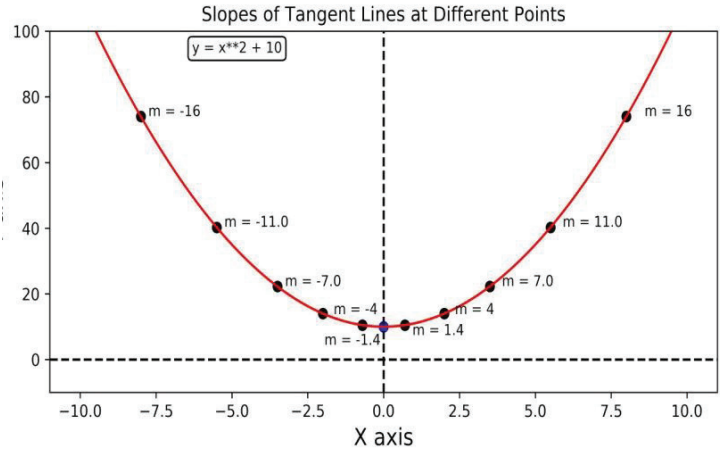
As the slope m = 16 is positive ($\frac{\partial J}{\partial c_k} > 0$), this tells us that at this point, the function value is increasing, and the value of 16 tells us that the value increases with a strong uprise steep. So now we know that, from this point, we do not want to move in the right direction. We should move back (to the left of the current point). So we take a step back and take x = 5.5.

Here, the slope is 11. It is still positive, but the magnitude is lower, which means the function is less steep, so we might be moving toward the basement/valley of the function, which is the minima of the function.



Similarly, we keep taking steps downhill, and as we can see in the graph below, the slope keeps decreasing, and eventually, we get m = 0 at x = 0 (blue point).

Consider starting from x = -8 and try reaching the blue point. Here, we will have the slope = -16 at x = -8. The negative sign tells that the function is going downhill at this point, so we need to move to the right (in the downhill direction). We need to keep moving in the right direction as we did in the previous example until we get the slope = 0 with x = 0.

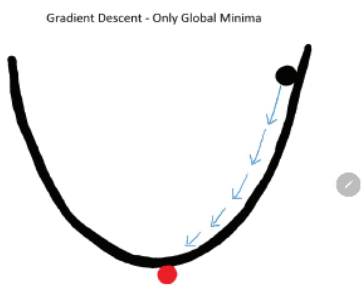


So, we randomly start from a point on function and update our parameters as:

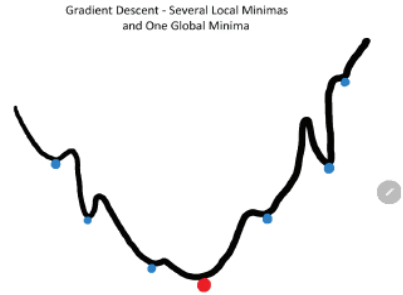
$$c_k = c_k - \alpha \frac{\partial J}{\partial c_k}$$

Here, C_K is the K^{th} parameter of our model, $\frac{\partial J}{\partial c_k}$ is derivative of a cost function with respect of parameter C_K and α is the learning rate.

Although a gradient descent iteratively minimizes the cost function, it does not guarantee to find global optimum minima of the function because loss functions usually look like the figure below with several local optimas (blue dots show local minimas, and red dots show global minimas). So, this highly depends on the initial values of the parameters. For each different initial value, we might end up in different local minima.



(a) Convex Loss Function

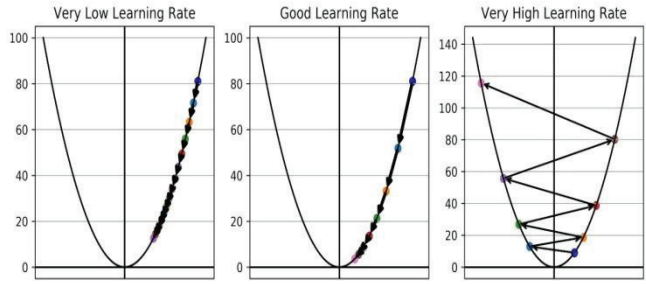


(b) Non-Convex Loss Function

Learning Rate

A learning rate is the means to control the rate of change in parameter values. The figure below shows a loss function with respect to one parameter. In the figure on the left, loss

convergence is extremely slow as the learning rate is very low. A very high learning rate might yield oscillating loss and might never converge (as shown in the rightmost figure). We have to be careful in choosing a learning rate so that it is neither too low to take forever to converge nor too high to explode (the middle figure shows loss convergence with a well-chosen learning rate). There are some other adaptive learning rate algorithms that adjust the learning rate on the run, e.g., Adagrad, Adaboost, RMSprop, etc.



Mathematical Notations

Each iteration of gradient descent gives new parameter values on which we evaluate our model. Reconsidering our earlier linear regression example, we need to see how much effect does C_D and C_1 have on the cost function. Using derivatives, we can see how a minute difference in each of the constant values will affect the cost function. We can rewrite the equation of line as:

$$p^{(i)} = c_0 \, x_0{}^{(i)} + c_1 \, x_1{}^{(i)}$$

Here, the subscripts $\mathbf{0}$ and $\mathbf{1}$ of the variable $\boldsymbol{x}$ are only to show the indexes of parameters, and $\boldsymbol{i}$ shows the $\boldsymbol{i^{th}}$ training set.

Consider the mean squared error function for calculating loss:

$$J = \frac{1}{m}\sum_{i=0}^{m-1}(p^{(i)} - y^{(i)})^2$$

Here, $\boldsymbol{p^{(i)}}$ is a predicted value by the model, and $\boldsymbol{y^{(i)}}$ is the actual target value for the $\boldsymbol{i^{th}}$ training set. For gradient descent, we need a derivative of this cost function:

$$\begin{aligned}\frac{\partial J}{\partial c_0} &= \frac{\partial J}{\partial c_0}(\frac{1}{m}\sum_{i=0}^{m-1}((c_0x_0^{(i)} + c_1x_1^{(i)}) - y^{(i)})^2) \\ &= \frac{2}{m}\sum_{i=0}^{m-1}((c_0x_0^{(i)} + c_1x_1^{(i)}) - y^{(i)})x_0^{(i)} \\ \frac{\partial J}{\partial c_1} &= \frac{\partial J}{\partial c_1}(\frac{1}{m}\sum_{i=0}^{m-1}((c_0x_0^{(i)} + c_1x_1^{(i)}) - y^{(i)})^2) \\ &= \frac{2}{m}\sum_{i=0}^{m-1}((c_0x_0^{(i)} + c_1x_1^{(i)}) - y^{(i)})x_1^{(i)}\end{aligned}$$

So, each parameter is updated as follows:

$$\begin{aligned}c_0 &= c_0 - \alpha\frac{\partial J}{\partial c_0} \\ c_1 &= c_1 - \alpha\frac{\partial J}{\partial c_1}\end{aligned}$$

3.7. Multivariable Linear Regression

We can extend one variable regression to multivariable linear regression by modifying our equations. The cost function will remain the same. We can rewrite single variable equations in a generic form:

$$
\begin{aligned}
p^{(i)} &= c_0 x_0^{(i)} + c_1 x_1^{(i)} + c_2 x_2^{(i)} + \cdots + c_n x_n^{(i)} \\
&= \sum_{j=0}^{n} c_j x_j^{(i)} \\
\frac{\partial J}{\partial c_k} &= \frac{2}{n} \sum_{i=0}^{m-1} (p^{(i)} - y^{(i)}) x_k^{(i)} \\
\implies c_k &= c_k - \alpha \frac{\partial J}{\partial c_k}
\end{aligned}
$$

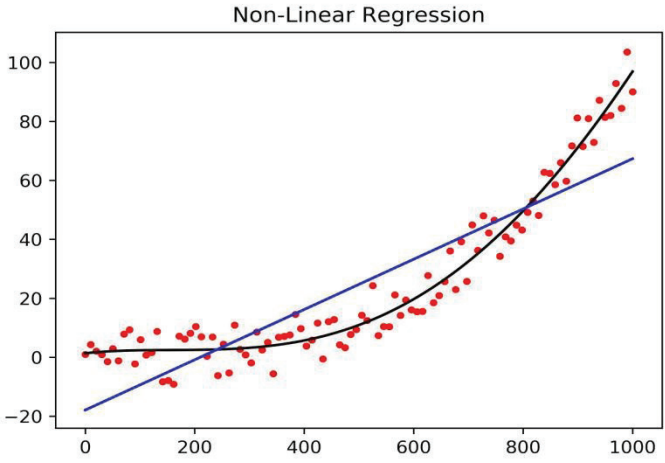
Here, m is the number of training examples, and n is the number of parameters in each training example.

3.8. Nonlinear Regression

With linear regression, we can fit our data for which the input features/independent variables have a linear correlation with a dependent variable. For non-linear data, like shown in the figure below, a linear fit (blue line) will not make a good model, but a nonlinear fit (black curve) will provide us with a good predictive model. Most of the real-world data is non-linear in nature. Some nonlinear regression models are Lasso and Ridge regression and Support Vector Machines.

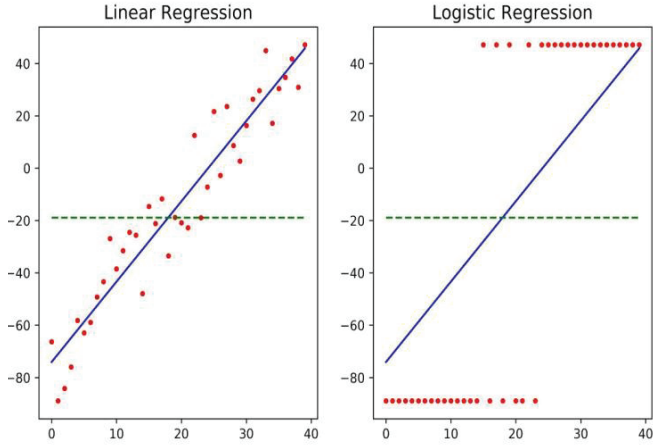
For nonlinear regression, we need to introduce high-level terms in our line fitting equation to make it a polynomial fitting equation:

$$
\begin{aligned}
p^{(i)} &= c_0 + c_1 x_1^{(i)} + c_2 (x_2^2)^{(i)} + c_3 (x_3^3)^{(i)} + c_4 (x_4^4)^{(i)} \\
x_0 &= 1 \\
=> p^{(i)} &= c_0 x_0 + c_1 x_1^{(i)} + c_2 (x_2^2)^{(i)} + c_3 (x_3^3)^{(i)} + c_4 (x_4^4)^{(i)} \\
=> p^{(i)} &= \sum_{j=0}^{n} c_j (x_j^j)^{(i)}
\end{aligned}
$$



3.9. Logistic Regression

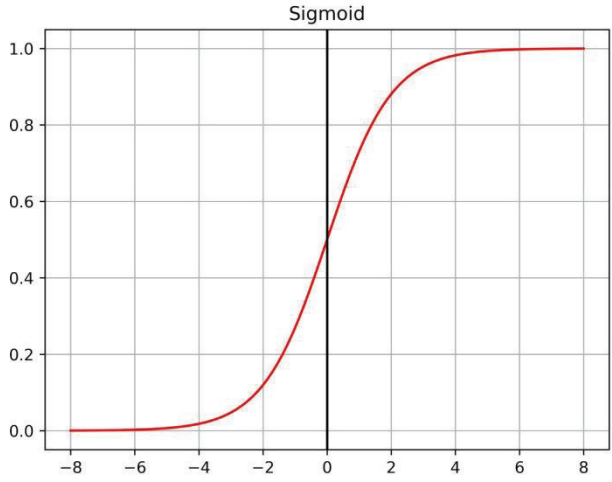
In logistic regression, we build models to predict discrete values as compared to continuous values in linear regression. Classifying an email as spam/not spam, classifying a tumor as malignant/non-malignant, or even classifying gender from the name are all examples of logistic regression. We can transform our output from linear regression to logistic regression by simply thresholding the values.



For example, in the figure above, the green dotted line shows thresholding value (points above that line are assigned maximum value, and points below that line are assigned minimum value), the blue line shows fitted regression line, and red dots show data points.

The problem with this threshold approach is the variation in data. With a different range of output values of our functions, the threshold value will vary, and we cannot hard code value every time. A function called sigmoid is introduced to tackle this challenge. It has the property to squash the values between 0 and 1. This function asymptotes to 1 for positive values of y, and to -1 for negative values of y, as shown in the figure below. By passing our output through the sigmoid, we can always get a threshold at $\boldsymbol{y = 0.5}$ for binary logistic regression. In general, the mathematical representation of this function is:

$$y = \frac{1}{1 + e^{-x}}$$



Now, as we know from linear regression:

$$p^{(i)} = c_0 {x_0}^{(i)} + c_1 {x_1}^{(i)}$$

To transform it into logistic regression, we apply the sigmoid function to our output. Hence, our final model prediction becomes:

$$p^{(i)} = \frac{1}{1 + e^{-(c_0 x_0^{(i)} + c_1 x_1^{(i)})}}$$

Since the cost function of linear regression cannot be used for logistic regression, a new cost function for logistic regression is introduced:

$$J = \begin{cases} -log(p^{(i)}) & y = 1 \\ -log(1 - p^{(i)}) & y = 0 \end{cases}$$

Finally, the cost function and gradient update rule:

$$\frac{\partial J}{\partial c_k} = \frac{2}{n} \sum_{i=0}^{m-1} (p^{(i)} - y^{(i)}) x_k^{(i)}$$

$$\implies c_k = c_k - \alpha \frac{\partial J}{\partial c_k}$$

4

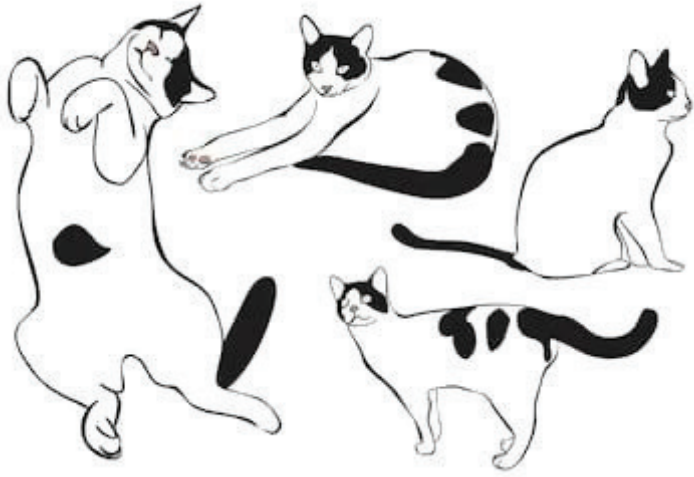
Machine Learning and Deep Learning

This chapter explains what machine learning and deep learning are. It also contains a brief history of how deep learning evolved with time, what past invented methodologies are being used today, and why deep learning has become so important today. It also gives the reader an idea of how simple mathematical functions were initially used and how they led to the development of complex deep learning models.

4.1. How can a Machine Learn?

Machines learn to carry out certain tasks by using algorithms and programs. For example, we can write a simple program to find the factorial of a number and a complex program to render the image of a black hole. In both cases, a human writes the step by step instructions (algorithm) to how to perform the task, and computers execute those sets of instructions. We design algorithms to solve different types of tasks, but there are many complex problems for which it would be extremely hard to develop explicit algorithms, e.g., recognizing cat faces or handwritten digits. Consider the example of recognizing a cat in the image. There are so many challenges that we cannot

hardcode, explicit features to consider all possible scenarios like occlusion, deformation, background clutter, and even intraclass variation (as shown in the examples below).



In machine learning, we do not program step by step instructions for the machine. Rather, machines use data to learn themselves. The process of learning includes data observation to look for patterns and modify the behavior for better results. More precisely, we build algorithms that use statistical analysis on data to look for patterns and produce outputs for a specific task without using explicit instructions. The Director of AI Research at Facebook says: "What makes us intelligent is our ability to learn, and this is what we're trying to reproduce in machines."

4.2. What is Deep Learning?

Deep learning is a branch of machine learning which employs algorithms to process data and provide output. Deep learning uses layers to feed forward data information from one layer

to the next layer. These layers learn to automatically extract meaningful features from the data and use those features to construct the results. In short, deep learning is a technique to learn by examples. The main feature of deep learning is that it eliminates the requirement of manual feature selection. It has the capability to learn feature extraction from data itself and learn representations.

For example, to recognize a dog in a photo, a neural net will be trained with a set of labeled dog images. This teaches the algorithm to differentiate between images containing a dog and an image without a dog. Given the amount and diversity of data, the model will be able to classify the images of a dog.

4.3. Why Deep Learning?

With increasing computing power and big data, deep learning is becoming a key part of various kinds of tasks. The main reason for its popularity is that it is powered by massive amounts of data. Hence, we are able to achieve the results that no other machine learning tool can provide. As there is so much data today, we can use deep learning to train our models and achieve better results than any traditional machine learning algorithms. There is no need for an expert to do the manual work of feature extraction. Deep learning algorithms automatically learn high-level features from data. Recent advances in deep learning have reached the point where these models surpass humans in tasks like object classification in images.

There are some quite complex tasks, e.g., recognizing cat faces in a variety of deformed and transformed images. It will be really hard to extract HOG features of different cat faces and

using kernels from traditional computer vision to match and find cat faces in the image. With deep learning, we just have to provide labeled data. Models can then automatically extract and learn important features. This gives immense power to the domain of deep learning because we can now apply it to any task without worrying about extracting important features from data (which requires domain experts). With the availability of huge amounts of data and with the computing power of GPUs, deep learning models can be trained on terabytes of data in a few hours using GPU clusters and cloud computing.

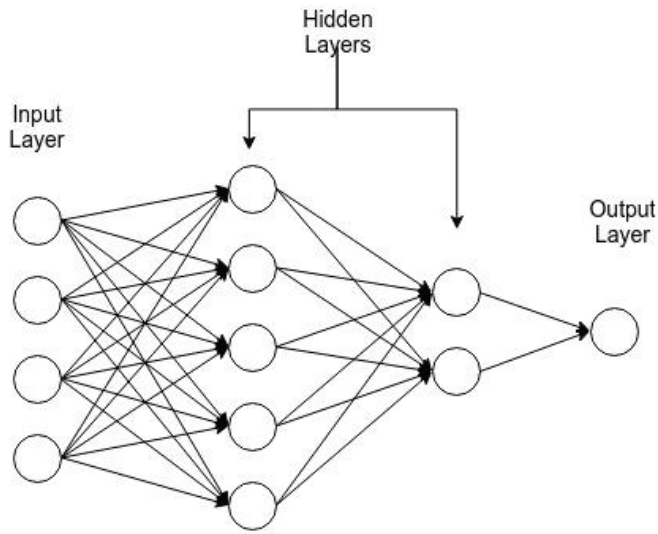
The idea of applying deep learning is simple. However, there are several complications within (e.g., initialization, activations, hyperparameters, etc.) that make it complex to achieve the target results. Hence, there is a lot of room for creative ideas in this domain.

4.4. Biological Inspiration

The human brain is the primary motivator for neural networks and deep learning. Artificial Neural networks (ANNs) are inspired by the idea of neuron firing and activation in the brain. These networks are based on how humans learn and process information. Our memory and all of our actions are controlled through our nervous system, which is composed of neurons. These neurons are connected in various ways with relative strengths. This same concept is applied in Artificial Neural Networks. On a basic level, an artificial neural network is a bunch of artificial neurons, called perceptrons, connected together to mimic biological neurons in a brain. An artificial neuron is essentially a mathematical function applied to an

input signal or value. These artificial neurons interact by passing information to other connected artificial neurons. They are organized as layers in a hierarchical structure. Neurons in each layer are connected to neurons in the next layer (as shown in the figure below). A neural network contains an input layer to get input data feed, a hidden layer to process that data, and an output layer to provide the output. A neural network with multiple hidden layers is called a deep neural network. Hence the term, deep learning.

Although we haven't been able to figure out exactly how the human brain works, most of the concepts and ideas used in deep learning or neural networks are directly inspired by biology. For example, this hierarchy of layers in Artifical Neural Networks (ANNs) is analogous to visual streams in Brodmann areas in the brain. The receptive field of convolutional neurons emulate biological receptive field, and suppression models are analogous to surround suppression to enable detection of object edges and shapes.



These algorithms do not perfectly emulate the brain, and we haven't been able to match the complexity of the brain yet. But still, these algorithms perform amazingly well on complex tasks. This is an active area of research in neuroscience to further understand visual cognition and information processing in the brain; e.g., how biological neurons perceive and process information, what events fire them the most, and which parts of the brain are active at what time.

4.5. A Brief History of Machine Learning and Deep Learning

The history of deep learning can be traced back to 1943 when Warren McCulloch and Walter Pitts published a paper with a concept of Artificial Neuron (AN) to mimic the thought process. This artificial neuron was based on the characteristic of a biological neuron of either being fully active to stimulation or not at all. This behavior of biological neurons was observed in microelectrode readings from the brain.

In 1957, Frank and Rosenblatt presented Mark I Perceptron Machine as the first implementation of the perceptron algorithm. The idea was to resemble the working of a biological neuron to create an agent that can learn. This perceptron was a supervised binary linear classifier with adjustable weights. This functionality was implemented through the following function:

$$f(x) = \begin{cases} 1 \; if \; w \cdot X + b > 0 \\ 0 \; otherwise \end{cases}$$

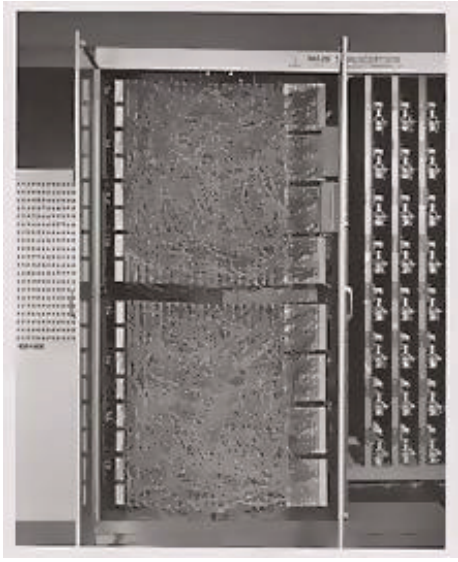
Here, $\boldsymbol{w}$ is a weights vector, $\boldsymbol{x}$ is inputs, and b is a bias.

For each input and output pair, this formula provided classification results. If the result/prediction did not match with output, the weight vector was updated through:

$$w = w + (y_{pred} - y\ )X$$

Here, $\boldsymbol{y_{pred}}$ is a predicted output of the function, $\boldsymbol{y}$ is the actual output, $\boldsymbol{x}$ is the input vector, and $\boldsymbol{w}$ is the weight vector.

It should be noted that, back at that time, they implemented this functionality through a hardware machine with wires and connections (as shown in the figure below).

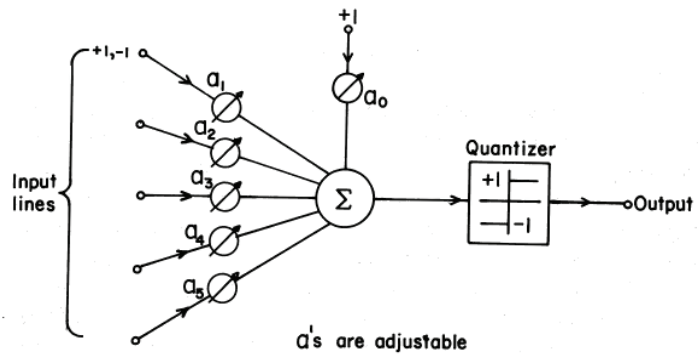


In 1960, Widrow and Hoff stacked these perceptrons and built a 3-layered (input layer, hidden layer, and output layer), fully connected, feed-forward architecture for classification as a hardware implementation, called ADALINE. The architecture presented in the paper is shown in the image below.

In 1960, Henry J. Kelley introduced a continuous backpropagation model, which is currently used in the learning weights

of the model. In 1962, a simpler version of backpropagation based on chain rule was introduced by Stuart Dreyfus. However, these methods were inefficient. The backpropagation that is currently being used in models was actually presented in the 1980s.

In 1979, Fukushima designed a multi-layered convolutional neural network architecture, called Neocognitron, that could learn to recognize patterns in images. The network resembled current day architectures but wasn't exactly the same. It also allowed to manually adjust the weight of certain connections. Many concepts from Neocognitron continue to be used. The layered connections in perceptrons allowed us to develop a variety of neural networks. For several patterns present in the data, the Selective Attention Model could distinguish and separate them.



In 1970, Seppo Linnainmaa presented automatic differentiation to efficiently compute the derivative of a differentiable composite function using the chain rule. Its application, later in 1986, led to the backpropagation of errors in multilayer perceptrons. This was when Geoff Hinton, Williams, and Rumelhart presented a paper to demonstrate that backpropagation in neural networks provides interesting distribution represen-

tations. In 1989, Yann LeCun, currently, Chief AI Scientist at Facebook, provided the first practical demonstration of backpropagation in convolutional neural networks to read handwritten digits at Bell Labs. But even with backpropagation, deep neural networks were not able to be trained well.

In 1995, Vapnik and Cortes introduced support vector machines for regression and classification of data. In 1997, Schmidhuber and Hochreiter introduced Long Short Term Memory (LSTM) for recurrent neural networks.

In all these years, a major hindering constraint was the computing power. But in 1999, computers started to become faster at processing data, and Graphical Processing Units (GPUs) were introduced. This increased computing power immensely.

In 2006, Hinton and Salakhutdinov presented a paper that reinvigorated research in deep learning. This was the first time when a 10-layer convolutional neural network was trained properly. Instead of training 10 layers using backpropagation, they came up with an unsupervised pre-training scheme called the Restricted Boltzmann Machine. This was a 2-step approach for training. In the first step, each layer of the network was trained using unsupervised objective. In the second step, all the layers were stacked together for backpropagation.

Later in 2009, Fei-Fei Li, a professor at Stanford University, launched ImageNet, a large visual database designed for visual object recognition research containing more than 14 million hand-annotated images of 20,000 different object categories. This gave neural networks a huge edge as data of this order made it possible to train neural networks and achieve good results.

In 2010, neural networks got a lot of attention from the research community when Microsoft presented a paper on speech recognition. Neural networks performed really well compared to other machine learning tools like SVMs and kernels. Specifically, they introduced a neural network as a part of a Gaussian Mixture Model (GMM) and Hidden Markov Model (HMM) framework and achieved huge improvements.

In 2012, a paper by Krizhevsky, Sutskever, and Hinton showed that huge improvements are achieved through deep learning in the visual recognition domain. Their model, AlexNet, outperformed all the other traditional computer vision methods in a visual recognition task and won several international competitions. Since then, the field has exploded, and several network architectures and ideas like Generative Adversarial Networks (GANs) have been introduced.

5

Fundamentals of Machine Learning

5.1. Model Training

A model learns the relationship between an object and its label by looking at examples, e.g., learning to recognize a cat on pictures. Training a model means teaching the model to memorize the features of an object and recognize them by looking at its examples. With each example from training data, the model tunes itself or updates its internal parameters to correct itself. For example, we feed a large number of cat images to the model, and the model learns to classify if an image contains a cat or not.

Training Data

The model learns from training data. It consists of example inputs and their true outputs; e.g., in our case, images and their correct labels (‘cat’ or ‘not cat’) are our training data.

Validation Data

The model does not learn from validation data. It is used to frequently evaluate the model. Validation results are used to

select the best performing model to tune hyperparameters. So, validation data indirectly affects the performance of the model.

Test Data

Test data is similar to training data, but it is not used to train the model. We only use this data to get predictions of the model after the model has been trained. The results on the test set tell how generalized the model is for unseen data.

There are three methods of training a model:

5.1.1. Stochastic Training

In a stochastic training, we randomly select a sample from a dataset, do a forward pass, then backpropagate to calculate gradients and update model parameters. As we update the parameters based on every single example in the dataset, a lot of noise gets added, and we might get oscillating loss values.

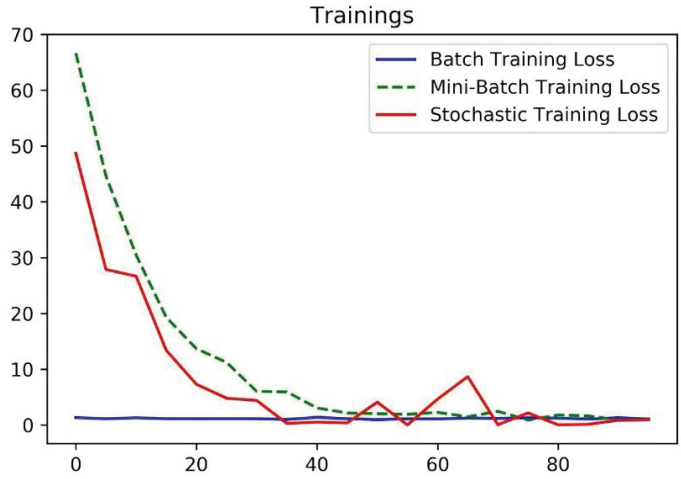
5.1.2. Batch Training

In a batch training, we use the entire dataset to forward propagate, calculate gradients, and update the model parameters. In this case, we always move toward the minima, and we do not zig-zag around the path.

5.1.3. Mini-Batch Training

In a mini-batch training, we use the in-between of batch training and stochastic training. For each iteration, we train our model on a randomly selected small number of examples from the training set. This reduces the noise as it gets the

average across the batch. We update our model frequently and can achieve loss minima in less time.



5.2. Types of Machine Learning

There are mainly three types of machine learning:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

Some other minor types include self-supervised and semi-supervised learning. While promising work has been shown in the latest research work of reinforcement learning, supervised and unsupervised learning remain the most common and widely used types.

5.2.1. Supervised Learning

Machine learning is more like approximating a function. Training a model means that it learns to map the input data to its correct output. To learn this mapping, we need to provide

some examples to the model. These examples consist of input data and their correct labels. In supervised learning, our goal is to build a model that can learn from limited training data (seen data) and can make accurate predictions on large test data (unseen data). Supervised machine learning models learn by looking at example input data and their corresponding true output (also called a label). These true outputs evaluate the model (how off the predictions are from the true output). So, the model tunes its parameters to predict values closer to true example outputs.

Three terms are used in this context:

Generalization

If a model can make accurate predictions equally on training data and test (unseen) data, it is said to have generalized well from training data to test data. In other words, same result metrics (e.g., accuracy) is achieved for training and test dataset.

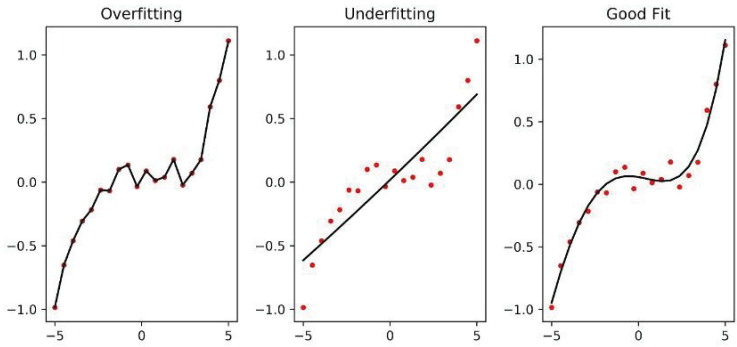
Overfitting

If a model performs well (sometimes with near-perfect accuracy and near-zero loss) on a training set but performs poorly on a test dataset, the model is said to overfit the training data. This can happen when the model is too complex. One approach to make the model simple is to use regularization, which means dropping higher-order terms in a regression model.

Underfitting

If a model performs poorly on both training dataset and test dataset, the model is said to underfit the data. Usually, this

happens when the model is not complex enough to learn the structure and distribution of data.



Common supervised learning algorithms are:

1. Linear Regression
2. Logistic Regression
3. Decision Trees
4. Support Vector Machine
5. Neural Networks

5.2.2. Unsupervised Learning

In unsupervised learning, there is no labeled data (correct outputs). Thus, there is no right or wrong, and there is no evaluation of the actual accuracy. Rather, the model itself learns the underlying structure and distribution of data and looks for inherent similarities to group data together. Some of the unsupervised learning techniques are Clustering, Autoencoders, Principal Component Analysis, and deep learning-based Generative Adversarial Networks (GANs).

5.2.3. Reinforcement Learning

This model of learning is all about reinforcing the desired behavior by rewarding correct behavior and punishing bad behavior. When this feedback system is not present, the system learns on its own by exploring its environment. It considers curiosity as a reward signal that reinforces the model to explore the environment and learn the correct behavior. This reinforcement learning model is similar to how newborn babies learn by observing and through curiosity, which is essentially learning from experience.

For example, we forbid a child to lie. If he lies, he will be punished for bad behavior. Similarly, if he speaks the truth, he gets a reward, such as a new toy or ice cream. In reinforcement learning, the model learns by interacting with its environment. Another example is to learn that fire burns and that it hurts. If a baby gets close to fire, he can feel the warmth, but if he tries to catch it, it will hurt him—but eventually, the baby would learn to keep a distance from the fire and not catch it.

Process

The reinforcement learning process consists of three basic components: state (representation of environment), action (what to do), and reward (based on the action). The goal of the machine learning model is to get the maximum reward. For example, consider a flappy bird game. The state is a single image frame, and the agent needs to learn to rise and fall to avoid collisions and stay in the air. If it touches a pipe or falls to the ground, the game is ended, which the agent considers a negative reward. A positive reward is a time to keep dodging the pipes and remaining in the air.

In deep reinforcement learning, we use deep neural networks to solve reinforcement learning problems. In traditional reinforcement learning, the agent tries to predict the best action based on a given state to maximize the rewards. This way, the model learns state-action pairs, i.e., the right action to perform for a particular state. The problem is that this knowledge accumulation of state-action pairs would explode in a complex problem. For example, if we need to play Chess or Go, then the complexity of the game would require the agent to store an immense number of state-action pairs, which is not feasible. So, we use neural networks that can generalize this knowledge and can predict the actions based on states rather than storing all this information.

5.2.4. Self-supervised

In supervised learning, the most tedious task is data engineering, which includes data gathering, cleaning, and labeling. For example, to train a model to classify cats in images, we need to train the model with a large number of feed-forward to cater to all possible varieties. These labels or correct outputs fed to the model are usually selected and annotated by humans.

Self-supervised learning is similar to supervised learning because we feed the correct labels of data to train a model. The difference is that we do not need humans to label data. The model automatically labels input data by extracting natural context, correlations, and metadata from it. This technique is well suited for models with dynamic input data or dynamic output labels and for those models which need to be retrained with new incoming information.

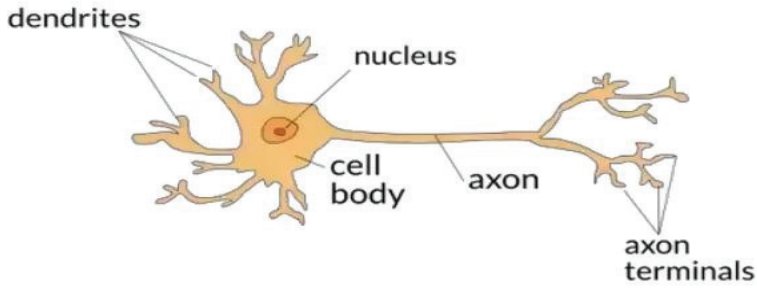
5.2.5. Semi-supervised

We need a huge amount of data to train a model that can generalize well. Usually, it is infeasible to label data of such magnitude because of the required human power and its associated cost. In all of the above machine learning types, either there are labels for all the datasets, or there are none. This machine learning type comes in between. Semi-supervised learning is for cases when we have a large amount of data, but only a small subset is labeled (or has true outputs). In this type, the model learns from both labeled and unlabeled data. The model is trained on small labeled data to identify groups of data and then trained on unlabeled data to define the boundaries of those groups, and also to identify new groups.

6

Fundamentals of Deep Learning

6.1. Biological Neurons

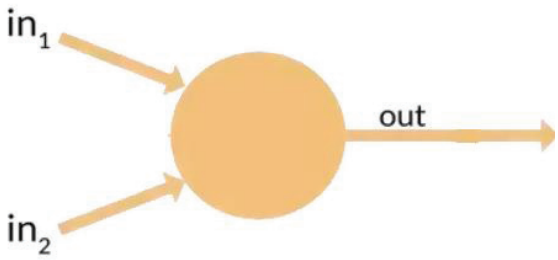


The brain is an information processing unit. It processes information in the form of signals. All senses: visual, hearing, and touch are converted to signals and passed to the brain to process it. Our brain consists of millions of small processing units called neurons. A neuron is shown in the figure above. It takes input from dendrites (which are actually output connections from other neurons to this neuron). Soma (cell body) processes those input signals through dendrites and sends the output signal on its axon, which is further connected to the dendrites of other neurons.

6.2. Perceptron

Neural Networks consist of computational neurons, also called perceptrons, which work like a biological neuron. A computation neuron takes input, processes that input, and provides output.

Suppose we need to recognize a fruit image. We have two inputs for that purpose: the color and the shape of some fruits and a single binary output, which is the fruit name.



Once the machine has learned all these properties, we can give it a new image of a fruit, *one it hasn't seen before*. The machine will hopefully classify it correctly and be able to tell us whether it is an orange or a banana.

The perceptron learns from the existing data and knows how to decide between multiple information by using weights. A higher weight means our perceptron considers that input more important compared to other inputs.

For this example, let's deliberately set suitable weights for our two inputs: 2 for the fruit shape and 4 for the fruit color.

The perceptron calculates and chooses the right fruit by simply multiplying the input with its respective weight and summing up all the values it gets for all the inputs.

Let's consider that we have two shapes: round and long. If the shape is round, the input value is 1, and if it is not round, the value is 0. We'll repeat the same process with the color: red takes the value of 1, and yellow color takes the value of 0.

Based on this information, if the fruit is round and red, the perceptron would do the following calculation:

$$Total = round \times shape\ weight + red \times color\ weight$$

$$Total = 1 \times 2 + 1 \times 4 = 6$$

This calculation is known as a linear combination.

To understand how to make a prediction of the right output using this value 6, we first need to define the threshold value.

The perceptron's output is either 0 for banana or 1 for orange. If the value of the linear combination is higher than the threshold value, the output is 1, and if it is not, the output is 0.

If the threshold value is 3, which means that if the calculation gives you a number less than 3, we have a banana. But if it's equal to or more than 5, then we have an orange.

That is the perceptron and how it works.

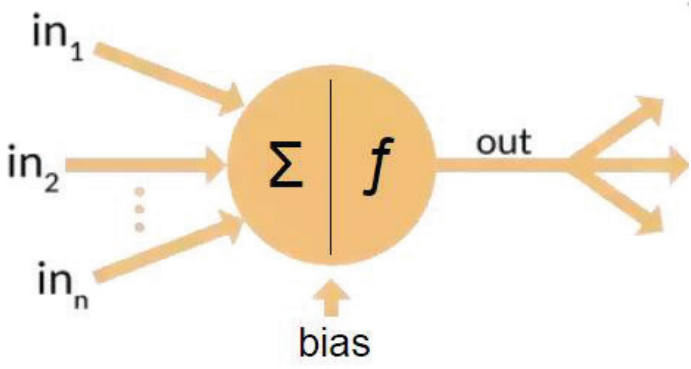
In reality, we set the weights to random values, and then the network adjusts those weights based on the output errors it made using the previous weights. That is called neural network training. In the next section, we will give more details about the wall mechanism.

In the mathematical language, the perceptron algorithms work like this:

$$output = \begin{cases} 0 \text{ if } \sum_i w_i x_i < threshold \\ 1 \text{ if } \sum_i w_i x_i \geqslant threshold \end{cases}$$

The *threshold* is sometimes moved to the other side of the inequality, and replaced with what's known as the neuron's *bias* to make things a little simpler for *training*.

$$output = \begin{cases} 0 \text{ if } \sum_i w_i x_i \ + \ bias < 0 \\ 1 \text{ if } \sum_i w_i x_i \ + \ bias \geqslant 0 \end{cases}$$



Now, with *bias*, we only need to make changes to the left side of the equation, while the right side can remain constant at zero.

This left side of the equation is a function. It is a function that transforms the values or states the conditions for the decision of the output neuron. It is known as an **activation function**.

The formula above is just one of several activation functions (and the simplest one) used in deep learning, and it is called the **Heaviside step function**.

$$f(h) = \begin{cases} 0 \text{ if } h < 0 \\ 1 \text{ if } h \geqslant 0 \end{cases}$$

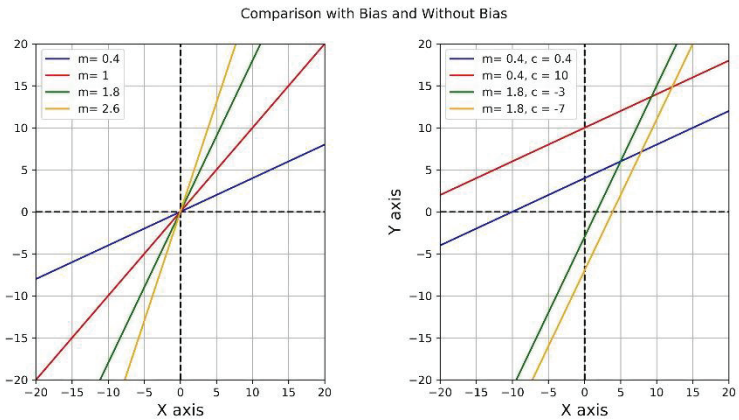
In reality, we can also use other activation functions. For example, we can use the *sigmoid function* presented in the next paragraph.

6.3. Why the Bias Value?

The bias value is a constant value that is added to the weighted input of a perceptron. It is just like the y-intercept (c) in our line equation:

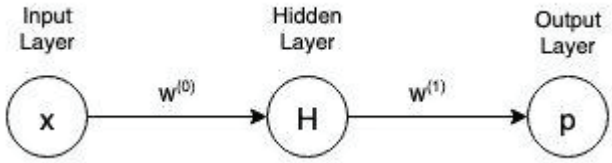
$$y = mx + c$$

It helps the model to fit the data best because, if we don't add bias value, our model can only represent data patterns for which the line passes through the origin. Using the $\boldsymbol{m}$ parameter (or weights in case of neural network), we can represent lines of different slopes, but all these lines would pass through the origin. Consider the figure below. All the lines in the figure on the left have $\boldsymbol{c} = \mathbf{0}$. Thus, these lines pass through origin while the lines on the figure on the right have different y-intercept values.



Using a bias value makes sure that our output is not $\mathbf{0}$ if all of our input values are $\mathbf{0}$. For example, consider the following image with one input neuron, one hidden neuron, and one output neuron. Now, if our input $\boldsymbol{x} = \mathbf{0}$, then no matter what the value of the input is, our output will be $\mathbf{0}$. There are several

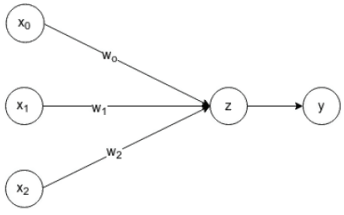
cases when we don't want our model to predict $\mathbf{0}$ if the input is $\mathbf{0}$. For such cases, we need a bias value.



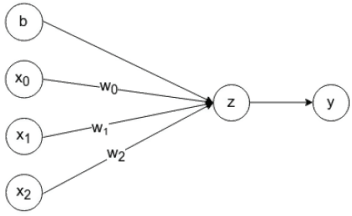
The bias value can be learned during the training phase.

6.4. Perceptron Examples

Now, we will take a look at more examples of an artificial neuron that takes several binary inputs and produces a binary output. It can be thought of as a digital logic gate. The figure (a) below shows a perceptron with three binary input units and one binary output unit. The underlying working of this neuron is more like logistic regression explained in chapter 2. $\{x_0, x_1, x_2\}$ are input values. Each of the input units has a weight associated with it, and those input values are multiplied to their respective weights. The result is binary thresholded.



(a) Perceptron with 3 input units and 1 output units



(b) Preceptron with 3 input units,1 bias unit and 1 output unit.

Figure (a) above shows a perceptron with three input units and an output unit, while figure (b) shows a perceptron with three input units, a bias unit, and an output unit. The value $\boldsymbol{z}$ is calculated by multiplying input values to their weights

and adding bias value to the result, which is then binary thresholded.

$$z = w_0x_0 + w_1x_1 + w_2x_2$$

$$y = \begin{cases} 0 & z \leq 0 \\ 1 & z > 0 \end{cases}$$

$$z = w_0x_0 + w_1x_1 + w_2x_2 + b$$

$$y = \begin{cases} 0 & z \leq 0 \\ 1 & z > 0 \end{cases}$$

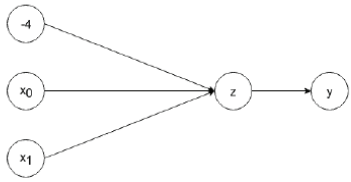
For better understanding, consider the example of perceptron models for AND, OR, and NOT binary operations.

AND Operation			
x_0	x_1	z	$y(threshold = 0)$
0	0	-4	0
0	1	-1	0
1	0	-1	0
1	1	2	1

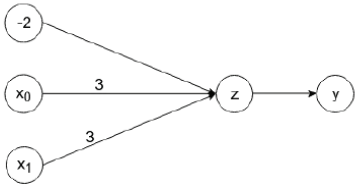
OR Operation			
x_0	x_1	z	$y(threshold = 0)$
0	0	-2	0
0	1	1	1
1	0	1	1
1	1	4	1

NOT Operation		
x_0	z	$y(threshold = 0)$
0	2	1
1	-1	0

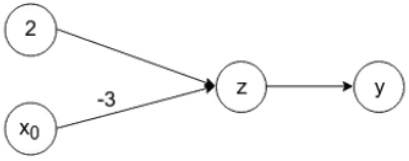
Figure (a), (b), and (c) show perceptrons for AND, OR, and NOT operations, respectively.



(a) Perceptron with 3 input units and 1 output units



(b) Perceptron with 3 input units and 1 output units

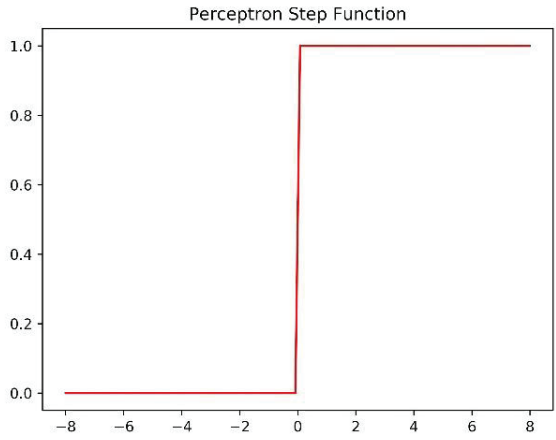


(c) Perceptron with 3 input units and 1 output units

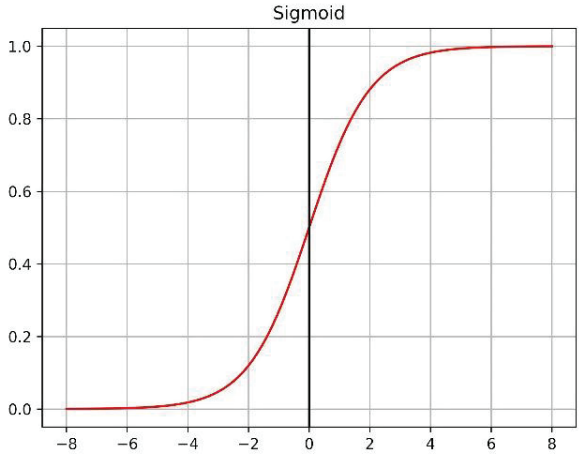
6.5. Sigmoid Function for the Perceptron

Training a machine learning or deep learning model means to iteratively tune the weights of the model to achieve the desired results. This means that a small change in weight should have a small change in output, so we can tweak our output minutely by a very small change in our parameters. To understand, consider an example, where we have to train a model to recognize digits (0–9) from an image. Suppose our model can recognize all digits but sometimes cannot distinguish between 6 and 8. We need to tweak our weights a little bit so that the model can distinguish and recognize those pixels of the digit/ at the same time, we need to make sure that the new weights do not affect the performance of the model for other digits. Having a strict threshold value would not make it possible as a little change in parameters will either flip the outcome or will not affect the outcome at all. This can

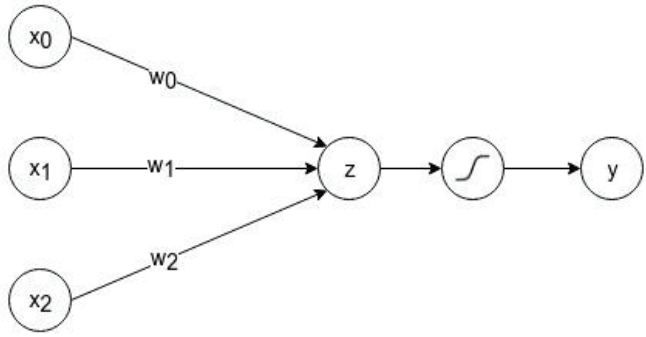
be better visualized in the figure below, which represents a perceptron thresholded at 0 (yielding a step function).



To avoid this problem, we use the sigmoid function (shown in the figure below). It performs similarly when the output is a very large negative number or a very large positive number. However, when the value is in-between, a slight change in input will give us a slight change in output. It would also squash all the output values between 0 and 1, so we do not have to choose different threshold values for different functionalities.

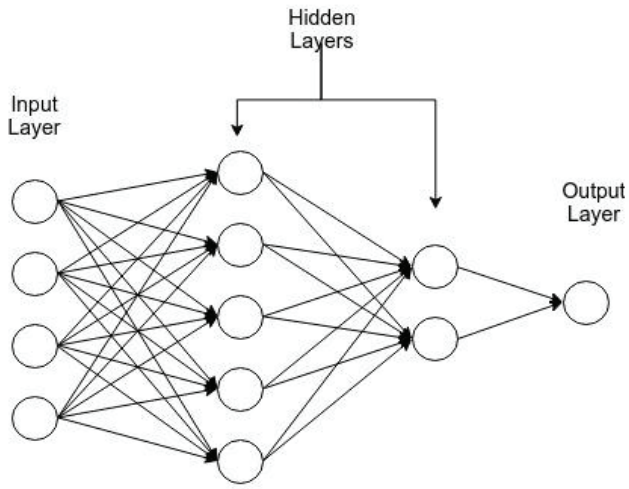


So, our perceptron model will be changed to the following:



6.6. Multilayer Feed-Forward Networks

With the configuration of neurons and weights explained above, the perceptron can perform basic operations. Using different values of weights, we can build different models, but it cannot make complex decisions. This same, simple decision-making perceptron structure can be stacked and merged together to develop complex decision-making models. Such a model structure would look as shown in the figure below.



In this figure, there are input neurons, output neurons, and some intermediate neurons. All input neurons make up the input layer, and all the output neurons, combined, are said to be the output layer. The input and output layer neurons just store input and output values, respectively. The intermediate neurons are said to be the hidden layers and are actually perceptrons.

The important part is the links that connect neurons in one layer to neurons in another layer. These links have some values associated (called weights) which tell, how much importance should be given to that link. It can be positive, negative, or even zero. A large positive value means that the neuron at the input side of link highly affects the outcome of end neuron, a large negative value means that the neuron at the input side of link negatively affects the outcome of end neuron, and a value of 0 means that the input neuron does not have any effect on output neuron. By using these links, the output of a perceptron from one layer is fed as input to perceptrons in the next layer. These hidden layers are used to make complex nonlinear models. It is a feed-forward neural network because the input data propagates forward through hidden layers and weight parameters to the final model prediction. This means that all these internal weights affect the final outcome, and different weight configurations can result in different outcomes.

6.7. Training Neural Networks with Backpropagation

In earlier chapters, we tried to explore and understand the working of the linear regression through equations and parameter updates using gradient descent. Now, we will use

neurons (neural network) to implement the same. Training a neural network includes multiple iterations of the following steps:

1. Forward pass to calculate model output $prediction = input * weights$
2. Calculate loss $= (target - prediction)^2$
3. Backward Pass (backpropagation) that calculates the contribution of each weight to final loss using a derivative of loss with respect to all parameters.

$$contribution\ of\ weight\ to\ loss = \frac{derivative\ of\ loss}{w.r.t.\ weights}$$

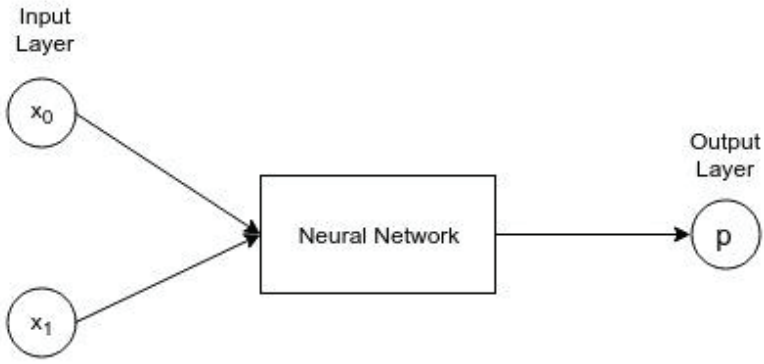
4. Parameter Update that uses optimization algorithms to modify the model parameters. The weights are updated in proportion to their contribution to the error:

$$weight = weight -$$

$$loss \times contribution\ of\ weight\ to\ loss$$

Neural networks are trained in the same way as other machine learning models, but its internal workings are a little complex. For simplicity, we will try to understand neural networks in an abstract way.

Think of a neural network as a black box and consider the following example of a neural network for a regression problem (predicting direct continuous values). The input layer has two input units, $\boldsymbol{x_0}$ and $\boldsymbol{x_1}$, and the model's output layer (prediction) has a single output unit $\boldsymbol{p}$.

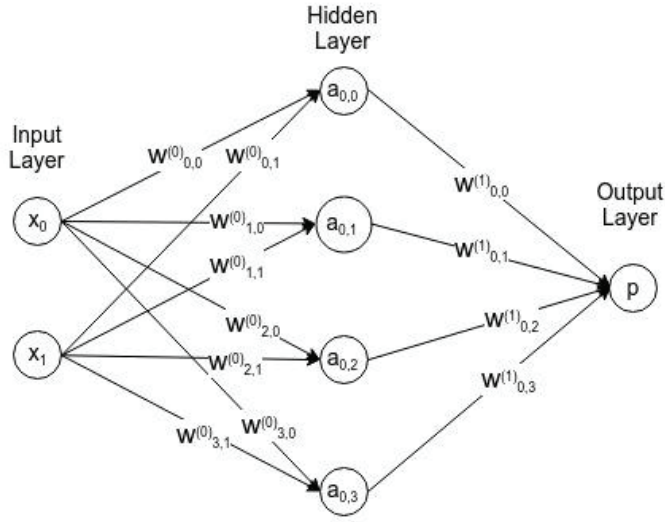


Just as we trained models before, we compare the model's prediction with our expected outcome and calculate the loss. We need a loss function that can quantify how off the predictions are from the actual or expected outcome. There are several loss functions that we can use (loss functions are explained in detail in deep learning chapter). Mean Squared loss function for regression is given by:

$$J = \frac{1}{m}\sum_{i=0}^{m-1}(p^{(i)} - y^{(i)})^2$$

where, $\boldsymbol{y^{(i)}}$ and $\boldsymbol{p^{(i)}}$ are the target and predicted values for $\boldsymbol{i^{th}}$ training set. To minimize this loss function, we find the gradient of the loss function and use gradient descent to iteratively reduce the loss by tuning model parameters.

Let's open the black box and consider the multilayer feed-forward network inside it, as shown in the figure below.



This feed-forward network has one input layer of two neurons, one hidden layer of four neurons, and one output layer of one neuron, as shown in the figure above. Hidden layer activations are represented using ***"a"*** with the first subscript showing the index of the hidden layer, and the second subscript showing the index of the neuron in the hidden layer. Superscripts of weights show the layer number of incoming input values. The first subscript shows the neuron number in the second layer, and the second subscript shows the neuron number in the first layer. For example, $w_{2,1}^{(0)}$ is a weight between layer 0 (input layer) and layer 1 (hidden layer), from the neuron of index 1 in layer 0 to neuron of index 2 in layer 1.

We can write this Multilayer Perceptron (MLP) structure in the following way:

$$\begin{bmatrix} a_{(0,0)} \\ a_{(0,1)} \\ a_{(0,2)} \\ a_{(0,3)} \end{bmatrix} = \begin{bmatrix} w_{(0,0)} & w_{(0,1)} \\ w_{(1,0)} & w_{(1,1)} \\ w_{(2,0)} & w_{(2,1)} \\ w_{(3,0)} & w_{(3,1)} \end{bmatrix} \begin{bmatrix} x_{(0)} \\ x_{(1)} \end{bmatrix}$$

$$p = \begin{bmatrix} w_{(0,0)} & w_{(0,1)} & w_{(0,2)} & w_{(0,3)} \end{bmatrix} \begin{bmatrix} a_{(0,0)} \\ a_{(0,1)} \\ a_{(0,2)} \\ a_{(0,3)} \end{bmatrix}$$

The issue with this MLP model is that we can only model linear relationships. We cannot model nonlinearity with this structure. We could skip the hidden layer and still be able to achieve the same results. To understand, let's combine the two equations above:

$$p = \begin{bmatrix} w_{(0,0)} & w_{(0,1)} & w_{(0,2)} & w_{(0,3)} \end{bmatrix} \begin{bmatrix} w_{(0,0)} & w_{(0,1)} \\ w_{(1,0)} & w_{(1,1)} \\ w_{(2,0)} & w_{(2,1)} \\ w_{(3,0)} & w_{(3,1)} \end{bmatrix} \begin{bmatrix} x_{(0)} \\ x_{(1)} \end{bmatrix}$$

Now, if we multiply these two weight matrices, we will get only one weight matrix of dimensions (1 x 2). So, we can eliminate this hidden layer and have only two connections from two input neurons to one output neuron.

So, we need to introduce nonlinearity in those perceptrons. For that, we need nonlinear activation functions for each hidden layer. We can choose from several available nonlinear activation functions, such as Sigmoid, Tanh, ReLU, LeakyRelu, etc. Let's use the sigmoid function. So, our hidden layer and output layer activations will be:

$$\begin{bmatrix} a_{(0,0)} \\ a_{(0,1)} \\ a_{(0,2)} \\ a_{(0,3)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{(0,0)} & w_{(0,1)} \\ w_{(1,0)} & w_{(1,1)} \\ w_{(2,0)} & w_{(2,1)} \\ w_{(3,0)} & w_{(3,1)} \end{bmatrix} \begin{bmatrix} x_{(0)} \\ x_{(1)} \end{bmatrix} \right)$$

$$p = \sigma\left(\begin{bmatrix} w_{(0,0)} & w_{(0,1)} & w_{(0,2)} & w_{(0,3)} \end{bmatrix} \begin{bmatrix} a_{(0,0)} \\ a_{(0,1)} \\ a_{(0,2)} \\ a_{(0,3)} \end{bmatrix}\right)$$

The output is directly calculated from the weights of the last layer and activations from the hidden layer, while the activations of the hidden layer are calculated using weights of the first layer and the input values. And so, the weights in the first layer do not directly affect the output value. Rather, those weights affect the activations of the hidden layer, and these activations, in turn, affect the output. The model is trained by learning what configuration or set of weights will predict the value closest to the expected value. So, the weights are jittered to find optimum values for the model. This jittering of weights is performed by calculating derivatives as the derivatives can tell how much the weight value should be increased or decreased to reduce the loss. This derivative calculation process to tune weights is called **backpropagation**.

As we know from earlier examples, to see the effect of each weight on output, we need to calculate the partial derivative of the loss function with respect to each weight parameter. For example,

1. To see how a change in weights of the last layer $w_{i,j}^{(1)}$, will affect the loss value, we directly calculate the partial derivative of the loss function with respect to weight $w_{i,j}^{(1)}$.
2. We cannot directly compute the gradient of the loss function with respect to the weights of the first layer $(w_{i,j}^{(0)})$ because these weights do not influence the loss value directly. Therefore, we first need to calculate the

effect of these weights on activations $a_{0,i}$ (a partial derivative of activations with respect of initial weights), and the effect of change in those activations on final output or loss value (a partial derivative of loss with respect to activations). For this, we use the chain rule.

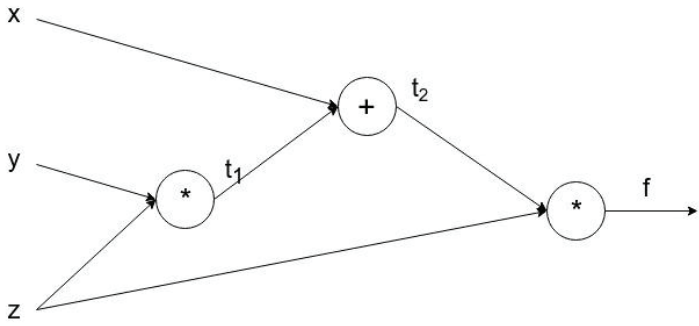
$$\frac{\partial Loss}{\partial w_{i,j}^{(0)}} = \frac{\partial Loss}{\partial a_{0,i}} * \frac{\partial a_{0,i}}{\partial w_{i,j}^{(0)}}$$

Finally, an optimization algorithm is used to update the weights in proportion to their effect on loss value.

To understand how backpropagation works, consider a simple example of the following function:

$$f = z(x + y \times z)$$

We have three inputs: x, y, and z, and one output f. The computational graph for this function is shown below:



We introduce new variables t_1, t_2 for middle computations:

$$t_1 = y * z \quad \Rightarrow \frac{\partial t_1}{\partial y} = z, \frac{\partial t_1}{\partial z} = y$$

$$t_2 = x + t_1 \quad \Rightarrow \frac{\partial t_2}{\partial x} = 1, \frac{\partial t_2}{\partial t_1} = 1$$

$$f = z \times t_2 \quad \Rightarrow \frac{\partial f}{\partial z} = t_2, \frac{\partial f}{\partial t_2} = z$$

Now, we want to see how initial x, y, and z values affect the final function f.

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial t_2} * \frac{\partial t_2}{\partial x} = z * 1 = z$$

Now, let's understand this partial differentiation. The effect of input x on the final output is equal to z. This means that one unit increment in x will result in z units increment in output or one unit decrement in x will result in z units decrement in output.

For example, let's assume $z = -4,\ x = 9,\ y = 3$

$$f = z(x + y * z) = 12$$

Now, let's increment x by one, so we will have x = 10. Now:

$$f = z(x + y * z) = 8$$

As we can see that one unit change in x resulted in z units increment in output (since z is negative, so the output value is decreased by z units)

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial t_2} * \frac{\partial t_2}{\partial t_1} * \frac{\partial t_1}{\partial y} = z * 1 * z = z^2$$

Therefore, change in y will result in z^2 change in output.

Similarly, to calculate the effect of input 'z' on function, we need to consider two links because z affects the function in two ways:

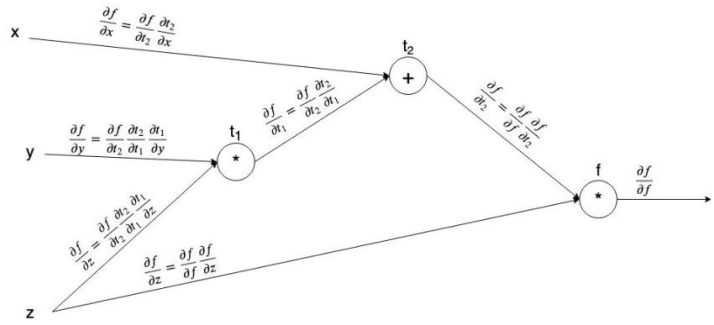
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial t_2} * \frac{\partial t_2}{\partial t_1} * \frac{\partial t_1}{\partial z} + \frac{\partial f}{\partial z}$$

$$= z * 1 * y + t_2 = z^2$$

$$= yz + (x + t_1)$$

$$= yz + (x + yz)$$

$$= x + 2yz$$



6.8. Parameters and Hyper-Parameters

Parameters and hyper-parameters are the settings or configuration variables that help us determine the performance of the model. These are the agents that are fine-tuned to improve the performance of the model.

6.9. Parameters

Parameters are inherent to the model, and they can be effectively determined from the model's data itself. These are directly related to the model. Models are categorized as parametric or nonparametric, depending on whether they

have a fixed or changing number of parameters. There are some characteristics of the parameters listed below.

1. Usually, they are automatically set
2. They are determined from the model's learned data
3. Models rely on them when making predictions

Some of the major examples of model parameters include:

1. Coefficient involved in linear regression
2. Weights of the deep neural network
3. Support vectors of a Support Vector Machine (SVM)

6.10. Hyper-Parameters

Hyper-parameters are the configuration variables, the values of which are not indigenous to the model and can't be directly estimated from the model's data. Unlike parameters, these are external to the model but still affect its performance. Whenever we are setting a parameter of our own, we are actually setting a hyper-parameter. Hyper-parameters can be of two categories:

1. Model Structure Specific Hyper-Parameters
    a. Number of hidden layers
    b. Number of units in a hidden layer
    c. Activation functions
2. Model Optimization Hyper-Parameters
    a. Learning rate
    b. Epochs
    c. Mini-batch size
    d. Dropout

e. Weights initialization

Some of these hyper-parameters are described below.

6.10.1. Number of Hidden Layers

Deciding the number of hidden layers mostly depends on the nature of data. For simple and linearly separable data, we do not need a hidden layer, but most of the real-world data is complex and nonlinear in nature. In such cases, we need hidden layers to draw boundaries in higher dimensions for data with multiple features.

We can solve most of the problems with just one hidden layer. And with two hidden layers, we can approximate any complex function as the second layer can learn abstract representations of data building on the output of the first layer. Using deeper networks with more than two hidden layers does not really affect the outcome (but deep convolutional neural networks do affect the results). It simply consumes more memory, storing weights and gradients during backpropagation.

6.10.2. Number of Neurons

Neural networks approximate functions, and more neuron units mean that the model will be better at approximating complex functions. The number of neurons decides how complex the decision boundary of a model can be. A model will need a few neurons to learn a simple function and will need a large number of neurons in each layer to learn a complex function. We need to carefully decide the number of neurons because very few neurons will result in underfitting the data, and a large number of units will lead to data overfitting. An observation that gives

better results is to have more neurons in the first hidden layer than the number of units in the input layer.

6.10.3. Learning Rate

The learning rate is known as the mother of all hyper-parameters. It is the rate at which a model updates its parameters. If the learning rate is too low, it will take a longer time for the model to converge, and if it is too large, the model might overshoot and never converge. So, we have to decide the learning rate carefully. A decaying learning rate is preferred, which reduces the learning rate with the number of iterations. The learning rate has been explained in detail in the topics above.

6.10.4. Epochs

The number of epochs is the number of iterations to train the model. In each iteration, we go through our dataset and improve the performance of the model by updating its parameters. We need to decide when to stop these training iterations. It is possible to attain a minimum loss way before completing the total number of iterations. It is also possible that we have iterated for the set number of epochs, but the loss hasn't reached the minimum possible value. For such cases, we need to keep iterating as long as the loss on the test set keeps decreasing. Based on such factors, we can determine when to stop iterating. For example, we can stop when there is a negligible difference in test dataset loss for the last 10 iterations.

6.10.5. Mini-Batch Size

It has been proven that models do not generalize well with very large mini-batch size because large batch size methods tend to converge to sharp minimizers and cannot escape from the basin of those minimas. A large batch size means that we need a huge amount of memory for model training.

A small mini-batch size adds noise in error calculations, but these methods tend to converge to flat minimizers and avoid getting stuck in local minima during training. Some good values for a mini-batch size are 4, 8, 16, and 32.

6.10.6. Dropout

Generally, we use a dropout rate of 20 percent – 25 percent. A value that is too high would result in under-learning of the network, and a value that is too low might not have a significant effect on the outcome. A dropout provides better results on large networks (a large number of neurons and multiple layers).

6.10.7. Weights Initialization

As explained earlier, the initial parameter values affect the minimum loss value attained and the time to converge because of the complex nonlinear nature of loss function having several local minimas. Normally, the weights are initialized to random normal distribution or uniform distribution.

6.11. Layers of a Deep Neural Network

A neural network is similar to the human brain. Just like our brain observes the happenings around us, stores data, and at a

later point in time, it helps us predict future patterns, keeping in view the previously held data; neural networks use the available data to predict future patterns or recognize objects.

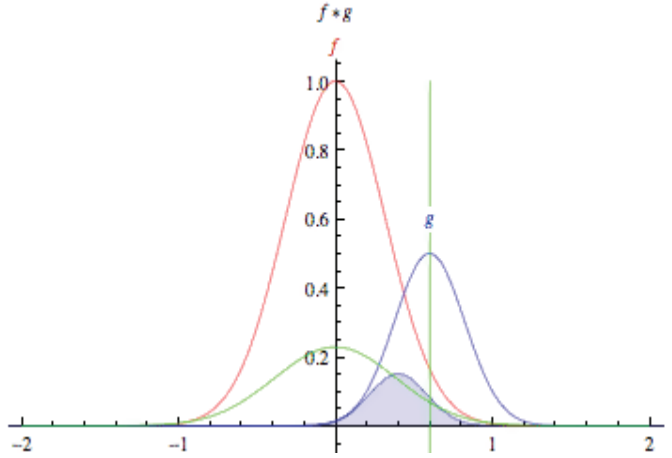
Deep neural networks are basically the same as neural networks. It's just that they contain more layers than what a normal neural network does. This is the reason they are called deep neural networks.

The layers of a deep neural network can be best understood in this way. Let's consider an image *input-image*. The deep neural network will pass it through a series of different layers and produce an output that we name *output-image*. Input-image will be passed through convolutional, pooling, nonlinear, and finally, fully connected the layer to produce an output. The various layers involved in the deep neural networks are described below.

To understand the convolution layer properly, first, we need to understand what convolution actually is.

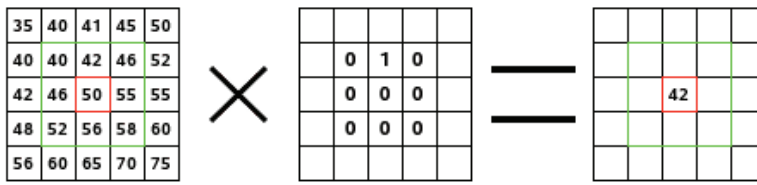
6.12. Convolution

The word *convolution* is derived from the Latin word *convolvere*, which means *to roll together*. In terms of maths, convolution is the measure of how much two different functions mutually roll over one another. To understand it better, think of convolution as two independent functions multiplied by each other.



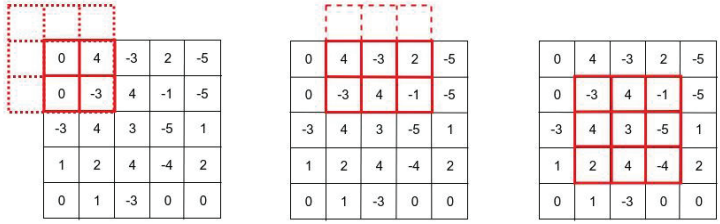
In the figure above, the green curve shows the convolution of blue over the red curve. This is what the basic concept is all about: two functions rolled over each other.

Another example is a kernel rolled over an image matrix. To understand it better, look at the figure below. On the leftmost of it is the matrix or the subject to be worked on, and in the middle is a 3*3 kernel. The digits in the boxes are the pixel values. When a kernel is applied onto the matrix, the rightmost figure is the answer. How does this happen? The green area in the matrix is the kernel size. The kernel multiplies its corresponding values with those of the matrix, and their sum is displayed in the center.

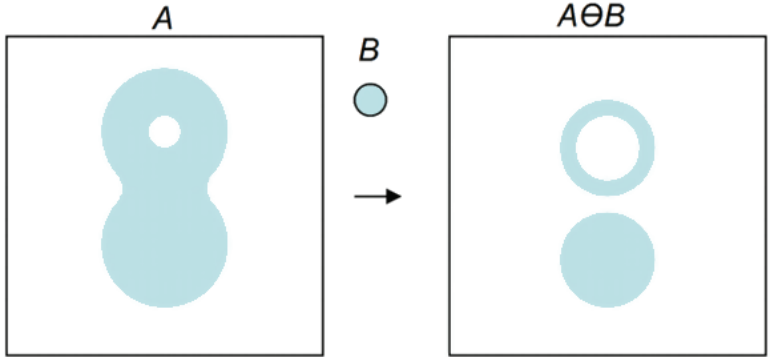


The following figure shows three different positions of the kernel being convolved with an image matrix. The dotted area

shows don't care region. The rest of the overlapping values in the kernel and matrix are multiplied and added together for each possible position.



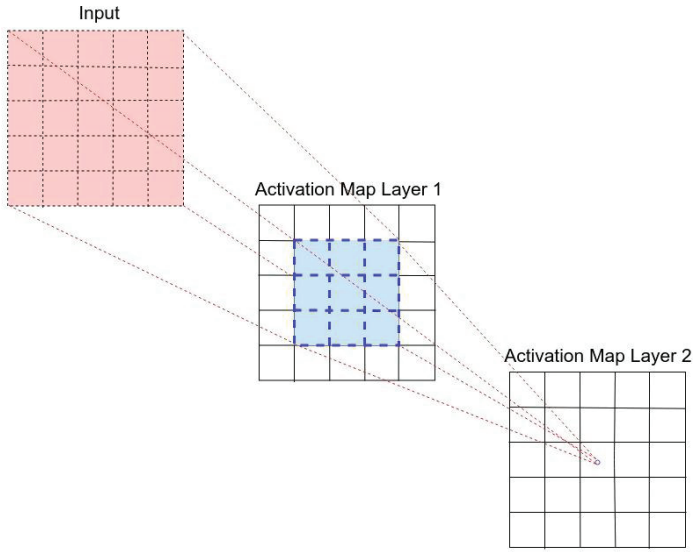
In deep learning, convolution is used for template matching or edge detection. One particular application of erosion in image processing is shown in the figure below. The box B is the filter/kernel, and box A is the subject. When a filter is applied to the subject, the figure on the right is the result.



6.12.1. Effective Receptive Field

Effective receptive field is the region of input space, which may affect any units in the networks. This input field can be the inputs of that region itself as well as the outputs of other regions. Keeping this in view, a receptive field can be calculated in relation to our input as well as the units under consideration.

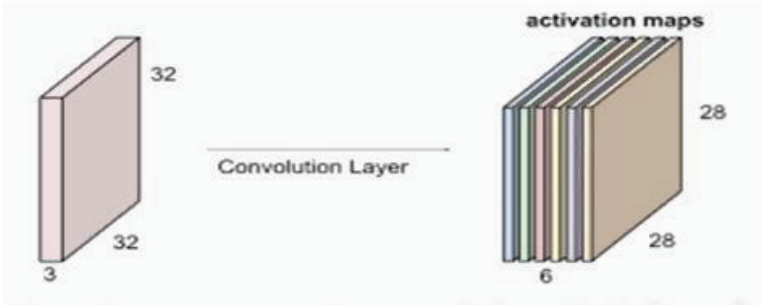
The following diagram shows that one value in the activation map from the 2nd layer has an effective receptive field of 5 x 5 area in input with a kernel of 3 x 3. The *receptive field* of the *neuron* in these layers would be the cross-section of the previous layer from which neurons are providing inputs.



6.12.2. Convolutional Layer

A convolutional layer is usually the very first layer in a convolutional neural network. Its main job is to identify local injunctions from the last layer and map them onto feature maps. Once convolution is applied, the perceptrons are generated. The perceptrons built-in into feature maps of size m2 x m3.

Once input is passed through a convolutional layer, many activation maps are generated, which are further operated on. What are activation maps?



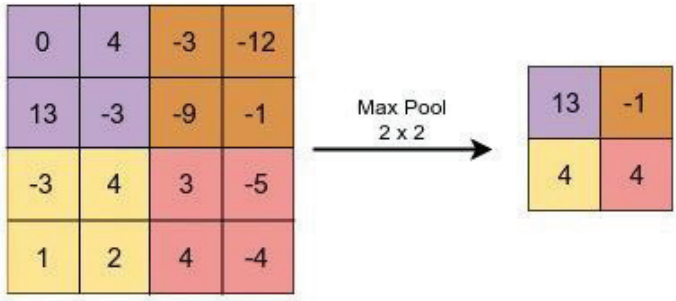
6.12.3. Activation Maps

Activation maps can simply be defined as the outputs of convolutional layers. These are used for the visualization of deep neural networks.

6.13. Pooling Layer

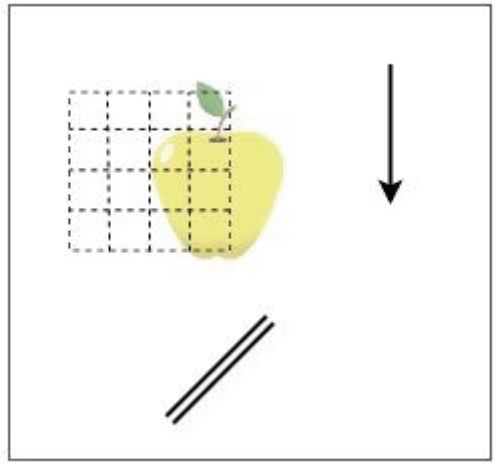
Pooling layers are inserted in between successive convolutional layers. These are inserted periodically. One purpose of the pooling layer is to reduce the number of parameters and computation significantly. Every slice of the input is operated by the pooling layer to resize it spatially. Pooling layers can use max, min, median, or average pooling.

Max pooling, as the name suggests, picks the maximum value from a kernel effective receptive field. Similarly, min pooling chooses the minimum value, median pooling chooses the median value, and average pooling averages all the values. Below is an example that illustrates max pooling further.



The kernel size is 2 x 2, and stride is 2. It means that the kernel looks at a 2 x 2 patch of the image, and stride means it slides two steps for the next operation. If you focus on the purple box on the top left, it contains four values: 0, 4, 13, and -3. Max pooling picks the maximum of these, i.e., 13 and max pools it to the next layer. This is what max pooling is. Similar work is done in other color boxes, too, and a 4 x 4 box-image is reduced to 2 x 2. We can set a bigger kernel, e.g., 4 x 4. Then for this same example, we would only have one value output.

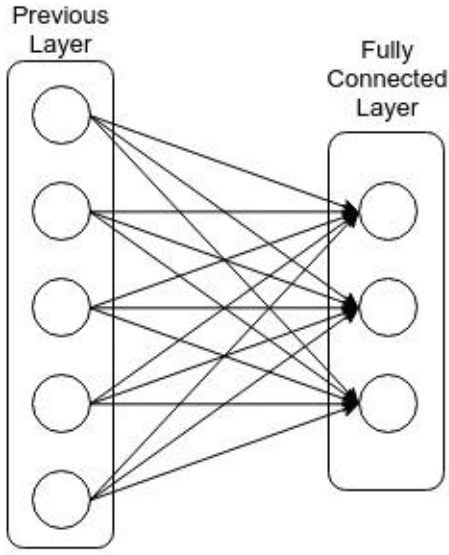
Another purpose or logic behind pooling is to focus on the presence of an object rather than its precise location in the image. For example, consider the following image containing an apple. We want to classify if there is an apple in the image or not. A 4 x 4 patch is shown in dotted black lines. This patch contains a significant part of our object. So, rather than keeping all the pixels inside this patch, we can extract the most important part of it (which can represent the object) and downsize this image.



6.14. Fully Connected Layer

This is the final layer of convolutional neural networks. In a fully connected layer, each and every neuron gets input from all the previous neurons. By doing so, the receptive field becomes larger as compared to the convolution layer, where the receptive field is small because all neurons do not receive input from all previous neurons.

Before a fully connected layer, all the high-level features are detected. A fully connected layer uses the output of the previous layer, i.e., the output of the pooling layer or ReLU, and produces a vector. The vector generated is N-dimensional, and N denotes the number of classes.



6.15. Softmax

Softmax is used in deep learning classification problems. It is a function that takes a vector of n real numbers as an input and returns the probability distribution of n numbers. Softmax assigns decimal probabilities to each class. One of the characteristics of softmax is that these probabilities must add up to 1.0. This simple feature helps in converging the training more quickly.

$$S(x) = \frac{e^{x}}{\sum_{i=1}^{n} e^{x_i}}$$

6.16. Batch Normalization

Batch normalization is used to increase the performance of models by normalizing the output of the previous activation

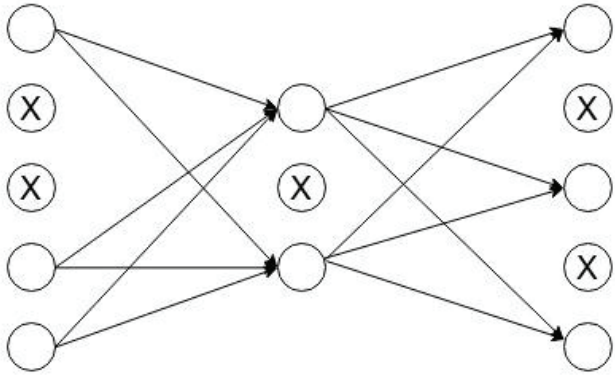
layer. It is done by taking its difference with batch mean and dividing it by batch standard deviation. Batch normalization is used to reduce the internal covariate shift, which is non-uniform distribution in hidden layers. This non-uniform distribution makes the training process very slow. Batch normalization normalizes the inputs mini-batch and uniformly distributes the hidden layers.

$$x^* = \frac{x - E[x]}{\sqrt{var(x)}}$$

Where x* is batch normalized value, E[x] is the mean of the batch, and var(x) is the variance of the batch.

6.17. Dropout

Dropout is used as a regularization technique to avoid overfitting the data. In this technique, neurons are randomly dropped (deactivated) during the training phase. Therefore, we get different architecture of the network in each forward pass, and each neuron tries to learn most features. We end up training multiple subnetworks, so this technique prevents co-adaptation of neurons, and the model can generalize well (perform well on unseen data) as it learns more robust features. An important point to remember is that the dropout is not applied during the testing phase.


X
X
X
X
X

7

Activation Functions

Activation functions calculate the weighted sum of inputs and add a bias value to it. These are used to treat the nonlinear signals. They convert the input signal of a node in a deep neural network to an output signal. This output signal will be used as input in the next layer. These functions are agents that decide which neurons will be pushing the values in the next layer. An activation function uses the sum of products of inputs and their respective weights, and then they apply the activation function to generate the output.

Some of the major activation function include:

1. Sigmoid Function
2. Tanh Function
3. ReLu Function
4. Leaky ReLU Function

7.1. Sigmoid Function

When given an input, the sigmoid function produces an output between 0 and 1. The unique thing about sigmoid function is that it is differentiable continuously. Moreover, it has an affixed output range.

$$Sigmoid = \frac{1}{1 + e^{-x}}$$

7.2. Tanh Function

Tanh function is quite similar to the sigmoid function. It can also be called a modified version of the sigmoid function. The only difference between Tanh and sigmoid function is in their range. In the sigmoid function, the output value range is between 0 and 1, but in Tanh function, it is bound between −1 and 1.

$$Tanh = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

7.3. ReLU Function

ReLU stands for the Rectified linear unit. This is the most commonly used activation function in deep learning networks. It simply returns a 0 whenever it gets a negative input. Apart from this, it returns the same value.

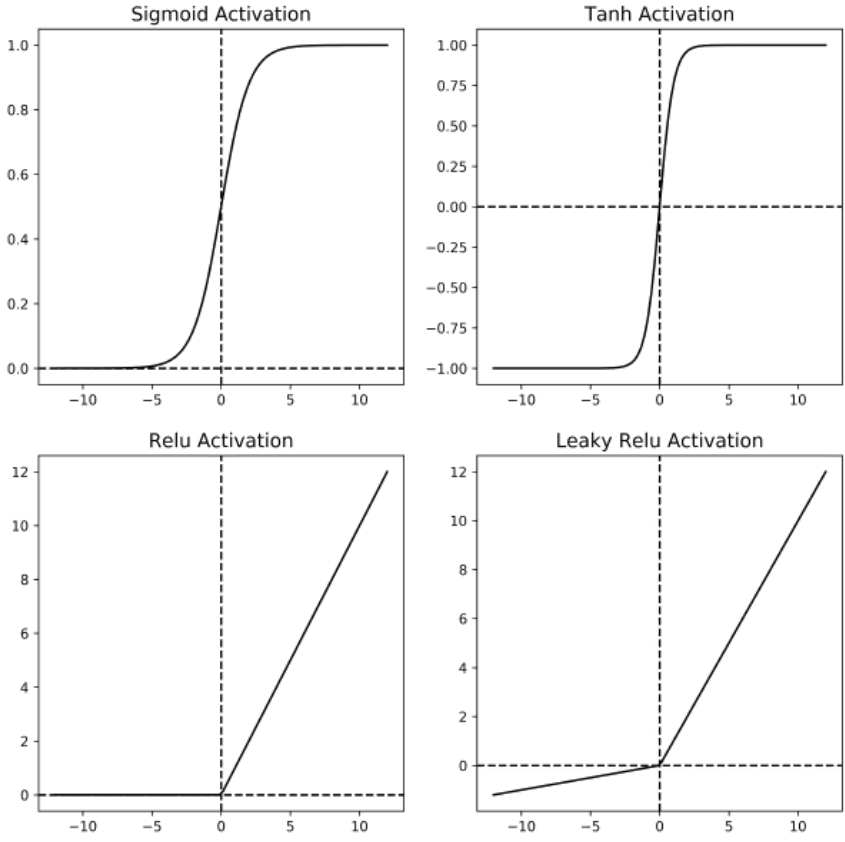
$$ReLU = max(0, x)$$

7.4. Leaky ReLU Function

It is similar to ReLU, but instead of dying for values less than 0, it has a negative slope to allow gradient flow.

$$Leaky\ ReLU = max(\alpha * x, x)$$

Most experiments use $\alpha = 0.01$


Sigmoid Activation
Tanh Activation
Relu Activation
Leaky Relu Activation

Loss Functions

The loss function is a measure of how well the algorithm models the given dataset. In other words, the loss function measures the efficiency of an algorithm.

To understand it in a simpler way, let's suppose that we have built a model to detect persons in an image. If we have an image that shows the faces of 10 people, when this image is fed to the model, it should recognize the persons on the image. For each person that it recognizes, it will be called a hit, and for every person it misses, it would be counted as a miss. In the end, the overall accuracy of the algorithm will depend on the number of hits and the number of misses. The lower the number of misses, the greater will be the accuracy. This is what the loss function does. It counts the losses or misses of an algorithm.

If the model is not good enough, it will miss a lot and hit fewer times; therefore, the loss function will give a higher value. On the contrary, if the loss function generates a low value, it indicates that there are more hits, and the model is efficient. Different models use different loss functions depending upon the complexity or type of problem. The major loss functions are:

Regression Loss Functions
• Mean Squared Loss • Mean Squared Logarithmic Loss • Mean Absolute Loss
Binary Classification Loss Functions
• Binary Cross-Entropy • Hinge Loss • Squared Hinge Loss
Multi-Class Classification Loss Functions
• Multi-Class Cross-Entropy Loss • Sparse Multiclass Cross-Entropy Loss • Kullback Leibler Divergence Loss

For the sake of understanding, we will be describing two of these loss functions: one from regression and one from classification.

8.1. Mean Squared Loss (MSL)

MSL function takes the average of the square of differences between actual and predicted values. The perfect score can be 0. Moreover, the value will always be positive irrespective of the sign of actual and predicted values. MSL is a regression loss function. Below is the mathematical formula for mean squared loss, also known as mean squared error.

$$\frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n}$$

8.2. Hinge Loss

Hinge loss is used for classifiers in training. This is used for maximum margin classification, and it is especially used for support vector machines (SVM). For an input t=1, hinge loss for the prediction y would be as follows:

$$\ell(y) = \max(0, 1 - t \cdot y)$$

9

Deep Learning Optimization Algorithms

As the name suggests, optimization algorithms are used to improve the performance of different algorithms by minimizing the objective function of those algorithms. Optimization algorithms work on the learnable parameters of the model. The X (input) values of the model are used to produce the target Y value.

Optimization algorithms can be of two types:

9.1. First-Order Optimization Algorithms

These are the type of optimizing algorithms which use the gradient values of that algorithm to minimize or maximize the loss function. The most commonly used first-order optimization algorithm is Gradient Descent.

9.2. Second-Order Optimization Algorithms

Second-order optimization algorithms use the second-order derivative, which is also called the Hessian. These algorithms

are not as frequently used as the former because it is very expensive to calculate the second-order derivative.

9.3. Major Optimization Algorithms

Optimization algorithms are wide and varied. The major types of these algorithms are mentioned below:

1. Stochastic Gradient Descent
2. AdaGrad Optimization
3. RMSprop Optimization
4. Adam Optimization
5. AdaBoost Optimization
6. Adadelta Optimization

Which Algorithm to Choose and Why?

The choice of optimization algorithms is often subjective. It depends on the type of problem, the dataset, and the parameters it has to optimize. Before you choose the algorithm, it is important to understand the constraints.

Besides all this, RMSprop is simply an extension of Adagrad, and Adam is the one that combines all the useful features of Adagrad and Gradient Descent. It is well suited in most cases.

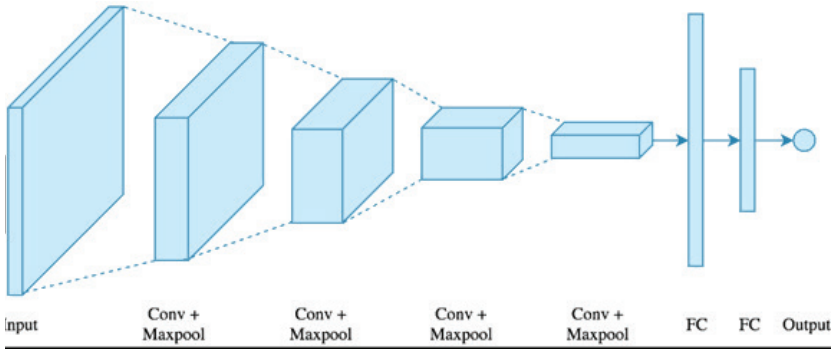
10

Major Architectures of Deep Networks

Since the day machine learning emerged as a separate field, several architectures have been discovered. Their range is varied and wide. Each has its own characteristics. The major ones among them are:

10.1. Convolutional Neural Network

Convolutional Neural Network (CNN) is an artificial neural network that carries out supervised learning for the analysis of data. Data analysis can be used in various cognitive tasks like image processing, object detection, or action detection. It uses several layers that convert the input into output after applying those layers. These layers include the convolutional layer, ReLU, and Max-pooling, and the fully connected layer, which is the last of these. Besides these, there can be many other hidden layers that are involved in the process.



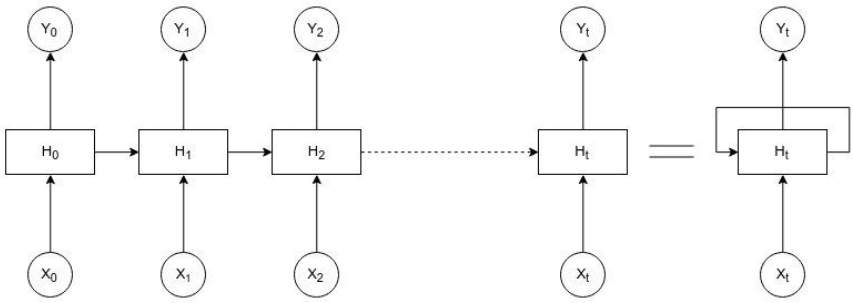
To understand the concept further, we need to understand what the terms convolution and pooling mean. Convolution can simply be defined as a process that takes in the input of two images and produces the output. One image is *the actual image to be transformed*, and the other can be considered as *the filter that is to be applied*. Convolution applies the filter over the actual image and produces the output. It is the same as taking two numbers and doing some operations over them to produce an output.

Pooling can be defined as filtering or selecting an input out of the given data. It can be of two types—max pooling and min pooling. Max pooling, as the name suggests, picks the maximum value, and min pooling chooses the minimum value.

10.2. Recurrent Neural Networks

Recurrent Neural Networks (RNN) is one of the very first model architectures to have emerged in machine learning. It is foundational to many of the architectures that we have today. RNN has a unique characteristic that it will not have all the feed-forward connection; it may have some connections that go into the previous layer or that current layer. This helps

it to maintain the memory. Mostly, RNN is used for speech recognition and handwriting recognition.

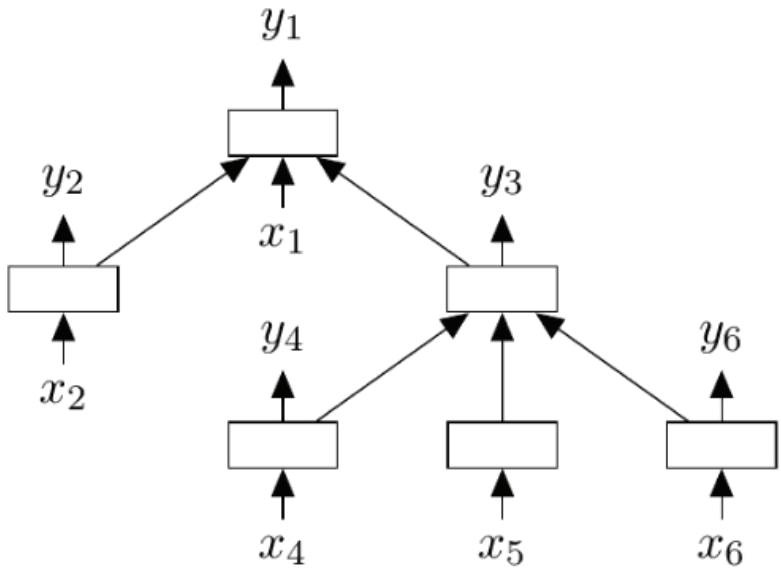


10.3. LSTM

Long Short Term Memory (LSTM) is a special type of Recurrent Neural Network (RNN) designed to learn long term dependencies. The main feature of LSTMs is that these remember information for longer periods of time. Therefore, they are widely used in natural language text compression, handwriting identification, speech recognition, gesture recognition, and image captioning.

10.4. Recursive Neural Networks

Recursive Neural Network uses the same weights recursively on structured input. T uses a tree structure, and it has a fixed number of branches too. In Recursive NN, the node of each parent is identical to its parent. These networks are complex and computationally very expensive. Therefore, they are not very widely used. Moreover, these models use back-propagation (most of the time) as a training method.


y_1
y_2
x_1
y_3
x_2
y_4
y_6
x_4
x_5
x_6

Congrats on completing this book

Hopefully, this book has demystified the notion of deep learning. But that is just the beginning. Now that you are familiar with the logic behind the different deep learning algorithms, understand the role of linear algebra, probability and statistics, and know how to create simple algorithms, it is time to move on. We have more books and courses to reinforce your efforts and guide you at every stage.

Deep learning is a crucial development in today's world. The concepts behind it have been around for more than a decade, but the age of deep learning, AI and related models—such as artificial intelligence, data science, and more—is now. The change is just happening and it is fast.

You have made a great decision to start your journey into the world of data science and deep learning with this book. Today, the knowledge and the ability to use data science and AI is a competitive advantage. Tomorrow, it will be a mere necessity.

Deep learning techniques have already started to change the world of business, by creating a new value for data. The future will be even more exciting. Very soon, most of the devices and apps that we use daily will be fueled by deep learning

algorithms. Many of them already are. Now you have a chance to become a part of this major development. Congrats on your decision and don't forget to check out our other books!

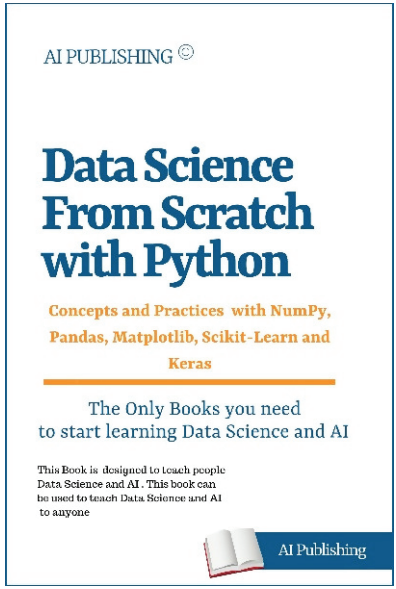
If you want to help us produce more material like this, then please leave an honest review. It really does make a difference.

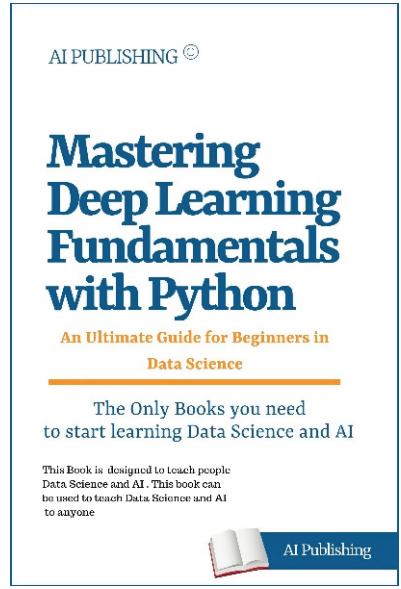
If you have any feedback, please let us know by sending an email to contact@aispublishing.net.

This feedback is highly valued, and we look forward to hearing from you. It will be very helpful for us to improve the quality of our books.

Until next time, happy analyzing.

From the same Publisher





AI PUBLISHING