R22 B.Tech. CSE (DS)

MACHINE LEARNING

UNIT - I

Learning –

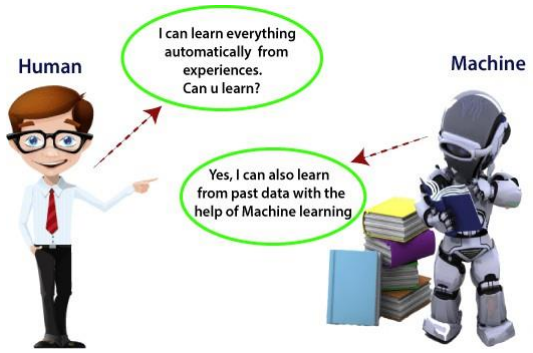
The Machine Learning Tutorial covers both the fundamentals and more complex ideas of machine learning. Students and professionals in the workforce can benefit from our machine learning tutorial.

A rapidly developing field of technology, machine learning allows computers to automatically learn from previous data. For building mathematical models and making predictions based on historical data or information, machine learning employs a variety of algorithms. It is currently being used for a variety of tasks, including speech recognition, email filtering, auto-tagging on Facebook, a recommender system, and image recognition.

You will learn about the many different methods of machine learning, including reinforcement learning, supervised learning, and unsupervised learning, in this machine learning tutorial. Regression and classification models, clustering techniques, hidden Markov models, and various sequential models will all be covered.

What is Machine Learning

In the real world, we are surrounded by humans who can learn everything from their experiences with their learning capability, and we have computers or machines which work on our instructions. But can a machine also learn from experiences or past data like a human does? So here comes the role of **Machine Learning**.



Introduction to Machine Learning

A subset of artificial intelligence known as machine learning focuses primarily on the creation of algorithms that enable a computer to independently learn from data and

previous experiences. Arthur Samuel first used the term "machine learning" in 1959. It could be summarized as follows:

Without being explicitly programmed, machine learning enables a machine to automatically learn from data, improve performance from experiences, and predict things.

Machine learning algorithms create a mathematical model that, without being explicitly programmed, aids in making predictions or decisions with the assistance of sample historical data, or training data. For the purpose of developing predictive models, machine learning brings together statistics and computer science. Algorithms that learn from historical data are either constructed or utilized in machine learning. The performance will rise in proportion to the quantity of information we provide.

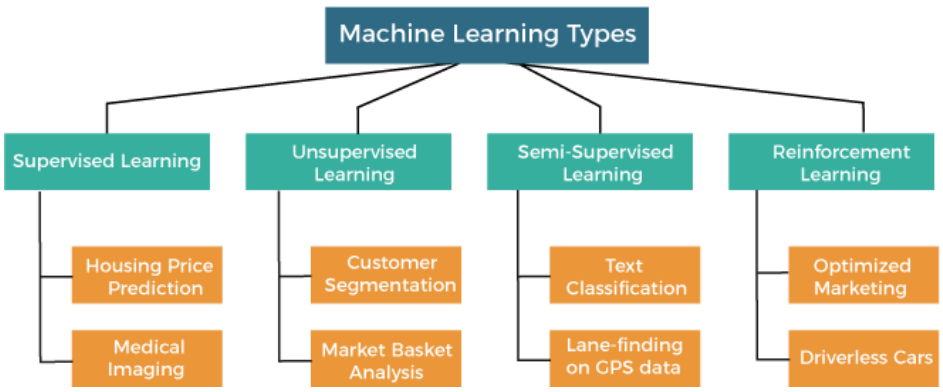
Types of Machine Learning

Machine learning is a subset of AI, which enables the machine to automatically learn from data, improve performance from past experiences, and make predictions. Machine learning contains a set of algorithms that work on a huge amount of data. Data is fed to these algorithms to train them, and on the basis of training, they build the model & perform a specific task.

These ML algorithms help to solve different business problems like Regression, Classification, Forecasting, Clustering, and Associations, etc.

Based on the methods and way of learning, machine learning is divided into mainly four types, which are:

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Semi-Supervised Machine Learning
4. Reinforcement Learning



. Supervised Machine Learning

As its name suggests, Supervised machine learning is based on supervision. It means in the supervised learning technique, we train the machines using the "labelled" dataset, and based on the training, the machine predicts the output. Here, the labelled data specifies that some of the inputs are already mapped to the output. More preciously, we can say; first, we train the machine with the input and corresponding output, and then we ask the machine to predict the output using the test dataset.

Let's understand supervised learning with an example. Suppose we have an input dataset of cats and dog images. So, first, we will provide the training to the machine to understand the images, such as the **shape & size of the tail of cat and dog, Shape of eyes, colour, height (dogs are taller, cats are smaller), etc.** After completion of training, we input the picture of a cat and ask the machine to identify the object and predict the output. Now, the machine is well trained, so it will check all the features of the object, such as height, shape, colour, eyes, ears, tail, etc., and find that it's a cat. So, it will put it in the Cat category. This is the process of how the machine identifies the objects in Supervised Learning.

The main goal of the supervised learning technique is to map the input variable(x) with the output variable(y). Some real-world applications of supervised learning are **Risk Assessment, Fraud Detection, Spam filtering,** etc.

- **Classification**
- **Regression**

a) Classification

Classification algorithms are used to solve the classification problems in which the output variable is categorical, such as **"Yes" or No, Male or Female, Red or Blue, etc**. The classification algorithms predict the categories present in the dataset. Some real-world examples of classification algorithms are **Spam Detection, Email filtering, etc.**

Some popular classification algorithms are given below:

- **Random Forest Algorithm**
- **Decision Tree Algorithm**
- **Logistic Regression Algorithm**
- **Support Vector Machine Algorithm**

b) Regression

Regression algorithms are used to solve regression problems in which there is a linear relationship between input and output variables. These are used to predict continuous output variables, such as market trends, weather prediction, etc.

Some popular Regression algorithms are given below:

- **Simple Linear Regression Algorithm**
- **Multivariate Regression Algorithm**
- **Decision Tree Algorithm**
- **Lasso Regression**

Advantages and Disadvantages of Supervised Learning

Advantages:

- Since supervised learning work with the labelled dataset so we can have an exact idea about the classes of objects.
- These algorithms are helpful in predicting the output on the basis of prior experience.

Disadvantages:

- These algorithms are not able to solve complex tasks.
- It may predict the wrong output if the test data is different from the training data.
- It requires lots of computational time to train the algorithm.

Unsupervised Machine Learning

Unsupervised learning is different from the Supervised learning technique; as its name suggests, there is no need for supervision. It means, in unsupervised machine learning, the machine is trained using the unlabeled dataset, and the machine predicts the output without any supervision.

In unsupervised learning, the models are trained with the data that is neither classified nor labelled, and the model acts on that data without any supervision.

The main aim of the unsupervised learning algorithm is to group or categories the unsorted dataset according to the similarities, patterns, and differences. Machines are instructed to find the hidden patterns from the input dataset.

Let's take an example to understand it more preciously; suppose there is a basket of fruit images, and we input it into the machine learning model. The images are totally unknown to the model, and the task of the machine is to find the patterns and categories of the objects.

So, now the machine will discover its patterns and differences, such as colour difference, shape difference, and predict the output when it is tested with the test dataset.

Categories of Unsupervised Machine Learning

Unsupervised Learning can be further classified into two types, which are given below:

- **Clustering**
- **Association**

1) Clustering

The clustering technique is used when we want to find the inherent groups from the data. It is a way to group the objects into a cluster such that the objects with the most similarities remain in one group and have fewer or no similarities with the objects of other groups. An example of the clustering algorithm is grouping the customers by their purchasing behaviour.

Some of the popular clustering algorithms are given below:

- **K-Means Clustering algorithm**
- **Mean-shift algorithm**
- **DBSCAN Algorithm**
- **Principal Component Analysis**
- **Independent Component Analysis**

2) Association

Association rule learning is an unsupervised learning technique, which finds interesting relations among variables within a large dataset. The main aim of this learning algorithm is to find the dependency of one data item on another data item and map those variables accordingly so that it can generate maximum profit. This algorithm is mainly applied in **Market Basket analysis, Web usage mining, continuous production**, etc.

Some popular algorithms of Association rule learning are **Apriori Algorithm, Eclat, FP-growth algorithm.**

Advantages and Disadvantages of Unsupervised Learning Algorithm

Advantages:

- These algorithms can be used for complicated tasks compared to the supervised ones because these algorithms work on the unlabeled dataset.
- Unsupervised algorithms are preferable for various tasks as getting the unlabeled dataset is easier as compared to the labelled dataset.

Disadvantages:

- The output of an unsupervised algorithm can be less accurate as the dataset is not labelled, and algorithms are not trained with the exact output in prior.
- Working with Unsupervised learning is more difficult as it works with the unlabelled dataset that does not map with the output.

Applications of Unsupervised Learning

- **Network Analysis:** Unsupervised learning is used for identifying plagiarism and copyright in document network analysis of text data for scholarly articles.
- **Recommendation Systems:** Recommendation systems widely use unsupervised learning techniques for building recommendation applications for different web applications and e-commerce websites.
- **Anomaly Detection:** Anomaly detection is a popular application of unsupervised learning, which can identify unusual data points within the dataset. It is used to discover fraudulent transactions.
- **Singular Value Decomposition:** Singular Value Decomposition or SVD is used to extract particular information from the database. For example, extracting information of each user located at a particular location.

Semi-Supervised Learning

Semi-Supervised learning is a type of Machine Learning algorithm that lies between Supervised and Unsupervised machine learning. It represents the intermediate ground between Supervised (With Labelled training data) and

Unsupervised learning (with no labelled training data) algorithms and uses the combination of labelled and unlabeled datasets during the training period.
Although Semi-supervised learning is the middle ground between supervised and unsupervised learning and operates on the data that consists of a few labels, it mostly consists of unlabeled data. As labels are costly, but for corporate purposes, they may have few labels. It is completely different from supervised and unsupervised learning as they are based on the presence & absence of labels.

To overcome the drawbacks of supervised learning and unsupervised learning algorithms, the concept of Semi-supervised learning is introduced. The main aim of semi-supervised learning is to effectively use all the available data, rather than only labelled data like in supervised learning. Initially, similar data is clustered along with an unsupervised learning algorithm, and further, it helps to label the unlabeled data into labelled data. It is because labelled data is a comparatively more expensive acquisition than unlabeled data.

We can imagine these algorithms with an example. Supervised learning is where a student is under the supervision of an instructor at home and college. Further, if that student is self-analysing the same concept without any help from the instructor, it comes under unsupervised learning. Under semi-supervised learning, the student has to revise himself after analyzing the same concept under the guidance of an instructor at college.

Advantages and disadvantages of Semi-supervised Learning

Advantages:

- It is simple and easy to understand the algorithm.
- It is highly efficient.
- It is used to solve drawbacks of Supervised and Unsupervised Learning algorithms.

Disadvantages:

- Iterations results may not be stable.
- We cannot apply these algorithms to network-level data.
- Accuracy is low.

4. Reinforcement Learning

Reinforcement learning works on a feedback-based process, in which an AI agent (A software component) automatically explore its surrounding by hitting & trail, taking action, learning from experiences, and improving its performance. Agent gets rewarded for each good action and get punished for each bad action; hence the goal of reinforcement learning agent is to maximize the rewards.

In reinforcement learning, there is no labelled data like supervised learning, and agents learn from their experiences only.

The reinforcement learning process is similar to a human being; for example, a child learns various things by experiences in his day-to-day life. An example of reinforcement learning is to play a game, where the Game is the environment, moves of an agent at each step define states, and the goal of the agent is to get a high score. Agent receives feedback in terms of punishment and rewards.

Due to its way of working, reinforcement learning is employed in different fields such as **Game theory, Operation Research, Information theory, multi-agent systems.**

A reinforcement learning problem can be formalized using **Markov Decision Process(MDP).** In MDP, the agent constantly interacts with the environment and performs actions; at each action, the environment responds and generates a new state.

Categories of Reinforcement Learning

Reinforcement learning is categorized mainly into two types of methods/algorithms:

- **Positive Reinforcement Learning:** Positive reinforcement learning specifies increasing the tendency that the required behaviour would occur again by adding something. It enhances the strength of the behaviour of the agent and positively impacts it.
- **Negative Reinforcement Learning:** Negative reinforcement learning works exactly opposite to the positive RL. It increases the tendency that the specific behaviour would occur again by avoiding the negative condition.

Real-world Use cases of Reinforcement Learning

- **Video Games:**
  RL algorithms are much popular in gaming applications. It is used to gain super-human performance. Some popular games that use RL algorithms are **AlphaGO** and **AlphaGO Zero**.

- **Resource Management:**
  The "Resource Management with Deep Reinforcement Learning" paper showed that how to use RL in computer to automatically learn and schedule resources to wait for different jobs in order to minimize average job slowdown.
- **Robotics:**
  RL is widely being used in Robotics applications. Robots are used in the industrial and manufacturing area, and these robots are made more powerful with reinforcement learning. There are different industries that have their vision of building intelligent robots using AI and Machine learning technology.
- **Text Mining**
  Text-mining, one of the great applications of NLP, is now being implemented with the help of Reinforcement Learning by Salesforce company.

Advantages and Disadvantages of Reinforcement Learning

Advantages

- It helps in solving complex real-world problems which are difficult to be solved by general techniques.
- The learning model of RL is similar to the learning of human beings; hence most accurate results can be found.
- Helps in achieving long term results.

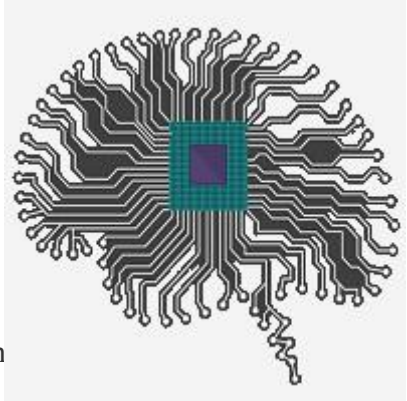
Disadvantage

- RL algorithms are not preferred for simple problems.
- RL algorithms require huge data and computations.
- Too much reinforcement learning can lead to an overload of states which can weaken the results.

The Brain and the Neuron

THE BRAIN AND THE NEURON In animals, learning occurs within the brain. If we can understand how the brain works, then there might be things in there for us to copy and use for our machine learning systems. While the brain is an impressively powerful and complicated system, the basic building
blocks that it is made up of are fairly simple and easy to understand. We'll look at them shortly, but it's worth noting that in computational terms the brain does exactly what we want. It deals with noisy and even inconsistent data, and produces answers that are usually correct from very high
dimensional data (such as images) very quickly. All amazing for something that weighs about 1.5 kg and is losing parts of itself all the time (neurons die as you age at impressive/depressing rates), but

Artificial Neural Network Tutorial provides basic and advanced concepts of ANNs. Our Artificial Neural Network tutorial is developed for beginners as well as professions.



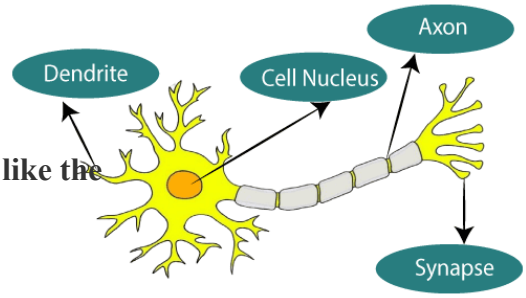
The term "Artificial neural network" refers to a biologically inspired sub-field of artificial intelligence modeled after the brain. An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain. Similar to a human brain has neurons intercon neural networks also have neurons that are linked to each networks. These neurons are known as nodes.

Artificial neural network tutorial covers all the aspects related to the artificial neural network. In this tutorial, we will discuss ANNs, Adaptive resonance theory, Kohonen self-organizing map, Building blocks, unsupervised learning, Genetic algorithm, etc.

What is Artificial Neural Network?

The term "**Artificial Neural Network**" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.

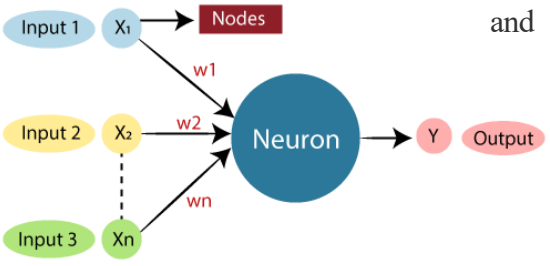
The given figure illustrates the typical diagram of Biological Neural Network.



The typical Artificial Neural Network looks something like the given figure.

Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

Relationship between Biological neural network and artificial neural network:



Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

An **Artificial Neural Network** in the field of **Artificial intelligence** where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.

There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.

We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate, which takes two inputs. If one or both the inputs are "On," then we get "On" in output. If both the inputs are "Off," then we get "Off" in output. Here the output depends upon input. Our brain does not perform the same task. The outputs to inputs relationship keep changing because of the neurons in our brain, which are "learning."

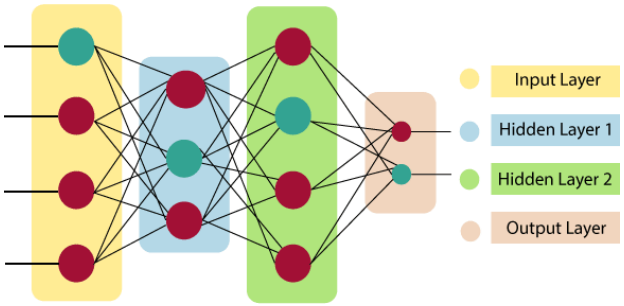
The architecture of an artificial neural network:

To understand the concept of the architecture of an artificial neural network, we have to understand what a neural network consists of. In order to define a neural network that consists of a large number of artificial neurons, which are termed units arranged in a sequence of layers. Lets us look at various types of layers available in an artificial neural network.

Artificial Neural Network primarily consists of three layers:

Input Layer:

As the name suggests, it accepts inputs in several different formats provided by the programmer.



Hidden Layer:

The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

Output Layer:

The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

It determines weighted total $\sum_{i=1}^{n} Wi * Xi + b$ is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.

Advantages of Artificial Neural Network (ANN)

Parallel processing capability:

Artificial neural networks have a numerical value that can perform more than one task simultaneously.

Storing data on the entire network:

Data that is used in traditional programming is stored on the whole network, not on a database. The disappearance of a couple of pieces of data in one place doesn't prevent the network from working.

Capability to work with incomplete knowledge:

After ANN training, the information may produce output even with inadequate data. The loss of performance here relies upon the significance of missing data.

Having a memory distribution:

For ANN is to be able to adapt, it is important to determine the examples and to encourage the network according to the desired output by demonstrating these examples to the network. The succession of the network is directly proportional to the chosen instances, and if the event can't appear to the network in all its aspects, it can produce false output.

Having fault tolerance:

Extortion of one or more cells of ANN does not prohibit it from generating output, and this feature makes the network fault-tolerance.

Disadvantages of Artificial Neural Network:

Assurance of proper network structure:

There is no particular guideline for determining the structure of artificial neural networks. The appropriate network structure is accomplished through experience, trial, and error.

Unrecognized behavior of the network:

It is the most significant issue of ANN. When ANN produces a testing solution, it does not provide insight concerning why and how. It decreases trust in the network.

Hardware dependence:

Artificial neural networks need processors with parallel processing power, as per their structure. Therefore, the realization of the equipment is dependent.

Difficulty of showing the issue to the network:

ANNs can work with numerical data. Problems must be converted into numerical values before being introduced to ANN. The presentation mechanism to be resolved here will directly impact the performance of the network. It relies on the user's abilities.

The duration of the network is unknown:

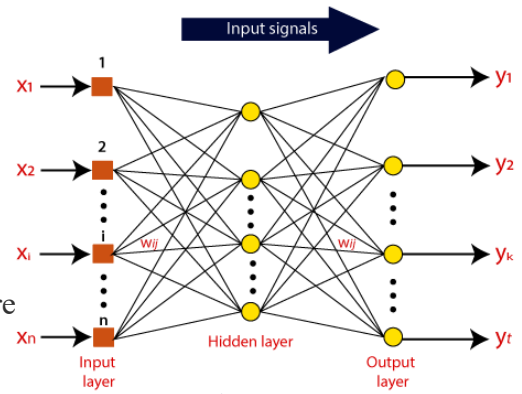
The network is reduced to a specific value of the error, and this value does not give us optimum results.

Science artificial neural networks that have steeped into the world in the mid-20th century are exponentially developing. In the present time, we have investigated the pros of artificial neural networks and the issues encountered in the course of their utilization. It should not be overlooked that the cons of ANN networks, which are a flourishing science branch, are eliminated individually, and their pros are increasing day by day. It means that artificial neural networks will turn into an irreplaceable part of our lives progressively important.

How do artificial neural networks work?

Artificial Neural Network can be best represented as a weighted directed graph, where the artificial neurons form the nodes. The association between the neurons outputs and neuron inputs can be viewed as the directed edges with weights. The Artificial Neural Network receives the input signal from the external source in the form of a pattern and image in the form of a vector. These inputs are then mathematically assigned by the notations x(n) for every n number of inputs.

Afterward, each of the input is multiplied by its corresponding weights (these weights are the details utilized by the artificial neural networks to solve a specific problem). In general terms, these weights normally represent the strength of the interconnection between neurons inside the artificial neural network. All the weighted inputs are summarized inside the computing unit.



If the weighted sum is equal to zero, then bias is added to make the output non-zero or something else to scale up to the system's response. Bias has the same input, and weight equals to 1. Here the total of weighted inputs can be in the range of 0 to positive infinity. Here, to keep the response in the limits of the desired value, a certain maximum value is benchmarked, and the total of weighted inputs is passed through the activation function.

Choosing the Training Experience:

The first design choice we face is to choose the type of training experience from which our system will learn. The type of training experience available can have a significant impact on success or failure of the learner. One key attribute is whether the training experience provides direct or indirect feedback regarding the choices made by the performance system. For example, in learning to play checkers, the system might learn from direct training examples consisting of individual checkers board states and the correct move for each.

In order to complete the design of the learning system, we must now choose

1. the exact type of knowledge to be learned

2. a representation for this target knowledge

3. a learning mechanism

Choosing the Target Function:

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program. Let us begin with a checkers-playing program that can generate the legal moves from any board state. The program needs only to learn how to choose the best move from among these legal moves.

Choosing a Representation for the Target Function:

X1: the number of black pieces on the

board x2: the number of red pieces on

the board x3: the number of black kings

on the board x4: the number of red kings

on the board

x5: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)

X6: the number of red pieces threatened by black

Thus, our learning program will represent V(b) as a linear function of the form

$$V(b)=w0+w1x1+w2x2+w3x3+w4x4+w5x5+w6x6$$

where wo through w6 are numerical coefficients, or weights, to be chosen by the learning algorithm.

Partial design of a checkers learning program:

Task T: playing checkers

Performance measure P: percent of games won in the world tournament

Training experience E: games played against itself

Target function: V:Board□R

Target function representation

V(b)=w0+w1x1+w2x2+w3x3+w4x4+w5x5+w6x6

Choosing a Function Approximation Algorithm

In order to learn the target function f we require a set of training examples, each describing a specific board state b and the training value Vtrain(b) for b.

ESTIMATING TRAINING VALUES

Rule for estimating training values.

Vtrain (b) □V(Successor(b))

ADJUSTING THE WEIGHTS

$$E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \ training\ \ examples} (V_{train}(b) - \hat{V}(b))^2$$

The Final Design:

The Performance System is the module that must solve the given per- formance task, in this case playing checkers, by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.

The **Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function. As shown in the diagram, each training example in this case corresponds to some game state in the trace, along with an estimate Vtrain, of the target function value for this example.

The **Generalizer** takes as input the training examples and produces an output hypothesis that is its estimate of the target function. It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.

The **Experiment Generator** takes as input the current hypothesis (currently learned
function) and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.

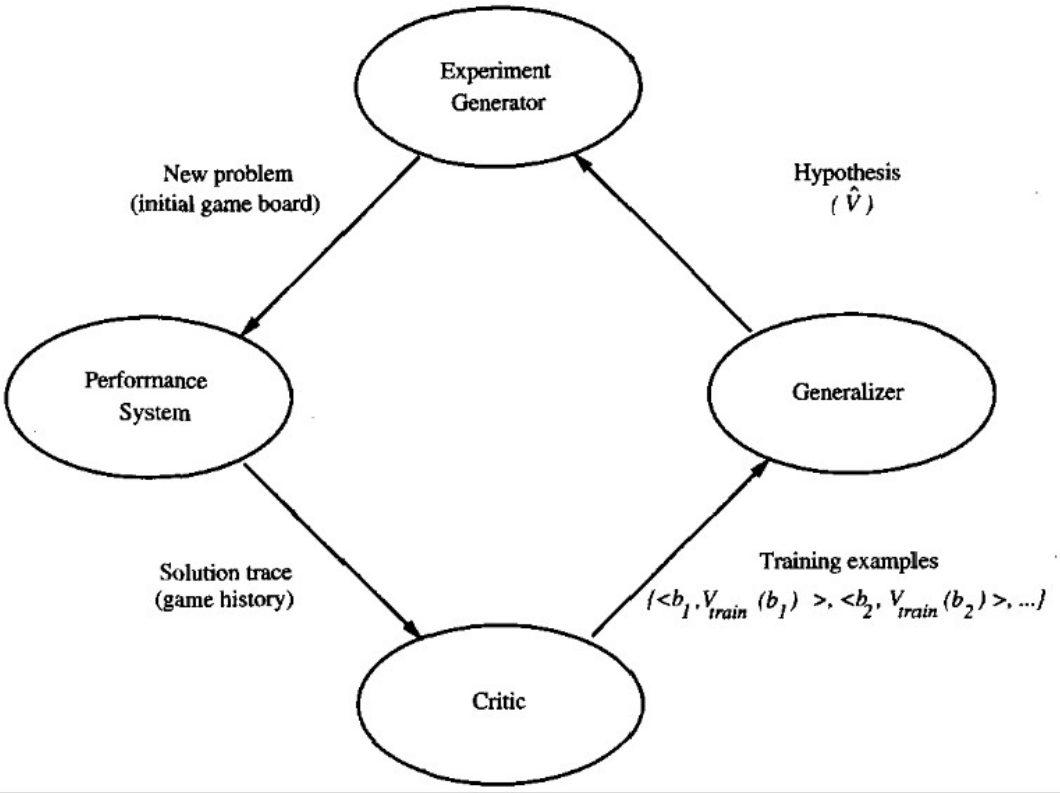


Fig: Final Design of Checkers Learning Problem

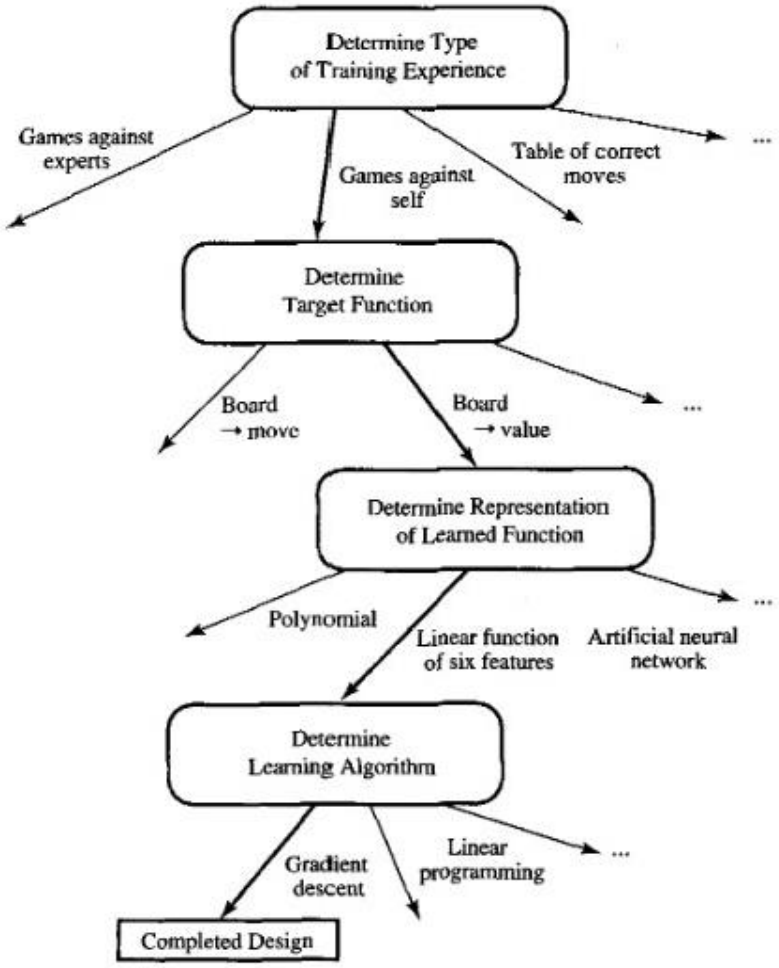


Fig: Summary of choices in designing checkers learning problem.

PERSPECTIVES AND ISSUES IN MACHINE LEARNING:

One useful perspective on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge held by the learner. For example, consider the space of hypotheses that could in principle be output by the above checkers learner. This hypothesis space consists of all evaluation functions that can be represented by some choice of values for the weights wo through w6.

The learner's task is thus to search through this vast space to locate the hypothesis that is most consistent with the available training examples. The LMS algorithm for fitting weights achieves this goal by iteratively tuning the weights, adding a correction to each weight each time the hypothesized evaluation function predicts a value that differs

from the training value.

This algorithm works well when the hypothesis representation considered by the learner defines a continuously parameterized space of potential hypotheses.

Issues in Machine Learning

- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?
- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?
- What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

Concept Learning:

Concept learning: Inferring a boolean-valued function from training examples of its input and output.

A CONCEPT LEARNING TASK:

What hypothesis representation shall we provide to the learner in this case? Let us begin by considering a simple representation in which each hypothesis consists of a conjunction of constraints on the instance attributes. In particular, let each hypothesis be a vector of six constraints, specifying the values of the six attributes Sky, AirTemp, Humidity, Wind, Water, and Forecast. For each attribute, the hypothesis will either

indicate by a "?' that any value is acceptable for this attribute,

specify a single required value (e.g., Warm) for the attribute, or

indicate by a "**θ**" that no value is acceptable.

The inductive learning hypothesis. Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

CONCEPT LEARNING AS SEARCH Concept learning can be viewed as the task of
searching through a large space of hypotheses implicitly defined by the hypothesis representation. The goal of this search is to find the hypothesis that best fits the training examples. It is important to note that by selecting a hypothesis representation, the designer of the learning algorithm implicitly defines the space of all hypotheses that the program can ever represent and therefore can ever learn.

General-to-Specific Ordering of Hypotheses Many algorithms for concept learning
organize the search through the hypothesis space by relying on a very useful structure that exists for any concept learning problem: a general-to-specific ordering of hypotheses. By taking advantage of this naturally occurring structure over the hypothesis space, we can design learning algorithms that exhaustively search even infinite hypothesis spaces without explicitly enumerating every hypothesis.

To illustrate the general-to-specific ordering, consider the two hypotheses

h1 = (Sunny, ?, ?, Strong, ?, ?)

h2 = (Sunny, ?, ?, ?, ?, ?)

Now consider the sets of instances that are classified positive by hl and by h2. Because h2 imposes fewer constraints on the instance, it classifies more instances as positive. In fact, any instance classified positive by hl will also be classified positive by h2. Therefore, we say that h2 is more general than hl.

Definition: Let hj and hk be boolean-valued functions defined over X. Then hj is more general-than-or-equal-to hk (written hj >= hk) if and only if

$$(\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS:

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
   - For each attribute constraint a_i in h
     If the constraint a_i is satisfied by x
     Then do nothing
     Else replace a_i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Fig: Find-S Algorithm

VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM:

The CANDIDATE-ELIMINATION algorithm finds all describable hypotheses that are consistent with the observed training examples. In order to define this algorithm precisely, we begin with a few basic definitions. First, let us say that a hypothesis is consistent with the training examples if it correctly classifies these examples.

Definition: A hypothesis h is **consistent** with a set of training examples D if and only if $h(x) = c(x)$ for each example $\langle x, c(x)\rangle$ in D.

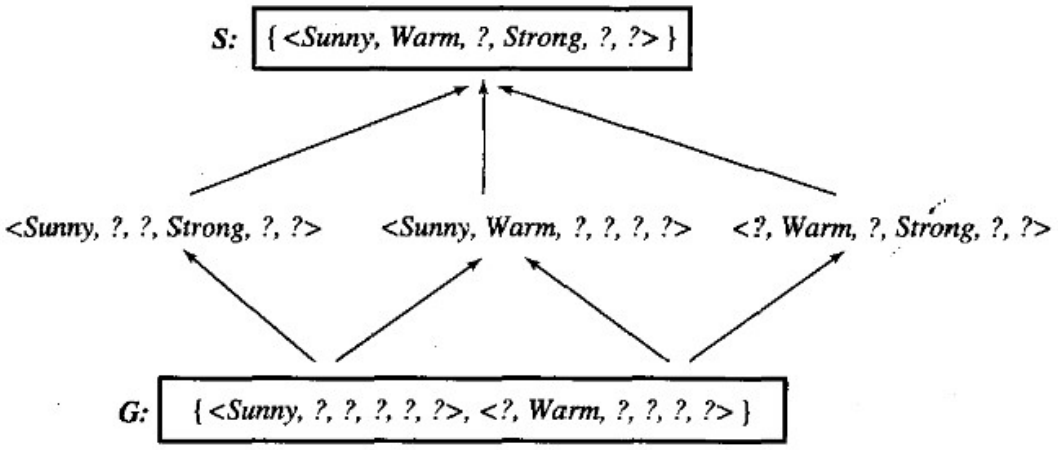
$$Consistent\,(h, D) \equiv (\forall \langle x, c(x)\rangle \in D)\ h(x) = c(x)$$

Definition: The **version space**, denoted $VS_{H,D}$, with respect to hypothesis space H and training examples D, is the subset of hypotheses from H consistent with the training examples in D.

$$VS_{H,D} \equiv \{h \in H | Consistent(h, D)\}$$

The LIST-THEN-ELIMINATE Algorithm

1. $VersionSpace \leftarrow$ a list containing every hypothesis in H
2. For each training example, $\langle x, c(x)\rangle$
   remove from $VersionSpace$ any hypothesis h for which $h(x) \neq c(x)$
3. Output the list of hypotheses in $VersionSpace$



Definition: The **general boundary** G, with respect to hypothesis space H and training data D, is the set of maximally general members of H consistent with D.

$$G \equiv \{g \in H | Consistent(g, D) \wedge (\neg \exists g' \in H)[(g' >_g g) \wedge Consistent(g', D)]\}$$

Definition: The **specific boundary** S, with respect to hypothesis space H and training data D, is the set of minimally general (i.e., maximally specific) members of H consistent with D.

$$S \equiv \{s \in H | Consistent(s, D) \wedge (\neg \exists s' \in H)[(s >_g s') \wedge Consistent(s', D)]\}$$

CANDIDATE-ELIMINATION Learning Algorithm:

Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For each training example d, do

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
      - h is consistent with d, and some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that
      - h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

REMARKS ON VERSION SPACES AND CANDIDATE-ELIMINATION ALGORITHM:

Will the CANDIDATE-ELIMINATION Algorithm Converge to the Correct Hypothesis?

The version space learned by the CANDIDATE-ELIMINATION algorithm will converge toward the hypothesis that correctly describes the target concept, provided (1) there are no errors in the training examples, and (2) there is some hypothesis in H that correctly describes the target concept.

What Training Example Should the Learner Request Next? Up to this point we have assumed that training examples are provided to the learner by some external teacher. Suppose instead that the learner is allowed to conduct experiments in which it chooses the next instance, then obtains the correct classification for this instance from an external oracle (e.g., nature or a teacher).

How Can Partially Learned Concepts Be Used? Suppose that no addit onal training

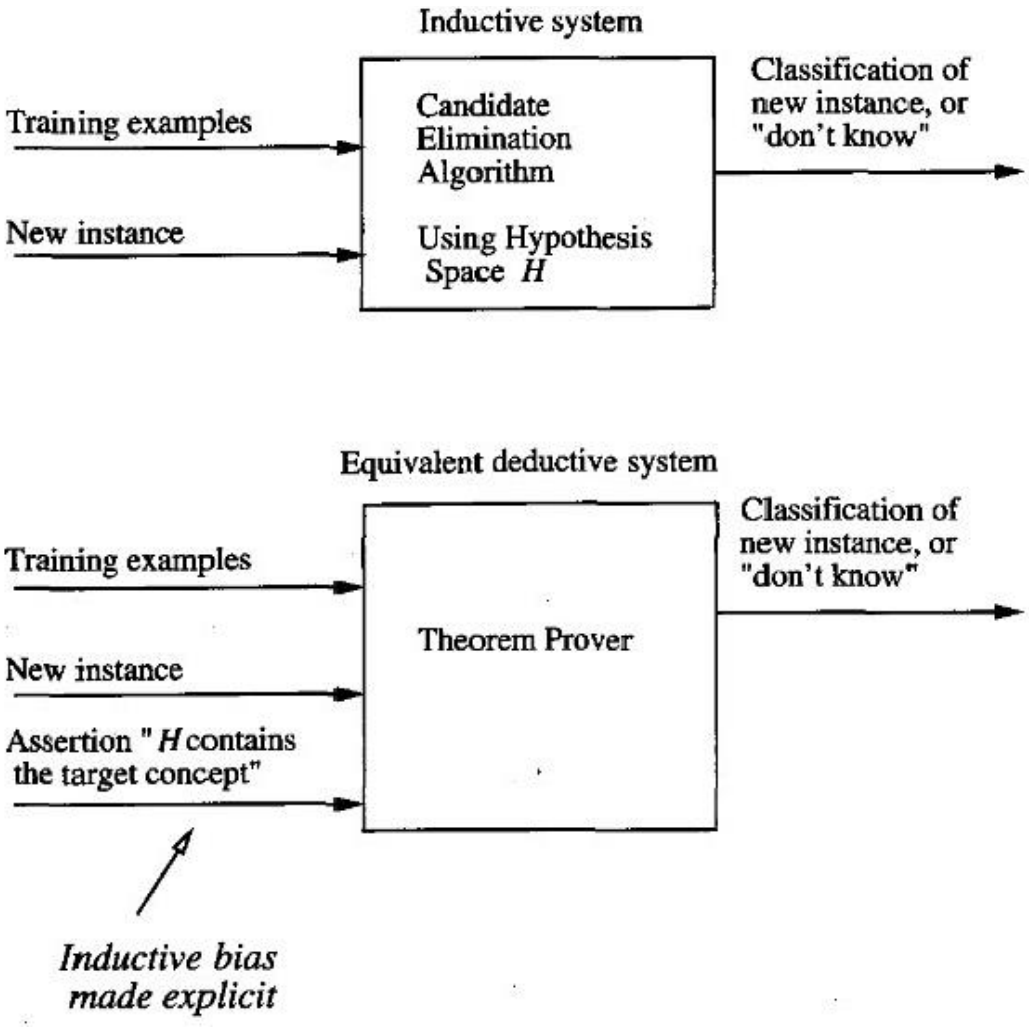
examples are available beyond the four in our example above, but that the learner is now required to classify new instances that it has not yet observed. Even though the version space still contains multiple hypotheses, indicating that the target concept has not yet been fully learned, it is possible to classify certain examples with the same degree of confidence as if the target concept had been uniquely identified.

INDUCTIVE BIAS:

Definition: Consider a concept learning algorithm L for the set of instances X. Let c be an arbitrary concept defined over X, and let $D_c = \{\langle x, c(x)\rangle\}$ be an arbitrary set of training examples of c. Let $L(x_i, D_c)$ denote the classification assigned to the instance x_i by L after training on the data D_c. The **inductive bias** of L is any minimal set of assertions B such that for any target concept c and corresponding training examples D_c

$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)]$$

Inductive bias of CANDIDATE-ELIMINATION algorithm. The concept c is target contained in the given hypothesis space H.


Inductive system
Training examples
New instance
Candidate Elimination Algorithm
Using Hypothesis Space H
Classification of new instance, or "don't know"
Equivalent deductive system
Training examples
New instance
Assertion "H contains the target concept"
Theorem Prover
Classification of new instance, or "don't know"
Inductive bias made explicit

Linear Discriminant Analysis (LDA) in Machine Learning

Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is also known as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA).

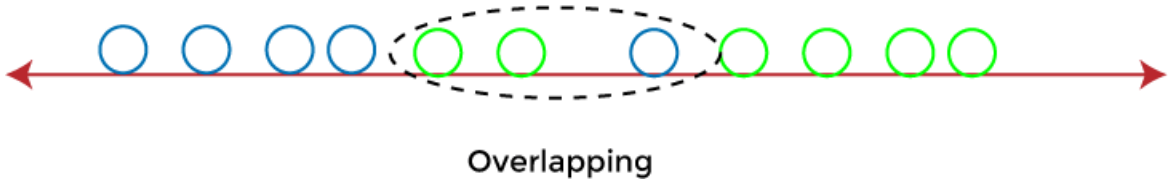
This can be used to project the features of higher dimensional space into lower-dimensional space in order to reduce resources and dimensional costs. In this topic, "**Linear Discriminant Analysis (LDA) in machine learning"**, we will discuss the LDA algorithm for classification predictive modeling problems, limitation of logistic regression, representation of linear Discriminant analysis model, how to make a prediction using LDA, how to prepare data for LDA, extensions to LDA and much more. So, let's start with a quick introduction to Linear Discriminant Analysis (LDA) in machine learning.

What is Linear Discriminant Analysis (LDA)?

Although the logistic regression algorithm is limited to only two-class, linear Discriminant analysis is applicable for more than two classes of classification problems.

Linear Discriminant analysis is one of the most popular dimensionality reduction techniques used for supervised classification problems in machine learning. It is also considered a pre-processing step for modeling differences in ML and applications of pattern classification.

Whenever there is a requirement to separate two or more classes having multiple features efficiently, the Linear Discriminant Analysis model is considered the most common technique to solve such classification problems. For e.g., if we have two classes with multiple features and need to separate them efficiently. When we classify them using a single feature, then it may show overlapping.

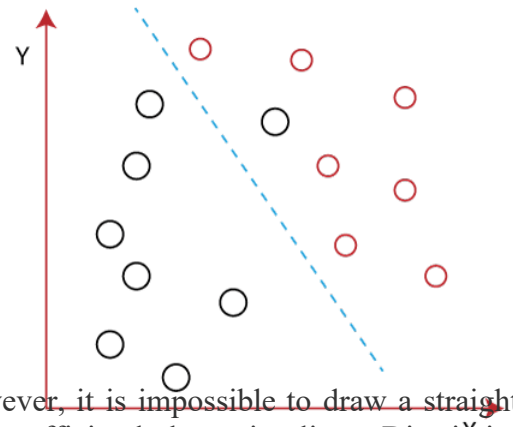


Overlapping

To overcome the overlapping issue in the classification process, we must increase the number of features regularly.

Example:

Let's assume we have to classify two different classes having two sets of data points in a 2-dimensional plane as shown below image:



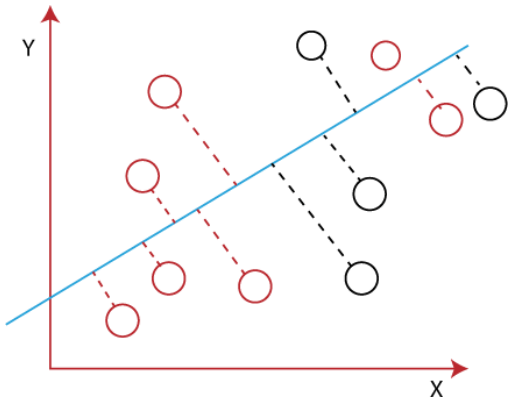
However, it is impossible to draw a straight line in a 2-d plane that can separate these data points efficiently but using linear Discriminant analysis; we can dimensionally reduce the 2-D plane into the 1-D plane. Using this technique, we can also maximize the separability between multiple classes.

How Linear Discriminant Analysis (LDA) works?

Linear Discriminant analysis is used as a dimensionality reduction technique in machine learning, using which we can easily transform a 2-D and 3-D graph into a 1-dimensional plane.

Let's consider an example where we have two classes in a 2-D plane having an X-Y axis, and we need to classify them efficiently. As we have already seen in the above example that LDA enables us to draw a straight line that can completely separate the two classes of the data points. Here, LDA uses an X-Y axis to create a new axis by separating them using a straight line and projecting data onto a new axis.

Hence, we can maximize the separation between these classes and reduce the 2-D plane into 1-D.



To create a new axis, Linear Discriminant Analysis uses the following criteria:

- It maximizes the distance between means of two classes.
- It minimizes the variance within the individual class.

Using the above two conditions, LDA generates a new axis in such a way that it can maximize the distance between the means of the two classes and minimizes the variation within each class.

In other words, we can say that the new axis will increase the separation between the data points of the two classes and plot them onto the new axis.

Why LDA?

- Logistic Regression is one of the most popular classification algorithms that perform well for binary classification but falls short in the case of multiple classification problems with well-separated classes. At the same time, LDA handles these quite efficiently.
- LDA can also be used in data pre-processing to reduce the number of features, just as PCA, which reduces the computing cost significantly.
- LDA is also used in face detection algorithms. In Fisherfaces, LDA is used to extract useful data from different faces. Coupled with eigenfaces, it produces effective results.

Drawbacks of Linear Discriminant Analysis (LDA)

Although, LDA is specifically used to solve supervised classification problems for two or more classes which are not possible using logistic regression in machine learning.

But LDA also fails in some cases where the Mean of the distributions is shared. In this case, LDA fails to create a new axis that makes both the classes linearly separable.

To overcome such problems, we use **non-linear Discriminant analysis** in machine learning.

Extension to Linear Discriminant Analysis (LDA)

Linear Discriminant analysis is one of the most simple and effective methods to solve classification problems in machine learning. It has so many extensions and variations as follows:

1. **Quadratic Discriminant Analysis (QDA):** For multiple input variables, each class deploys its own estimate of variance.
2. **Flexible Discriminant Analysis (FDA):** it is used when there are non-linear groups of inputs are used, such as splines.
3. **Flexible Discriminant Analysis (FDA):** This uses regularization in the estimate of the variance (actually covariance) and hence moderates the influence of different variables on LDA.

Real-world Applications of LDA

Some of the common real-world applications of Linear discriminant Analysis are given below:

- **Face Recognition**
  Face recognition is the popular application of computer vision, where each face is represented as the combination of a number of pixel values. In this case, LDA is used to minimize the number of features to a manageable number before going through the classification process. It generates a new template in which each dimension consists of a linear combination of pixel values. If a linear combination is generated using Fisher's linear discriminant, then it is called Fisher's face.
- **Medical**
  In the medical field, LDA has a great application in classifying the patient disease on the basis of various parameters of patient health and the medical treatment which is going on. On such parameters, it classifies disease as mild, moderate, or severe. This classification helps the doctors in either increasing or decreasing the pace of the treatment.

- **Customer Identification**
  In customer identification, LDA is currently being applied. It means with the help of LDA; we can easily identify and select the features that can specify the group of customers who are likely to purchase a specific product in a shopping mall. This can be helpful when we want to identify a group of customers who mostly purchase a product in a shopping mall.
- **For Predictions**
  LDA can also be used for making predictions and so in decision making. For example, "will you buy this product" will give a predicted result of either one or two possible classes as a buying or not.
- **In Learning**
  Nowadays, robots are being trained for learning and talking to simulate human work, and it can also be considered a classification problem. In this case, LDA builds similar groups on the basis of different parameters, including pitches, frequencies, sound, tunes, etc.

Perceptron in Machine Learning

In Machine Learning and Artificial Intelligence, Perceptron is the most commonly used term for all folks. It is the primary step to learn Machine Learning and Deep Learning technologies, which consists of a set of weights, input values or scores, and a threshold. ***Perceptron is a building block of an Artificial Neural Network***. Initially, in the mid of 19th century, **Mr. Frank Rosenblatt** invented the Perceptron for performing certain calculations to detect input data capabilities or business intelligence. Perceptron is a linear Machine Learning algorithm used for supervised learning for various binary classifiers. This algorithm enables neurons to learn elements and processes them one by one during preparation. In this tutorial, "Perceptron in Machine Learning," we will discuss in-depth knowledge of Perceptron and its basic functions in brief. Let's start with the basic introduction of Perceptron.

What is the Perceptron model in Machine Learning?

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, ***Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence***.

Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., **input values, weights and Bias, net sum, and an activation function.**

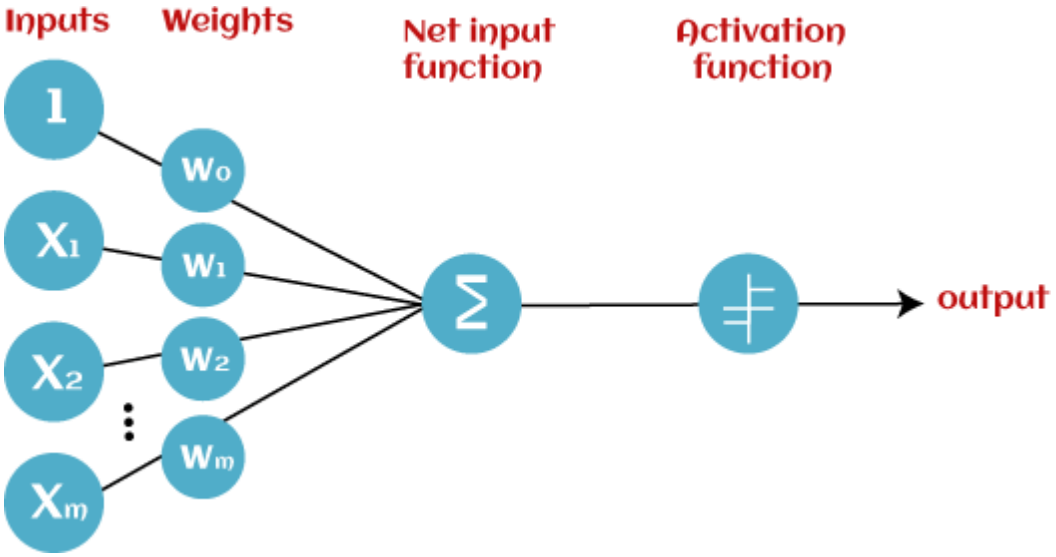
What is Binary classifier in Machine Learning?

In Machine Learning, binary classifiers are defined as the function that helps in deciding whether input data can be represented as vectors of numbers and belongs to some specific class.

Binary classifiers can be considered as linear classifiers. In simple words, we can understand it as a ***classification algorithm that can predict linear predictor function in terms of weight and feature vectors.***

Basic Components of Perceptron

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:



Input Nodes or Input Layer:

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

- **Wight and Bias:**

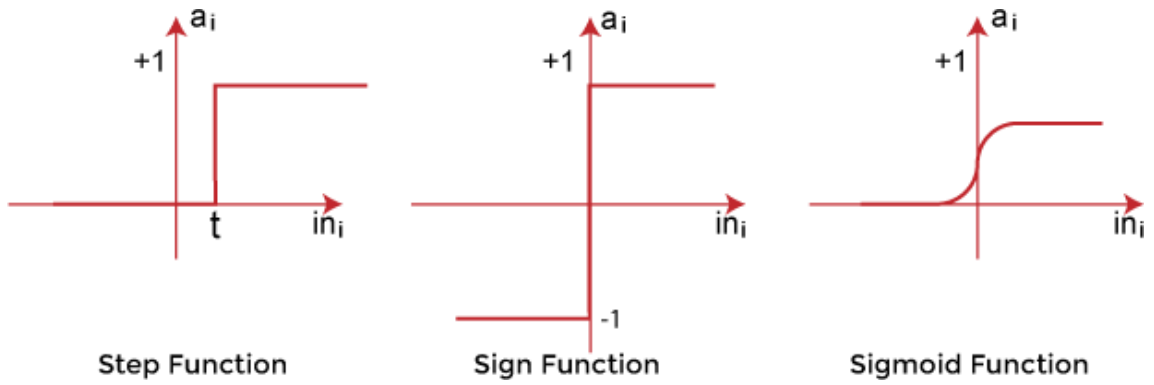
Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

- **Activation Function:**

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

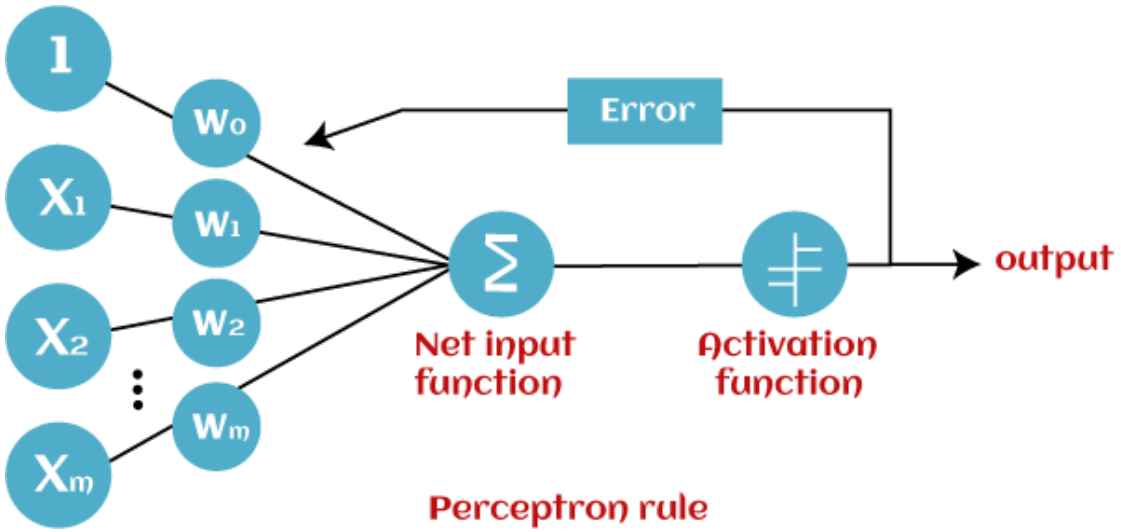
- Sign function
- Step function, and
- Sigmoid function



The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs. Activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.

How does Perceptron work?

In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function. The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the **step function** and is represented by **'f'**.



This step function or Activation function plays a vital role in ensuring that output is mapped between required values (0,1) or (-1,1). It is important to note that the weight of input is indicative of the strength of a node. Similarly, an input's bias value gives the ability to shift the activation function curve up or down.

Perceptron model works in two important steps as follows:

Step-1

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

$\sum wi*xi = x1*w1 + x2*w2 + \ldots wn*xn$

Add a special term called **bias 'b'** to this weighted sum to improve the model's performance.

$\mathbf{\sum wi*xi + b}$

Step-2

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

$\mathbf{Y = f(\sum wi*xi + b)}$

Linear separability

Linear separability is an important concept in machine learning, particularly in the field of supervised learning. It refers to the ability of a set of data points to be separated into distinct categories using a linear decision boundary. In other words, if there exists a straight line that can cleanly divide the data into two classes, then the data is said to be linearly separable.

Linear separability is a concept in machine learning that refers to the ability to separate data points in binary classification problems using a linear decision boundary. If the data points can be separated using a line, linear function, or flat hyperplane, they are considered linearly separable. Linear separability is an important concept in neural networks, and it is introduced in the context of linear algebra and optimization theory.

In the context of machine learning, linear separability is an important property because it makes classification problems easier to solve. If the data is linearly separable, we can use a linear classifier, such as logistic regression or support vector machines (SVMs), to accurately classify new instances of data.

Linearly separable data points can be separated using a line, linear function, or flat hyperplane. In practice, there are several methods to determine whether data is linearly

separable[3]. One method is linear programming, which defines an objective function subjected to constraints that satisfy linear separability. Another method is to train and test on the same data - if there is a line that separates the data points, then the accuracy or AUC should be close to 100%. If there is no such line, then training and testing on the same data will result in at least some error. Multi-layer neural networks can learn hidden features and patterns in data that linear classifiers cannot

To understand the concept of linear separability, it is helpful to first consider a simple two-dimensional example. Imagine we have a set of data points in a two-dimensional space, where each point is labeled either "red" or "blue". If these data points can be separated by a straight line, such that all the red points are on one side of the line and all the blue points are on the other side, then the data is linearly separable.

Python provides several methods to determine whether data is linearly separable. One method is linear programming, which defines an objective function subjected to constraints that satisfy linear separability. Another method is clustering, where if two clusters with cluster purity of 100% can be found using some clustering methods such as k-means, then the data is linearly separable.

However, not all data sets are linearly separable. In some cases, it may be impossible to draw a straight line that can separate the data into distinct categories. For example, imagine a set of data points that are arranged in a circular pattern, with red and blue points interspersed throughout. In this case, it is impossible to draw a straight line that separates the data into two classes.

When faced with data that is not linearly separable, machine learning algorithms must use more complex decision boundaries to accurately classify the data. For example, a decision tree or a neural network may be able to accurately classify data that is not linearly separable.

Linear separability is not only important in the context of machine learning, but it also has applications in other fields such as physics, biology, and economics. For example, in physics, linear separability can be used to analyze the relationship between two physical quantities. In biology, it can be used to study the behavior of animals or to analyze genetic data. In economics, it can be used to analyze the relationship between two economic variables.

Example:

One way to test for linear separability is to use linear programming. Linear programming defines an objective function subject to constraints that satisfy linear separability. The scipy.optimize.linprog() function in Python can be used to solve linear programming problems. Here's an example of using scipy.optimize.linprog() to test for linear separability:

```
import numpy as np
from scipy.optimize import linprog

# Define the data points
X = np.array([[1, 2], [2, 3], [3, 1], [4, 3]])
y = np.array([1, 1, -1, -1])

# Define the objective function and constraints
c = np.zeros(X.shape[1] + 1)
c[-1] = 1
A = np.zeros((X.shape, X.shape[1] + 1))
A[:, :-1] = -y[:, np.newaxis] * X
A[:, -1] = -y
b = -np.ones(X.shape)

# Solve the linear programming problem
res = linprog(c, A_ub=A, b_ub=b)

if res.success:
    print("The data is linearly separable.")
else:
    print("The data is not linearly separable.")
```

In this example, we define a set of data points X and their corresponding labels y. We then define the objective function and constraints for the linear programming problem. The objective function is simply a vector of zeros with a 1 in the last position, which corresponds to the bias term in the linear decision boundary. The constraints are defined such that the dot product of each data point with the weight vector and bias term is greater than or equal to -1 for negative examples and less than or equal to 1 for positive examples. We then solve the linear programming problem using scipy.optimize.linprog() and check if the solution is successful. If the solution is successful, the data is linearly separable.

Another way to test for linear separability is to use a linear classifier or support vector machine (SVM) with a linear kernel. For linearly separable datasets, a linear classifier or SVM with a linear kernel can achieve 100% accuracy to classify data[4]. Here's an example of using a linear SVM to test for linear separability:

```
from sklearn import svm

# Define the data points
X = np.array([[1, 2], [2, 3], [3, 1], [4, 3]])
y = np.array([1, 1, -1, -1])

# Train a linear SVM
clf = svm.SVC(kernel='linear')
clf.fit(X, y)
if clf.score(X, y) == 1.0:
    print("The data is linearly separable.")
else:
    print("The data is not linearly separable.")
```

In this example, we define a set of data points X and their corresponding labels y. We then train a linear SVM using the svm.SVC() function from scikit-learn with a linear kernel. We then check if the accuracy of the SVM on the training data is 100%.

Moreover, it is important to note that linear separability is not the only criterion for the effectiveness of a classification algorithm. In some cases, even if the data is linearly separable, a linear classifier may not be the best choice. For example, if the data is high-dimensional or contains complex nonlinear relationships, a nonlinear classifier may be more effective. In such cases, machine learning algorithms such as decision trees, random forests, or neural networks may be more appropriate

In practice, determining whether a set of data points is linearly separable can be challenging. There are several approaches that can be used to determine linear separability, including visual inspection and mathematical analysis. One common approach is to use a scatter plot to visualize the data and see if a straight line can be drawn that separates the data into two classes.

Another approach is to use mathematical tools to determine if the data is linearly separable. For example, the Perceptron algorithm is a simple algorithm that can be used to determine linear separability. The algorithm works by iterating through the data points and updating a set of weights that define the decision boundary. If the algorithm is able to converge on a set of weights that separates the data, then the data is linearly separable.

Another important aspect to consider is that linear separability is not an inherent property of the data itself, but rather a property of the representation of the data. In other words, the way in which the data is transformed or encoded can affect whether

or not it is linearly separable. This means that preprocessing techniques such as feature selection,

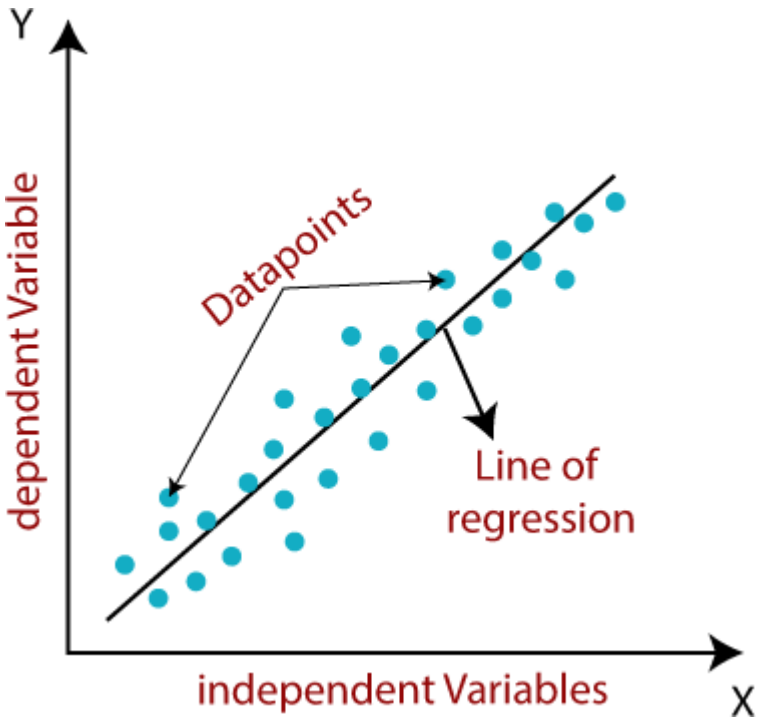
In conclusion, linear separability is an important concept in machine learning that refers to the ability of a set of data points to be separated into distinct categories using a linear decision boundary. Linear separability makes classification problems easier to solve, but not all data sets are linearly separable. linear separability is an important concept in machine learning that allows us to determine if a binary classification problem can be separated using a linear decision boundary. In practice, determining whether a set of data points is linearly separable can be challenging, and there are several approaches that can be used to determine linear separability, including visual inspection and mathematical analysis.

Linear Regression in Machine Learning

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price,** etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Mathematically, we can represent a linear regression as:

$y= a0+a1x+ \varepsilon$

Here,

Y= Dependent Variable (Target Variable)
X= Independent Variable (predictor Variable)
a0= intercept of the line (Gives an additional degree of freedom) a1 = Linear regression coefficient (scale factor to each input value). ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Types of Linear Regression

Linear regression can be further divided into two types of the algorithm:

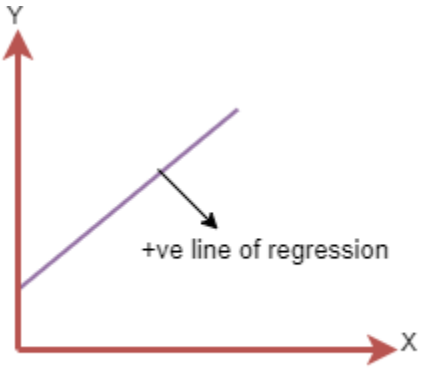
- **Simple Linear Regression:**
  If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

- **Multiple Linear regression:**
  If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Linear Regression Line

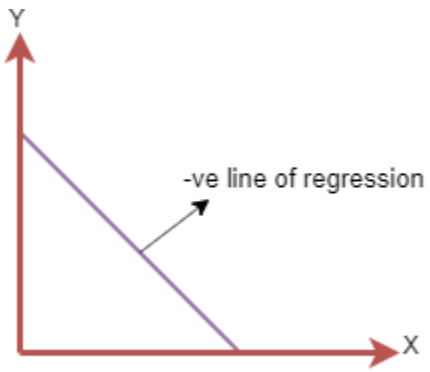
A linear line showing the relationship between the dependent and independent variables is called a **regression line**. A regression line can show two types of relationship:

- **Positive Linear Relationship:**
  If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



The line equation will be: $Y= a_0+a_1x$

- **Negative Linear Relationship:**
  If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.



The line of equation will be: $Y= -a_0+a_1x$

Finding the best fit line:

When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.

The different values for weights or the coefficient of lines (a_0, a_1) gives a different line of regression, so we need to calculate the best values for a_0 and a_1 to find the best fit line, so to calculate this we use cost function.

Cost function-

- The different values for weights or coefficient of lines (a_0, a_1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.
- Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.
- We can use the cost function to find the accuracy of the **mapping function**, which maps the input variable to the output variable. This mapping function is also known as **Hypothesis function**.

For Linear Regression, we use the **Mean Squared Error (MSE)** cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

For the above linear equation, MSE can be calculated as:

$$MSE = 1\frac{1}{N}\sum_{i=1}^{n}(y_i - (a_1x_i + a_0))^2$$

Where,

N=Total number of observation
Yi = Actual value
$(a_1x_i+a_0)$= Predicted value.

Residuals: The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Gradient Descent:

- Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.
- It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

Model Performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called **optimization**. It can be achieved by below method:

1. R-squared method:

- R-squared is a statistical method that determines the goodness of fit.
- It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.
- The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a **coefficient of determination,** or **coefficient of multiple determination** for multiple regression.
- It can be calculated from the below formula:

$$\text{R-squared} = \frac{\text{Explained variation}}{\text{Total Variation}}$$

Assumptions of Linear Regression

Below are some important assumptions of Linear Regression. These are some formal checks while building a Linear Regression model, which ensures to get the best possible result from the given dataset.

- **Linear relationship between the features and target:** Linear regression assumes the linear relationship between the dependent and independent variables.
- **Small or no multicollinearity between the features:** Multicollinearity means high-correlation between the independent variables. Due to

multicollinearity, it may difficult to find the true relationship between the predictors and target variables. Or we can say, it is difficult to determine which predictor variable is affecting the target variable and which is not. So, the model assumes either little or no multicollinearity between the features or independent variables.

- **Homoscedasticity Assumption:**
  Homoscedasticity is a situation when the error term is the same for all the values of independent variables. With homoscedasticity, there should be no clear pattern distribution of data in the scatter plot.
- **Normal distribution of error terms:**
  Linear regression assumes that the error term should follow the normal distribution pattern. If error terms are not normally distributed, then confidence intervals will become either too wide or too narrow, which may cause difficulties in finding coefficients.
  It can be checked using the **q-q plot**. If the plot shows a straight line without any deviation, which means the error is normally distributed.
- **No autocorrelations:**
  The linear regression model assumes no autocorrelation in error terms. If there will be any correlation in the error term, then it will drastically reduce the accuracy of the model. Autocorrelation usually occurs if there is a dependency between residual errors.

UNIT – II

Multi-layer Perceptron– Going Forwards – Going Backwards: Back Propagation Error – Multi-layer Perceptron in Practice – Examples of using the MLP – Overview – Deriving Back-Propagation – Radial Basis Functions and Splines – Concepts – RBF Network – Curse of Dimensionality – Interpolations and Basis Functions – Support Vector Machines

Multi-layer Perceptron :

The multilayer perceptron is an artificial neural network structure and is a nonparametric estimator that can be used for classification and regression. We discuss the back propagation algorithm to train a multilayer perceptron for a variety of applications.

We have pretty much decided that the learning in the neural network happens in the weights. So, to perform more computation it seems sensible to add more weights. There are two things that we can do:

add some backwards connections, so that the output neurons connect to the inputs again, or add more

neurons.

The **first approach** leads into recurrent networks. These have been studied, but are not that commonly used.We will instead consider the **second approach**.

Multi-layer Perceptron :.

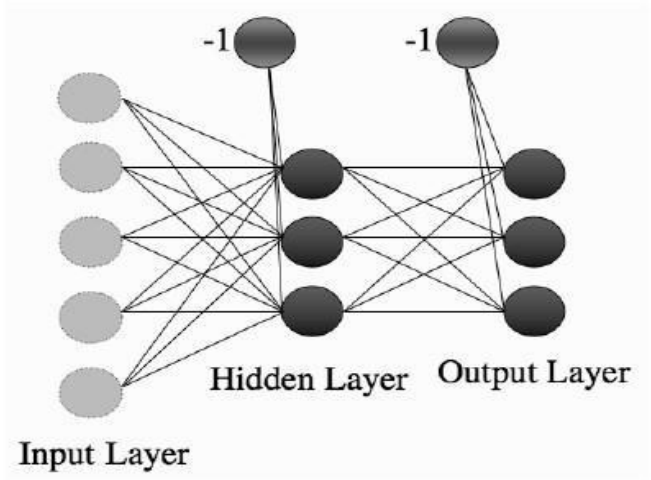


FIGURE 4.1 The Multi-layer Perceptron network, consisting of multiple layers of connected neurons.

we can check that a prepared network can solve the two-dimensional XOR problem, something that we have seen is not possible for a linear model like the Perceptron.

A suitable network is shown in Figure 4.2. To check that it gives the correct answers, all that is required is to put in each input and work through the network, treating it as **two different Perceptrons**, first computing the activations of the neurons in the middle layer (labelled as C and D in Figure 4.2) and then using those activations as the inputs to the single neuron at the output. As an example, I'll work out what happens when you put in (1, 0) as an input;

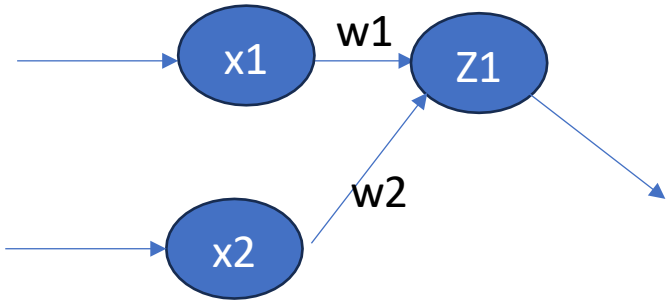
Multilayer Perceptron with XOR example y= z1+z2
Where z1= $x1\overline{x2}$ z2 = $\overline{x1}x2$

Z1= $x1\overline{x2}$

$\overline{y} = x1\overline{x2} + \overline{x1}x2$

f(Yn) = {1 if Yin≥0

{0 Yin<0



X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Activation Fu

Multi-layer Perceptron with XOR

Let us consider the weights for w11 = 1 w21 =1 Threshold = 1 learning rate = 1.5

(0,0) → Zin = Wij*Xi = *1*0+1*0 = 0 (out = 0)*

(0,1) → Zin = Wij*Xi = *1*0+1*1 = 1 (out = 1)*

- wij =wij +n(t-o)xi ∴ *-w11 = 1+1.5 *(0-1)*0=0* *w21 = 1+1.5 * (0-1) * 1 = -0.5*

X1	X2	Z
0	0	0
0	1	0
1	0	1
1	1	0

Multi-layer Perceptron with XOR

X1	X2	Z1
0	0	0
0	1	0
1	0	1
1	1	0

First function = Z1= $x1x2$

Let us consider the weights for w11 = 1 w21 =1 updated weights:w11= 1, w21= -0.5

Threshold = 1 learning rate = 1.5

(0,0) ⟶ Zin = Wij*Xi = *1*0+(-0.5)*0 = 0 (out = 0)* (0,1) ⟶ Zin = Wij*Xi = *1*0+(-0.5)*1 = -0.5 (out = 0) (1,0)* ⟶ *Zin = Wij*Xi = 1*1+(-0.5)*0 = 1 (out=1) (1,1)* ⟶ *Zin = Wij*Xi = 1*1+(-0.5) * 1 = 0.5 (out =0)*

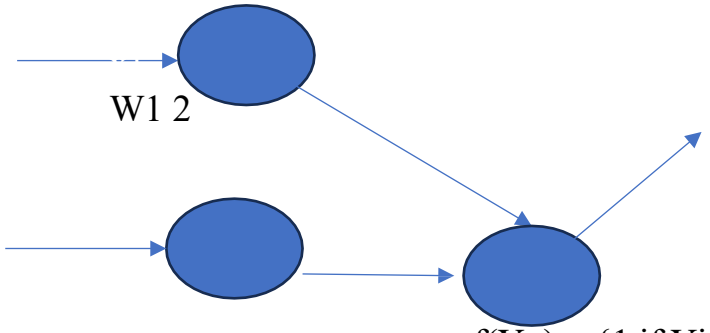
- wij =wij +n(t-o)xi ∴ *-w11 = 1+1.5 *(0-1)*0=1* *w21 = 1+1.5 * (0-1) * 1 = -0.5*

Multilayer Perceptron with XOR example y= z1+z2
Where z1= $x1x2$ z2 = $x1x2$

$\overline{z2} = x1x2$

W1 2

W2 2



f(Yn) = {1 if Yin≥0

{0 Yin<0

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Activation Function

Multi-layer Perceptron with XOR First function = z2 = x1x2
Let us consider the weights for w12 = 1 w2 2 = 1 updated weights:w12= -0.5, w2 2=1

Threshold = 1 learning rate = 1.5

→

(0,0) Z2in = Wij*Xi = *1*0+(1)*0 = 0 (out = 0)*

→

(0,1) Z2in = Wij*Xi = *1* 0 +(1)*1 =* ***1*** *(out =* ***1****)*

→

*(1,0) Zin = Wij*Xi = 1*1+(1)*0 = 1 (out=0)*

Fyin = {1 if yin ≥ 0
{0 if yin ≤ 0

Activation function

- wij =wij +n(t-o)xi *∴ -w12 = 1+1.5 *(0-1)* 1=-0.5* *w22 = 1+1.5 * (0-1) * 0 = 1*

Multi-layer Perceptron with XOR First function =
z2 = x1x2

Let us consider the weights for w12 = 1 w2 2 = 1 updated weights:w12= -0.5, w2 2=1

Threshold = 1 learning rate = 1.5

(0,0) ⟶ Z2in = Wij*Xi = *-0.5* (0) + 1 * 0 = 0 (out = 0)*

(0,1) ⟶ Z2in = Wij*Xi = *-0.5* 0 +(1)*1 =* ***1*** *(out =* ***1****)*

*(1,0) ⟶ Z2in = Wij*Xi = -0.5*1+(1)*0 = -0.5 (out=0)*

*(1,1) ⟶ Z2in = Wij*xi = -0.5 * 1 + 1 * 1 =0.5 (out = 0)*

X1	X2	Z2
0	0	0
0	1	**1**
1	0	0
1	1	0

Fyin = {1 if yin ≥ 0
Activation function {0 if yin ≤ 0

- wij =wij +n(t-o)xi *∴ -w12 = 1+1.5 *(0-1)* 1=-0.5* *w22 = 1+1.5 * (0-1) * 0 = 1*

- Y = Z1 OR Z2
- => yin = Z1v1 +z2v2
- => v1 = 1 ; v2 = 1

Threshold = 1 and learning rate = 1.5

X1	X2	Z1	Z2	Y
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

✓ (0,0) -> yin = vi * xi = 1*0+1*0 = 0 (out =0)

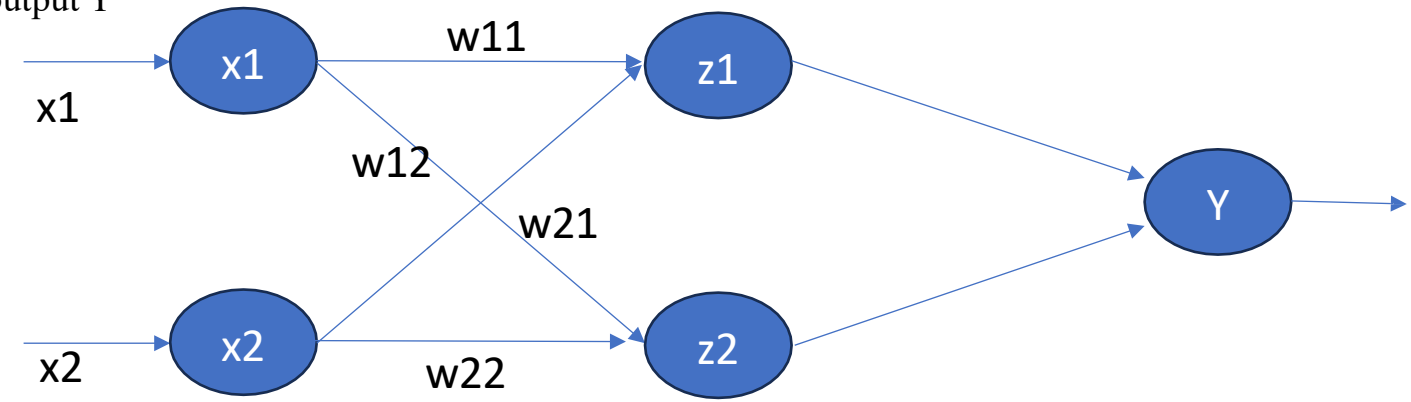
✓ (0,1) -> yin = vi*xi = 1*0+1*1 = 1 (out = 1)

✓ (1,0)-> yin = vi*xi = 1*1+1*0 = 1 (out = 1)

✓ (0,0)-> yin = vi*xi = 0*1 + 0*1 = 0 (out =0)

- Final weights with output Y

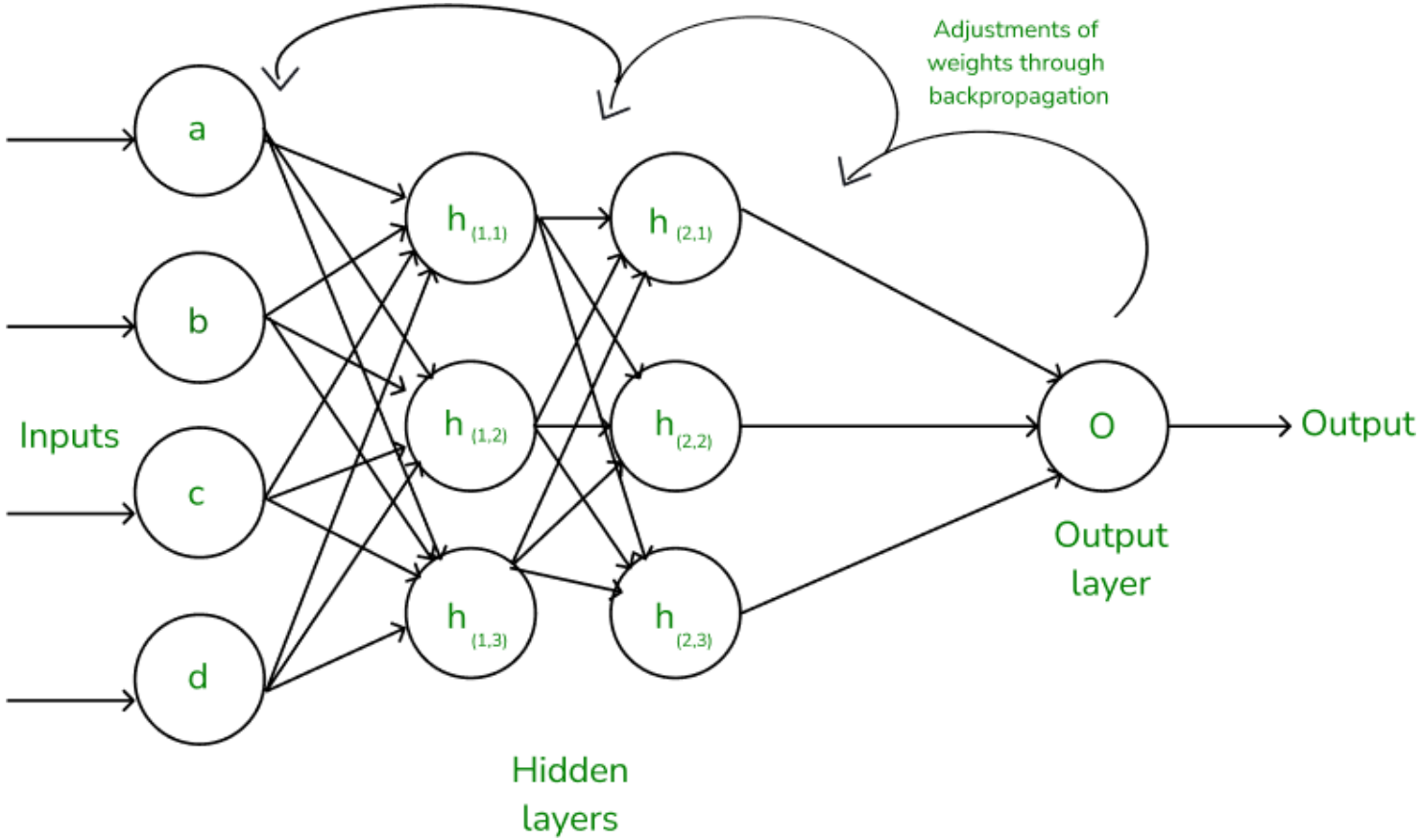
Y



- In machine learning, backpropagation is an effective algorithm used to train artificial neural networks, especially in feed-forward neural networks.

- Backpropagation is an iterative algorithm, that helps to minimize the cost function by determining which weights and biases should be adjusted. During every epoch, the model learns by adapting the weights and biases to minimize the loss by moving down toward the gradient of the error. Thus, it involves the two most popular optimization algorithms,
- such as gradient descent or stochastic gradient descent.

- Computing the gradient in the backpropagation algorithm helps to minimize the cost function and it can be implemented by using the mathematical rule called chain rule from calculus to navigate through complex layers of the neural network.


Adjustments of
weights through
backpropagation
a
b
c
d
Inputs
h (1,1)
h (1,2)
h (1,3)
h (2,1)
h (2,2)
h (2,3)
Hidden
layers
O
Output
Output
layer

Advantages of Using the Backpropagation Algorithm in Neural Networks

Backpropagation, a fundamental algorithm in training neural networks, offers several advantages that make it a preferred choice for many machine learning tasks. Here, we discuss some key advantages of using the backpropagation algorithm:

1. ***Ease of Implementation*:** Backpropagation does not require prior knowledge of neural networks, making it accessible to beginners. Its straightforward nature simplifies the programming process, as it primarily involves adjusting weights based on error derivatives.
2. ***Simplicity and Flexibility:*** The algorithm's simplicity allows it to be applied to a wide range of problems and network architectures. Its flexibility makes it suitable for various scenarios, from simple feedforward networks to complex recurrent or convolutional neural networks.
3. ***Efficiency*:** Backpropagation accelerates the learning process by directly updating weights based on the calculated error derivatives. This e fficiency is particularly advantageous in training deep neural networks, where learning features of a function can be time-consuming.
4. ***Generalization:*** Backpropagation enables neural networks to generalize well to unseen data by iteratively adjusting weights during training. This generalization ability is crucial for developing models that can make accurate predictions on new, unseen examples.
5. ***Scalability*:** Backpropagation scales well with the size of the dataset and the complexity of the network. This scalability makes it suitable for large-scale machine learning tasks, where training data and network size are significant factors.

Working of Backpropagation Algorithm

The Backpropagation algorithm works by two different passes, they are:

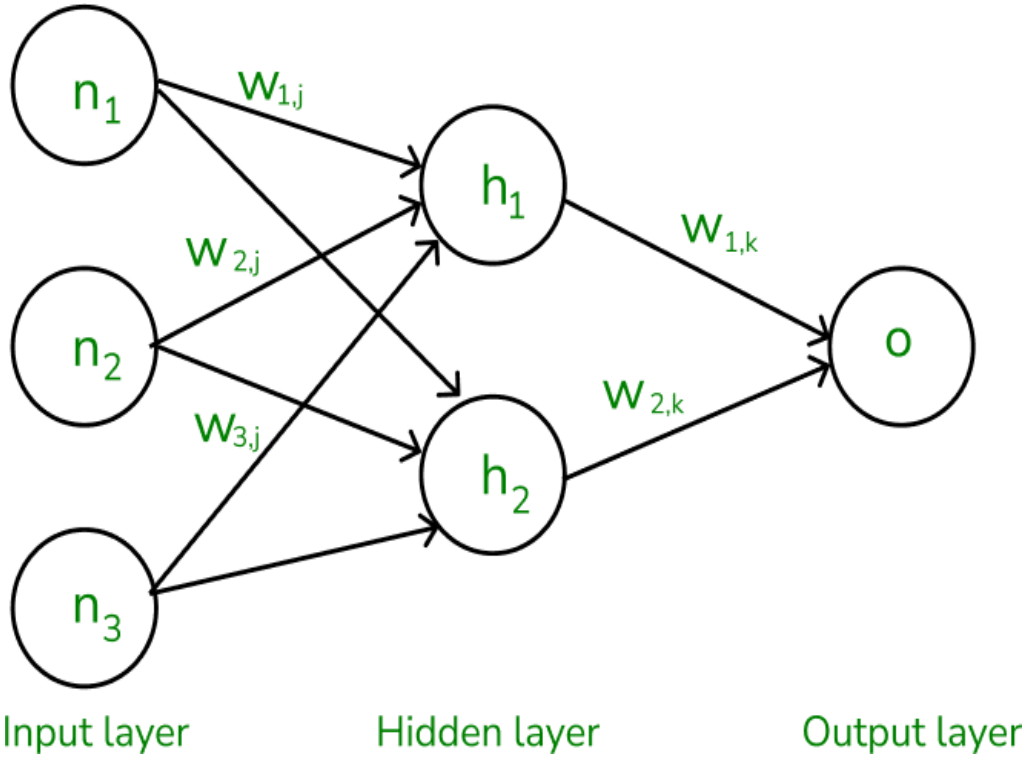
Forward pass Backward pass

In forward pass, initially the input is fed into the input layer. Since the inputs are raw data, they can be used for training our neural network.
The inputs and their corresponding weights are passed to the hidden layer. The hidden layer performs the computation on the data it receives. If there are two hidden layers in the neural network, for instance, consider the illustration fig(a), h_1 and h_2 are the two hidden layers, and the output of h_1 can be used as an input of h_2. Before applying it to the activation function, the bias is added.

To the weighted sum of inputs, the activation function is applied in the hidden layer to each of its neurons. One such activation function that is commonly used is ReLU can also be used, which is responsible for returning the input if it is positive otherwise it returns zero. By doing this so, it introduces the non-linearity to our model, which enables the network to learn the complex relationships in the data. And finally, the weighted outputs from the last hidden layer are fed into the output to compute the final prediction, this layer can also use the activation function called the **softmax function** which is responsible for converting the weighted outputs into probabilities for each class.

Working of Backpropagation Algorithm

The Backpropagation algorithm works by two different passes, they are: Forward pass



Working of Backpropagation Algorithm

The Backpropagation algorithm works by two different passes, they are: Backward pass

- In the backward pass process shows, the error is transmitted back to the network which helps the network, to improve its performance by learning and adjusting the internal weights.

- To find the error generated through the process of forward pass, we can use one of the most commonly used methods called mean squared error which calculates the difference between the predicted output and desired output. The formula for mean squared error is:
  Mean squared error = (predicted output – actual output)^2

- Once we have done the calculation at the output layer, we then propagate the error backward through the network, layer by layer.

- The key calculation during the backward pass is determining the gradients for each weight and bias in the network. This gradient is responsible for telling us how much each weight/bias should be adjusted to minimize the error in the next forward pass. The chain rule is used iteratively to calculate this gradient efficiently.

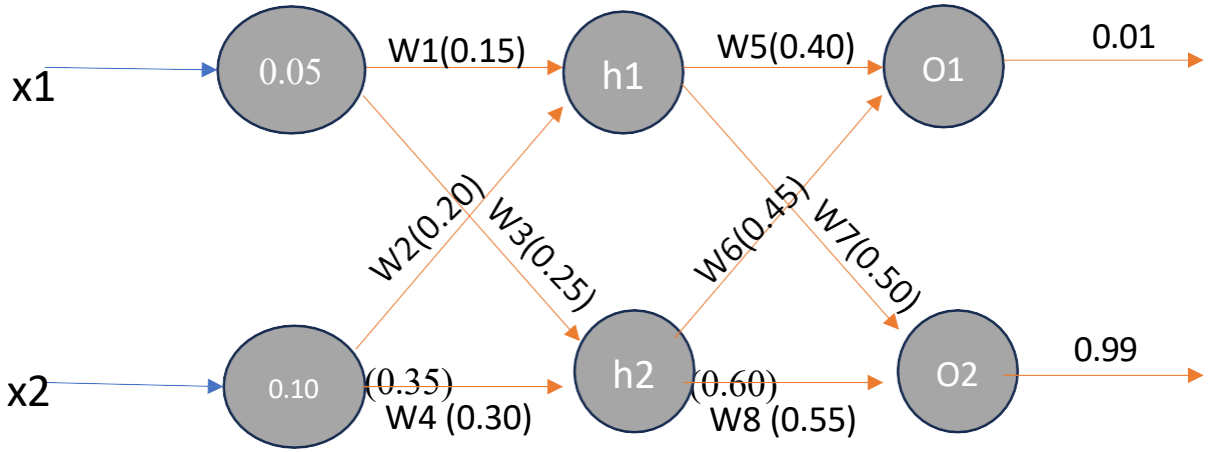
- In addition to gradient calculation, the activation function also plays a crucial role in backpropagation, it works by calculating the gradients with t he help of the derivative of the activation function.

Working of Backpropagation Algorithm with an example

part 1:
calculate forward propagation error
calculate h1(in and out)
where h1 = w1x1+w2x2+b1

Here h1 and h2 hidden layers
- **w1 to w8 are weights**
- **b1 and b2 are bias factor**

where b1 between input and hidden layer
where b2 between hidden and output layer



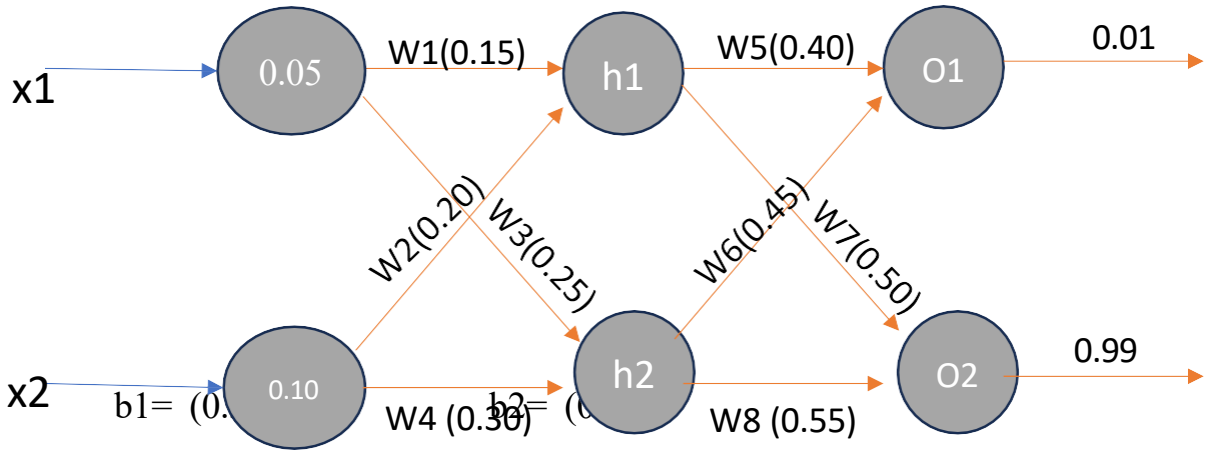
Working of Backpropagation Algorithm with an example

part 1:
calculate forward propagation error calculate h1(in and out)

where h1 = w1x1+w2x2+b1
h1(in)
= 0.15*0.05+0.20*0.10+0.35
= 0.377

1
= **H2 (out)** $\frac{}{1+e^{-h1(in}}$
)

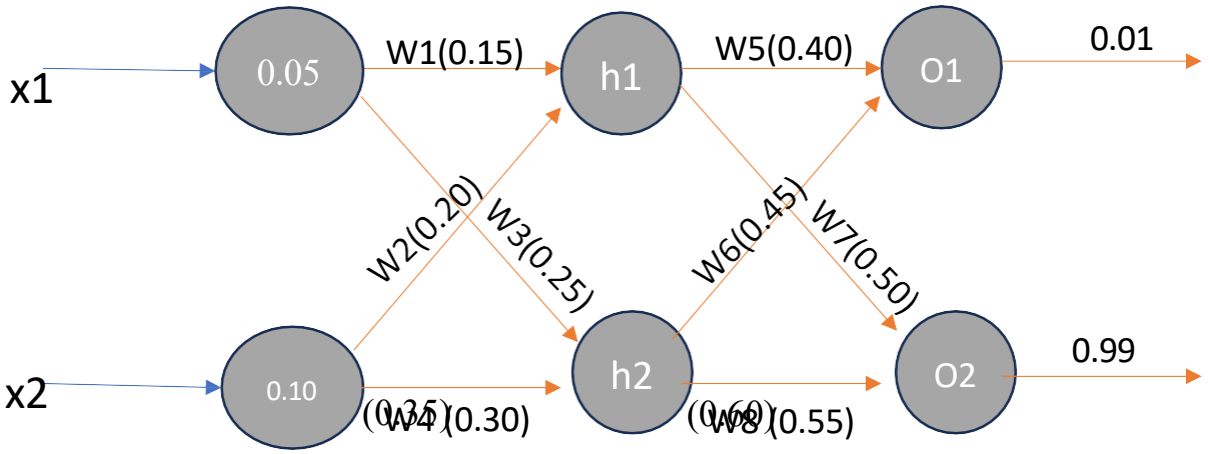
H2 (out) = $\frac{1}{1+e^{-0.3777}}$ **= 0.5932**



Working of Backpropagation Algorithm with an example

part 2:

calculate forward propagation error calculate h2(in and out)



where h2 = x1w3+x2w4+b1

h2(in)

= 0.05*0.25+0.10*0.3+0.35

= 0.0125+0.03+0.35 = 0.3925

1

= **h2 (out)**

H2 (out) = $\frac{1}{1+e^{-h2(in)}}$

$\frac{1}{1+e^{-0.3925}}$ **= 0.5968**

Working of Backpropagation Algorithm with an example

part 3:

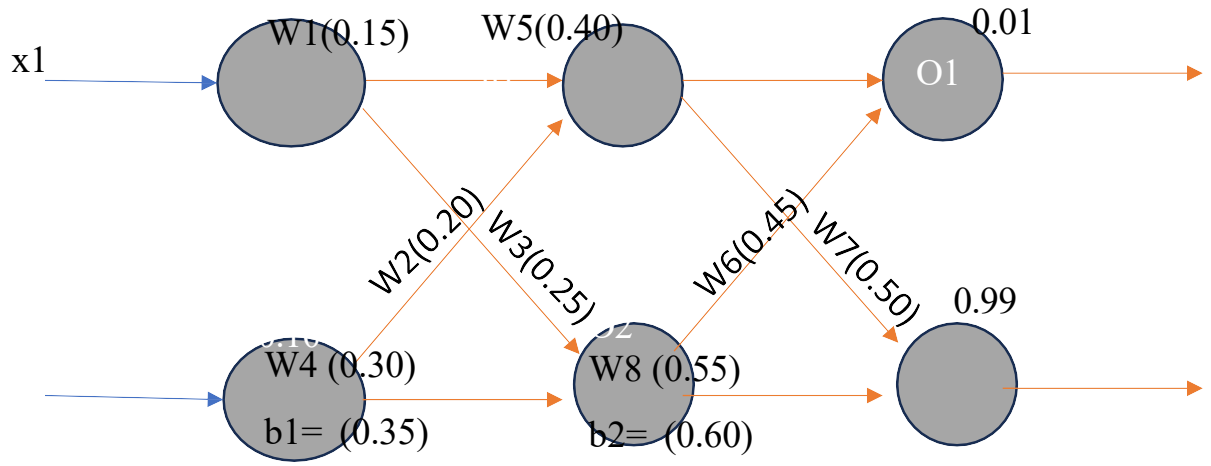
calculate forward propagation error calculate O1(in and out)

where O1 = h1(out)*w5+h2(out)*w6+b2

O1(in)

= 0.593*0.4+0.596*0.45+0.6

= 1.105



= **O1 (out)**

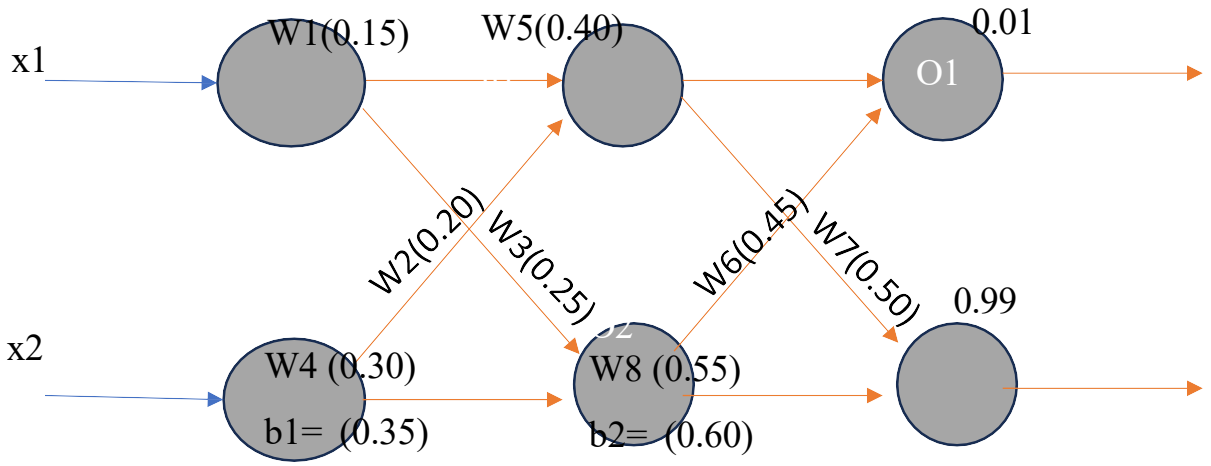
H2 (out) = $\frac{}{1+e^{-O1(in)}}$

$$\frac{1}{1+e^{-1.105}} = 0.7513$$

Working of Backpropagation Algorithm with an example

part 3:

calculate forward propagation error calculate O2(in and out)



where O2 = h1(out)*w7+h2(out)*w8+b2

O2(in)

= 0.5932*0.50+0.5968*0.55+0.6

= 1.22484

$$= \textbf{O2 (out)} \quad \frac{}{1+e^{-O2(in)}}$$

H2 (out) =

$$\frac{1}{1+e^{-1.22484}} = 0.7729$$

Working of Backpropagation Algorithm with an example

part 3:

calculate forward propagation error

calculate E total where E represents Error

$$\text{Etotal} = \sum \frac{1}{2}(target - Output)^2$$

$$= E_{O1} + E_{O2}$$

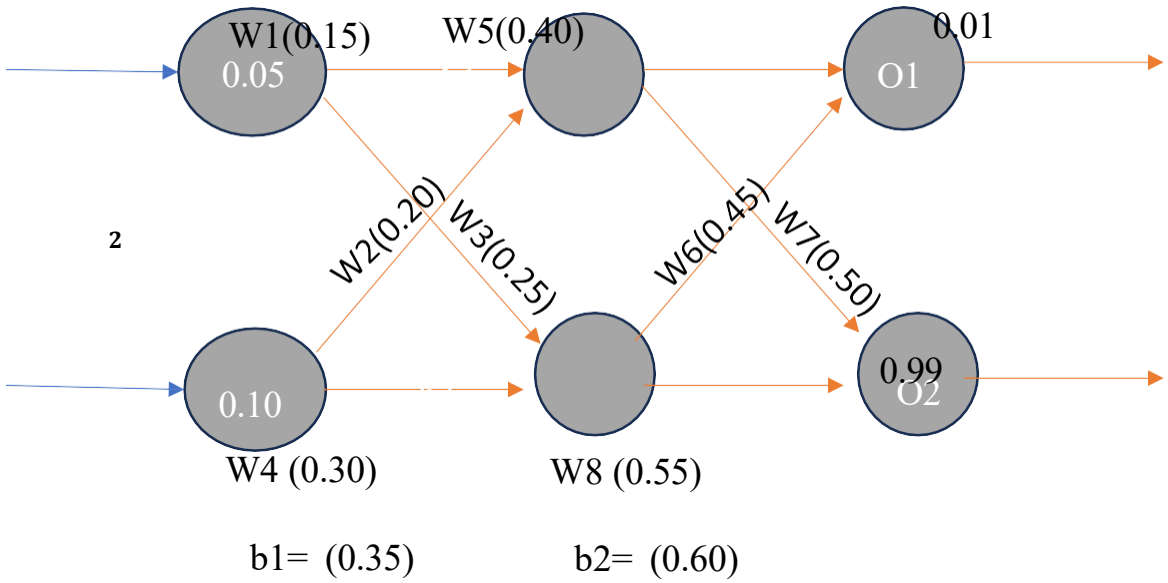
where error at output O1

where error at output O2

$$= \frac{1}{2}(0.01 - 0.7513)^2 + \frac{1}{2}(0.99 - 0.7729)^2$$

$$= \frac{1}{2}(-0.7413)^2 \quad x2 \quad \frac{1}{2}(0.2171)^2$$

= 0.274+ 0.0235 = 0.29837 (appprx)



Working of Backpropagation Algorithm with an example

part 3:

calculate Backward propagation error (output layer to hidden layer)

W5,W6,W7,W8

$*W5 = W5 - n\frac{\partial Etotal}{\partial W5}$ where x1

∂ Partial Differential

Where n is Learning Rate with 0.6

$$\frac{\partial Etotal}{\partial W5} = \frac{\partial Etotal}{\partial out\ O1} * \frac{\partial Out\ O1}{\partial net\ O1} * \frac{\partial net\ O1}{\partial W5}$$

x2

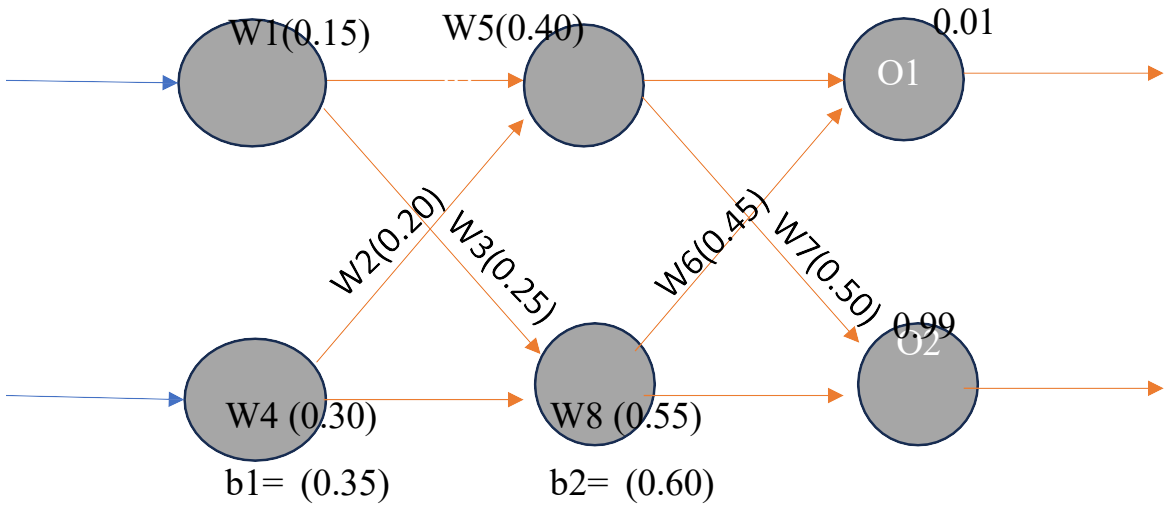
$\frac{\partial Etotal}{\partial out\ O1}$ =Out O1 – Target O1

= 0.751365-0.01 = 0.7413565

$\frac{\partial Out\ O1}{\partial net\ O1}$ = Out O1 (1-Out O1)

= 0.751365(1-0.751365)

= 0.186815602



Working of Backpropagation Algorithm with an example

part 3:

calculate Backward propagation error (output layer to hidden layer)

W5,W6,W7,W8

$*W5 = W5 - n\frac{\partial Etotal}{\partial W5}$ where x1

∂ Partial Differential

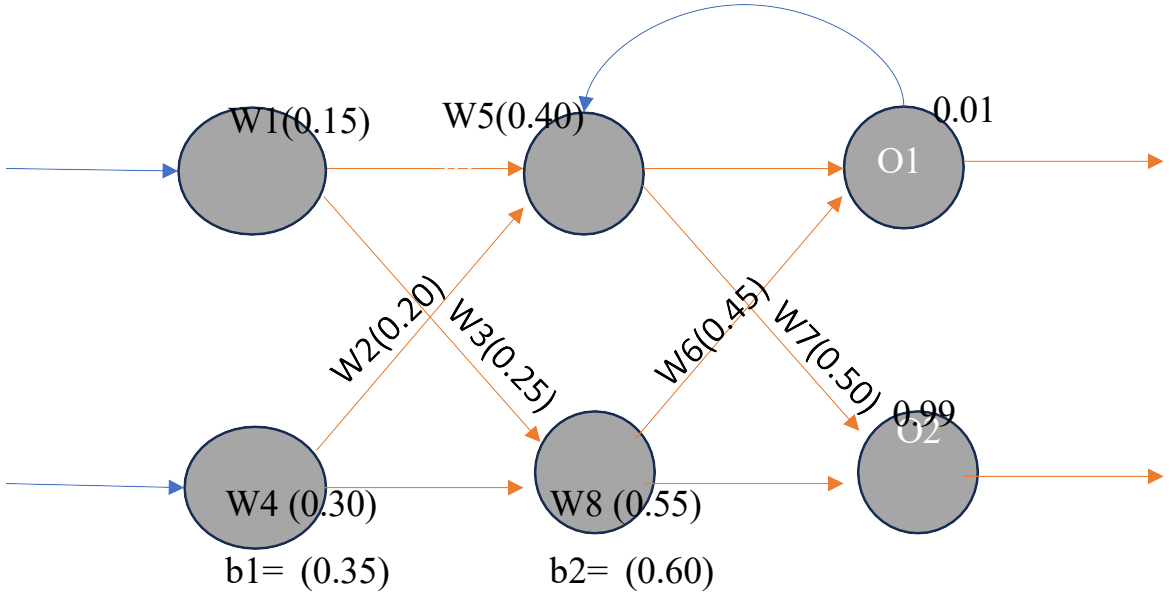
Where n is Learning Rate with 0.6

$$\frac{\partial Etotal}{\partial W5} = \frac{\partial Etotal}{\partial out\ O1} * \frac{\partial Out\ O1}{\partial net\ O1} * \frac{\partial net\ O1}{\partial W5}$$

x2

$\frac{\partial net\ O1}{\partial W5}$ = out h1 = 0.59326992 [h1 (out)]

$\frac{\partial Out\ O1}{\partial net\ O1}$ = Out O1 (1-Out O1)



= 0.751365(1-0.751365)
= 0.186815602

Working of Backpropagation Algorithm with an example
part 3:
calculate Backward propagation error (output layer to hidden layer)
W5,W6,W7,W8

$*W5 = W5 - n\frac{\partial Etotal}{\partial W5}$ where x1

∂ Partial Differential

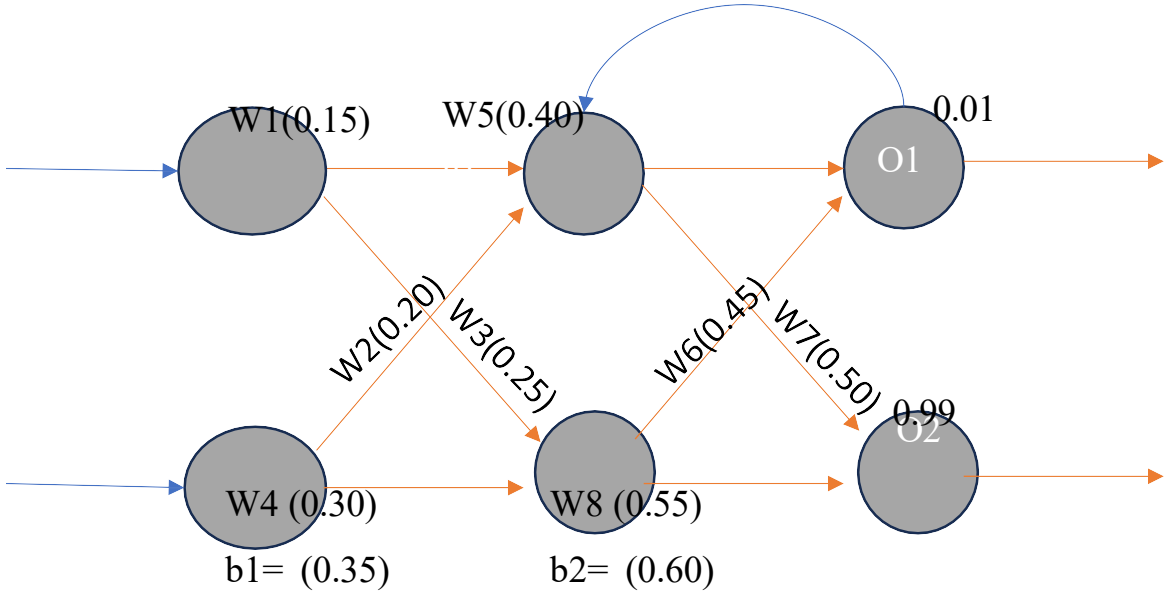
Where n is Learning Rate with 0.6

$$\frac{\partial Etotal}{\partial W5} = \frac{\partial Etotal}{\partial out\ O1} * \frac{\partial Out\ O1}{\partial net\ O1} * \frac{\partial net\ O1}{\partial W5}$$

x2

$\frac{\partial Etotal}{\partial W5}$ = 0.7413565*0.186815602*0.59326992
= 0.08216704

$W5^* = W5 - n\frac{\partial Etotal}{\partial W5}$ = 0.4-(0.6*0.08216704)
= 0.350699776



Working of Backpropagation Algorithm with an example

part 3:

calculate Backward propagation error (hidden layer to input layer)

W1,W2,W3,W4

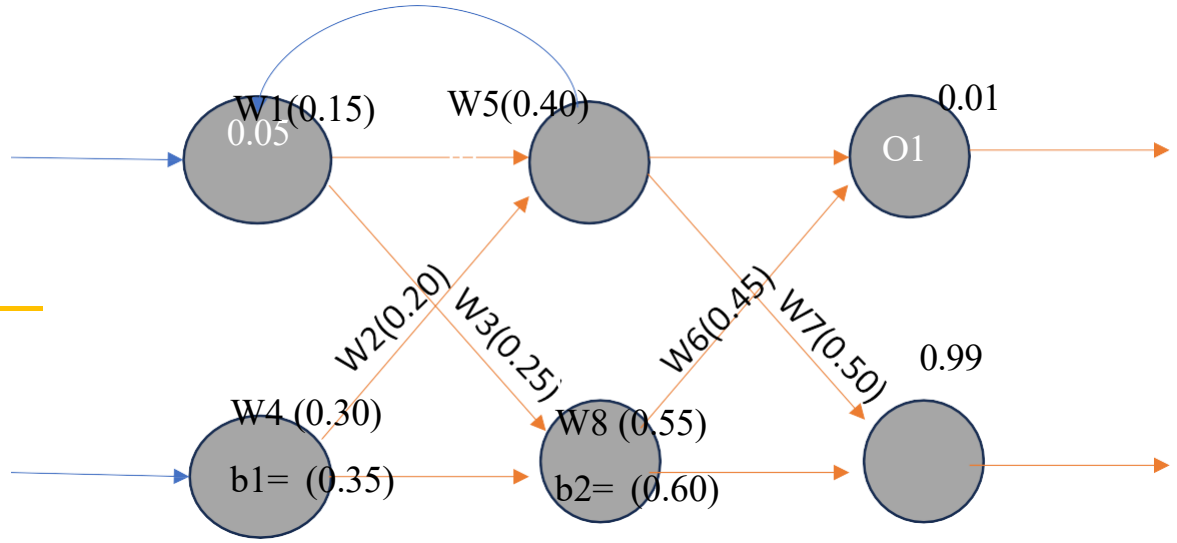
$$*W1 = W1 - n\frac{\partial Etotal}{\partial W1}$$

$$\frac{\partial Etotal}{\partial W1} = \frac{\partial Etotal}{\partial out\,h1} * \frac{\partial out\,h1}{\partial net\,h1} * \frac{\partial net\,h1}{\partial W1}$$

$$\frac{\partial Etotal}{\partial out\,h1} = \frac{\partial E\,O1}{\partial out\,h1} + \frac{\partial E\,O2}{\partial out\,h1}$$

$$\frac{\partial E\,O1}{\partial out\,h1} = \frac{\partial E\,O1}{\partial net\,O1} * \frac{\partial net\,O1}{\partial out\,O1}$$

$$\frac{\partial E\,O2}{\partial out\,h1} = \frac{\partial E\,O2}{\partial net\,O2} * \frac{\partial net\,O2}{\partial out\,h1}$$



Working of Backpropagation Algorithm with an example

part 3:

calculate Backward propagation error (hidden layer to input layer)

W1,W2,W3,W4

$$\frac{\partial E}{\partial out_{h1}} = \frac{\partial E_{O1}}{\partial net_{O1}} * \frac{\partial net_{O1}}{\partial out_{h1}} + \frac{\partial E_{O2}}{\partial net_{O2}} * \frac{\partial net_{O2}}{\partial out_{h1}}$$

$$\frac{\partial E}{\partial out_{h1}} = \frac{\partial E_{O2}}{\partial net_{O2}} * \frac{\partial net_{O2}}{\partial out_{h1}}$$

$$\frac{\partial E_{O2}}{\partial net_{O2}} = \frac{\partial E_{O2}}{\partial out_{O2}} * \frac{\partial out_{O2}}{\partial net_{O2}}$$

$$\frac{\partial E_{O2}}{\partial out_{O2}} = (\text{out O2} - \text{target O2})$$

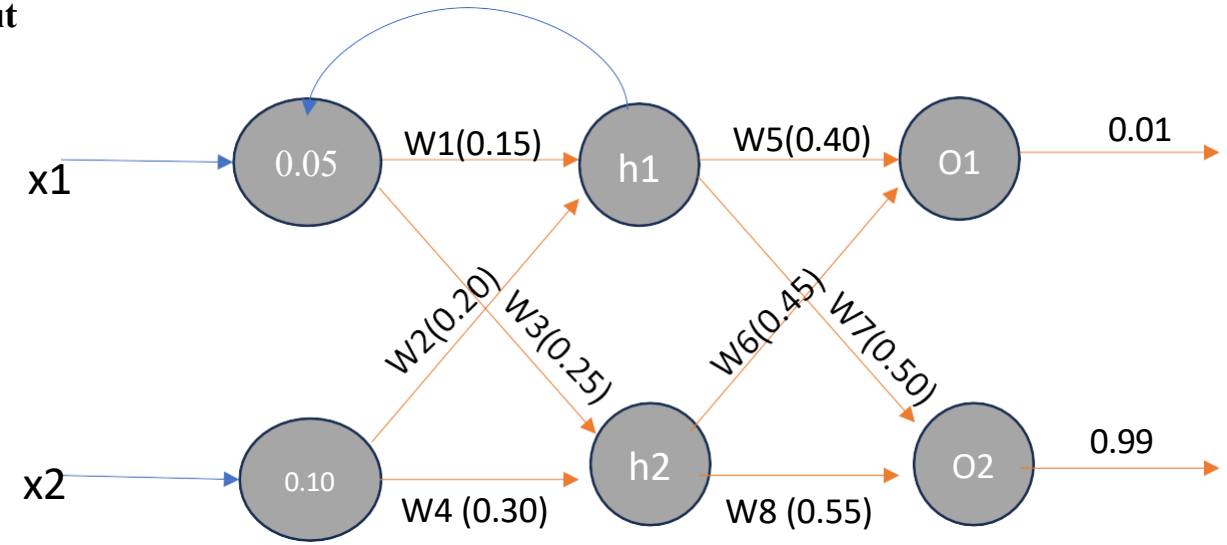
$$= 0.772928465 - 0.99$$

$$= -0.2170771535$$

$$\frac{\partial out_{O2}}{\partial net_{O2}} = \text{out O2} - (1 - \text{out O2})$$

$$= 0.772928465 - (1 - 0.772928465)$$

$$= 0.175510052$$



b1= (0.35) b2= (0.60)

Working of Backpropagation Algorithm with an example

part 3:

calculate Backward propagation error (hidden layer to input layer)

W1,W2,W3,W4

$$\frac{\partial E\ O1}{\partial out\ h1} = \frac{\partial E\ O1}{\partial net\ O1} * \frac{\partial net\ O1}{\partial out\ O1}$$

$$\frac{\partial E\ O2}{\partial out\ h1} = \frac{\partial E\ O2}{\partial net\ O2} * \frac{\partial net\ O2}{\partial out\ h1}$$

$$\frac{\partial E\ O2}{\partial net\ O2} = \frac{\partial E\ O2}{\partial out\ O2} * \frac{\partial out\ O2}{\partial net\ O2}$$

$\frac{\partial E\ O2}{\partial out\ O2}$ = (out O2- target O2)

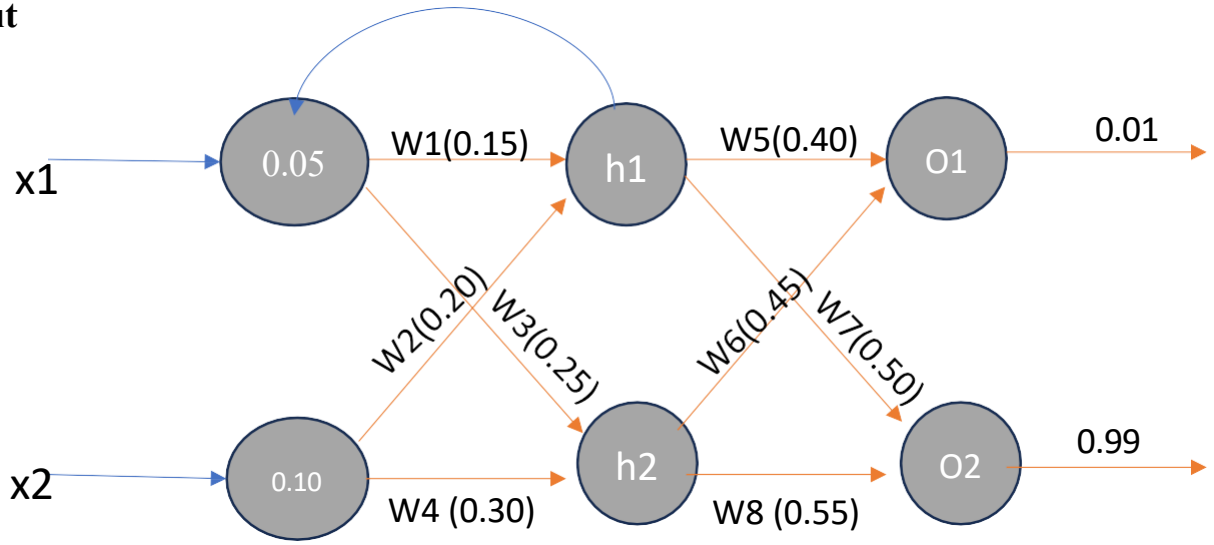
= 0.772928465-0.99

= -0.2170771535

$\frac{\partial out\ O2}{\partial net\ O2}$ = out O2 – (1-out O2)

= 0.772928465 – (1- 0.772928465)

= 0.175510052



b1= (0.35) b2= (0.60)

$\frac{\partial E\ O2}{\partial net\ O2}$ = -0.2170771535 * 0.175510052

= on total of O2 from h1 => W7

$\frac{\partial net\ O2}{\partial net\ O2}$ = 0.50

$\frac{}{\partial out\ h1}$

Working of Backpropagation Algorithm with an example part 3:

Calculate Backward propagation error (hidden layer to input layer)
W1,W2,W3,W4

$$\frac{\partial E}{\partial out\ h1} = \frac{\partial E\ O1}{\partial net\ O1} * \frac{\partial net\ O1}{\partial out\ h1}$$

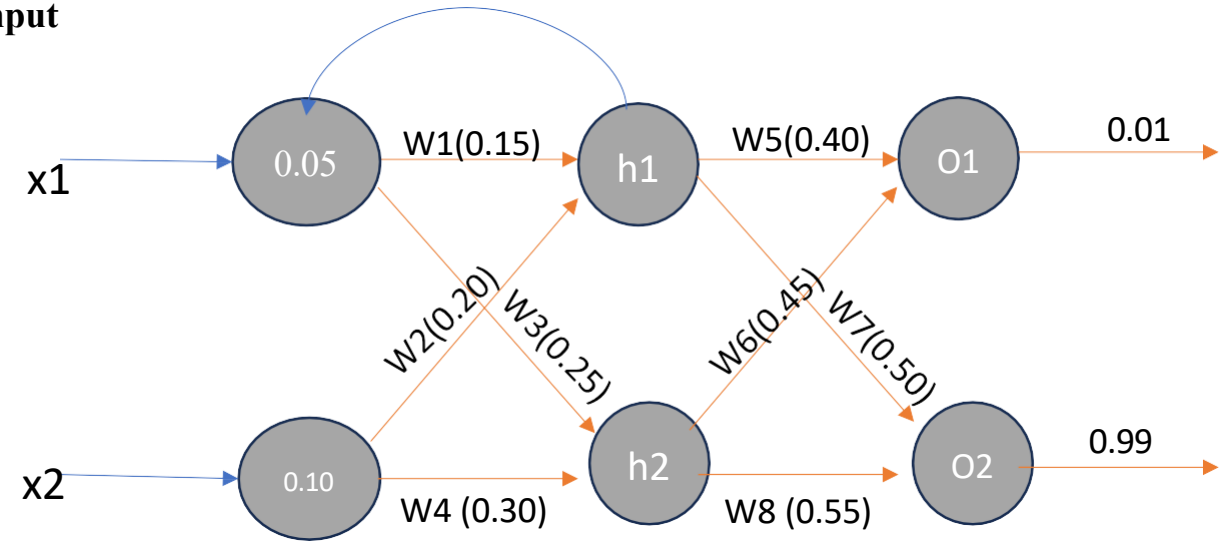
$$\frac{\partial E\ O2}{\partial out\ h1} = \frac{\partial E\ O2}{\partial net\ O2} * \frac{\partial net\ O2}{\partial out\ h1}$$

$$= -0.03809823 * 0.50$$

$$\frac{\partial E\ O2}{\partial net\ O2} = \frac{\partial E\ O2}{\partial out\ O2} * \frac{\partial out\ O2}{\partial net\ O2}$$

$$\frac{\partial E\ O2}{\partial out\ O2}$$

$$\partial out\ O2$$



b1= (0.35) b2= (0.60)

= 0.772928465 – (1- 0.772928465)

= (out O2- target O2)

= 0.772928465- 0.99

= -0.2170771535

= out O2 – (1-out O2)

$\partial E\ O2$ = -0.2170771535 * 0.175510052

= -0.03809823

$\frac{}{\partial\ net\ O2}$ = on total of O2 from h1 => W7

= 0.

$\partial net\ O2$

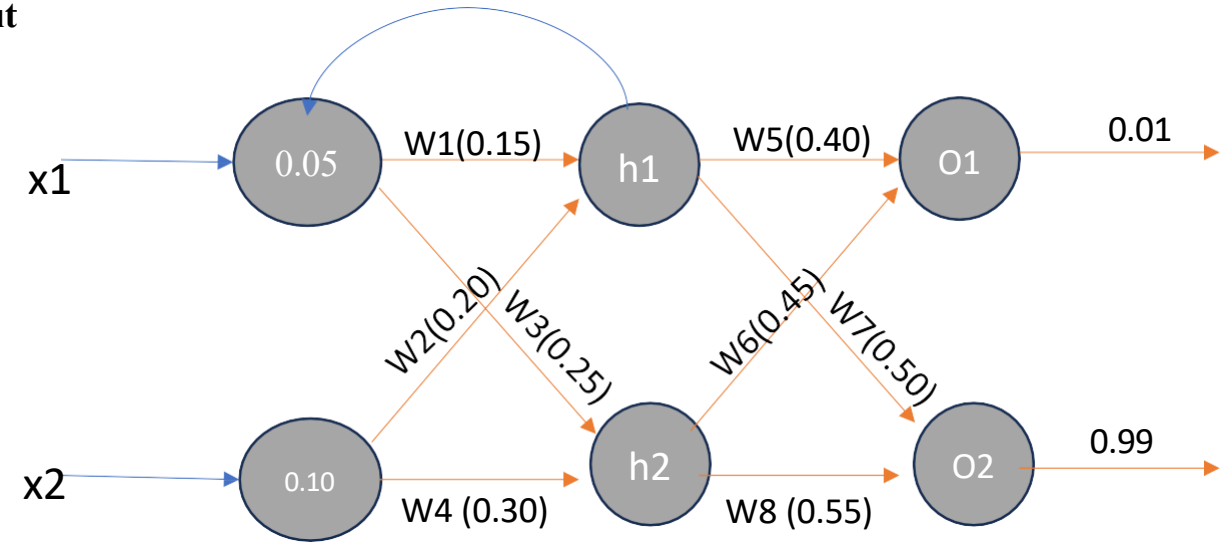
$\frac{}{\partial\ out\ h1}$

Working of Backpropagation Algorithm with an example

part 3:

calculate Backward propagation error (hidden layer to input layer)

W1,W2,W3,W4



b1= (0.35) b2= (0.60)

$$\frac{\partial E\ O1}{\partial out\ h1} = \frac{\partial E\ O1}{\partial net\ O1} * \frac{\partial net\ O1}{\partial out\ h1}$$

$$= 0.37257738 * 0.4$$

$$\frac{\partial E\ O2}{\partial out\ h1} = \frac{\partial E\ O2}{\partial net\ O2} * \frac{\partial net\ O2}{\partial out\ h1}$$

$$\frac{\partial E\ O1}{\partial net\ O1} = \frac{\partial E\ O1}{\partial out\ O1} * \frac{\partial out\ O1}{\partial net\ O1}$$

$$\frac{\partial E\ O1}{} = (out\ O1 - target\ O1) = 0.7513 - 0.01 = 0.7413$$

$$\frac{\partial out\ O1}{} = out\ O1 - (1 - out\ O1) = 0.7513 - (1 - 0.7513) = 0.5026$$

$$\frac{\partial out\ O1}{} = 0.5026$$

$$\frac{\partial net\ O1}{}$$

$$\frac{\partial E\ O1}{} = 0.7413 * 0.5026 = 0.37257738$$

$$\frac{\partial net\ O1}{} = \text{on total of O1 from h1} \Rightarrow W5 = 0.4$$

$$\frac{\partial net\ O1}{}$$

$$\frac{}{\partial out\ h1}$$

Working of Backpropagation Algorithm with an example

part 3:

calculate Backward propagation error (hidden layer to input layer)

W1,W2,W3,W4

$$\frac{\partial E}{\partial W1} = \frac{\partial E}{\partial out\ h1} * \frac{\partial out\ h1}{\partial net\ h1} * \frac{\partial net\ h1}{\partial W1}$$

$$\frac{\partial E}{\partial out\ h1} = \frac{\partial E\ O1}{\partial out\ h1} + \frac{\partial E\ O2}{\partial out\ h1}$$

$$\frac{\partial E\ O1}{\partial out\ h1} = \frac{\partial E\ O1}{\partial net\ O1} * \frac{\partial net\ O1}{\partial out\ O1}$$

=0.37257738*0.4

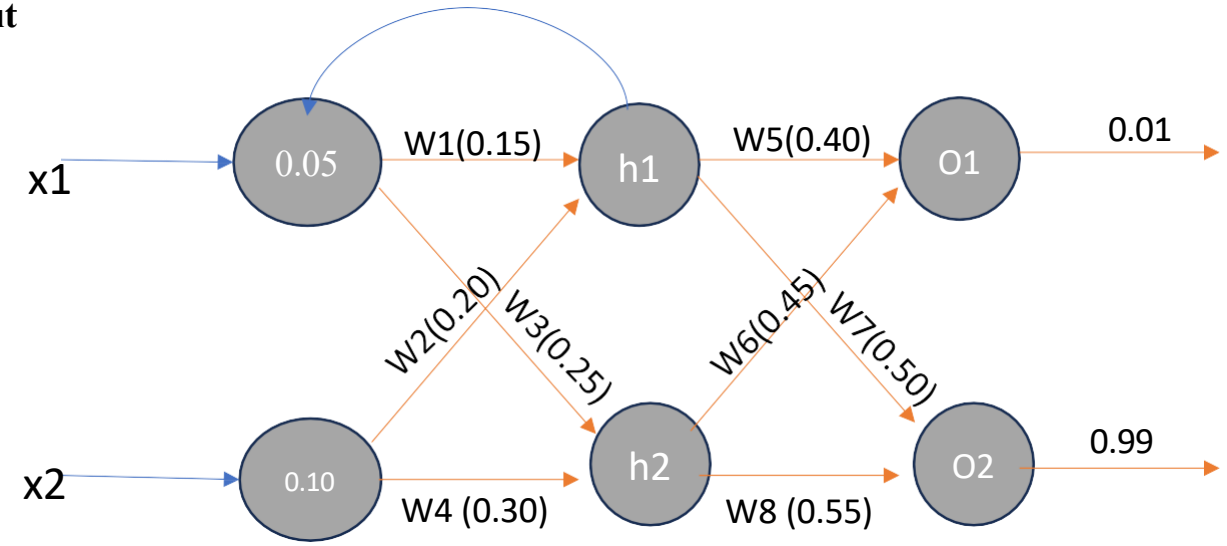
$$\frac{\partial E\ O2}{\partial out\ h1} = \frac{\partial E\ O2}{\partial net\ O2} * \frac{\partial net\ O2}{\partial out\ h1}$$

= -0.03809823 * 0.50

= (0.37257738*0.4) + (-0.03809823 * 0.50)

= 0.019049115+0.149030952

=0.168080067



b1= (0.35) b2= (0.60)

$$\frac{\partial out\ h1}{\partial net\ h1} = out\ h1(1-out\ h1) = 0.5932(1-0.5932)$$

=0.5932*0.4077 =0.24184764

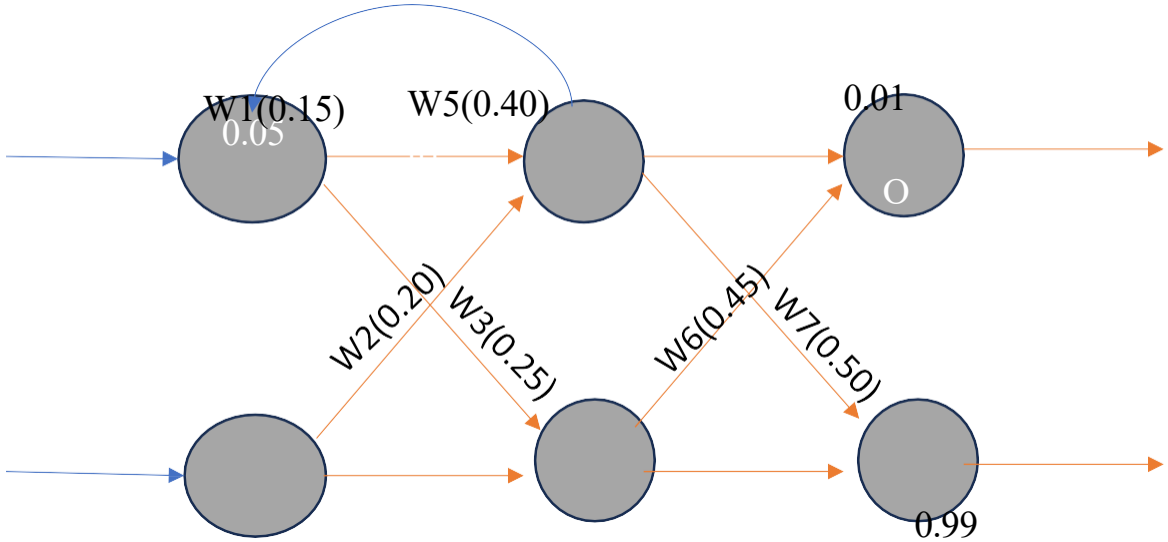
$$\frac{\partial net\ h1}{\partial W1} = \frac{\partial}{\partial w1}(w1x1+w2x2+b1) = (x1+0+0)=0.05$$

Working of Backpropagation Algorithm with an example

part 3:

calculate Backward propagation error (hidden layer to input layer)

W1,W2,W3,W4



x1

$$\frac{\partial Etotal}{\partial out\ h1} = \frac{\partial E\ O1}{\partial out\ h1} + \frac{\partial E\ O2}{\partial out\ h1}$$

= (0.37257738*0.4) + (-0.03809823 * 0.50)

= 0.019049115+0.149030952

=0.168080067

x2

W4 (0.30) W8 (0.55)

b1= (0.35) b2= (0.60)

$$\frac{\partial Etotal}{\partial W1} = \frac{\partial Etotal}{\partial out\ h1} * \frac{\partial out\ h1}{\partial net\ h1} * \frac{\partial net\ h1}{\partial W1}$$

= 0.168080067 * 0.24184764*0.05

= 0.00203248837

$\frac{\partial out\ h1}{\partial net\ h1}$ = out h1(1-out h1) =0.5932(1-0.5932)

=0.5932*0.4077 =0.24184764

*w1 = $W1-n\frac{\partial Etotal}{\partial W1}$

$= 0.15 - (0.6 * 0.00203248837)$

=0.15-0.000304873=0.149695163

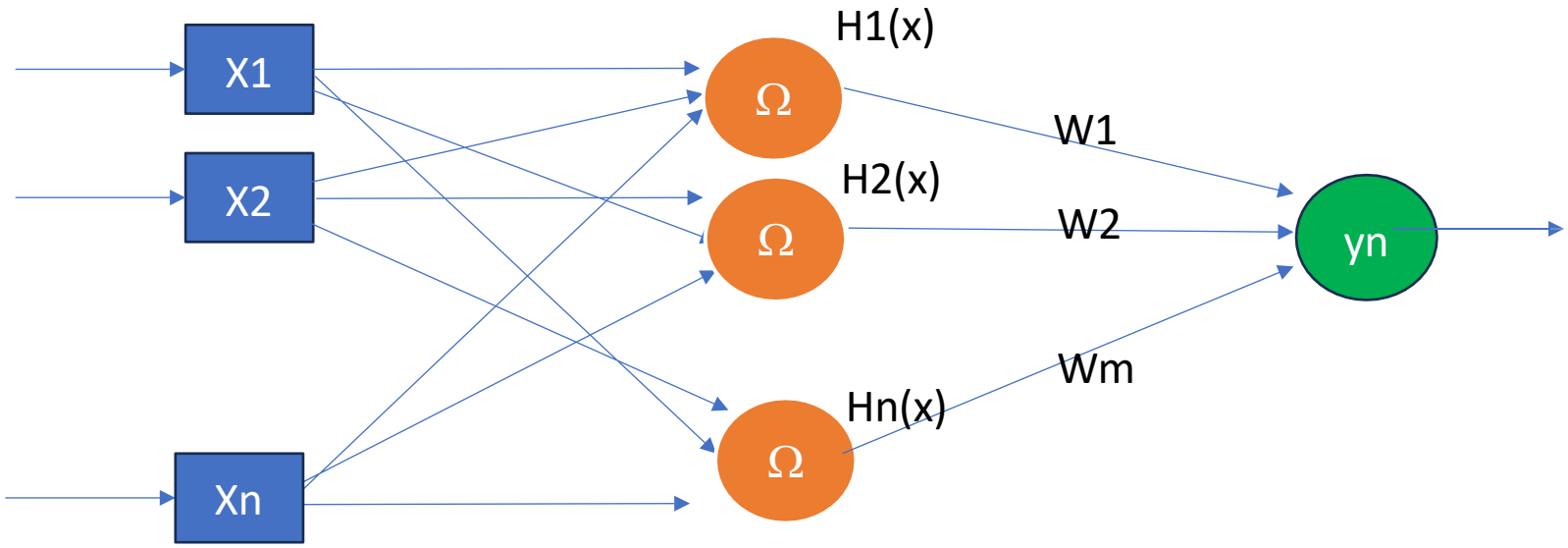
$\frac{\partial net\ h1}{\partial W1} = \frac{\partial}{\partial w1}(w1x1+w2x2+b1) = (x1+0+0)=0.05$

Radial Basis Functions and Splines – Concepts – RBF Network

Single layer perceptron can be used for classifying **linearly Separable** data Multilayer perceptron can be used for classifying **non-linearly Separable** data

Radial Basis Functions is a type of Multilayer perceptron
which has one input layer one output layer with strictly one hidden layer

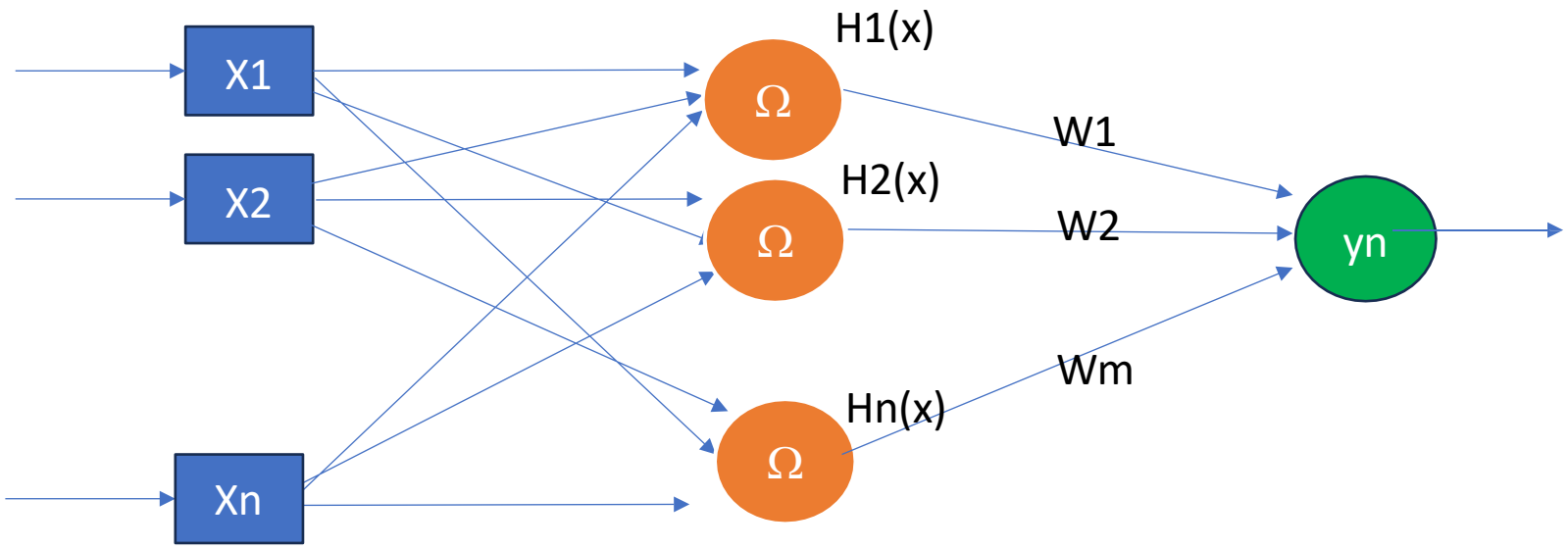
Y



Radial Basis Functions and Splines – Concepts – RBF Network

The hidden layer use a non – linear radial basis function as the **activation function**. Which converts the input parameter into high dimension space which is then fed into the network to linearly separate the problem.

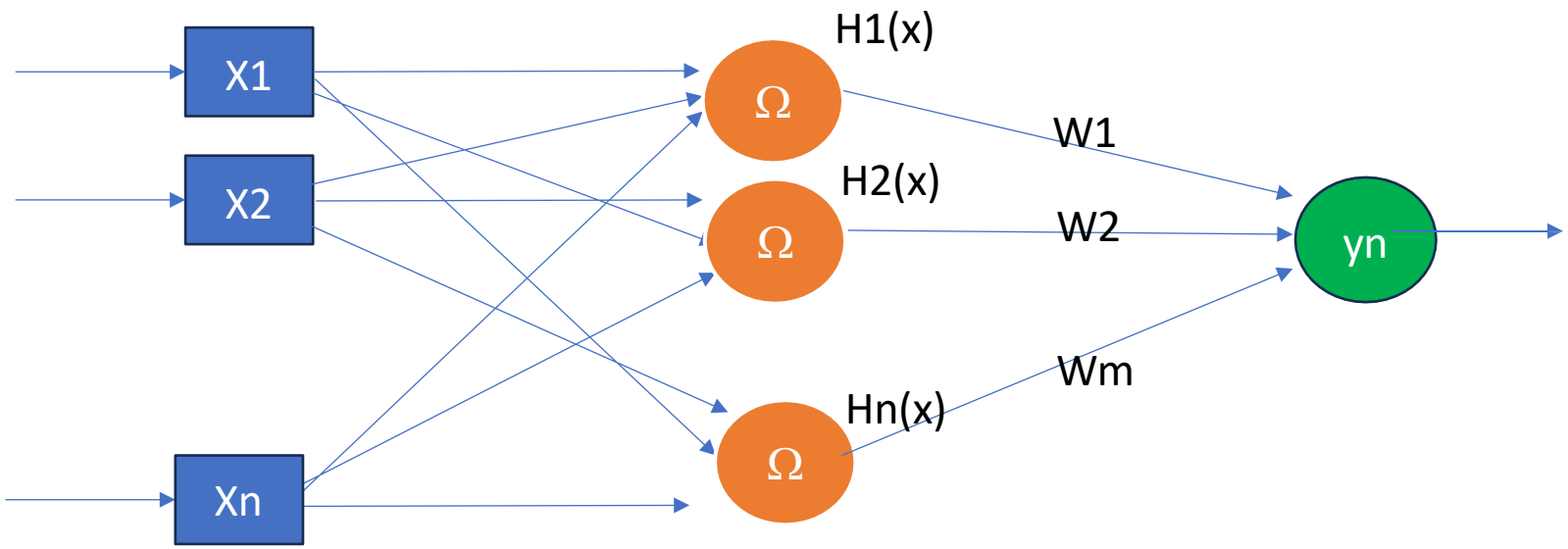
Y



Radial Basis Functions and Splines – Concepts – RBF Network

This cab be used in 1. Classification 2. interpolation 3. function approximation
4. Time series prediction 5. System control

Y



The Gaussian Radial basis function which monotonically decrease with distance from the centre.

$$H(x) = e^{\frac{-(x-r)^2}{r^2}}$$ where x = input

R = radius

C = centre

A multiquadratic RBF which monotonically increase with distance from the centre

$$H(x) = \sqrt{\frac{r^2+(x-c)^2}{r}}$$

Algorithm for RBF

where input : input vector : x1,x2…….xn

Output : yn

Assign random weights for every connection from the hidden layer to the output layer in the network in the range [-1 to +1]

Forward phase :

Calculate input and output in the input layer (input layer as direct transfer function) where the output of the nodes equals the input
Input at node I in the Input layer is **Ii=xi** where xi is the input received at node I

Output at node I “Oi” in the input layer is **Oi = Ii**

Step2: for each node j in the hidden layer, find the center (c) and the variance r
define hidden layer neurons with gaussian RB

$$Hi(x) = e^{\frac{-(xi-Cj)^2}{r^2}}$$

compute (x-cj) applying Euclidian distance measure between x and cj

Step 3 : for each node k in the output layer, compute linear weighted sum of the output of each neuron k from the hidden layer neurons j.

$$f_k(x) = \sum_{j=1}^{m} Wij\ Hj(x)$$

where Wij = weight in the link from the hidden layer neuron j to the output layer

H(x) is the **output of a hidden layer** neuron j for an input vector x

Backward phase :

1. Train the hidden layer using Back Propagation
2. update the weights between the hidden layer and output layer

RBF Networks are conceptually similar to K-Nearest Neighbor (k-NN) models, though their implementation is distinct. The fundamental idea is that an item's predicted target value is influenced by nearby items with similar predictor variable values. Here's how RBF Networks operate:

1. **Input Vector**: The network receives an n-dimensional input vector that needs classification or regression.
2. **RBF Neurons**: Each neuron in the hidden layer represents a prototype vector from the training set. The network computes the Euclidean distance between the input vector and each neuron's center.
1. **Activation Function**: The Euclidean distance is transformed using a Radial Basis Function (typically a Gaussian function) to compute the neuron's activation value. This value decreases exponentially as the distance increases.
2. **Output Nodes**: Each output node calculates a score based on a weighted sum of the activation values from all RBF neurons. For classification, the category with the highest score is chosen.

Key Characteristics of RBFs

Radial Basis Functions: These are real-valued functions dependent solely on the distance from a central point. The Gaussian function is the most commonly used type.

- **Dimensionality**: The network's dimensions correspond to the number of predictor variables.
- **Center and Radius**: Each RBF neuron has a center and a radius (spread). The radius affects how broadly each neuron influences the input space.

Architecture of RBF Networks

The architecture of an RBF Network typically consists of three layers:

Input Layer

Function: After receiving the input features, the input layer sends them straight to the hidden layer.

Components: It is made up of the same number of neurons as the characteristics in the input data. One feature of the input vector corresponds to each neuron in the input layer.

Hidden Layer

Function: This layer uses radial basis functions (RBFs) to conduct the non-linear transformation of the input data.

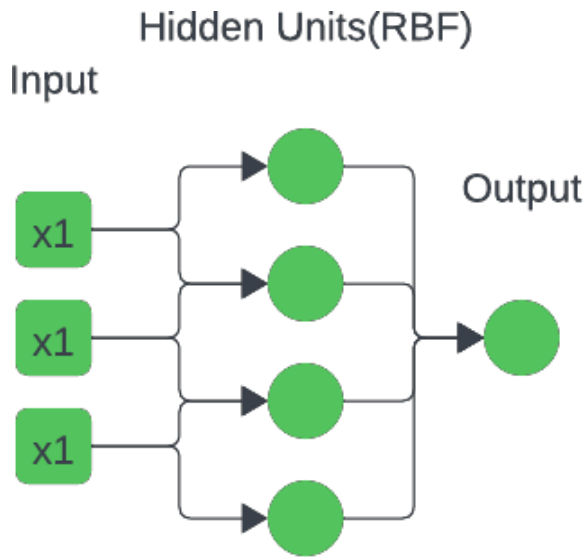
Components: Neurons in the buried layer apply the RBF to the incoming data. The Gaussian function is the RBF that is most frequently utilized.

RBF Neurons: Every neuron in the hidden layer has a spread parameter (σ) and a center, which are also referred to as prototype vectors. The spread parameter modulates the distance between the center of an RBF neuron and the input vector, which in turn determines the neuron's output.

Output Layer

Function: The output layer uses weighted sums to integrate the hidden layer neurons' outputs to create the network's final output.

Components: It is made up of neurons that combine the outputs of the hidden layer in a linear fashion. To reduce the error between the network's predictions and the actual target values, the weights of these combinations are changed during training.


Hidden Units(RBF)
Input
Output
x1
x1
x1

Training Process of radial basis function neural network

An RBF neural network must be trained in three stages: choosing the center's, figuring out the spread parameters, and training the output weights.

Step 1: Selecting the Centers

Techniques for Centre Selection: Centre's can be picked at random from the training set of data or by applying techniques such as k-means clustering.

K-Means Clustering: The center's of these clusters are employed as the center's for the RBF neurons in this widely used center selection technique, which groups the input data into k groups.

Step 2: Determining the Spread Parameters

The spread parameter (σ) governs each RBF neuron's area of effect and establishes the width of the RBF.

Calculation: The spread parameter can be manually adjusted for each neuron or set as a constant for all neurons. Setting σ based on the separation between the center's is a popular method, frequently accomplished with the help of a heuristic like dividing the greatest distance between canters by the square root of twice the number of center's

Step 3: Training the Output Weights

Linear Regression: The objective of linear regression techniques, which are commonly used to estimate the output layer weights, is to minimize the error between the anticipated output and the actual target values.

Pseudo-Inverse Method: One popular technique for figuring out the weights is to utilize the pseudo-inverse of the hidden layer outputs matrix

Advantages of RBF Networks

1. **Universal Approximation**: RBF Networks can approximate any continuous function with arbitrary accuracy given enough neurons.
2. **Faster Learning**: The training process is generally faster compared to other neural network architectures.
3. **Simple Architecture**: The straightforward, three-layer architecture makes RBF Networks easier to implement and understand.

Applications of RBF Networks

Classification: RBF Networks are used in pattern recognition and classification tasks, such as speech recognition and image classification.
Regression: These networks can model complex relationships in data for prediction tasks.
Function Approximation: RBF Networks are effective in approximating non-linear functions.

Example of RBF Network

Consider a dataset with two-dimensional data points from two classes. An RBF Network trained with 20 neurons will have each neuron representing a prototype in the input space. The network computes category scores, which can be visualized using 3-D mesh or contour plots. Positive weights are assigned to neurons belonging to the same category and negative weights to those from different categories. The decision boundary can be plotted by evaluating scores over a grid.

The **Curse of Dimensionality in Machine Learning** arises when working with high-dimensional data, leading to increased computational complexity, overfitting, and spurious correlations. Techniques like dimensionality reduction, feature selection, and careful model design are essential for mitigating its effects and improving algorithm performance. Navigating this challenge is crucial for unlocking the potential of high-dimensional datasets and ensuring robust machine-learning solutions.

What is the Curse of Dimensionality?

The Curse of Dimensionality refers to the phenomenon where the efficiency and effectiveness of algorithms deteriorate as the dimensionality of the data increases exponentially.

- In high-dimensional spaces, data points become sparse, making it challenging to discern meaningful patterns or relationships due to the vast amount of data required to adequately sample the space.
- The Curse of Dimensionality significantly impacts machine learning algorithms in various ways. It leads to increased computational complexity, longer

training times, and higher resource requirements. Moreover, it escalates the risk of overfitting and spurious correlations, hindering the algorithms' ability to generalize well to unseen data.

How to Overcome the Curse of Dimensionality?

To overcome the curse of dimensionality, you can consider the following strategies:

Dimensionality Reduction Techniques:

- Feature Selection: Identify and select the most relevant features from the original dataset while discarding irrelevant or redundant ones. This reduces the

dimensionality of the data, simplifying the model and improving its efficiency.

- Feature Extraction: Transform the original high-dimensional data into a lower-dimensional space by creating new features that capture the essential information. Techniques such as Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE) are commonly used for feature extraction.

Data Preprocessing:

- Normalization: Scale the features to a similar range to prevent certain features from dominating others, especially in distance-based algorithms.
- Handling Missing Values: Address missing data appropriately through imputation or deletion to ensure robustness in the model training process.

Feature Selection and Dimensionality Reduction

1. **Feature Selection:** SelectKBestis used to select the top k features based on a specified scoring function (f_classifin this case). It selects the features that are most likely to be related to the target variable.
2. **Dimensionality Reduction:** PCA(Principal Component Analysis) is then used to further reduce the dimensionality of the selected features. It transforms the data into a lower-dimensional space while retaining as much variance as possible.

Training the classifiers

Training Before Dimensionality Reduction: Train a Random Forest classifier (clf_before) on the original scaled features (X_train_scaled) without

1. **Evaluation Before Dimensionality Reduction:** Make predictions (y_pred_before) on the test set (X_test_scaled) using the classifier trained before dimensionality reduction, and calculate the accuracy (accuracy_before) of the model.
2. **Training After Dimensionality Reduction:** Train a new Random Forest classifier (clf_after) on the reduced feature set (X_train_pca) after dimensionality reduction.
3. **Evaluation After Dimensionality Reduction:** Make predictions (y_pred_after) on the test set (X_test_pca) using the classifier trained after dimensionality reduction, and calculate the accuracy (accuracy_after) of the model.

Interpolation in Machine Learning

The practice of guessing unknown values based on available data points is known as interpolation in the context of machine learning. In tasks like regression and classification, where the objective is to predict outcomes based on input features, it is important. Machine learning algorithms are capable of producing well-informed predictions for unknown or intermediate values by interpolating between known data points.

Interpolation Types

The intricacy and applicability of interpolation techniques varied for various kinds of data. Typical forms of interpolation include the following:

- **Interpolation in Linear Form**: By assuming a linear relationship between neighboring data points, linear interpolation calculates values along a straight line that connects them.
- **Equation-Based Interpolation:** By fitting a polynomial function to the data points, polynomial interpolation produces a more flexible approximation that is capable of capturing nonlinear relationships.
- **Interpolation of Splines**: By building piece wise polynomial functions that connect data points gradually, spline interpolation prevents abrupt changes in the interpolated function.
- **Interpolation of Radial Basis Function**: Values based on the separations between data points are interpolated using radial basis functions in radial basis function interpolation.

Interpolation in Linear Form

A straightforward but efficient technique for guessing values between two known data points is linear interpolation.

The value of y at any intermediate point x can be approximated using the following formula, given two data points: (1, 1) (x 1 ,y 1) and (2 , 2) (x 2 ,y 2). i.e y=y_1+(x−x_1)·(y_2−y_1)/x_2−x_1

Implementation

- This code snippet illustrates linear interpolation using **LinearNDInterpolator** from SciPy.
- It randomly generates 10 data points in 2D space with corresponding values.
- The **LinearNDInterpolator function** constructs an interpolation function based on these points. It then interpolates the value at a specified point and visualizes both the data points and the interpolated point on a scatter plot.
- Finally, the interpolated value at the specified point is printed.

Polynomial Interpolation

- Polynomial interpolation is a method of estimating values between known data points by fitting a polynomial function to the data. The goal is to find a polynomial that passes through all the given points.
- This method is useful for approximating functions that may not have a simple analytical form. One common approach to polynomial interpolation is to use the Lagrange polynomial or Newton's divided differences method to construct the interpolating polynomial.

Implementation

- This article demonstrates polynomial interpolation using the **interp1d function** from SciPy.
- It begins by generating sample data representing points along a sine curve. The interp1d function is then applied with a cubic spline interpolation method to approximate the curve between the data points.
- Finally, the original data points and the interpolated curve are visualized using matplotlib, showcasing the effectiveness of polynomial interpolation in approximating the underlying function from sparse data points.

```
import numpy as np
from scipy.interpolate import interp1d import matplotlib.pyplot as plt

# Generate some sample data x = np.linspace(0,
10, 10) y = np.sin(x)

# Perform polynomial interpolation poly_interp = interp1d(x, y,
kind='cubic')

# Generate points for plotting the interpolated curve x_interp =
np.linspace(0, 10, 100)
y_interp = poly_interp(x_interp)

# Plot the original data and the interpolated curve plt.scatter(x, y,
label='Original Data')
plt.plot(x_interp, y_interp, color='red', label='Polynomial
Interpolation') plt.xlabel('X')
plt.ylabel('Y')
plt.title('Polynomial Interpolation with interp1d') plt.legend()
plt.grid(True) plt.show()
```

Applications Of Interpolation in Machine Learning

Interpolation is a method used in various fields for estimating values between known data points. Some common applications of interpolation include:

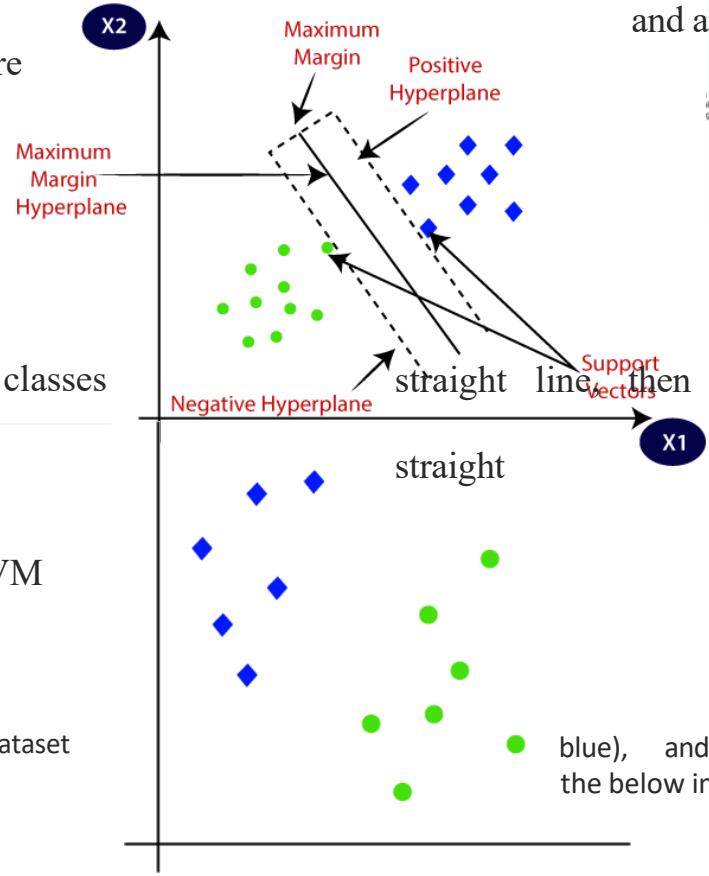
- **Image Processing:** Interpolation is used to resize images by estimating the values of pixels in the resized image based on the values of neighboring pixels in the original image.
- **Computer Graphics:** In computer graphics, interpolation is used to generate smooth curves and surfaces, such as Bezier curves and surfaces, which are used to create shapes and animations.
- **Numerical Analysis:** Interpolation is used in numerical analysis to approximate the value of a function between two known data points. This is useful in areas such as finite element analysis and computational fluid dynamics.
- **Signal Processing:** In signal processing, interpolation is used to upsample signals, which increases the number of samples in a signal without changing its frequency content.
- **Mathematical Modeling:** Interpolation is used in mathematical modeling to estimate unknown values based on known data points, such as in the construction of mathematical models for physical systems.
- **Geographic Information Systems (GIS):** Interpolation is used in GIS to estimate values of geographical features, such as elevation or temperature, at locations where data is not available.
- **Audio Processing:** In audio processing, interpolation is used to resample audio signals

Support Vector Machine Algorithm

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for
Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that
classified using a decision boundary or hyperplane:



Types of SVM
SVM can be of two types:

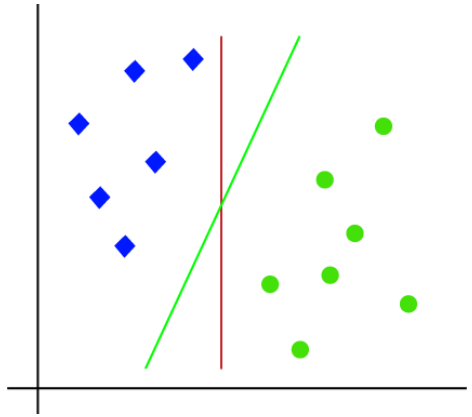
Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

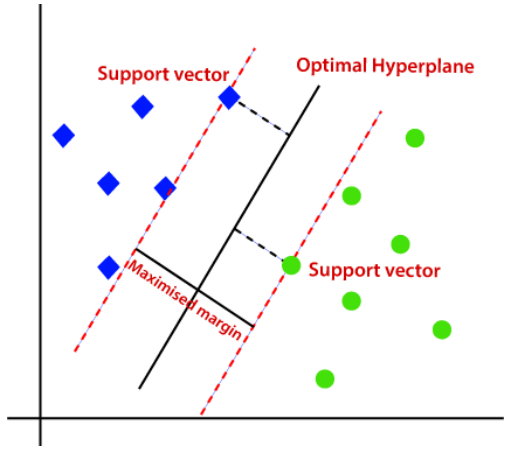
Linear SVM:

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:

So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:



the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the
classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin.

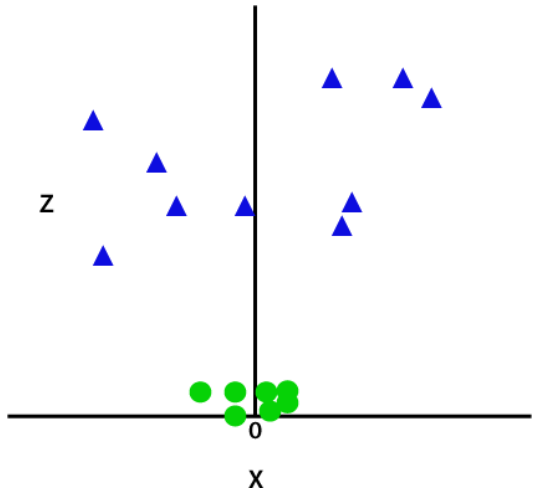


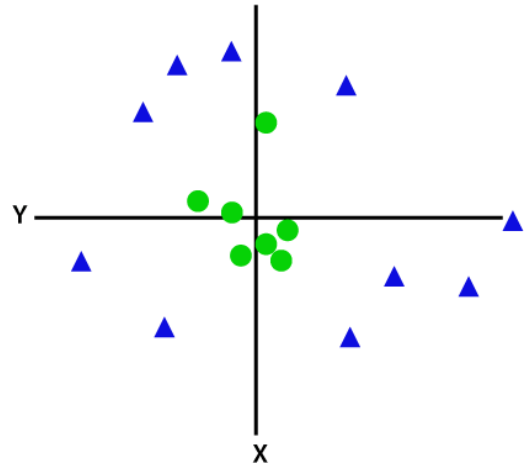
The **hyperplane** with maximum margin is called the **optimal hyperplane**.

Non-Linear SVM:

If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:

So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as: $z= x^2 + y^2$ By adding the third dimension, the sample space will become as below image:





become as:

Hence we get a circumference of radius 1 in case of non-linear data.

UNIT-III

Learning with Trees and Decision Trees

1. Learning with Trees

Introduction

Learning with Trees is a supervised machine learning technique in which a model learns by creating a tree-like structure from the training data. Every internal node represents a decision based on an attribute, every branch represents the outcome of that decision, and every leaf node represents the final prediction or class label.

Tree-based learning is one of the most popular machine learning approaches because it is simple to understand, easy to visualize, and capable of solving both classification and regression problems.

For example, suppose a bank wants to decide whether a customer is eligible for a loan. The decision depends on factors such as salary, age, employment status, and credit score. Instead of manually checking each application, a decision tree learns the decision-making process from historical data and predicts whether a new customer should receive the loan.

Why Tree-Based Learning?

Many real-world problems involve making a sequence of decisions. Human beings naturally think in the form of questions.

Example:

Should a student receive a scholarship?

Question 1:
Is the student's percentage greater than 85?

If Yes → Go to Question 2

If No → Scholarship Rejected

Question 2:
Is the family income below ₹3,00,000?

If Yes → Scholarship Approved

If No → Partial Scholarship

This sequence of questions forms a decision tree.

Basic Components of a Tree

1. Root Node

The first node of the tree.

It represents the entire dataset.

Example:

Is Salary > 50000?

Decision Node

A node where the data is split based on a condition.

Example:

Age > 30?

Leaf Node

The final node.

It contains the prediction.

Example:

Approved

or

Rejected

4. Branch

A branch connects two nodes.

Example

Salary > 50000

Yes
|
V
Loan Approved

Structure of a Decision Tree

Salary > 50000

/ \

Yes No

Age>30 Reject Loan

/ \

Yes No

Approve Review

Every path from the root to a leaf re

How Learning Takes Place?

Suppose we have the following training dataset.

Salary	**Experience**	**Loan**
High	High	Yes
High	Low	Yes
Low	High	No
Low	Low	No

The algorithm examines every feature and determines which feature best separates the classes.

It may decide that Salary is the best feature.

The first split becomes

Salary?

High Low

Yes No

hus the model learns automatically from the data.

Advantages of Tree Learning

1. Easy to understand.
2. Easy to visualize.
3. No complex mathematical calculations.
4. Handles numerical and categorical data.
5. Works for classification and regression.
6. Requires less data preprocessing.
7. Feature scaling is not required.

Disadvantages

1. Can overfit training data.
2. Small changes in data may produce different trees.
3. Large trees become complex.
4. Greedy algorithms may not produce the optimal tree.

Applications

Medical diagnosis

Loan approval

Fraud detection

Email spam detection

Customer segmentation

Disease prediction

Student performance prediction

Crop recommendation

Real-Life Example

A doctor diagnoses dengue.

Question 1

Does the patient have fever?

If No

Healthy

If Yes

Question 2

Platelet count below 1.5 lakh?

If Yes

Dengue

If No

Viral Fever

The doctor is unconsciously following a decision tree.

2. Decision Trees

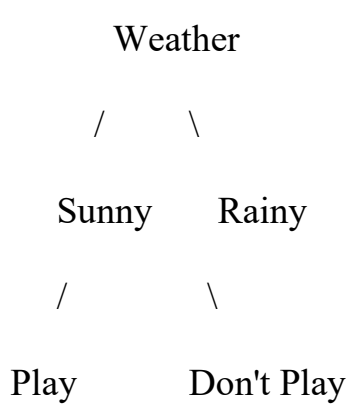
Definition

A Decision Tree is a supervised machine learning algorithm that predicts the output by asking a sequence of questions. The algorithm repeatedly

divides the dataset into smaller subsets until all records in a subset belong to the same class.

A decision tree resembles an upside-down tree.

Example



Working Principle

A decision tree works using Divide and Conquer.

The complete dataset is divided into smaller subsets.

Each subset is again divided.

This process continues until all records belong to one class.

Steps of Decision Tree Algorithm

Step 1

Take the training dataset.

Step 2

Calculate the best attribute for splitting.

Step 3

Make the root node.

Step 4

Split the dataset.

Step 5

Repeat the process for every subset.

Step 6

Stop when every leaf contains only one class.

Example Dataset

Weather	Wind	Play Cricket
Sunny	Weak	Yes
Sunny	Strong	Yes
Rainy	Weak	No

Rainy Strong No

The tree becomes

Weather

/ \

Sunny Rainy

Yes No

Prediction

Weather = Sunny

Output = Play Cricket

Weather = Rainy

Output = Don't Play

Important Terminologies

Parent Node

A node that is divided into child nodes.

Example

Weather

Child Node

Nodes obtained after splitting.

Sunny

Rainy

Sibling Nodes

Nodes having the same parent.

Example

Sunny Rainy

Subtree

A small tree inside the main tree.

Depth

The number of edges from the root node to a node.

Height

The longest path from the root node to a leaf node.

Splitting

Splitting means dividing the dataset into smaller groups.

Example

Students

Marks > 60?

Yes

No

Stopping Criteria

The tree stops growing when

• All records belong to one class.

• No more attributes are available.

• Maximum depth is reached.

• Minimum number of samples is reached.

Overfitting

Sometimes the tree becomes too large and memorizes the training data.

Example

100% Training Accuracy

65% Testing Accuracy

This is called overfitting.

Underfitting

If the tree is too small, it cannot learn patterns.

Example

Training Accuracy = 55%

Testing Accuracy = 54%

The model performs poorly on both datasets.

Pruning

Pruning means removing unnecessary branches from the tree.

Types

Pre-Pruning

Stop the tree before it becomes very large.

Post-Pruning

Grow the complete tree and then remove unnecessary branches.

Pruning improves generalization and reduces overfitting.

Decision Tree Algorithms

1. ID3 (Iterative Dichotomiser 3)

Uses Information Gain.

1. C4.5

Uses Gain Ratio.

Handles continuous attributes and missing values.

1. CART (Classification and Regression Trees)

Uses Gini Index.

Supports both classification and regression.

1. CHAID

Uses Chi-Square test.

Suitable for marketing and survey analysis.

Applications

Medical diagnosis

Disease prediction

Bank loan approval

Credit card fraud detection

Face recognition

Customer churn prediction

Recommendation systems

Agriculture

Spam filtering

Intrusion detection systems

Constructing Decision Trees

A decision tree is constructed by repeatedly dividing the dataset into smaller and smaller groups until all records in each group belong to the same class. This process is called **recursive partitioning** because the same splitting process is repeated at every level of the tree.

The main objective while constructing a decision tree is to find the **best attribute** that separates the data into pure groups.

For example, suppose we want to predict whether a person buys a laptop.

Age	Income	Student	Buy Laptop
Young	High	No	No
Young	High	No	No
Middle	High	No	Yes
Senior	Medium	No	Yes
Senior	Low	Yes	Yes
Senior	Low	Yes	No
Middle	Low	Yes	Yes
Young	Medium	No	No

The algorithm examines every attribute (Age, Income, Student) and determines which one separates "Yes" and "No" most effectively.

Steps to Construct a Decision Tree

Step 1: Collect the Training Dataset

The algorithm first collects all available training samples.

Example

Student	Attendance	Pass
A	Good	Yes
B	Poor	No
C	Good	Yes
D	Poor	No

Initially all records belong to one dataset.

Entire Dataset

Step 2: Calculate Impurity

The algorithm checks how mixed the dataset is.

If all records belong to one class

Yes
Yes
Yes
Yes

Impurity = 0

If records belong to different classes

Yes
No
Yes
No

Impurity becomes high.

Decision trees try to **reduce impurity** after every split.

Entropy

Entropy measures the uncertainty or impurity in a dataset.

If all records belong to one class,

Entropy = 0

If records are equally divided,

Entropy = Maximum

Formula

Entropy(S)=−∑pilog2(pi)Entropy(S)=-\sum p_i\log_2(p_i)Entropy(S)=−∑pilog2(pi)

Where

S = Dataset

pi = Probability of each class

Example 1

Dataset

Result
Yes
Yes
Yes
Yes

Probability

Yes = 4/4 = 1

No = 0

Entropy

$= -(1 \times \log_2 1)$

$= 0$

Therefore

Entropy = 0

The dataset is perfectly pure.

Example 2

Dataset

Result
Yes
Yes
No
No

Probability

Yes = 2/4 = 0.5

No = 2/4 = 0.5

Entropy

$= -(0.5 \log_2 0.5 + 0.5 \log_2 0.5)$

= 1

Entropy = 1

This represents maximum uncertainty.

Interpretation of Entropy

Entropy	**Meaning**
0	Pure dataset
0.5	Partially mixed
1	Completely mixed

The objective of a decision tree is always

Maximum Information Gain

or

Minimum Entropy

Information Gain

Information Gain tells us how much uncertainty decreases after splitting.

Formula

Information Gain

= Entropy(Parent)

− Weighted Entropy(Children)

The attribute having the **highest Information Gain** becomes the root node.

Simple Example

Suppose

Entropy before split

= 1

Entropy after split

= 0.2

Information Gain

= 1 − 0.2

= 0.8

A higher value means a better split.

Example

Suppose three attributes produce

Attribute	Information Gain
Age	0.72
Income	0.41
Student	0.18

The algorithm selects

Age

because it has the highest Information Gain.

Choosing the Root Node

Suppose

Attribute	Gain
Outlook	0.246
Humidity	0.151
Wind	0.048
Temperature	0.029

The highest gain is

Outlook

Therefore

Outlook becomes the Root Node.

Outlook

/ | \

Sunny Rainy Overcast

Recursive Partitioning

After selecting the root node,

the remaining subsets are again divided.

Example

Dataset

↓

Split using Age

↓

Young

Middle

Senior

↓

Split each subset again

↓

Continue until every leaf contains one class

The algorithm repeats the same process recursively.

Stopping Conditions

The algorithm stops constructing the tree when

1. All samples belong to one class.

Example

Yes

Yes

Yes

No further split is required.

1. No attributes are left.

1. Dataset becomes empty.

1. Maximum depth is reached.

1. Minimum number of samples is reached.

Splitting Example

Suppose

Salary	Loan
High	Yes
High	Yes
Low	No
Low	No

Split

Salary

High

Low

Result

High → Yes

Low → No

Perfect split.

Good Split vs Bad Split

Good Split

Salary

High

Yes

Low

No

Every child contains only one class.

Bad Split

Salary

High

Yes

No

Low

Yes

No

Classes are still mixed.

Therefore this split is not useful.

Advantages of Information Gain

1. Easy to calculate.
2. Produces accurate trees.
3. Widely used in ID3.
4. Reduces uncertainty.

Limitations of Information Gain

Information Gain prefers attributes having many distinct values.

Example

Suppose an attribute

Student_ID

Every record has a unique ID.

Information Gain becomes very high,

but Student_ID is useless for prediction.

To overcome this problem,

C4.5 introduced **Gain Ratio**.

Gain Ratio

Gain Ratio improves Information Gain by penalizing attributes with many unique values.

Formula

Gain Ratio

= Information Gain / Split Information

The attribute with the highest Gain Ratio is selected.

Example

Suppose

Attribute	Gain	Gain Ratio
Student ID	0.95	0.10
Age	0.72	0.65

Information Gain prefers Student ID.

Gain Ratio selects Age because it provides a better and more meaningful split.

Gini Index

The CART algorithm does not use Entropy.

Instead, it uses the **Gini Index** to measure impurity.

Formula

Gini=1−∑pi2Gini = 1 - \sum p_i^2Gini=1−∑pi2

Where

pip_ipi = Probability of each class.

Lower Gini value indicates a purer dataset.

Example

Dataset

Result
Yes
Yes
No
No

Probability

Yes = 0.5

No = 0.5

Gini

= 1 − (0.5² + 0.5²)

= 1 − (0.25 + 0.25)

= 0.5

Suppose another dataset

Yes

Yes

Yes

No

Probability

Yes = 0.75

No = 0.25

Gini

$= 1 - (0.75^2 + 0.25^2)$

$= 1 - (0.5625 + 0.0625)$

$= 0.375$

Since **0.375 < 0.5**, the second dataset is purer.

Difference Between Entropy and Gini Index

Entropy	Gini Index
Used in ID3 and C4.5	Used in CART
Uses logarithm	Does not use logarithm
Slightly slower	Faster
Measures uncertainty	Measures impurity
Value ranges from 0 to 1	Value ranges from 0 to 0.5 (binary classification)

Real-Life Example

Imagine a teacher wants to divide students into groups.

Instead of randomly creating groups, the teacher asks questions:

- Are the marks above 80?
- Is attendance above 75%?
- Have assignments been submitted?

Each answer divides the students into smaller groups. The teacher continues asking questions until each group contains students with similar performance.

This is exactly how a **decision tree is constructed**—by repeatedly asking the most informative question at each step until the data is organized into pure groups.

Classification and Regression Trees (CART)

Introduction

Classification and Regression Trees (CART) is a supervised machine learning algorithm developed by **Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone** in 1984.

Unlike ID3 and C4.5, which mainly focus on classification problems, CART can solve **both classification and regression problems**.

The main idea behind CART is to divide the dataset into smaller and smaller subsets until each subset becomes as pure as possible.

CART always creates a **Binary Tree**, meaning each internal node has only **two branches**.

Why CART?

Earlier algorithms like ID3 use Information Gain and C4.5 uses Gain Ratio. These algorithms have some limitations.

For example,

A hospital wants to predict whether a patient has diabetes.

Patient information includes:

- Age
- Weight
- Blood Sugar
- Blood Pressure

CART automatically finds the best feature and divides the patients into two groups repeatedly until accurate predictions are obtained.

Definition

Classification and Regression Trees (CART) is a decision tree algorithm that uses the **Gini Index** for classification problems and **Mean Squared Error (MSE)** for regression problems. It always performs **binary splitting**.

Features of CART

1. Works for Classification.
2. Works for Regression.
3. Uses Binary Splitting.
4. Uses Gini Index for Classification.
5. Uses Mean Squared Error for Regression.
6. Easy to interpret.
7. Supports numerical and categorical attributes.

Binary Splitting

The most important characteristic of CART is **Binary Splitting**.

Every node is divided into only **two child nodes**.

Example

Instead of

Weather

Sunny
Rainy
Cloudy

CART converts it into

Weather

Weather

/ \

Sunny Others

or

Weather

/ \

Rainy Not Rainy

Only **two branches** are created.

Structure of CART

Example

Suppose a bank wants to approve loans.

Salary > 50000

/ \

Yes No

Experience>5 Reject

/ \

Yes No

Approve Review

Every decision divides the data into exactly **two parts**.

Classification Tree

A Classification Tree predicts **categorical outputs**.

Examples

Disease

Diabetes

No Diabetes

Loan

Approved

Rejected

Email

Spam

Not Spam

Student

Pass

Fail

The output always belongs to a category.

Example of Classification Tree

Dataset

Salary	**Loan**
High	Yes
High	Yes
Low	No
Low	No

Tree

Salary

High

Loan = Yes

Low

Loan = No

New Customer

Salary = High

Prediction

Loan Approved

Regression Tree

A Regression Tree predicts **continuous numerical values**.

Examples

House Price

₹50 Lakhs

₹70 Lakhs

₹1 Crore

Temperature

25°C

32°C

Salary Prediction

₹45,000

₹80,000

Rainfall Prediction

120 mm

Instead of categories, regression trees predict numbers.

Example of Regression Tree

Suppose

Experience	Salary
1	25000
2	30000
4	45000
8	80000

Tree

Experience > 3

Yes

Salary = 80000

No

Salary = 30000

Prediction

Experience = 2 years

Predicted Salary

₹30,000

Gini Index

CART selects the best split using the **Gini Index**.

The Gini Index measures impurity.

Lower Gini value means better separation.

Formula

Gini=1−∑pi2Gini = 1-\sum p_i^2Gini=1−∑pi2

where

pip_ipi is the probability of each class.

Example 1

Dataset

Result
Yes
Yes
No
No

Probability

Yes

2/4 = 0.5

No

2/4 = 0.5

Gini

$=1-(0.5^2+0.5^2)$

=1-(0.25+0.25)

=0.5

Example 2

Dataset

Yes

Yes

Yes

No

Probability

Yes

3/4

No

1/4

Gini

$=1-(0.75^2+0.25^2)$

$=1-(0.5625+0.0625)$

$=0.375$

Since

$0.375 < 0.5$

The second dataset is purer.

Therefore CART prefers the second split.

Selecting the Best Split

Suppose

Attribute	**Gini**
Salary	0.21
Age	0.39

Attribute Gini

Experience 0.44

The algorithm chooses

Salary

because it has the **lowest Gini Index**.

CART Algorithm

Step 1

Read the dataset.

↓

Step 2

Calculate the Gini Index for every attribute.

↓

Step 3

Choose the attribute with the lowest Gini value.

↓

Step 4

Split the dataset into two groups.

↓

Step 5

Repeat the same process for every child node.

↓

Step 6

Stop when every leaf becomes sufficiently pure.

Example

Dataset

Marks	Pass
90	Yes
80	Yes
40	No
35	No

Possible split

Marks > 60

Yes

Pass

No

Fail

Prediction

Student Marks

85

Output

Pass

Student Marks

30

Output

Fail

Regression Tree Splitting

Regression trees do not use the Gini Index.

Instead, they minimize **Mean Squared Error (MSE)**.

Formula

MSE=1n∑(y−y^)2MSE=\frac{1}{n}\sum (y-\hat y)^2MSE=n1∑(y−y^)2

where

yyy = Actual Value

y^\hat yy^ = Predicted Value

The split with the **lowest MSE** is selected.

Example

House Prices

Area (sq.ft)	**Price**
800	30 Lakhs
1000	40 Lakhs
1500	60 Lakhs
2000	80 Lakhs

Possible split

Area >1200

Yes

High Price

No

Low Price

Prediction

Area = 1700 sq.ft

Predicted Price

₹60 Lakhs

Pruning in CART

Sometimes the tree becomes too large.

A large tree memorizes training data instead of learning patterns.

This problem is called **Overfitting**.

To avoid overfitting,

CART uses **Pruning**.

Pruning removes unnecessary branches.

Example

Large Tree

A

|

B

|

C

|

D

|

E

After Pruning

A

|

B

|

C

The simpler tree performs better on new data.

Cost Complexity Pruning

CART uses **Cost Complexity Pruning**.

It balances

- Tree Size
- Prediction Error

A very large tree may have low training error but poor testing accuracy.

Cost Complexity Pruning removes branches that contribute very little to prediction accuracy.

Advantages of CART

1. Simple to understand.
2. Easy to visualize.
3. Supports classification and regression.
4. Handles missing values reasonably well.
5. No feature scaling required.
6. Handles numerical and categorical data.
7. Produces accurate predictions for many real-world problems.

Disadvantages of CART

1. Can overfit training data.
2. Sensitive to small changes in the dataset.
3. Large trees become difficult to interpret.
4. May not perform well on highly complex relationships.
5. Greedy splitting does not always produce the globally optimal tree.

Applications of CART

Healthcare

Disease diagnosis

Cancer prediction

Heart disease prediction

Banking

Loan approval

Credit risk analysis

Fraud detection

Agriculture

Crop prediction

Yield estimation

Soil quality analysis

Education

Student performance prediction

Placement prediction

Dropout prediction

E-commerce

Customer recommendation

Product demand prediction

Customer churn prediction

Cyber Security

Intrusion Detection Systems (IDS)

Malware detection

Network attack classification

Difference Between ID3, C4.5 and CART

Feature	ID3	C4.5	CART
Splitting Criterion	Information Gain	Gain Ratio	Gini Index
Tree Type	Multiway	Multiway	Binary
Classification	Yes	Yes	Yes
Regression	No	Limited	Yes
Missing Values	No	Yes	Yes (better support)
Pruning	No	Yes	Yes
Continuous Data	Limited	Yes	Yes

Real-Life Example

Imagine a doctor diagnosing a patient.

- Is the patient's temperature above 38°C?
  - **Yes** → Go to the next question.
  - **No** → Patient is likely healthy.

If **Yes**:

- Is the blood sugar level above 180 mg/dL?
  - **Yes** → Diabetes risk.
  - **No** → Viral fever.

Notice that **every question has only two possible outcomes (Yes/No)**. This binary decision-making process is exactly how the CART algorithm builds its tree.

Ensemble Learning

Introduction

In real life, we usually do not depend on the opinion of a single person to make an important decision. Instead, we ask several experts and then make the final decision based on the majority opinion. Ensemble Learning follows the same principle.

Instead of using only one machine learning model, Ensemble Learning combines multiple models to produce a more accurate and reliable prediction.

The word **Ensemble** means **a group working together**.

Definition

Ensemble Learning is a machine learning technique in which multiple models, called **base learners** or **weak learners**, are combined to build a stronger and more accurate prediction model.

Instead of depending on a single classifier, the algorithm combines the predictions of several classifiers.

Why Ensemble Learning?

Suppose a hospital wants to detect heart disease.

Three doctors examine the same patient.

Doctor 1 → Disease

Doctor 2 → Disease

Doctor 3 → No Disease

Since two doctors predict "Disease", the final decision is

Disease

This is exactly how Ensemble Learning works.

Need for Ensemble Learning

A single model may make mistakes because

- Training data may contain noise.
- The model may overfit.
- The model may underfit.
- The model may not capture all relationships.

Combining several models reduces these problems and improves prediction accuracy.

Basic Idea

Suppose we have three classifiers.

Classifier A

Spam

Classifier B

Spam

Classifier C

Not Spam

Final Prediction

Spam

because the majority predicts Spam.

Working Principle

Step 1

Train multiple models using the same dataset or different datasets.

↓

Step 2

Each model makes its own prediction.

↓

Step 3

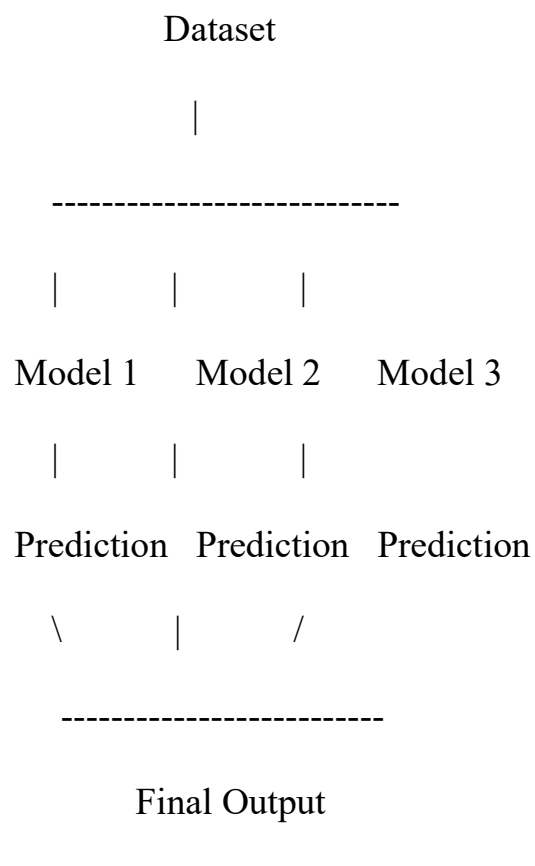
Combine all predictions.

↓

Step 4

Generate one final prediction.

Diagram



Weak Learner

A Weak Learner is a model whose accuracy is only slightly better than random guessing.

Example

Decision Stump

A decision stump contains only one decision.

Salary > 50000?

Yes

No

Accuracy

65%

This is a weak learner.

Strong Learner

A Strong Learner is obtained by combining many weak learners.

Example

Weak Learner 1

65%

Weak Learner 2

68%

Weak Learner 3

70%

Combined Model

95%

Thus many weak learners together become a strong learner.

Example

Suppose five classifiers predict whether an email is spam.

Classifier	**Prediction**
C1	Spam
C2	Spam
C3	Spam
C4	Not Spam
C5	Spam

Majority prediction

Spam

Final Output

Spam

Types of Ensemble Learning

There are mainly three methods.

1. Bagging
2. Boosting
3. Stacking

1. Bagging

Bagging means

Bootstrap Aggregating

It trains multiple models independently.

Each model receives a different random sample of the training data.

Final prediction is obtained using majority voting.

Example

Tree 1

Loan Approved

Tree 2

Loan Approved

Tree 3

Loan Rejected

Final Output

Loan Approved

2. Boosting

Boosting trains models one after another.

Every new model focuses on correcting the mistakes made by the previous model.

Example

Model 1

Accuracy

70%

Model 2

Learns previous mistakes

Accuracy

82%

Model 3

Learns previous mistakes

Accuracy

91%

3. Stacking

Different algorithms are trained together.

Example

Decision Tree

Random Forest

Support Vector Machine

Logistic Regression

The outputs are combined by another model called the **Meta Learner**.

Voting Methods

Majority Voting

The class receiving the maximum votes is selected.

Example

Model	**Prediction**
M1	Yes
M2	Yes
M3	No
M4	Yes
M5	No

Final Output

Yes

Average Voting

Used mainly for regression.

Example

Predicted House Prices

Model 1

$200,000

Model 2

$210,000

Model 3

$190,000

Average

200000+210000+1900003=200000\frac{200000+210000+190000}{3}=2000000320000+210000+190000=200000

Final Prediction

$200,000

Advantages of Ensemble Learning

1. Higher prediction accuracy.
2. Reduces overfitting.
3. Produces stable models.
4. Handles complex datasets.
5. Improves generalization.
6. Better than a single classifier.

7. More reliable predictions.

Disadvantages

1. Requires more computation.
2. More memory consumption.
3. Longer training time.
4. Difficult to interpret.
5. Complex implementation.

Applications

Healthcare

Disease diagnosis

Cancer detection

Heart disease prediction

Banking

Loan approval

Fraud detection

Credit scoring

Cyber Security

Intrusion Detection Systems

Malware detection

Spam filtering

Agriculture

Crop prediction

Weather forecasting

Education

Student performance prediction

E-commerce

Recommendation systems

Customer churn prediction

Real-Life Example

Imagine a school is selecting the "Best Student."

Instead of asking only one teacher, the principal asks five teachers to evaluate the students based on attendance, marks, discipline, and participation.

Teacher 1 → Student A

Teacher 2 → Student A

Teacher 3 → Student B

Teacher 4 → Student A

Teacher 5 → Student A

Since most teachers selected Student A, the principal also selects Student A.

Similarly, Ensemble Learning combines predictions from multiple models to make a more accurate final decision.

Difference Between Single Learning and Ensemble Learning

Single Model	Ensemble Learning
One classifier	Multiple classifiers
Lower accuracy	Higher accuracy
Higher chance of overfitting	Lower chance of overfitting

Single Model	Ensemble Learning
Less computation	More computation
Easier to implement	More complex
Less robust	More robust

Important Points for Exams

- Ensemble Learning combines **multiple machine learning models**.
- The goal is to improve **accuracy, robustness, and generalization**.
- Base learners are called **weak learners**.
- The combined model is called a **strong learner**.
- The three major ensemble techniques are **Bagging, Boosting, and Stacking**.

Bagging (Bootstrap Aggregating)

Introduction

Bagging stands for **Bootstrap Aggregating**. It is one of the most popular ensemble learning techniques used to improve the accuracy and stability of machine learning models.

Instead of training only one model, Bagging trains multiple models independently using different random samples of the training dataset. The predictions of all models are then combined to produce the final prediction.

Bagging mainly helps in **reducing overfitting** and **improving prediction accuracy**.

Definition

Bagging is an ensemble learning technique in which multiple models are trained independently on different bootstrap samples of the training

dataset, and their predictions are combined using majority voting (classification) or averaging (regression).

Why Bagging?

Suppose one doctor diagnoses a disease.

Doctor's prediction

Disease

The doctor may make a mistake.

Instead, five doctors examine the patient.

Doctor 1 → Disease

Doctor 2 → Disease

Doctor 3 → Disease

Doctor 4 → No Disease

Doctor 5 → Disease

Final Decision

Disease

Since most doctors predict Disease, the final prediction becomes more reliable.

Bagging works exactly like this.

Meaning of Bootstrap

Bootstrap means **sampling with replacement**.

Suppose the original dataset contains

A
B
C
D
E

One bootstrap sample may be

A
C
E
A
D

Notice

A appears twice.

B is missing.

This is possible because sampling is done **with replacement**.

Meaning of Aggregating

Aggregating means

Combining the predictions of all models.

For Classification

Majority Voting

For Regression

Average Prediction

Steps in Bagging

Step 1

Collect the original dataset.

A B C D E

↓

Step 2

Generate multiple bootstrap samples.

Sample 1

A C D A E

Sample 2

B D B C A

Sample 3

E D A C C

↓

Step 3

Train one model on each sample.

↓

Step 4

Each model predicts independently.

↓

Step 5

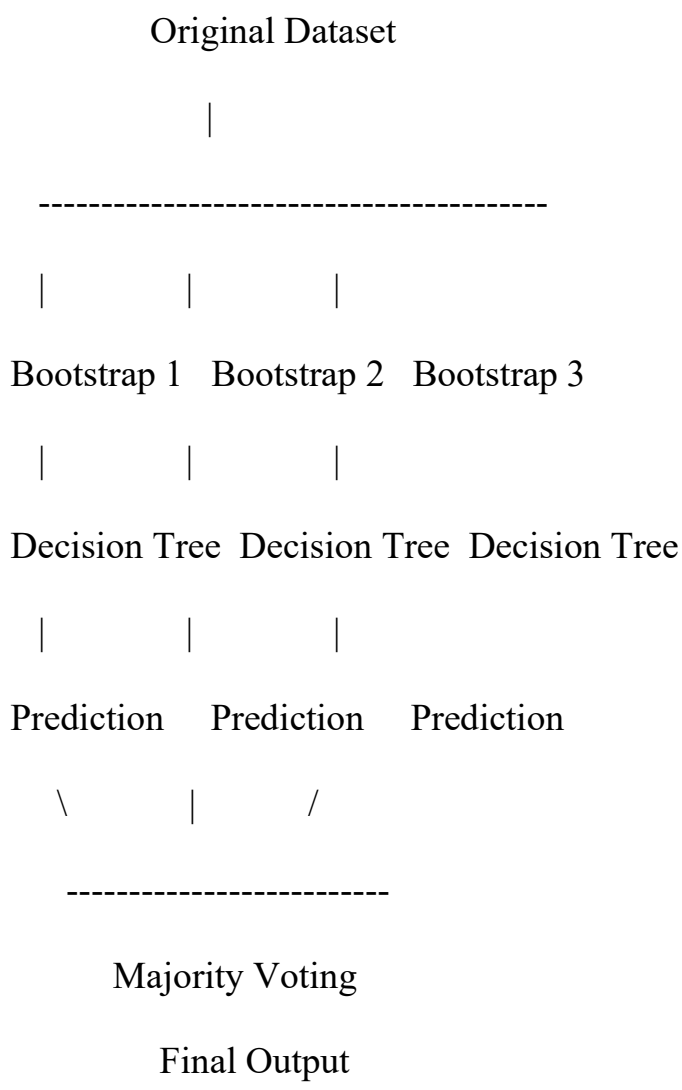
Combine all predictions.

↓

Step 6

Produce the final prediction.

Diagram



Example

Suppose three decision trees predict whether a customer will buy a product.

Tree 1

Yes

Tree 2

No

Tree 3

Yes

Majority Voting

Yes = 2

No = 1

Final Prediction

Yes

Majority Voting

Majority Voting is used in **Classification**.

Example

Model Prediction
Tree 1 Yes
Tree 2 Yes
Tree 3 No
Tree 4 Yes
Tree 5 No

Votes

Yes = 3

No = 2

Final Prediction

Yes

Averaging

Used for **Regression Problems**.

Example

Three trees predict the price of a house.

Tree 1

250000

Tree 2

260000

Tree 3

240000

Average

250000+260000+2400003=250000\frac{250000+260000+240000}{3}
=2500003250000+260000+240000=250000

Final Prediction

250000

Bootstrap Sampling Example

Suppose

Original Dataset

1
2
3
4
5

Possible Samples

Sample 1

1
2
3
3
5

Sample 2

2
2
4
5
1

Sample 3

5
4
4
2
3

Each sample is slightly different.

Therefore every model learns different patterns.

Random Forest

Random Forest is the most famous application of Bagging.

It consists of many Decision Trees.

Each tree is trained on a different bootstrap sample.

The final prediction is obtained using majority voting.

Example

Tree 1

Healthy

Tree 2

Healthy

Tree 3

Disease

Tree 4

Healthy

Tree 5

Healthy

Final Prediction

Healthy

Why Random Forest Performs Better

Different trees make different mistakes.

When combined,

many mistakes cancel each other.

Therefore

Accuracy increases.

Out-of-Bag (OOB) Samples

Since bootstrap sampling uses replacement,

some records are not selected.

These unused records are called

Out-of-Bag Samples.

Example

Original Dataset

A
B
C
D
E

Bootstrap Sample

A
C
D
A
E

Unused Record

B

B is an Out-of-Bag sample.

Out-of-Bag Error

Out-of-Bag samples are used to test the model.

Therefore,

a separate testing dataset is often unnecessary.

Advantages

- Saves time.
- Estimates model performance.
- Reduces overfitting.

Bagging Algorithm

Step 1

Take the original training dataset.

↓

Step 2

Generate N bootstrap samples.

↓

Step 3

Train one model on each sample.

↓

Step 4

Repeat until all models are trained.

↓

Step 5

Collect predictions from every model.

↓

Step 6

Use Majority Voting (classification) or Averaging (regression).

↓

Step 7

Generate the final prediction.

Advantages of Bagging

1. Improves prediction accuracy.
2. Reduces overfitting.

3. Reduces variance.
4. Produces stable models.
5. Works well with Decision Trees.
6. Handles noisy datasets.
7. Parallel training is possible because models are independent.

Disadvantages

1. High computational cost.
2. Requires more memory.
3. Training multiple models takes longer.
4. Less interpretable than a single decision tree.

Applications

Healthcare

- Disease prediction
- Cancer diagnosis
- Medical image classification

Banking

- Loan approval
- Credit risk analysis
- Fraud detection

Cyber Security

- Intrusion Detection Systems
- Malware detection
- Network attack prediction

Agriculture

- Crop yield prediction
- Soil classification

Education

- Student performance prediction

E-Commerce

- Recommendation systems
- Customer churn prediction

Real-Life Example

Suppose a university wants to recruit a professor.

Instead of allowing only the Principal to decide,

the university forms an interview panel.

Panel Member 1 → Selected

Panel Member 2 → Selected

Panel Member 3 → Not Selected

Panel Member 4 → Selected

Panel Member 5 → Selected

Majority decision

Selected

Bagging follows exactly the same idea.

Instead of depending on one model,

it combines multiple models.

Difference Between Decision Tree and Bagging

Decision Tree	Bagging
Single model	Multiple models
Higher chance of overfitting	Less overfitting
Lower accuracy	Higher accuracy
High variance	Reduced variance
Simple	More complex
One prediction	Combined prediction

Important Exam Points

- Bagging = **Bootstrap + Aggregating**
- Uses **sampling with replacement**
- Models are trained **independently**
- Classification uses **Majority Voting**
- Regression uses **Average Prediction**
- Random Forest is the **best-known Bagging algorithm**
- Main objective: **Reduce variance and improve accuracy**

7. Boosting (Introduction)

Boosting is another ensemble learning technique, but unlike Bagging, the models are **trained one after another (sequentially)**. Each new model focuses on correcting the mistakes made by the previous model.

You can think of Boosting as a teacher helping a student. After the first test, the teacher identifies the student's weak areas. The next lesson focuses on those weak areas. After the second test, the teacher again identifies the remaining mistakes and continues improving the student's performance. This process continues until the student performs much better.

Boosting algorithms such as **AdaBoost**, **Gradient Boosting**, and **XGBoost** follow this idea to build highly accurate predictive models.

The next section will explain **Boosting** in full detail, including AdaBoost, Gradient Boosting, XGBoost, algorithms, formulas, and examples.

Boosting

Introduction

Boosting is one of the most powerful Ensemble Learning techniques used in Machine Learning. Unlike Bagging, where all models are trained independently, **Boosting trains models sequentially (one after another).**

Each new model tries to **correct the mistakes made by the previous model**. Because of this continuous improvement, the final model becomes much more accurate.

Boosting converts many **weak learners** into a **strong learner**.

Definition

Boosting is an ensemble learning technique that combines multiple weak learners by training them sequentially, where each new learner focuses on correcting the errors made by the previous learner.

Why Boosting?

Suppose a teacher gives a Mathematics test.

Student Score

40/100

The teacher identifies the student's mistakes and teaches those topics again.

Second Test

65/100

Again the teacher identifies mistakes.

Third Test

85/100

Fourth Test

95/100

The student's performance keeps improving because every test focuses on previous mistakes.

Boosting follows the same idea.

Basic Idea of Boosting

Instead of training all models at once,

Boosting trains

Model 1

↓

Find mistakes

↓

Model 2

↓

Correct mistakes

↓

Model 3

↓

Correct remaining mistakes

↓

Final Strong Model

Diagram

Training Dataset

|

Weak Learner 1

|

Identify Errors

|

Weak Learner 2

|

Identify Errors

|

Weak Learner 3

|

Final Prediction

Working Principle

Step 1

Train the first weak learner.

↓

Step 2

Calculate errors.

↓

Step 3

Increase the importance (weight) of incorrectly classified samples.

↓

Step 4

Train another learner using the updated weights.

↓

Step 5

Repeat until the required number of models is trained.

↓

Step 6

Combine all models to produce the final prediction.

Example

Suppose we want to predict whether an email is Spam.

Model 1

Correct Predictions

8

Wrong Predictions

2

Boosting gives more importance to those 2 incorrect emails.

Model 2

Learns those mistakes.

Correct Predictions

9

Wrong Predictions

1

Model 3

Learns the remaining mistake.

Correct Predictions

10

Wrong Predictions

0

The final model becomes highly accurate.

Weak Learner

A weak learner performs only slightly better than random guessing.

Example

Decision Stump

Age > 30

Yes

No

Accuracy

60%

Strong Learner

Many weak learners combined sequentially become

Strong Learner

Accuracy

95%

Characteristics of Boosting

1. Sequential learning.
2. Focuses on previous errors.
3. Produces high accuracy.
4. Converts weak learners into strong learners.
5. Sensitive to noisy data.
6. Reduces bias.

AdaBoost (Adaptive Boosting)

AdaBoost is the first successful Boosting algorithm.

The word

Ada

means

Adaptive

because the algorithm adapts itself based on previous mistakes.

Working of AdaBoost

Step 1

Assign equal weight to every training sample.

Example

Suppose we have five records.

Record	Weight
A	0.20
B	0.20
C	0.20
D	0.20
E	0.20

Initially,

every record has equal importance.

Step 2

Train the first weak learner.

Suppose

A and B

are classified correctly.

C

is classified incorrectly.

D

correctly.

E

incorrectly.

Step 3

Increase weights of wrong predictions.

Updated Weights

Record	Weight
A	0.10
B	0.10
C	0.35
D	0.10
E	0.35

Now

C and E

receive more attention.

Step 4

Train another weak learner.

It mainly focuses on

C and E.

Step 5

Repeat until the required accuracy is achieved.

AdaBoost Flow

Dataset

↓

Weak Learner 1

↓

Increase weights of wrong predictions

↓

Weak Learner 2

↓

Increase weights

↓

Weak Learner 3

↓

Final Prediction

Advantages of AdaBoost

1. High prediction accuracy.
2. Easy to implement.
3. Reduces bias.
4. Works well with Decision Trees.
5. Good for classification.

Limitations

1. Sensitive to noisy data.
2. Sensitive to outliers.
3. Slower than Bagging.

Gradient Boosting

Gradient Boosting is another Boosting algorithm.

Instead of increasing weights,

it reduces the prediction error using

Gradient Descent.

Each new model tries to minimize the loss function.

Working Principle

Model 1

↓

Calculate Error

↓

Model 2 learns residual errors

↓

Calculate remaining error

↓

Model 3 learns remaining residuals

↓

Final Prediction

Residual

Residual means

Difference between

Actual Value

and

Predicted Value.

Formula

Residual=Actual−PredictedResidual = Actual - PredictedResidual=Actual−Predicted

Example

Actual House Price

250000

Predicted Price

230000

Residual

250000 − 230000

=

20000

The next model tries to reduce this error.

Advantages of Gradient Boosting

1. Very high accuracy.
2. Handles complex datasets.
3. Works for regression.
4. Works for classification.
5. Excellent predictive performance.

Disadvantages

1. Slow training.
2. Can overfit.
3. Requires parameter tuning.

XGBoost (Extreme Gradient Boosting)

XGBoost is an improved version of Gradient Boosting.

It is one of the most widely used machine learning algorithms in competitions and real-world applications.

Features of XGBoost

1. Faster than Gradient Boosting.
2. Parallel processing.
3. Handles missing values.
4. Built-in regularization.
5. Prevents overfitting.
6. High prediction accuracy.

Why XGBoost is Popular

Many companies use XGBoost because

- High speed.
- Excellent accuracy.
- Handles very large datasets.
- Efficient memory usage.

Applications of Boosting

Healthcare

Disease prediction

Cancer diagnosis

Heart disease prediction

Medical image analysis

Banking

Fraud detection

Loan approval

Credit scoring

Cyber Security

Intrusion Detection Systems

Malware detection

Spam filtering

Agricultur

Crop prediction

Weather prediction

Disease detection

E-Commerce

Recommendation systems

Customer churn prediction

Demand forecasting

Education

Student performance prediction

Placement prediction

Difference Between Bagging and Boosting

Bagging	Boosting
Models are trained independently	Models are trained sequentially
Reduces variance	Reduces bias
Uses Bootstrap Sampling	Focuses on previous mistakes
Parallel training	Sequential training
Majority Voting	Weighted combination
Example: Random Forest	Examples: AdaBoost, Gradient Boosting, XGBoost

Real-Life Example

Imagine a cricket coach training a batsman.

Day 1

The player struggles against fast bowling.

The coach notices the weakness.

Day 2

The coach practices only fast bowling.

The player improves.

Day 3

The player struggles against spin bowling.

The coach now focuses only on spin.

Day 4

The player becomes good at both fast and spin bowling.

Each training session focuses on correcting previous mistakes.

This is exactly how **Boosting** works.

Important Exam Points

- Boosting trains models **one after another**.
- Each new model **corrects the mistakes** of the previous model.
- Converts **weak learners into strong learners**.
- **AdaBoost** uses **sample weights**.
- **Gradient Boosting** minimizes **residual errors**.
- **XGBoost** is a faster and more optimized version of Gradient Boosting.
- Main objective: **Reduce bias and improve prediction accuracy**.

R22 B.Tech. CSE (DS)

MACHINE LEARNING

B.Tech. III Year I Sem.

UNIT – IV

Dimensionality Reduction – Linear Discriminant Analysis – Principal Component Analysis – Factor Analysis – Independent Component Analysis – Locally Linear Embedding – Isomap – Least Squares Optimization

Evolutionary Learning – Genetic algorithms – Genetic Offspring: - Genetic Operators – Using Genetic Algorithms

Dimensionality reduction is the process of reducing the number of features (or dimensions) in a dataset while retaining as much information as possible. This can be done for a variety of reasons, such as to reduce the complexity of a model, to improve the performance of a learning algorithm, or to make it easier to visualize the data. There are several techniques for dimensionality reduction, including principal component analysis (PCA), singular value decomposition (SVD), and linear discriminant analysis (LDA). Each technique uses a different method to project the data onto a lower-dimensional space while preserving important information.

In machine learning,
high-dimensional data refers to data with a large number of features or variables. The curse of dimensionality is a common problem in machine learning, where the performance of the model deteriorates as the number of features increases. This is because the complexity of the model increases with the number of features, and it becomes more difficult to find a good solution.

In addition, high-dimensional data can also lead to overfitting, where the model fits the training data too closely and does not generalize

There are two main approaches to dimensionality reduction: feature selection and feature extraction.

Feature Selection:

Feature selection involves selecting a subset of the original features that are most relevant to the problem at hand. The goal is to reduce the dimensionality of the dataset while retaining the most important features. There are several

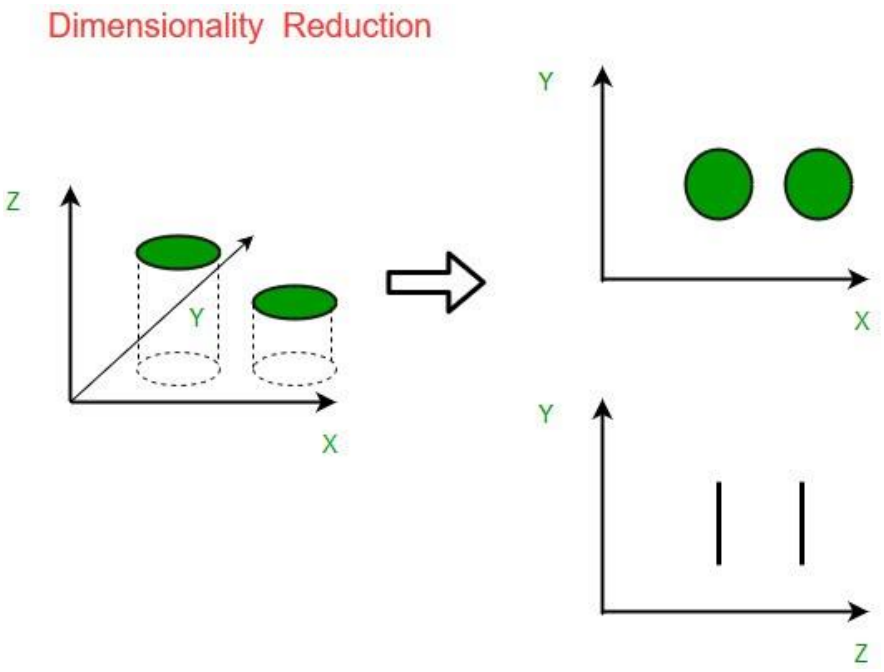
methods for feature selection, including filter methods, wrapper methods, and embedded methods. Filter methods rank the features based on their relevance to the target variable, wrapper methods use the model performance as the criteria for selecting features, and embedded methods combine feature selection with the model training process.
There are two main approaches to dimensionality reduction: feature selection and feature extraction.

Feature Extraction:

Feature extraction involves creating new features by combining or transforming the original features. The goal is to create a set of features that captures the essence of the original data in a lower-dimensional space. There are several methods for feature extraction, including principal component analysis (PCA), linear discriminant analysis (LDA), and t-distributed stochastic neighbor embedding (t-SNE). PCA is a popular technique that projects the original features onto a lower-dimensional space while preserving as much of the variance as possible.

Why is Dimensionality Reduction important in Machine Learning and Predictive Modeling?
An intuitive example of dimensionality reduction can be discussed through a simple e-mail classification problem, where we need to classify whether the e-mail is spam or not. This can involve a large number of features, such as whether or not the e-mail has a generic title, the content of the e-mail, whether the e-mail uses a template, etc. However, some of these features may overlap.

In another condition, a classification problem that relies on both humidity and rainfall can be collapsed into just one underlying feature, since both of the aforementioned are correlated to a high degree. Hence, we can reduce the number of features in such problems. A 3-D classification problem can be hard to visualize, whereas a 2-D one can be mapped to a simple 2-dimensional space, and a 1-D problem to a simple line. The below figure illustrates this concept, where a 3-D feature space is split into two 2-D feature spaces, and later, if found to be correlated, the number of features can be reduced even further.



Methods of Dimensionality Reduction

The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

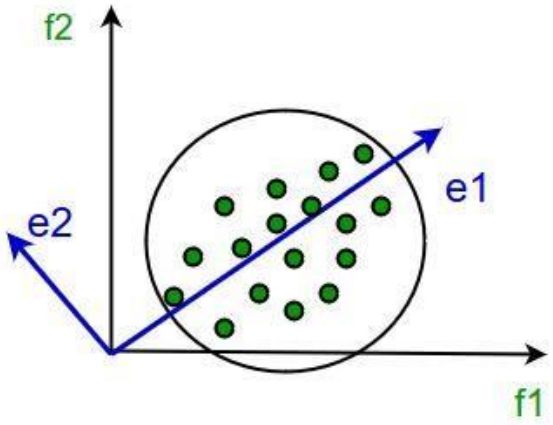
Dimensionality reduction may be both linear and non-linear, depending upon the method used. The prime linear method, called Principal Component Analysis, or PCA,

Principal Component Analysis

This method was introduced by Karl Pearson. It works on the condition that while the data in a higher dimensional space is mapped to data in a lower dimension space,

the variance of the data in the lower dimensional

space should be maximum.



It involves the following steps:

- Construct the covariance matrix of the data.
- Compute the eigenvectors of this matrix.
- Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction

of variance of the original data.

Advantages of Dimensionality Reduction

- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.
- Improved Visualization: High dimensional data is difficult to visualize, and dimensionality reduction techniques can help in visualizing the data in 2D or 3D, which can help in better understanding and analysis.
- Overfitting Prevention: High dimensional data may lead to overfitting in machine learning models, which can lead to poor generalization performance. Dimensionality reduction can help in reducing the complexity of the data, and hence prevent overfitting.
- Feature Extraction: Dimensionality reduction can help in extracting important features from high dimensional data, which can be useful in feature selection for machine learning models.
- Data Preprocessing: Dimensionality reduction can be used as a preprocessing step before applying machine learning algorithms to reduce the dimensionality of the data and hence improve the performance of the model.

- Improved Performance: Dimensionality reduction can help in improving the performance of machine learning models by reducing the complexity of the data, and hence reducing the noise and irrelevant information in the data.

Disadvantages of Dimensionality Reduction

- It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.
- We may not know how many principal components to keep- in practice, some thumb rules are applied.
- Interpretability: The reduced dimensions may not be easily interpretable, and it may be difficult to understand the relationship between the original features and the reduced dimensions.
- Overfitting: In some cases, dimensionality reduction may lead to overfitting, especially when the number of components is chosen based on the training data.
- Sensitivity to outliers: Some dimensionality reduction techniques are sensitive to outliers, which can result in a biased representation of the data.
- Computational complexity: Some dimensionality reduction techniques, such as manifold learning, can be computationally intensive, especially when dealing with large datasets.

What is Factor Analysis?

Factor analysis, a method within the realm of statistics and part of the general linear model (GLM), serves to condense numerous variables into a smaller set of factors. By doing so, it captures the maximum shared variance among the variables and condenses them into a unified score, which can subsequently be utilized for further

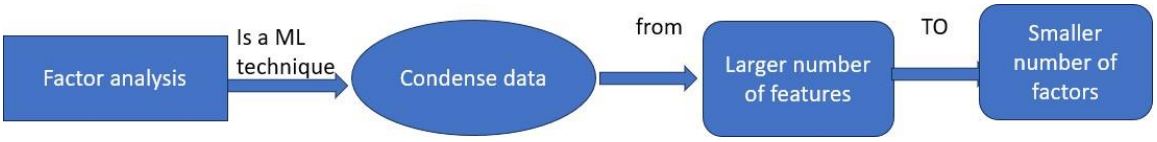
analysis.Factor analysis operates under several assumptions: linearity in relationships, absence of multicollinearity among variables, inclusion of relevant variables in the analysis, and genuine correlations between variables and factors. While multiple methods exist, principal component analysis stands out as the most prevalent approach in practice.

What does Factor mean in Factor Analysis?

In the context of factor analysis, a "factor" refers to an underlying, unobserved variable or latent construct that represents a common source of variation among a set of observed variables. These observed variables, also known as indicators or manifest variables, are the measurable variables that are directly observed or measured in a study.

How to do Factor Analysis (Factor Analysis Steps)?

Factor analysis is a statistical method used to describe variability among observed, correlated variables in terms of a potentially lower number of unobserved variables called factors. Here are the general steps involved in conducting a factor analysis:



1. **Determine the Suitability of Data for Factor Analysis**
    - **Bartlett's Test:** Check the significance level to determine if the correlation matrix is suitable for factor analysis.
    - **Kaiser-Meyer-Olkin (KMO) Measure:** Verify the sampling adequacy. A value greater than 0.6 is generally considered acceptable.
2. **Choose the Extraction Method**
    - **Principal Component Analysis (PCA):** Used when the main goal is data reduction.
    - **Principal Axis Factoring (PAF):** Used when the main goal is to identify underlying factors.
3. **Factor Extraction**
    - Use the chosen extraction method to identify the initial factors.

- Extract eigenvalues to determine the number of factors to retain. Factors with eigenvalues greater than 1 are typically retained in the analysis.
- Compute the initial factor loadings.

4. Determine the Number of Factors to Retain

- **Scree Plot:** Plot the eigenvalues in descending order to visualize the point where the plot levels off (the "elbow") to determine the number of factors to retain.
- **Eigenvalues:** Retain factors with eigenvalues greater than 1.

5. Factor Rotation

- **Orthogonal Rotation (Varimax, Quartimax):** Assumes that the factors are uncorrelated.
- **Oblique Rotation (Promax, Oblimin):** Allows the factors to be correlated.
- Rotate the factors to achieve a simpler and more interpretable factor structure.
- Examine the rotated factor loadings.

6. Interpret and Label the Factors

- Analyze the rotated factor loadings to interpret the underlying meaning of each factor.
- Assign meaningful labels to each factor based on the variables with high loadings on that factor.

7. Compute Factor Scores (if needed)

- Calculate the factor scores for each individual to represent their value on each factor.

8. Report and Validate the Results

- Report the final factor structure, including factor loadings and communalities.
- Validate the results using additional data or by conducting a confirmatory factor analysis if necessary.

Types of Factor Analysis

There are two main types of Factor Analysis used in data science:

1. Exploratory Factor Analysis (EFA)

Exploratory Factor Analysis (EFA) is used to uncover the underlying structure of a set of observed variables without imposing preconceived notions about how many factors there are or how the variables are related to each factor. It explores complex

interrelationships among items and aims to group items that are part of unified concepts or constructs.

- Researchers do not make a priori assumptions about the relationships among factors, allowing the data to reveal the structure organically.
- Exploratory Factor Analysis (EFA) helps in identifying the number of factors needed to account for the variance in the observed variables and understanding the relationships between variables and factors.

2. Confirmatory Factor Analysis (CFA)

Confirmatory Factor Analysis (CFA) is a more structured approach that tests specific hypotheses about the relationships between observed variables and latent factors based on prior theoretical knowledge or expectations. It uses structural equation modeling techniques to test a measurement model, wherein the observed variables are assumed to load onto specific factors.

- Confirmatory Factor Analysis (CFA) assesses the fit of the hypothesized model to the actual data, examining how well the observed variables align with the proposed factor structure.
- This method allows for the evaluation of relationships between observed variables and unobserved factors, and it can accommodate measurement error.
- Researchers hypothesize the relationships between variables and factors before conducting the analysis, and the model is tested against empirical data to determine its validity.

In summary, while Exploratory Factor Analysis (EFA) is more exploratory and flexible, allowing the data to dictate the factor structure, Confirmatory Factor Analysis (CFA) is more confirmatory, testing specific hypotheses about how the observed variables are related to latent factors. Both methods are valuable tools in understanding the underlying structure of data and have their respective strengths and applications.

Types of Factor Extraction Methods

Some of the Type of Factor Extraction methods are dicussed below:

1. **Principal Component Analysis (PCA)**:
    - PCA is a widely used method for factor extraction.

- It aims to extract factors that account for the maximum possible variance in the observed variables.
- Factor weights are computed to extract successive factors until no further meaningful variance can be extracted.
- After extraction, the factor model is often rotated for further analysis to enhance interpretability.

2. **Canonical Factor Analysis**:
    - Also known as Rao's canonical factoring, this method computes a similar model to PCA but uses the principal axis method.
    - It seeks factors that have the highest canonical correlation with the observed variables.
    - Canonical factor analysis is not affected by arbitrary rescaling of the data, making it robust to certain data transformations.
3. **Common Factor Analysis**:
    - Also referred to as Principal Factor Analysis (PFA) or Principal Axis Factoring (PAF).
    - This method aims to identify the fewest factors necessary to account for the common variance (correlation) among a set of variables.
    - Unlike PCA, common factor analysis focuses on capturing shared variance rather than overall variance.

Assumptions of Factor Analysis

Let's have a closer look onto the assumptions of factorial analysis, that are as follows:

1. **Linearity**: The relationships between variables and factors are assumed to be linear.
2. **Multivariate Normality**: The variables in the dataset should follow a multivariate normal distribution.
3. **No Multicollinearity**: Variables should not be highly correlated with each other, as high multicollinearity can affect the stability and reliability of the factor analysis results.

4. **Adequate Sample Size**: Factor analysis generally requires a sufficient sample size to produce reliable results. The adequacy of the sample size can depend on factors such as the complexity of the model and the ratio of variables to cases.
5. **Homoscedasticity**: The variance of the variables should be roughly equal across different levels of the factors.
6. **Uniqueness**: Each variable should have unique variance that is not explained by the factors. This assumption is particularly important in common factor analysis.
7. **Independent Observations**: The observations in the dataset should be independent of each other.
8. **Linearity of Factor Scores**: The relationship between the observed variables and the latent factors is assumed to be linear, even though the observed variables may not be linearly related to each other.
9. **Interval or Ratio Scale**: Factor analysis typically assumes that the variables are measured on interval or ratio scales, as opposed to nominal or ordinal scales.

Violation of these assumptions can lead to biased parameter estimates and inaccurate interpretations of the results. Therefore, it's important to assess the data for these assumptions before conducting factor analysis and to consider potential remedies or alternative methods if the assumptions are not met.

What is Independent Component Analysis?

Independent Component Analysis (ICA) is a statistical and computational technique used in machine learning to separate a multivariate signal into its independent non-Gaussian components. The goal of ICA is to find a linear transformation of the data such that the transformed data is as close to being statistically independent as possible.

The heart of ICA lies in the principle of statistical independence. ICA identify components within mixed signals that are statistically independent of each other.

Statistical Independence Concept:

It is a probability theory that if two random variables X and Y are statistically independent. The joint probability distribution of the pair is equal to the product of their individual probability distributions, which means that knowing the outcome of one variable does not change the probability of the other outcome.

or

Assumptions in ICA

1. The first assumption asserts that the source signals (original signals) are statistically independent of each other.
2. The second assumption is that each source signal exhibits non-Gaussian distributions.

Mathematical Representation of Independent Component Analysis

The observed random vector is , representing the observed data with m components. The hidden components are

represented by the random vector , where n is the number of hidden sources.

Linear Static Transformation

The observed data X is transformed into hidden components S using a linear static transformation representation by the matrix W.

Here, W = transformation matrix.

The goal is to transform the observed data x in a way that the resulting hidden components are independent. The independence is measured by

some function . The task is to find the optimal transformation matrix W that maximizes the independence of the hidden components.

Advantages of Independent Component Analysis (ICA):

- ICA is a powerful tool for **separating mixed signals** into their independent components. This is useful in a variety of applications, such as signal processing, image analysis, and data compression.

- ICA is a **non-parametric approach**, which means that it does not require assumptions about the underlying probability distribution of the data.
- ICA is an **unsupervised learning technique**, which means that it can be applied to data without the need for labeled examples. This makes it useful in situations where labeled data is not available.
- ICA can be **used for feature extraction**, which means that it can identify important features in the data that can be used for other tasks, such as classification.

Disadvantages of Independent Component Analysis (ICA):

- ICA assumes that the underlying sources are non-Gaussian, which may not always be true. If the underlying sources are Gaussian, ICA may not be effective.
- ICA assumes that the sources are mixed linearly, which may not always be the case. If the sources are mixed nonlinearly, ICA may not be effective.
- ICA can be computationally expensive, especially for large datasets. This can make it difficult to apply ICA to real-world problems.
- ICA can suffer from convergence issues, which means that it may not always be able to find a solution. This can be a problem for complex datasets with many sources.

Cocktail Party Problem

Consider *Cocktail Party Problem* or *Blind Source Separation* problem to understand the problem which is solved by independent component analysis.

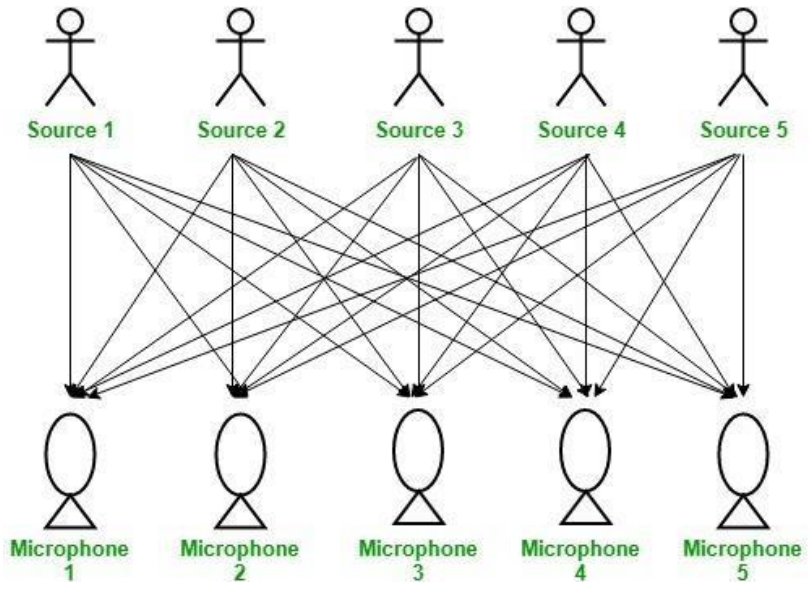
Problem: To extract independent sources' signals from a mixed signal composed of the signals from those sources.

Given: Mixed signal from five different independent sources.

Aim: To decompose the mixed signal into independent sources:

- Source 1
- Source 2
- Source 3
- Source 4
- Source 5

Solution: Independent Component Analysis



Here, there is a party going into a room full of people. There is 'n' number of speakers in that room, and they are speaking simultaneously at the party. In the same room, there are also 'n' microphones placed at different distances from the speakers, which are recording 'n' speakers' voice signals. Hence, the number of speakers is equal to the number of microphones in the room. Step 4: Visualize the signals

- Python3

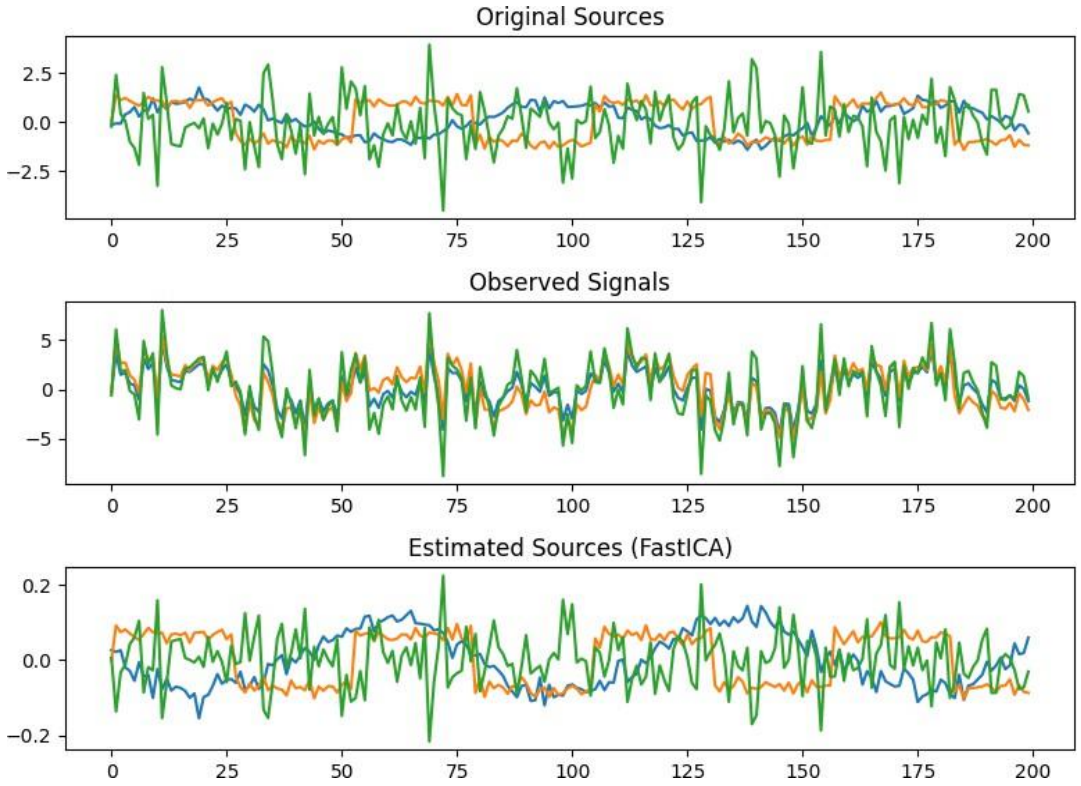
```
# Plot the results plt.figure(figsize=(8, 6))
 plt.subplot(3, 1, 1) plt.title('Original
Sources') plt.plot(S)
plt.subplot(3, 1, 2) plt.title('Observed
Signals') plt.plot(X)
```

```
plt.subplot(3, 1, 3) plt.title('Estimated Sources
(FastICA)') plt.plot(S_)
 plt.tight_layout() plt.show()
```

Now, using these microphones' recordings, we want to separate all the 'n' speakers' voice signals in the room, given that each microphone recorded the voice signals coming from each speaker of different intensity due to the difference in distances between them.

Decomposing the mixed signal of each microphone's recording into an independent source's speech signal can be done by using the machine learning technique, independent component analysis.

where, X1, X2, …, Xn are the original signals present in the mixed signal and Y1, Y2, …, Yn are the new features and are independent components that are independent of each other.



LLE**(Locally Linear Embedding)** is an unsupervised approach designed to transform data from its original high-dimensional space into a lower-dimensional representation, all while striving to retain the essential geometric characteristics of the underlying non-linear feature structure.
LLE operates in several key steps:

- Firstly, it constructs a nearest neighbors graph to capture these local relationships. Then, it optimizes weight values for each data point, aiming to minimize the reconstruction error when expressing a point as a linear combination of its neighbors. This weight matrix reflects the strength of connections between points.
- Next, LLE computes a lower dimensional representation of the data by finding eigenvectors of a matrix derived from the weight matrix. These eigenvectors represent the most relevant directions in the reduced space. Users can specify the desired dimensionality for the output space, and LLE selects the top eigenvectors accordingly.

Mathematical Implementation of LLE Algorithm

The key idea of LLE is that locally, in the vicinity of each data point, the data lies approximately on a linear subspace. LLE attempts to unfold or unroll the data while preserving these local linear relationships.
Here is a mathematical overview of the LLE algorithm:
Minimize: $\sum i|xi - \sum j\ Wijxj|^2$
Subject to : $\sum jwij = 1$
Where:

- xi represents the i-th data point.
- wij are the weights that minimize the reconstruction error for data point xi using its neighbors.

It aims to find a lower-dimensional representation of data while preserving local relationships. The mathematical expression for LLE involves minimizing the reconstruction error of each data point by expressing it as a weighted sum of its k nearest neighbors' contributions. This optimization is subject to constraints ensuring that the weights sum to 1 for each data point.
Locally Linear Embedding (LLE) is a dimensionality reduction technique used in machine learning and data analysis. It focuses on preserving local relationships between data points

when mapping high-dimensional data to a lower-dimensional space. Here, we will explain the LLE algorithm and its parameters.

Mathematical Implementation of LLE Algorithm

The key idea of LLE is that locally, in the vicinity of each data point, the data lies approximately on a linear subspace. LLE attempts to unfold or unroll the data while preserving these local linear relationships.

Here is a mathematical overview of the LLE algorithm:

Minimize:

Subject to :

Where:

- xi represents the i-th data point.
- wij are the weights that minimize the reconstruction error for data point xi using its neighbors.

Locally Linear Embedding Algorithm

The LLE algorithm can be broken down into several steps:

- **Neighborhood Selection:** For each data point in the high-dimensional space, LLE identifies its k-nearest neighbors. This step is crucial because LLE assumes that each data point can be well approximated by a linear combination of its neighbors.
- **Weight Matrix Construction:** LLE computes a set of weights for each data point to express it as a linear combination of its neighbors. These weights are determined in such a way that the reconstruction error is minimized. Linear regression is often used to find these weights.
- **Global Structure Preservation:** After constructing the weight matrix, LLE aims to find a lower-dimensional representation of the data that best preserves the local linear relationships. It does this by seeking a set of coordinates in the lower-dimensional space for each data point that minimizes a cost function. This cost function evaluates how well each data point can be represented by its neighbors.
- **Output Embedding:** Once the optimization process is complete, LLE provides the final lower-dimensional representation of the data. This representation captures the essential structure of the data while reducing its dimensionality.

Parameters in LLE Algorithm

LLE has a few parameters that influence its behavior:

- **k (Number of Neighbors):** This parameter determines how many nearest neighbors are considered when constructing the weight matrix. A larger k captures more global relationships but may introduce noise. A smaller k focuses on local relationships but can be sensitive to outliers. Selecting an appropriate value for k is essential for the algorithm's success.
- **Dimensionality of Output Space:** You can specify the dimensionality of the lower-dimensional space to which the data will be mapped. This is often chosen based on the problem's requirements and the trade-off between computational complexity and information preservation.
- **Distance Metric:** LLE relies on a distance metric to define the proximity between data points. Common choices include Euclidean distance, Manhattan distance, or custom-defined distance functions. The choice of distance metric can impact the results.
- **Regularization (Optional):** In some cases, regularization terms are added to the cost function to prevent overfitting. Regularization can be useful when dealing with noisy data or when the number of neighbors is high.
- **Optimization Algorithm (Optional):** LLE often uses optimization techniques like Singular Value Decomposition (SVD) or eigenvector methods to find the lower-dimensional representation. These optimization methods may have their own parameters that can be adjusted.

Advantages of LLE

The dimensionality reduction method known as locally linear embedding (LLE) has many benefits for data processing and visualization. The following are LLE's main benefits:

- **Preservation of Local Structures**: LLE is excellent at maintaining the in-data local relationships or structures. It successfully captures the inherent geometry of nonlinear

manifolds by maintaining pairwise distances between nearby data points.

- **Handling Non-Linearity**: LLE has the ability to capture nonlinear patterns and structures in the data, in contrast to linear techniques like Principal Component Analysis (PCA). When working with complicated, curved, or twisted datasets, it is especially helpful.
- **Dimensionality Reduction**: LLE lowers the dimensionality of the data while preserving its fundamental properties. Particularly when working with high-dimensional datasets, this reduction makes data presentation, exploration, and analysis simpler.

Disavantages of LLE

- **Curse of Dimensionality**: LLE can experience the "curse of dimensionality" when used with extremely high-dimensional data, just like many other dimensionality reduction approaches. The number of neighbors required to capture local interactions rises as dimensionality does, potentially increasing the computational cost of the approach.
- **Memory and computational Requirements**: For big datasets, creating a weighted adjacency matrix as part of LLE might be memory-intensive. The eigenvalue decomposition stage can also be computationally taxing for big datasets.
- **Outliers and Noisy data**: LLE is susceptible to anomalies and jittery data points. The quality of the embedding may be affected and the local linear relationships may be distorted by outliers.

Implementation of Locally Linear Embedding

```
#importing Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_swiss_roll
from sklearn.manifold import LocallyLinearEmbedding
# Generating a Synthetic Dataset (Swiss Roll)
# Code for Generating a synthetic dataset (Swiss Roll) n_samples =
1000

# Define the number of neighbors for LLE
```

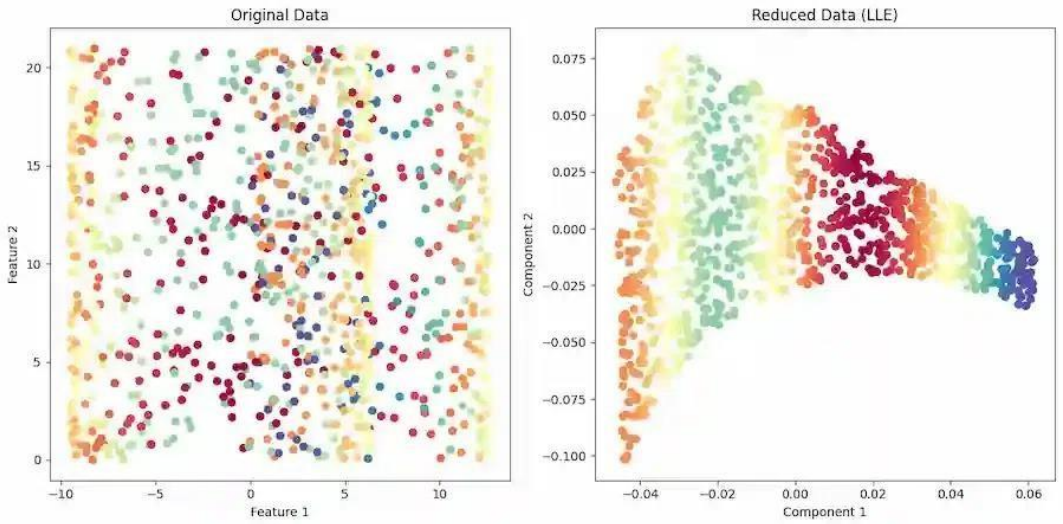
```
n_neighbors = 10

X, _ = make_swiss_roll(n_samples=n_samples)

#It generates a synthetic dataset resembling a Swiss Roll using the
make_swiss_roll function from scikit-learn.
n_samples specifies the number of data points to generate.
n_neighbors defines the number of neighbors used in the LLE algorithm.
#Applying Locally Linear Embedding (LLE)

# Including Locally Linear Embedding

lle = LocallyLinearEmbedding(n_neighbors=n_neighbors,

n_components=2) X_reduced = lle.fit_transform(X)
```

An instance of the LLE algorithm is created with LocallyLinearEmbedding.
The n_neighbors parameter determines the number of neighbors to consider during the embedding process.
The LLE algorithm is then fitted to the original data X using the fit_transform method. This step reduces the dataset to two dimensions (n_components=2).

```
#Visualizing the Original and Reduced Data

# Code for Visualizing the original Versus

reduced data plt.figure(figsize=(12, 6))


plt.subplot(121)

plt.scatter(X[:, 0], X[:, 1], c=X[:, 2],

cmap=plt.cm.Spectral) plt.title("Original

Data")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")
```

```
plt.subplot(122)
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=X[:, 2],
cmap=plt.cm.Spectral) plt.title("Reduced Data (LLE)")
```

```
plt.xlabel("Component 1")
plt.ylabel("Component 2")

plt.tight_layout()
plt.show()
```



ISOMAP

A nonlinear dimensionality reduction method used in data analysis and machine learning is called isomap, short for isometric mapping. Isomap was developed to maintain the inherent geometry of high-dimensional data as a substitute for conventional techniques like Principal Component Analysis (PCA). Isomap creates a low-dimensional representation, usually a two- or three-dimensional map, by focusing on the preservation of pairwise distances between data points.

This technique works especially well for extracting the underlying structure from large, complex datasets, like those from speech recognition, image analysis, and biological systems. Finding patterns and insights in a variety of scientific and engineering domains is made possible by Isomap's capacity to highlight the fundamental relationships found in data.

Isomap

An understanding and representation of complicated data structures are crucial for the field of machine learning. To achieve this, Manifold Learning, a subset of unsupervised learning, has a significant role to play. Among the manifold learning techniques, ISOMAP (Isometric Mapping) stands out for its prowess in capturing the intrinsic geometry of high-dimensional data. In the case of situations in which linear methods are lacking, they have proved particularly efficient.

ISOMAP is a flexible tool that seamlessly blends multiple learning and dimensionality reduction intending to obtain more detailed knowledge of the underlying structure of data. This article takes a look at ISOMAP's inner workings and sheds light on its parameters, functions, and proper implementation with SkLearn.

Isometric mapping is an approach to reduce the dimensionality of machine learning.

Relation between Geodesic Distances and Euclidean Distances Understanding the distinction between equatorial and elliptic distances is of vital importance for ISOMAP. The geodesic distance considers the shortest path along the curved surface of the manifold, as opposed to Euclidean distances which are measured by measuring straight Line distances in the input space. In order to provide a more precise representation of the data's internal structure, ISOMAP exploits these quantum distances.

ISOMAP Parameters

ISOMAP comes with several parameters, each influencing the dimensionality reduction process:

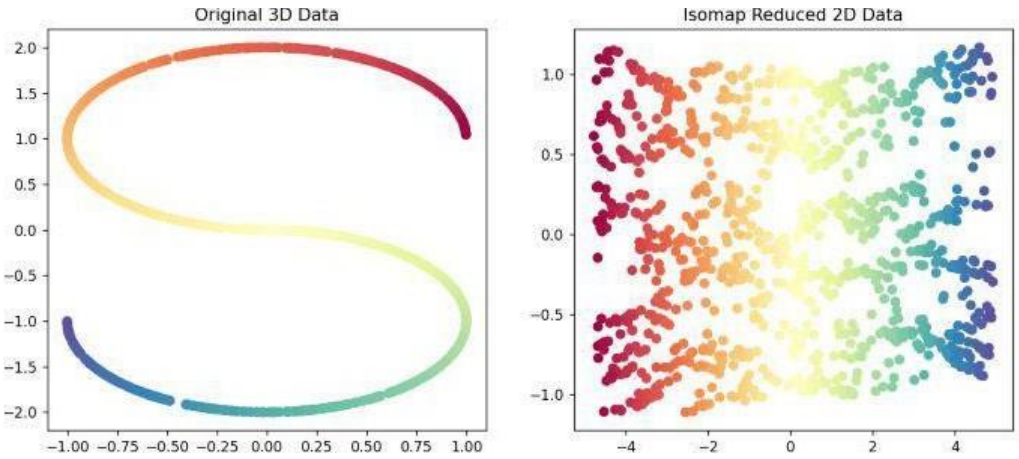
- **n_neighbors**: Determines the number of neighbors used to approximate geodesic distances. Higher values may make it possible to achieve higher results, but they still require more computing power.
- **n_components**: Determines the number of dimensions in a low dimensional representation.
- **eigen_solver**: Determines the method used for decomposing an Eigenvalue. There are options such as "auto", "arpack" and "dense."

- **radius**: You can designate a radius within which neighbors are taken into account in place of using a set number of neighbors. Outside of this range, data points are not regarded as neighbors.
- **tol**: tolerance in the eigenvalue solver to attain convergence. While a lower value might result in a more accurate solution, it might also lengthen the computation time.
- **max_iter**: The maximum number of times the eigenvalue solver can run. It continues if None is selected, unless convergence or additional stopping conditions are satisfied.
- **path_method**: chooses the approximation technique for geodesic distances on the graph. 'auto' (automatic selection) and 'FW' (Floyd-Warshall algorithm) are available options.
- **neighbors_algorithm**: A method for calculating the closest neighbors. 'Auto', 'ball_tree', 'kd_tree', and 'brute' are among the available options. 'auto' selects the best algorithm according to the input data.
- **metric**: The nearest neighbor search's distance metric. 'Minkowski' is the default; however, 'euclidean','manhattan', and several other options are also available.

Working of ISOMAP

- **Calculate the pairwise distances:** The algorithm starts by calculating the Euclidean distances between the data points.
- **Find nearest neighbors according to these distances:** For each data point, its k nearest neighbor is determined by that distance.
- **Create a neighborhood plot:** the edges of each point are aligned with their closest neighbors, which creates a diagram that represents the data's regional structure.
- **Calculate geodesic distances:** The Floyd algorithm sorts through all the pairs of data points in a neighborhood graph and finds the most distant paths. geodesic distances are represented by these shortest paths.
- **Perform dimensional reduction:** Classical Multi Scaling MDS is used for geodesic distance matrices that result in low dimensional embedding of data.

```
from sklearn.datasets import make_s_curve from
sklearn.manifold import Isomap import
matplotlib.pyplot as plt

# Generate S-curve data
X, color = make_s_curve(n_samples=1000, random_state=42)

# Apply Isomap
isomap = Isomap(n_neighbors=10, n_components=2) X_isomap =
isomap.fit_transform(X)

# Plot the original and reduced-dimensional data fig, ax =
plt.subplots(1, 2, figsize=(12, 5))

ax[0].scatter(X[:, 0], X[:, 2], c=color, cmap=plt.cm.Spectral)
ax[0].set_title('Original 3D Data')

ax[1].scatter(X_isomap[:, 0], X_isomap[:, 1], c=color,
cmap=plt.cm.Spectral)
ax[1].set_title('Isomap Reduced 2D Data')

plt.show()
```



This sample of code illustrates how to apply the dimensionality reduction method Isomap to a dataset of S curves. Plotting the original 3D data next to the reduced 2D data for visualization follows the generation of S-curve data with 3D coordinates using Isomap. The fundamental connections between data points in a lower-dimensional space are preserved by the Isomap transformation, which captures the underlying geometric structure. With the inherent patterns in the data still intact, the resultant visualization shows how effective Isomap is at unfolding the S-curve structure in a more manageable 2D representation.

Advantages and Disadvantages of Isomap

Advantages

- **Capturing non linear relationships:** Unlike linear dimensional reduction techniques such as PCA, Isomap is able to capture the underlying non linear structure of the data.
- **Global structure**: Isomap's goal is to preserve the overall relationship between data points, which will give a better representation of the entire manifold.
- **Globally optimised:** The algorithm guarantees that on the built neighborhood graph, where geodesic distances are defined, a global optimal solution will be found.

Disadvanatges

- **Computational cost**: for large datasets, computation of geodesic distance using Floyd's algorithm can be computationally expensive and lead to a longer run time.
- **Sensitive to parameter settings**: incorrect selection of the parameters may lead to a distortion or misleading insert.
- May be difficult for manifolds with holes or topological complexity, which may lead to inaccurate representations: Isomap is not capable of performing well in a manifold that contains holes or other topological complexity.

Applications of Isomap

- **Visualization**: High-dimensional data like face images can be visualized in a lower-dimensional space, enabling easier exploration and understanding.
- **Data exploration**: Isomap can help identify clusters and patterns within the data that are not readily apparent in the original high-dimensional space.
- **Anomaly detection**: Outliers that deviate significantly from the underlying manifold can be identified using Isomap.
- **Machine learning tasks**: Isomap can be used as a pre-processing step for other machine learning tasks, such as classification and clustering, by improving the performance and interpretability of the models.

- **Least Square method** is a fundamental mathematical technique widely used in **data analysis, statistics, and regression modeling** to identify the **best-fitting curve or line** for a given set of data points. This method ensures that the overall error is reduced, providing a highly accurate model for predicting future data trends.
- In statistics, when the data can be represented on a cartesian plane by using the independent and dependent variable as the x and y coordinates, it is called **scatter data.** This data might not be useful in making interpretations or predicting the values of the dependent variable for the independent variable. So, we try to get an **equation of a line that fits best to the given data points** with the help of the **Least Square Method**.

What is the Least Square Method?

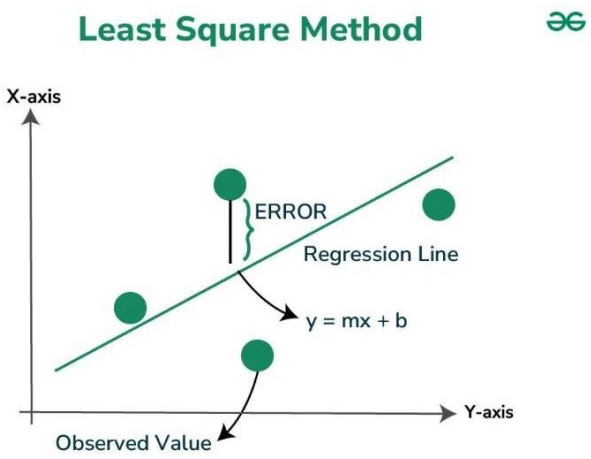
Least Square Method is used to derive a generalized linear equation between two variables. when the value of the dependent and independent variable is represented as the x and y coordinates in a 2D cartesian coordinate system. Initially, known values are marked on a plot. The plot obtained at this point is called a scatter plot.

Then, we try to represent all the marked points as a straight line or a **linear equation**. The equation of such a line is obtained with the help of the Least Square method. This is done to get the value of the dependent variable for an independent variable for which the value was initially unknown. This helps us to make predictions for the value of dependent variable.

Least Square Method Definition

Least Squares method is a statistical technique used to find the equation of best-fitting curve or line to a set of data points by minimizing the sum of the squared differences between the observed values and the values predicted by the model.

This method aims at minimizing the sum of squares of deviations as much as possible. The line obtained from such a method is called a **regression line** or **line of best fit**.



Formula for Least Square Method

Least Square Method formula is used to find the best-fitting line through a set of data points. For a simple linear regression, which is a line of the form $y=mx+c$, where y is the dependent variable, x is the independent variable, a

is the slope of the line, and *b* is the y-intercept, the formulas to

calculate the slope (m) and intercept (c) of the line are derived from the following equations:

1. **Slope (m) Formula:** $m = n(\sum xy) - (\sum x)(\sum y) / n(\sum x2) - (\sum x)2$
2. **Intercept (c) Formula:** $c = (\sum y) - a(\sum x) / n$

Where:

- n is the number of data points,
- $\sum xy$ is the sum of the product of each pair of x and y values,
- $\sum x$ is the sum of all x values,
- $\sum y$ is the sum of all y values,
- $\sum x2$ is the sum of the squares of x values.

Formula for Least Square Method

Least Square Method formula is used to find the best-fitting line through a set of data points. For a simple linear regression, which is a line of the form $y=mx+c$, where y is the dependent variable, x is the independent variable, a is the slope of the line, and b is the y-intercept, the formulas to calculate the slope (m) and intercept (c) of the line are derived from the following equations:

1. **Slope (m) Formula:** $m = n(\sum xy) - (\sum x)(\sum y) / n(\sum x2) - (\sum x)2$
2. **Intercept (c) Formula:** $c = (\sum y) - a(\sum x) / n$

Where:

- n is the number of data points,
- $\sum xy$ is the sum of the product of each pair of x and y values,
- $\sum x$ is the sum of all x values,
- $\sum y$ is the sum of all y values,
- $\sum x2$ is the sum of the squares of x values.

The steps to find the line of best fit by using the least square method is discussed below:

- **Step 1:** Denote the independent variable values as xi and the dependent ones as yi.
- **Step 2:** Calculate the average values of xi and yi as X and Y.
- **Step 3:** Presume the equation of the line of best fit as y = mx + c, where m is the slope of the line and c represents the intercept of the line on the Y-axis.
- **Step 4:** The slope m can be calculated from the following formula:

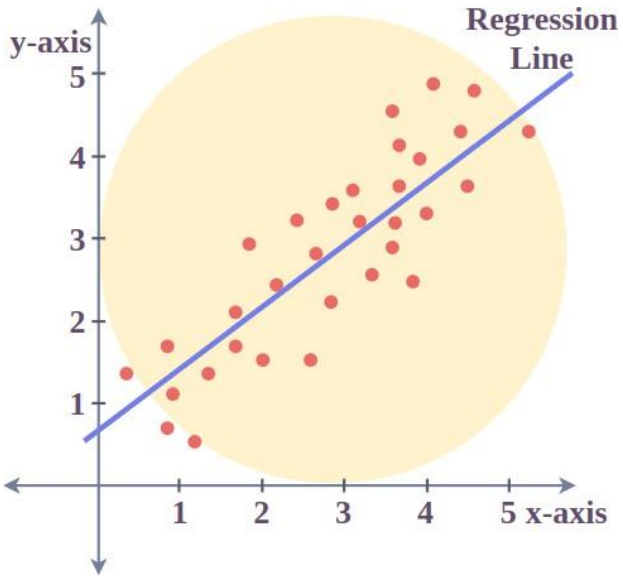
$$m = [\Sigma (X - xi) \times (Y - yi)] / \Sigma (X - xi)2$$

- **Step 5:** The intercept c is calculated from the following formula:

$$c = Y - mX$$

Thus, we obtain the line of best fit as y = mx + c, where values of m and c can be calculated from the formulae defined above.
These formulas are used to calculate the parameters of the line that best fits the data according to the criterion of the least squares, minimizing the sum of the squared differences between the observed values and the values predicted by the linear model.

Least Square Method Graph
Let us have a look at how the data points and the line of best fit obtained from the Least Square method look when plotted on a graph.



The red points in the above plot represent the data points for the sample data available. **Independent variables are plotted as x-coordinates and dependent ones are plotted as y-coordinates**. The equation of the line of best fit obtained from the Least Square method is plotted as the red line in the graph.

We can conclude from the above graph that how the **Least Square method helps us to find a line that best fits the given data points** and hence can be used to make further predictions about the value of the dependent variable where it is not known initially.

Limitations of the Least Square Method

The Least Square method assumes that the data is evenly distributed and doesn't contain any outliers for deriving a line of best fit. But, this method doesn't provide accurate results for unevenly distributed data or for data containing outliers.

Check: Least Square Regression Line

Least Square Method Solved Examples

Problem 1: Find the line of best fit for the following data points using the Least Square method: (x,y) = (1,3), (2,4), (4,8), (6,10), (8,15).

Solution:

Here, we have x as the independent variable and y as the dependent variable. First, we calculate the means of x and y values denoted by X and Y respectively.

$X = (1+2+4+6+8)/5 = 4.2$

$Y = (3+4+8+10+15)/5 = 8$

i	*yi*	*X – xi*	*Y – yi*	*(X-xi)*(Y-yi)*	*(X – xi)2*
1	*3*	*3.2*	*5*	*16*	*10.24*
2	*4*	*2.2*	*4*	*8.8*	*4.84*
4	*8*	*0.2*	*0*	*0*	*0.04*
6	*10*	*-1.8*	*-2*	*3.6*	*3.24*
8	*15*	*-3.8*	*-7*	*26.6*	*14.44*
Sum (Σ)		*0*	*0*	*55*	*32.8*

The slope of the line of best fit can be calculated from the formula as follows:

$$m = (\Sigma (X - xi)*(Y - yi)) / \Sigma(X - xi)2$$

m = 55/32.8 = 1.68 (rounded upto 2 decimal places) Now, the intercept will be calculated from the formula as follows:

$$c = Y - mX$$

$$c = 8 - 1.68*4.2 = 0.94$$

Thus, the equation of the line of best fit becomes, y = 1.68x + 0.94.

Genetic Algorithms

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random searches provided with historical data to direct the search into the region of better performance in solution space. **They are commonly used to generate high-quality solutions for optimization problems and search problems.**

Genetic algorithms simulate the process of natural selection which means those species that can adapt to changes in their environment can survive and reproduce and go to the next generation. In simple words, they simulate "survival of the fittest" among individuals of consecutive generations to solve a problem. **Each generation consists of a population of individuals** and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

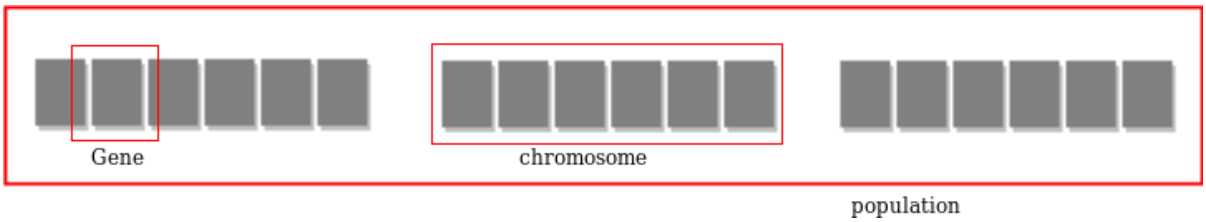
Foundation of Genetic Algorithms

Genetic algorithms are based on an analogy with the genetic structure and behavior of chromosomes of the population. Following is the foundation of GAs based on this analogy —

1. Individuals in the population compete for resources and mate
2. Those individuals who are successful (fittest) then mate to create more offspring than others
3. Genes from the "fittest" parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.
4. Thus each successive generation is more suited for their environment.

Search space

The population of individuals are maintained within search space. Each individual represents a solution in search space for given problem. Each individual is coded as a finite length vector (analogous to chromosome) of components. These variable components are analogous to Genes.
Thus a chromosome (individual) is composed of several genes (variable components).



Fitness Score

A Fitness Score is given to each individual which **shows the ability of an individual to "compete"**. The individual having optimal fitness score (or near optimal) are sought.

The GAs maintains the population of n individuals (chromosome/solutions) along with their fitness scores.The individuals having better fitness scores are given more chance to reproduce than others. The individuals with better fitness scores are selected who mate and produce **better offspring** by combining chromosomes of parents.

The population size is static so the room has to be created for new arrivals. So, some individuals die and get replaced by new arrivals eventually creating new generation when all the mating opportunity of the old population is exhausted. It is hoped that over successive generations better solutions will arrive while least fit die.

Each new generation has on average more "better genes" than the individual (solution) of previous generations. Thus each new generations have better **"partial solutions"** than previous generations. Once the offspring produced having no significant difference from offspring produced by previous populations, the population is converged. The algorithm is said to be converged to a set of solutions for the problem.

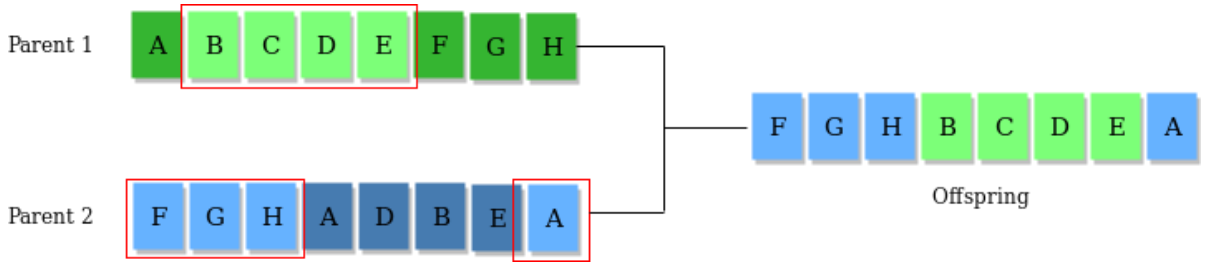
Operators of Genetic Algorithms

Once the initial generation is created, the algorithm evolves the generation using following operators —

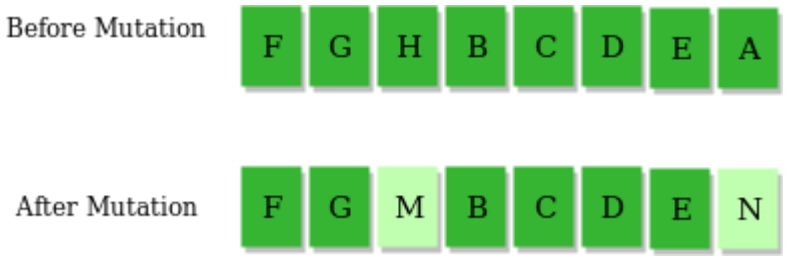
1) Selection Operator: The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generations.

2) Crossover Operator: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are

exchanged thus creating a completely new individual (offspring). For example —



Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence. For example —



The whole algorithm can be summarized as –

```
1) Randomly initialize populations p
2) Determine fitness of population
3) Until convergence repeat:
      a) Select parents from population
      b) Crossover and generate new population
      c) Perform mutation on new population
      d) Calculate fitness for new population
```

Why use Genetic Algorithms

- They are Robust
- Provide optimisation over large space state.
- Unlike traditional AI, they do not break on slight change in input or presence of noise

Application of Genetic Algorithms

Genetic algorithms have many applications, some of them are –

- Recurrent Neural Network
- Mutation testing

- Code breaking
- Filtering and signal processing
- Learning fuzzy rule base etc

R22 B.Tech. CSE (DS)

MACHINE LEARNING

B.Tech. III Year I Sem.

Unit – 5

Reinforcement Learning – Overview – Getting Lost Example

Markov Chain Monte Carlo Methods – Sampling – Proposal Distribution – Markov Chain Monte
Carlo – Graphical Models – Bayesian Networks – Markov Random Fields – Hidden Markov Models – Tracking Methods

Reinforcement Learning – Overview

A classic example of reinforcement learning using the concept of "getting lost" would be a
scenario where an autonomous robot is navigating an unfamiliar environment, like a maze, and must learn to reach a specific destination by trial and error, receiving positive rewards for moving closer to the goal and negative rewards for going further away or hitting obstacles, essentially "learning" how to navigate without getting lost through repeated attempts and feedback from the environment

Reinforcement Learning – Overview

Types of Reinforcement learning

There are mainly two types of reinforcement learning, which are:

Positive Reinforcement:

The positive reinforcement learning means adding something to increase the tendency that expected behavior would occur again. It impacts positively on the behavior of the agent and increases the
strength of the behavior.

This type of reinforcement can sustain the changes for a long time, but too much positive reinforcement may lead to an overload of states that can reduce the consequences.

Negative Reinforcement:

The negative reinforcement learning is opposite to the positive reinforcement as it increases the tendency that the specific behavior will occur again by avoiding the negative condition.

It can be more effective than the positive reinforcement depending on situation and behavior, but it provides reinforcement only to meet minimum behavior.

Elements of Reinforcement Learning

There are four main elements of Reinforcement Learning, which are given below:

1. Policy
2. Reward Signal
3. Value Function

Elements of Reinforcement Learning

There are four main elements of Reinforcement Learning, which are given below:

1. Policy

A policy can be defined as a way how an agent behaves at a given time. It maps the perceived states of the environment to the actions taken on those states. A policy is the core
element of the RL as it alone can define the behavior of the agent. In some cases, it may be a simple function or a lookup table, whereas, for other cases, it may involve general computation as a search process. It could be deterministic or a stochastic policy:

For deterministic policy: a = π(s)

For stochastic policy: π(a | s) = P[At =a | St = s]

Elements of Reinforcement Learning

There are four main elements of Reinforcement Learning, which are given below:

2) Reward Signal: The goal of reinforcement learning is defined by the reward signal. At each state, the environment sends an immediate signal to the learning agent, and this signal is known as
a **reward signal**. These rewards are given according to the good and bad actions taken by the agent. The agent's main objective is to maximize the total number of rewards for good actions. The reward signal can change the policy, such as if an action selected by the agent leads to low reward, then the policy may change to select other actions in the future.

3) Value Function: The value function gives information about how good the situation and action are and how much reward an agent can expect. A reward indicates the **immediate signal for each good and bad action**, whereas a value function specifies **the good state and action for the future**. The value function depends on the reward as, without reward, there could be no value. The goal of
estimating values is to achieve more rewards.

Elements of Reinforcement Learning

4) Model: The last element of reinforcement learning is the model, which mimics the behavior of the environment. With the help of the model, one can make inferences about how the environment will behave. Such as, if a state and an action are given, then a model can predict the next state and
reward.

The model is used for planning, which means it provides a way to take a course of action by considering all future situations before actually experiencing those situations. The approaches for solving the RL problems **with the help of the model** are termed as the **model-based approach**.
Comparatively, an approach **without using a model** is called a **model-free**

approach. Key points of this example:

- **State:**
- At any point in time, the robot's "state" is its current location within the maze, including information like proximity to walls and potential pathways.

- The robot can choose actions like "move forward", "turn left", "turn right" based on its current state.
- **Reward:**
- The robot receives a positive reward when it moves closer to the goal and a negative reward when it moves further away or hits a wall.
- **Learning process:**
- Through repeated trials, the robot learns which actions in each state lead to the highest cumulative reward, effectively building a strategy to navigate the maze without

getting lost How it works:

- **Exploration vs. Exploitation:**

  Initially, the robot might explore different paths randomly to understand the maze layout.

- **Policy Improvement:**

Over time, the robot gradually starts to prioritize actions that have led to positive rewards in the past, leading to a more efficient navigation strategy.

- **Q-Learning:**

A common reinforcement learning algorithm that can be used in this scenario involves creating a "Q-table" where each cell represents a state-action pair, storing the expected future reward for taking a particular action in a given state.

Real-world applications:

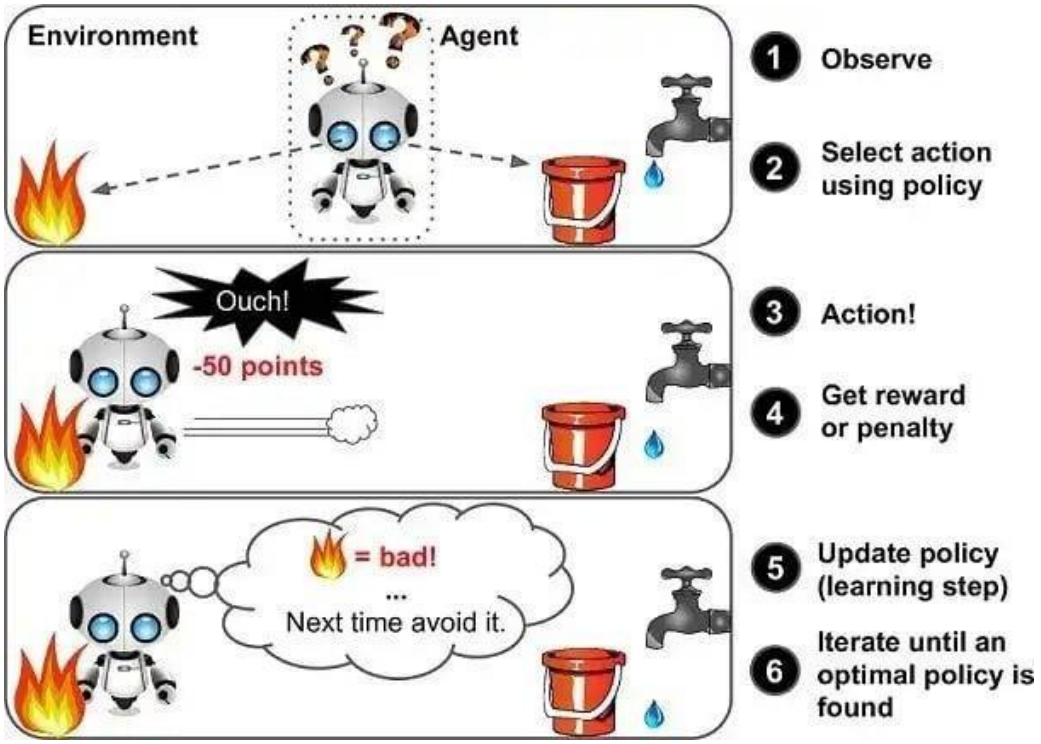
- **Robot navigation:**
- This concept is directly applicable to robots navigating complex environments like warehouses or disaster zones.
- **Autonomous vehicles:**
- Self-driving cars can use reinforcement learning to learn how to navigate roads, avoiding obstacles and making optimal route decisions.
- **Game playing:**
- AI agents in video games can learn to play effectively by receiving rewards for achieving goals within the game

Real-world applications:

- **Robot navigation:**

This concept is directly applicable to robots navigating complex environments like

warehouses or disaster zones.



Markov Chain Monte Carlo Methods – Sampling – Proposal Distribution – Markov Chain Monte
Carlo – Graphical Models – Bayesian Networks – Markov Random Fields – Hidden Markov Models – Tracking Methods

Monte Carlo methods, or **Monte Carlo experiments**, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. The name comes from the Monte Carlo Casino in Monaco, where the primary developer of the method.

Monte Carlo methods are mainly used in three distinct problem classes: **optimization, numerical integration,**

and generating draws from a probability distribution. They can also be used to model phenomena with significant uncertainty in inputs, such as calculating the risk of a nuclear power plant failure. Monte Carlo methods are often implemented using computer simulations, and they can provide approximate solutions to problems that are otherwise intractable or too complex to analyze
mathematically.

Sampling distribution is essential in various aspects of real life. Sampling distributions are important for inferential statistics. A sampling distribution represents the distribution of a statistic, like the mean or standard deviation, which is calculated from multiple samples of a population. It shows how these statistics vary across different samples drawn from the same population.

In this article, we will discuss the Sampling Distribution in detail and its types along with examples and go through some practice questions too.

Sampling distribution is also known as a **finite-sample distribution**. Sampling distribution is the probability distribution of a statistic based on random samples of a given population. It represents the distribution of frequencies on how spread apart various outcomes will be for a specific population.

Some important terminologies related to sampling distribution are given below:

- **Statistic:** A numerical summary of a sample, such as mean, median, standard deviation, etc.
- **Parameter:** A numerical summary of a population is often estimated using sample statistics.
- **Sample**: A subset of individuals or observations selected from a population.
- **Population:** Entire group of individuals or observations that a study aims to describe or draw conclusions about.
- **Sampling Distribution:** Distribution of a statistic (e.g., mean, standard deviation) across multiple samples taken from the same population.

Central Limit Theorem(CLT): A fundamental theorem in statistics stating that the sampling distribution of the sample mean tends to be approximately normal as the sample size increases, regardless of the shape of the population distribution

Some important terminologies related to sampling distribution are given below:

- **Standard Error:** Standard deviation of a sampling distribution, representing the variability of sample statistics around the population parameter.
- **Bias**: Systematic error in estimation or inference, leading to a deviation of the estimated statistic from the true population parameter.
- **Confidence Interval:** A range of values calculated from sample data that is likely to contain the population parameter with a certain level of confidence.
- **Sampling Method:** Technique used to select a sample from a population, such as simple random sampling, stratified sampling, cluster sampling, etc.
- **Inferential Statistics:** Statistical methods and techniques used to draw conclusions or make inferences about a population based on sample data.
- **Hypothesis Testing:** A statistical method for making decisions or drawing conclusions about a population parameter based on sample data and assumptions about the population.

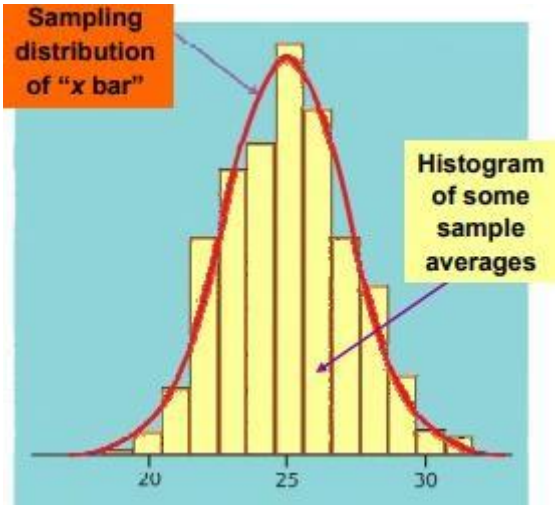
3 main factors influencing the variability of a sampling distribution are:

1. **Number Observed in a Population:** The symbol for this variable is "N." It is the measure of observed activity in a given group of data.
2. **Number Observed in Sample:** The symbol for this variable is "n." It is the measure of observed activity in a random sample of data that is part of the larger grouping.
3. **Method of Choosing Sample:** How you chose the samples can account for variability in some cases.

Sampling Distribution of Mean it focuses on calculating the mean

or rather the average of every sample group

- chosen from the population and plotting the data points. The graph shows a normal distribution where the center is the mean of the sampling distribution, which represents the mean of the entire population.
- Mean, or center of the sampling distribution of $\bar{x}$, is equal to the population mean, μ.
- $\mu_{\bar{x}} = \mu$



Standard deviation of the sampling distribution is

$\sigma/\sqrt{n}$, where n is the sample size.

$\sigma_{\bar{x}} = \sigma/\sqrt{n}$

Standard deviation of the sampling distribution

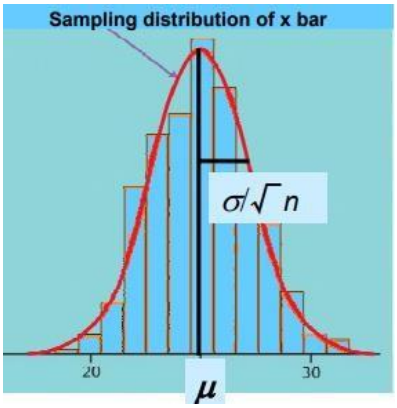
measures how much the sample statistic varies from

sample to

sample. It is smaller than the standard

deviation of the population by a factor of √n.

Averages are less variable than individual observations



T-Distribution

Sampling distribution involves a small population or a population about which you don't know much. It is used to estimate the mean of the population and other statistics such as confidence intervals, statistical differences and linear regression. T-distribution uses a t-score to evaluate data that wouldn't be appropriate for a normal distribution.

Formula for the t-score, denoted as t, is:

$$t = [x - \mu] / [s / \sqrt{(n)}]$$

where:

- **μ** is Population Mean (or an estimate of it)
- **s** is Sample Standard Deviation
- **n** is Sample Size

This formula calculates the difference between the sample mean and the population mean, scaled by the standard error of the sample mean. The t-score helps to assess whether the observed difference between the sample and population means is statistically significant.

Markov chain Monte Carlo methods create samples from a continuous random variable, with probability density proportional to a known function. These samples can be used to evaluate an integral over that variable, as its expected value or variance.

Practically, an ensemble of chains is generally developed, starting from a set of points arbitrarily chosen and sufficiently distant from each other. These chains are stochastic processes of "walkers" which move around randomly according to an algorithm that looks for places with a reasonably high contribution to the integral to move into next, assigning them higher probabilities.

Random walk Monte Carlo methods are a kind of random simulation or Monte Carlo method. However, whereas the random samples of the integrand used in a conventional Monte Carlo integration are statistically independent, those used in MCMC are autocorrelated. Correlations of samples introduces the need to use the Markov chain central limit theorem when estimating the error of mean values.

These algorithms create Markov chains such that they have an equilibrium distribution which is proportional to the function given.

In statistics, **Markov chain Monte Carlo** (**MCMC**) is a class of algorithms used to draw samples from a probability distribution. Given a probability distribution, one can construct a Markov chain whose elements' distribution approximates it – that is, the Markov chain's equilibrium distribution matches the target distribution. The more steps that are included, the more closely the distribution of the
sample matches the actual desired distribution.

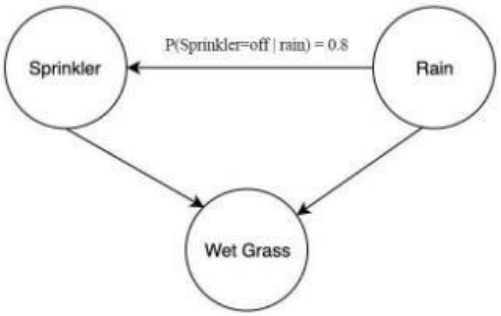
Markov chain Monte Carlo methods are used to study probability distributions that are too complex or too highly dimensional to study with analytic techniques alone. Various algorithms exist for constructing such Markov chains, including the Metropolis–Hastings algorithm.

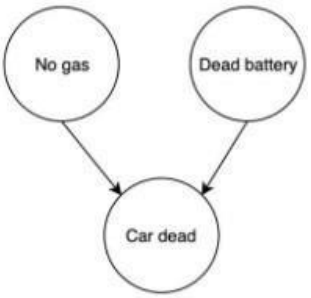
Graphical Model

One major difference between Machine Learning (ML) and Deep Learning (DL) is the amount of domain knowledge sought to solve a problem. ML algorithms regularly exploit domain knowledge. However, the solution can be biased if the knowledge is incomplete. However, if it is done correctly, we can solve problems more efficiently.
The Graphical model (GM) is a branch of ML which uses a graph to represent a domain problem. Many ML & DL algorithms, including Naive Bayes' algorithm, the Hidden Markov Model, Restricted Boltzmann machine and Neural Networks, belong to the GM. Studying it allows us a bird's eye view on many ML algorithms. In this article, we focus on the basic in representing a problem using a graphical model.
Later
in this series, we will discuss how inference is made and how the model is trained.

Probabilistic graphical modeling combines both probability and graph theory. The probabilistic part reason under uncertainty. So we can use probability theory to model and argue the real-world problems better. The graph part models the dependency or correlation. In GM, we model a domain problem with a collection of random variables ($X_1, \ldots, Xn$) as a joint distribution $p(X_1, \ldots, Xn)$. The model is represented by a graph. Each

node in the graph represents a variable with each edge representing the dependency or correlation between two variables.





For example, the sprinkler problem above is modeled by three variables:
1. whether the sprinkler is automatically on or off,
2. whether it is raining, and
3. whether the grass is wet.
And we model the problem with the joint probability $p(X_1, \ldots, X_n)$

The likelihood of the observations $P(E)$ — the chance of the lab results (Evidence E).

The marginal probability $P(X_1)$, $P(X_1, X_3)$, etc … — the chance of having Alzheimer when you are 70.

· Conditional probability (or a posterior belief based on evidence), $P(Y|E)$ — the chance of having Alzheimer given your parent has it.

· Maximum a Posterior (MAP), arg max $P(X, E)$ — the most likely disease given the lab results.

For example, the marginal probability of having the grass wet is computed by summing over other variables. For the conditional probability, we can first apply Bayes' Theorem followed by the corresponding marginal probabilities.

$$p(X_1) = \sum_{X_2} \sum_{X_3} \ldots \sum_{X_n} p(X_1, \ldots, X_n) \qquad p(X_1 \mid X_2) = \frac{\sum_{X_3} \ldots \sum_{X_n} p(X_1, \ldots, X_n)}{\sum_{X_1} \sum_{X_3} \ldots \sum_{X_n} p(X_1, \ldots, X_n)}$$

Joint probability

Which probability distributions below is more complex? Let's give some serious thoughts here because it helps us to understand machine learning (ML) better.

$$p(d, i, g, s, l), \quad p(d)\, p(i)\, p(g \mid d, i)\, p(s \mid i)\, p(l \mid g)$$

For the L.H.S. distribution, we can expand it using the chain rule.

$$p(d, i, g, s, l) = \frac{p(d)\, p(i \mid d)}{p(d, i)}\, p(g \mid d, i)\, p(s \mid d, i, g)\, p(l \mid d, i, g, s)$$

Let's assume that after further analysis of the domain problem, we discover the independence claims below. For example, the first claim is variable i is independent of d.

$i \perp d,\ s \perp \{d, g\},\ l \perp \{d, i, s\}$ ($a \perp b$ means a is independent of b)

Therefore, we can simplify the R.H.S. expression as:

Simplified to

$$p(d, i, g, s, l) = p(d)\, p(i)\, p(g \mid d, i)\, p(s \mid i)\, p(l \mid g)$$

Original: $p(d, i, g, s, l) = p(d)\, p(i \mid \cancel{d})\, p(g \mid d, i)\, p(s \mid \cancel{d}, \cancel{i}, \cancel{g})\, p(l \mid \cancel{d}, \cancel{i}, \cancel{g}, \cancel{s})$

s is independent of d or g

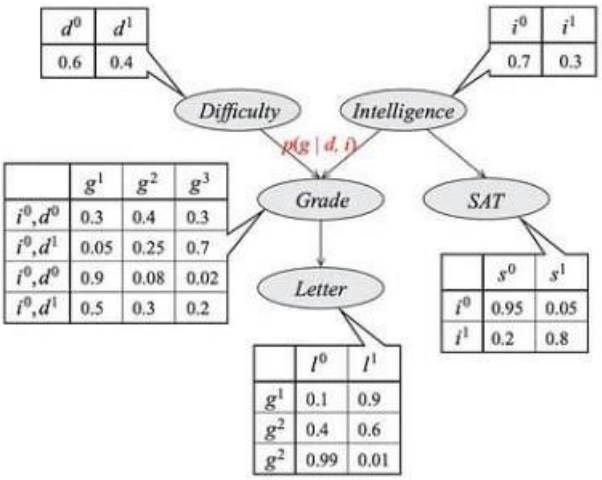
We can draw this dependency using a directed Graphical model. Each node represents a variable and to quantify the dependency, we associate a conditional probability for each node.

$$P(x_i \mid x_{A_i})$$

the parents of x_i

These model parameters are presented as the table entries below.

These model parameters are presented as the table entries below.



So what is the complexity for these models? $p(d, i, g, s, l)$ have 5 variables with the possible combinations of 32 (2^5). To model p, we sample data for 32 possibilities. On the contrary, the graph above has 4 tables with only 26 parameters. Don't underestimate the curse of exponential. 64 variables will have a combination of

18,446,744,073,709,551,616 (20 digits)

i.e. a simple 8 × 8 grayscale image will contain enough variables (pixels) that make an ML problem too complex to solve. In ML, it is important to discover independence to break the curse of this exponential complexity, for example, Naive Bayes classifier, Hidden Markov Model, Maximum likelihood estimation with i.i.d., etc… The distribution of a variable can be calculated given all its neighbors are know. In ML, the original grand design is missing. We spot global insight through localized findings.

Bayesian Belief Network

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as: "A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.

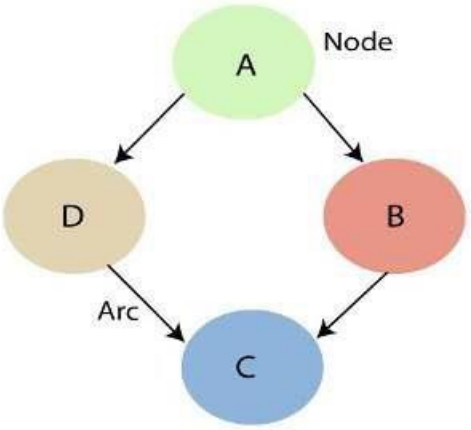
Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection. Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- **Directed Acyclic Graph**
- **Table of conditional probabilities.**

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:



Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.

- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph.

These links represent that one node directly influence the other node, and if there is

no directed link that means that nodes are independent with each other

- **In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.**
- **If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.**
- **Node C is independent of node A.**

The Bayesian network has mainly two components:

- **Causal Component**
- **Actual numbers**

Each node in the Bayesian network has condition probability distribution **P(Xi |Parent(Xi))**,

which determines the effect of the parent on that node.
Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables x1, x2, x3,..... , xn, then the probabilities of a different combination of x1,

x2, x3.. xn, are known as Joint probability distribution.

P[x1, x2, x3,. ., xn], it can be written as the following way in terms of the joint probability distribution.
= P[x1| x2, x3,....., xn]P[x2, x3,. , xn]
= P[x1| x2, x3,....., xn]P[x2|x3,....., xn]. P[xn-1|xn]P[xn].

In general for each variable Xi, we can write the equation as:
P(Xi|Xi-1,. ..... , X1) = P(Xi |Parents(Xi))

Explanation of Bayesian network:

Let's understand the Bayesian network through an example by creating a directed acyclic graph:

Problem:
Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.
Solution:

- The Bayesian network for the above problem is given below. The network structure is

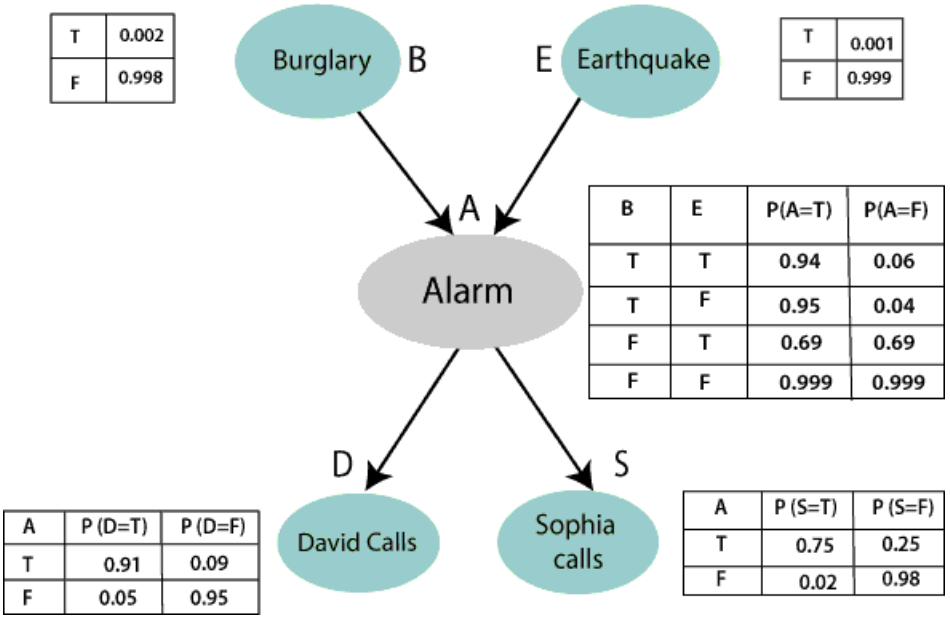
showing that burglary and earthquake is the parent node of the alarm and directly

affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.

- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table or CPT.
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.

can write the events of problem statement in the form of probability: **P[D, S, A, B, E]**, can rewrite the above probability statement using joint probability distribution:

P[D, S, A, B, E]= P[D | S, A, B, E]. P[S, A, B, E]
=P[D | S, A, B, E]. P[S | A, B, E]. P[A, B, E]
= P [D| A]. P [S| A, B, E]. P[A, B, E]
= P[D | A]. P[S | A]. P[A| B, E]. P[B, E]
= P[D | A]. P[S | A]. P[A| B, E]. P[B |E]. P[E]



T	0.002
F	0.998

T	0.001
F	0.999

B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P (D=T)	P (D=F)
T	0.91	0.09
F	0.05	0.95

A	P (S=T)	P (S=F)
T	0.75	0.25
F	0.02	0.98

Let's take the observed probability for the Burglary and earthquake component:

P(B= True) = 0.002, which is the probability of burglary. P(B= False)= 0.998, which is the probability of no burglary.

P(E= True)= 0.001, which is the probability of a minor earthquake
P(E= False)= 0.999, Which is the probability that an earthquake not occurred. We can provide the conditional probabilities as per the below tables: **Conditional probability table for Alarm A:**

The Conditional probability of Alarm A depends on Burglar and earthquake

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A= True)	P(A= False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

P(S, D, A, ¬B, ¬E) = P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E).

= 0.75* 0.91* 0.001* 0.998*0.999

= 0.00068045.

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

Gibbs Sampling

- Fix the values of observed variables
- Set the values of all non observed variables randomly
- Perform a random walk through the space of complete variable assignments. On each move
- 1. pick a variable X
- 2. calculate Pr (X=true all other variables)
- 3. set X to true with that probability

Repeat many times. Frequency with which any variable X is true is its posterior probability

Converges to true posterior when frequencies stop changing significantly

- Stationary distribution, mixing

calculate Pr (X=true all other variables)

Recall: a variable is independent of all others given its Markov Blanket

- parents - children - other parents of

children So problem becomes calculating

Pr(X=true

MB(x))

- we solve this sub-problem exactly

- fortunately, it is easy to solve

P(X)= α P(X| Parents(X)) π P(Y| Parents(Y))

P(X|A,B,C) $= \frac{P(X,A,B,C)}{P(A,B,C)}$

$= \frac{P(A)P(X|A)P(C)P(B|X,C)}{P(A,B,C)}$

= P(X|A)P(B|X,C)

= αP(X|A)P(B|X,C)

P(X|E) $= \frac{number\ of\ samples\ with\ X=x}{total\ number\ of\ samples}$

Advantages

- **No samples are discarded**
- **No problem with samples of low weight Can be implemented very efficiently**

- **Disadvantages**
- **Can get stuck if relationship between two variables**
- **Many variations have been devised to make MCMCM**

Markov Random fields

A Markov random Field is an undirected graph

where each node captures the (discrete or Gaussian) probability distribution of a variable and

the edges represent dependencies between those variables and are weighted to represent the relative strengths of the dependencies.

The defining difference between a Bayesian network and a Markov random field is that a Bayesian network is directed while a Markov random field is undirected.

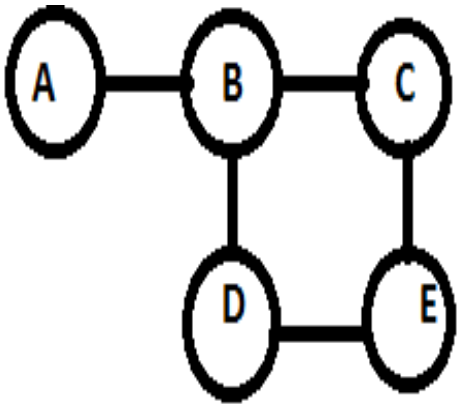
a hidden Markov model is directed and thus a sub-type of Bayesian network rather than a sub-type of Markov random field.

A **conditional random field** is a Markov random field that has specifically been set up to allow the values of a specific variable or variables to be predicted based on the values of the other variables

Undirected graphical models edge represents the potential between two variables, syntactically, Factorization distribution probabilities between variable.

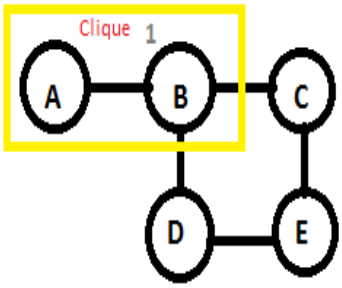
CPDs for Bayesian networks, we have tables to incorporate relations between nodes in **Markov networks**.

However, there are two crucial differences between these tables and CPDs.



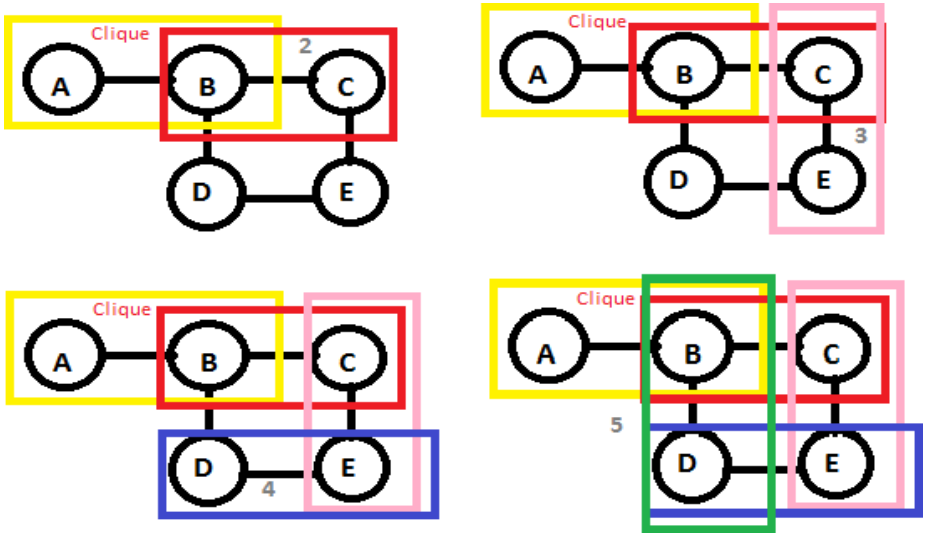
Clique in graph theory.it is a subset of vertices of an undirected graph.

P(A, B, C, D, E) α Φ(A,B) Φ (B,C) Φ (B,D) Φ (C,E) Φ (D,E)

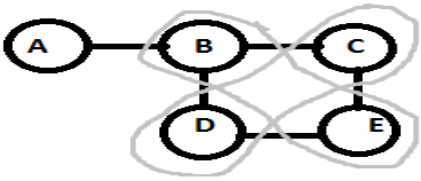


P(X) = $\frac{1}{z}\bar{\Pi}_{Ceclique}\phi_C(x_C)$ **(Potential functions)**

Such that: It induces sub graph is complete in every vertices in a clique is adjacent. So, clique in this graph adjust adjacently one by one.

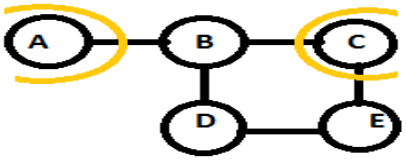


It is some different if we join **D,C** and **B,E** clique over here then it is also change its probability.



P(A, B, C, D, E) α Φ(A,B) Φ (B,C,D) Φ (C,D,E)

Some undirected graphic model has Markov Random Field. In MRF certain paths between **A** and **C**




```
A -> B -> C
A -> B -> D -> E -> C
```

Independence properties such as Markov properties

Any 2 subsets if variables are conditionally independent given a separating subset.

- ✓ If we take **'A'** as a subset and **'C'** as one subset then there is wore between them. So, there is no way to go

  between **'A'** and **'C'** without getting threw the subset. So, we are using **(A, B)** than **B, C, D, E**. **Therefore**, A and C are separating subsets

- ✓ Any 2 subset of variable are conditionally independent given a separating subset.
- ✓ {B,D}, {B,E} and {B,D,E} are separating subsets.

Hidden Markov Model

HMM is specified by a setoff states Q, a set of transition probabilities A, a set of observation likelihoods B, A defined start state and end state(s), and a set of observation symbols O, is not Constructed from the state set Q which means

observations may be disjoint from states.

AccordingtoSajja [2012]aMarkovchainisaspecialcaseofaweightedautomatonin which the input sequence uniquely determinesstates the automaton will go through for that input sequence. Since they can't represent inherently ambiguous problems. Markov chain is only useful for assigning probabilities to unambiguous sequences.

AMarkov chain is specifiedbythefollowing components:

Q= q1q2.. .qN aset ofstates
A = [aij]NxN a transition probability matrix *A*, each *aij*representing the probability of moving from state *i* to state *j*.
q0,*qend* a special start state and end state which are not associated with observations.
A Markov chain embodies an important assumption about these probabilities In a first-order Markov chain, the probabilityof a particular state is dependent onlyon the

immediateprevious state, Symbolically

Markov Assump琀椀on**:**P(qi|q1...qi−1) =P(qi|qi−1)

Sinceeach*aij*expressestheprobability*p*(*qj*|*qi*),thelawsofprobabilityrequirethatthe values

of the probabilities for some state must sum to 1.

· = ·1, ·2, ..., ·*N* an initial probability distribution over states. · *i*is the probabilitythattheMarkovchainwillstartinstate*i*.Some states*j*mayhave *j*=0,meaningthattheycannotbeinitial states.

Each *i*expressesthėprobability*p*(*qi*|*START*),allthe probabilitiesmustsumto1:

Markov chain is useful when we need to compute the probability for a sequence of events that we can observe in the world. It is useful even in events in which one is interested, but may not be directly observable in the world. For example forNER named entities tagsare not observed in real world as we observe sequence of various weather cold,hot,rain.ButsequenceofwordsobservedinrealworldandNERhasto infer the correct named entities tags from these word sequence.The named entity tags are said to be hidden because they are not observed.

HMM Assumptions

*MarkovAssumption***:**Afirst-orderHiddenMarkovModelinstantiatestwosimplifying assumptions. One: as incase of first-order Markov chain, the probability of a particular state is dependent only on the immediate previous state:

P(qi|q1... qi−1)=P(qi|qi−1)

Independent Assumption: The probability of an output observation *oi*is dependent only on the state that produced the observation *qi*, and not on any other state or any

other observations:

P(oi|q1. . .qi, . . . ,qn,o1, .. . ,oi, . . . ,on)=P(oi|qi)

*TheStationaryAssumption*Hereitisassumedthatstatetransitionprobabilitiesare independentofthe actualtimeatwhichthetransitionstakeplace. Mathematically, P(qt1+1 =j |qt1 =i) =P(qt2+1=j|qt2=i)fortime t1and t2

Having described the structure of an HMM, it will be logical to introduce the algorithms for computing things with them. An influential tutorial byRabiner [1989],

introduced the idea that Hidden Markov Models should be characterized by three fundamental problems: *Evaluation problem* can be used for isolated (word) recognition. *Decoding problem* is related to the continuous recognition as well as to the segmentation. *Learning problem* must be solved, if we want to train an HMM for

the subsequent use of recognition tasks.

Evaluation: Byevaluationitismeantwhatistheprobabilitythataparticular sequenceofsymbolsisproducedbyaparticularmodel?ThatisgivenanHMM

(*A*,*B*) and an observation sequence *O*, to determine the likelihood *P*(*O*| ·=·). For evaluation followingtwo algorithms are specified : the *forward algorithm* or the *backwards algorithm,* it may be notedthis does not mean the forward-backward algorithm.

Learning Generally, the learning problem is the adjustment of the HMM parameters, so that the given set of observations (called the *training set*) is represented by the model in the best way for the intended application. More explicitly given an observation sequence *O* and the set of states in the HMM, learn the HMM parameters
A and *B*. Thus it would be clear that the ``quantity'' which is to be optimized during the learning process can be different from application to application. In other words there may be several *optimization criteria* for learning, out of them a suitable one is selected depending on the application.
Inessenceitistofindthemodelthatbestfitsthedata.Forobtainingthedesired model the following 3 algorithms are applied:

- MLE(maximumlikelihood estimation)
- Viterbitraining(differentfromViterbidecoding)
- BaumWelch=forward-backwardalgorithm

AdvantagesandDisadvantagesof HMM

. Theunderlyingtheoreticalbasisismuchmoresound,elegantandeasyto understand.
Itiseasiertoimplementandanalyze.
: HMM taggers are very simple to train (just need to compile counts from thetraining corpus).
. Performsrelativelywell(over 90%performanceonnamedentities).
Statisticiansarecomfortablewiththetheoreticalbaseof HMM. Libertytomanipulatethe
· trainingandverification processes.
. Mathematical/theoreticalanalysisoftheresultsandprocesses.
Incorporatespriorknowledgeintothearchitecturewithgooddesign.
· Initializethemodelclosetosomething believedtobe correct.
: Iteliminateslabelbiasproblem.

It has also been proved effective for a number of other tasks, such as speechrecognition, handwriting recognitionand sign language recognition.

. Because each HMM uses onlypositive data, they scale well; since new words can be added without affecting learnt HMMs.

Disadvantages:

. In order to define joint probabilityover observation and label sequence HMM needs to enumerate all possible observation sequences.

. Maindifficultyis modeling probabilityofassigning atagto wordcan be very difficult if "words" are complex.

. Itisnotpracticaltorepresentmultipleoverlappingfeaturesandlongterm dependencies.

. Number of parameters to be evaluated is huge. So it needs a large data set for training.

· Itrequires hugeamountoftraininginordertoobtainbetter results.

. HMMs onlyuse positive data to train. In other words, HMM training involves maximizing the observed probabilities for examples belonging to a class. But it does not minimize the probability of observation of instances from other classes.

. It adopts the Markovian assumption: that the emission and the transition probabilities depend only on the current state, which does not map well to many real-world domains;

Tracking Methods

Object tracking aims at estimating bounding boxes and the identities of objects in videos. It takes in a set of initial object detection, develops a visual model for the objects, and tracks the objects as they move around in a video. Furthermore, object tracking enables us to assign a unique ID to each tracked object, making it possible for us to count unique objects in a video. Often, there's an indication around the object being tracked, for example, a surrounding square that follows the object, showing the user where the object is

Object tracking has a wide range of applications in computer vision, such as surveillance, human-computer interaction, traffic flow monitoring, human activity recognition.

Simple Online And Realtime Tracking (SORT)

Simple Online And Realtime Tracking (SORT) is a lean implementation of a tracking-by-detection framework. Keeping in line with Occam's Razor, it ignores appearance features beyond the detection component. SORT uses the position and size of the bounding boxes for both motion estimation and data association through frames. Faster RCNN is used as the object detector. The displacement of objects in the consecutive frames is estimated by a linear constant velocity model which is independent of other objects and camera motion. The state of each target is defined as x = [u, v, s, r, u,' v,' s'] where (u,v) represents the center of the bounding box r and u indicate scale and aspect ratio. The other variables are the respective velocities.

For the ID assignment, i.e., data association task the new target states are used to predict the bounding boxes that are later on compared with the detected boxes in the current timeframe. The IOU metric and the Hungarian algorithm are utilized for choosing the optimum box to pass on the identity. SORT achieves **74.6 MOTA** and 76.9 IDF1 on the MOT17 dataset with 291 ID switches and **30 FPS**.

DeepSORT

Despite achieving overall good performance in terms of tracking precision and accuracy, SORT has a high number of identity switches. It fails in many of the challenging scenarios like occlusions, different camera angles, etc. To overcome these limitations DeepSORT replaces the association metric with a more informed metric that combines motion and appearance information. In particular, a "deep appearance" distance metric is added. The core idea is to obtain a vector that can be used to represent a given image.

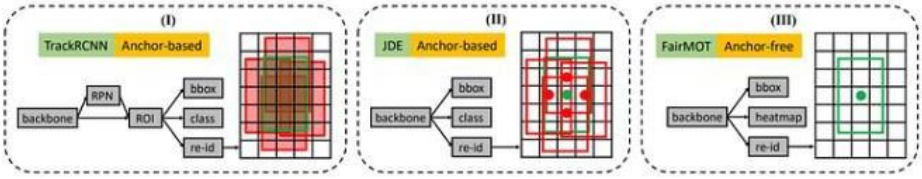
To do this DeepSORT creates a classifier and strips the final classification layer, this leaves us with a dense layer that produces a single feature vector.

In addition to that, DeepSORT adds extra dimensions to its motion tracking model. The state of each target is denoted on the eight-dimensional state space (u, v, γ, h, x,˙ y,˙ γ, ˙ h˙) that contains the bounding box center position (u, v), aspect ratio γ, height h, and their respective velocities in image coordinates. These additions enable DeepSORT to effectively handle challenging scenarios and reduce the number of identity switches by 45%. DeepSORT

achieves **75.4 MOTA** and 77.2 IDF1 on the MOT17 dataset with 239 ID switches but a lower **FPS of 13**.

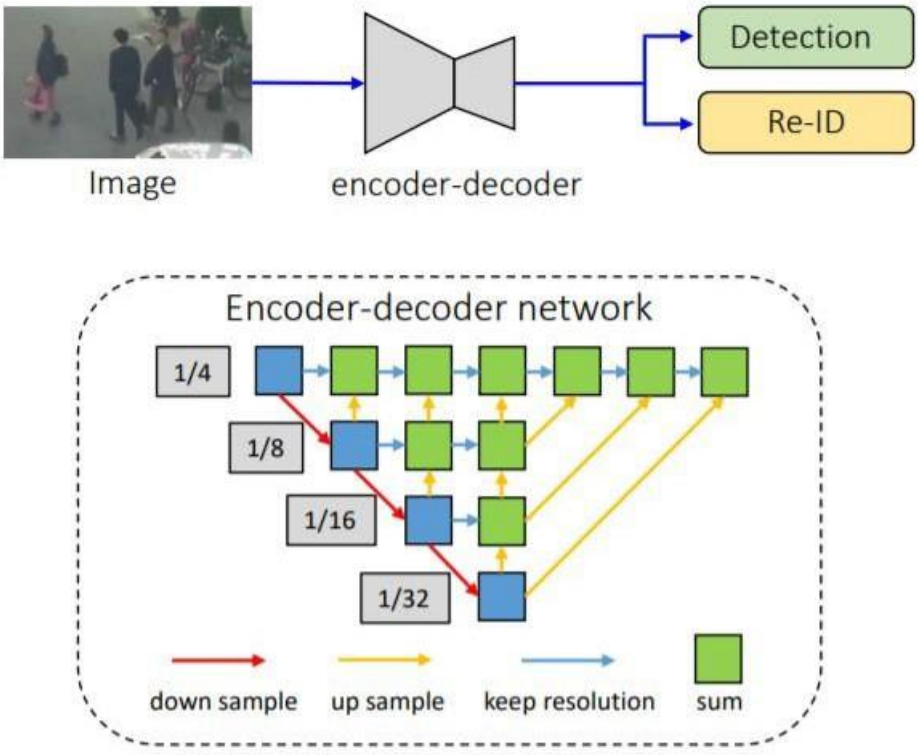
FairMOT

Approaching object tracking from a multi-task learning perspective of object detection and re-
ID in a single network is appealing since it allows shared optimization of the two tasks. However, the two tasks tend to compete with each other. In particular, previous works usually treat re-ID(re-Identification) as a secondary task whose accuracy is heavily affected by the primary detection task. As a result, the network is biased to the primary detection task which is not fair to the re-ID task.



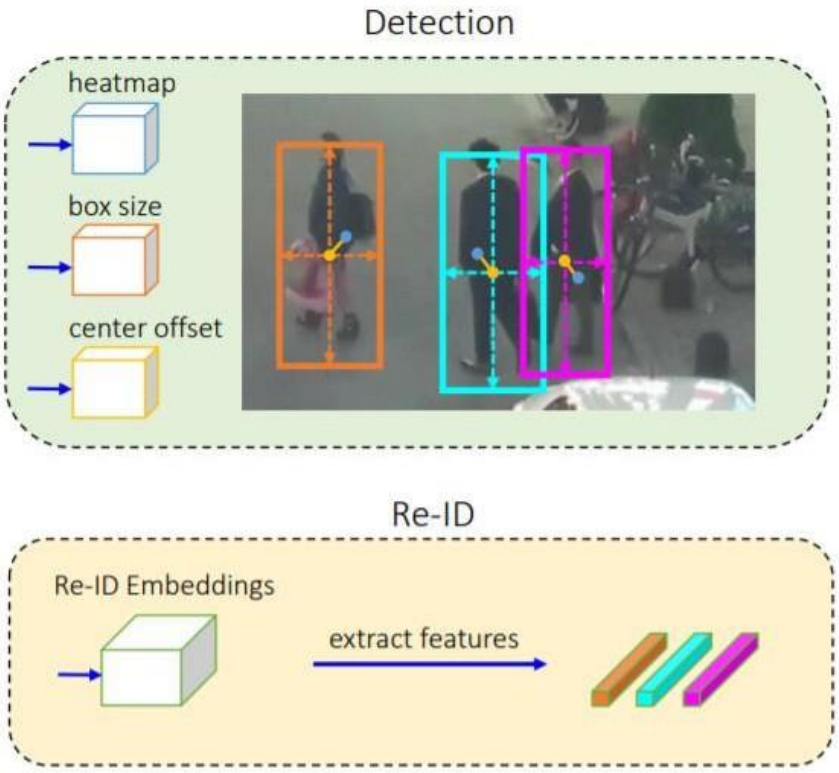
Comparison of the existing one-shot trackers and FairMOT. The three methods extract re-ID features differently. TrackR-CNN extracts re-ID features for all positive anchors using ROI-Align. JDE extracts re-ID features at the centers of all positive anchors. FairMOT extracts re-ID features at the object center

FairMOT is a new tracking approach built on top of the anchor-free object detection architecture CenterNet. The detection and re-ID tasks are treated equally in FairMOT which essentially differs from the previous “detection first, re-ID secondary” frameworks. It has a simple network structure that consists of two homogeneous branches for detecting objects and extracting re-ID features.



Overview of FairMOT and its encoder-decoder network

FairMOT adopts ResNet-34 as the backbone as it offers a good balance between accuracy and speed. An enhanced version of Deep Layer Aggregation (DLA) is applied to the backbone to fuse multi-layer. In addition, convolution layers in all up-sampling modules are replaced by deformable convolutions so that they can dynamically adjust the receptive field according to object scales and poses. These modifications also help to solve the alignment issue.



Detection and Re-ID branches

The detection branch is built on top of CenterNet, three parallel heads are appended to DLA-34 to estimate heatmaps, object center offsets, and bounding box sizes, respectively. The Re-ID branch aims to generate features that can distinguish objects. Ideally, affinity among different objects should be smaller than that between the same objects. To achieve this goal, FairMOT applies a convolution layer with 128 kernels on top of backbone features to extract re-ID features for each location. The re-ID features are learned through a classification task. All object instances of the same identity in the training set are treated as the same class. All these optimizations help FairMOT achieve **77.2 MOTA** and 79.8 IDF1 on the MOT17 dataset

with **25.9 FPS** running speed.

TransMOT

Advances in deep learning have inspired us to learn spatial-temporal relationships using deep
learning. Especially, the success of Transformer suggests a new paradigm of modeling temporal dependencies through the powerful self-attention mechanism. But transformer-based

trackers have not been able to achieve state-of-the-art performance for several

reasons

Videos can contain a large number of objects. Modeling the spatial-temporal relationships of these objects with a general Transformer is inefficient because it does not take the

spatial-temporal structure of the objects into consideration.

. It requires a lot of computation resources and data for a transformer to model long-term

temporal dependencies.

. The DETR-based object detector used in these works is still not state-of-the-art.

. TransMOT is a new spatial-temporal graph Transformer that solves all these issues. It arranges the trajectories of all the tracked objects as a series of sparse weighted graphs that are constructed using the spatial relationships of the targets. TransMOT then uses these graphs to create a spatial graph transformer encoder layer, a temporal transformer encoder layer, and a spatial transformer decoder layer to model the spatial-temporal relationships of the objects. The sparsity of the weighted graph representations makes it more computationally efficient during training and inference. It is also a more effective model than a regular transformer because it exploits the structure of the objects.

. To further improve the tracking speed and accuracy, TransMOT introduces a cascade association framework to handle low-score detections and long-term occlusions. TransMOT can also be combined with different object detectors or visual feature extraction sub-networks to form a unified end-to-end solution. This enables developers
to leverage state-of-the-art object detectors for object tracking. TransMOT achieves **76.7 MOTA** and 75.1 IDF1 on the MOT17 dataset with **9.6 FPS**.

· **ByteTrack**

. Most tracking methods obtain identities by associating detection boxes with scores higher than a threshold. The objects with low detection scores are simply ignored,

which brings non-negligible true object missing and fragmented trajectories.

BYTE is
an effective association method that utilizes all detection boxes from high scores to low ones in the matching process.