DEEP LEARNING

Unit -1

INTRODUCTION

UNIT I	INTRODUCTION	9
Introduction to machine learning- Linear models (SVMs and Perceptrons, logistic regression)- Intro to Neural Nets: What a shallow network computes- Training a network: loss functions, back propagation and stochastic gradient descent- Neural networks as universal function approximates.		

Introduction to Machine Learning:

A computer program which learns from experience is called a machine learning program or simply a learning program. Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data, or both.

Definition of learning:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E.

Examples:

i) Handwriting recognition learning problem

• Task T: Recognizing and classifying handwritten words within images

• Performance P: Percent of words correctly classified

• Training experience E: A dataset of handwritten words with given classifications

ii) A robot driving learning problem

• Task T: Driving on highways using vision sensors

• Performance measure P: Average distance traveled before an error

• Training experience: A sequence of images and steering commands recorded while observing a human driver

iii) A chess learning problem

• Task T: Playing chess

• Performance measure P: Percent of games won against opponents

• Training experience E: Playing practice games against itself

Components of Learning:

The learning process, whether by a human or a machine, can be divided into four components,namely, data storage, abstraction, generalization and evaluation. Figure 1.1 illustrates the various components and the steps involved in the learning process.

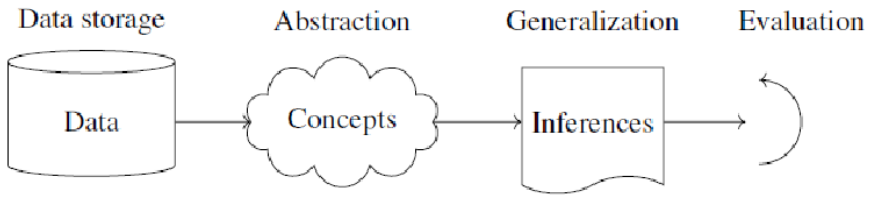


Figure 1.1: Components of learning process

1. Data storage

Facilities for storing and retrieving huge amounts of data are an important component of the learning process. Humans and computers alike utilize data storage as a foundation for advanced reasoning.

• In a human being, the data is stored in the brain and data is retrieved using electrochemical signals.

• Computers use hard disk drives, flash memory, random access memory and similar devices to store data and use cables and other technology to retrieve data.

2. Abstraction

The second component of the learning process is known as abstraction.

Abstraction is the process of extracting knowledge about stored data. This involves creating general concepts about the data as a whole. The creation of knowledge involves application of known models and creation of new models.

The process of fitting a model to a dataset is known as training. When the model has been trained, the data is transformed into an abstract form that summarizes the original information.

3. Generalization

The third component of the learning process is known as generalisation.

The term generalization describes the process of turning the knowledge about stored data into a form that can be utilized for future action. These actions are to be carried out on tasks that are similar, but not identical, to those what have been seen before. In generalization, the goal is to discover those properties of the data that will be most relevant to future tasks.

4. Evaluation

Evaluation is the last component of the learning process.

It is the process of giving feedback to the user to measure the utility of the learned knowledge.

This feedback is then utilized to effect improvements in the whole learning process.

Applications of machine learning:

1. In retail business, machine learning is used to study consumer behaviour.
2. In finance, banks analyze their past data to build models to use in credit applications, fraud detection, and the stock market.
3. In manufacturing, learning models are used for optimization, control, and troubleshooting.
4. In medicine, learning programs are used for medical diagnosis.
5. In telecommunications, call patterns are analyzed for network optimization and maximizing the quality of service.
6. In science, large amounts of data in physics, astronomy, and biology can only be analyzed fast enough by computers. The World Wide Web is huge; it is constantly growing and searching for relevant information cannot be done manually.
7. In artificial intelligence, it is used to teach a system to learn and adapt to changes so that the system designer need not foresee and provide solutions for all possible situations.
8. It is used to find solutions to many problems in vision, speech recognition, and robotics.
9. Machine learning methods are applied in the design of computer-controlled vehicles to steer correctly when driving on a variety of roads.
10. Machine learning methods have been used to develop programmes for playing games such as chess, backgammon etc..,

Linear Models:

The Linear Model is one of the most straightforward models in machine learning. It is the building block for many complex machine learning algorithms, including deep neural networks. Linear models predict the target variable using a linear function of the input features. The below are the most popular linear models commonly used:

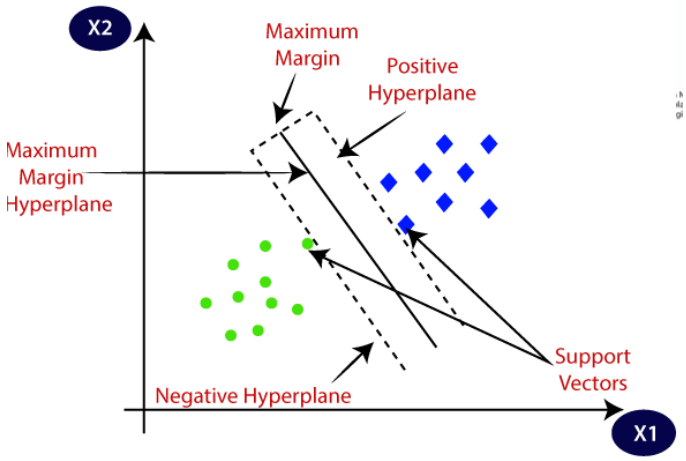
- Logistic regression
- Support Vector Machines
- Perceptrons

Support Vector Machines:

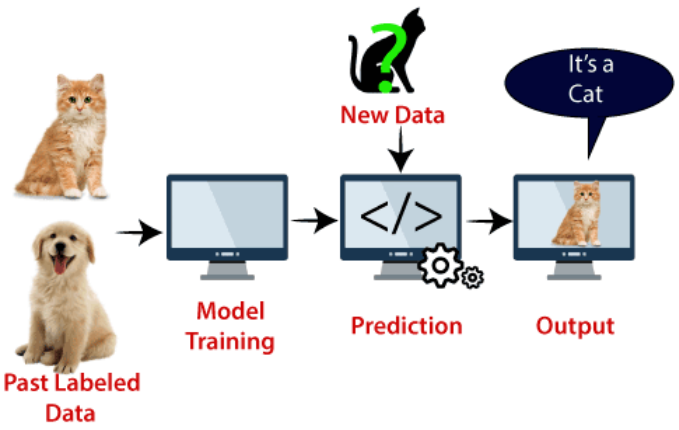
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes to put the new data point in the correct category. 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, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Example: Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Dimensions of Hyperplane:

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features , then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.The hyperplane should have maximum margin, which means the maximum distance between the data points.

Support Vectors:

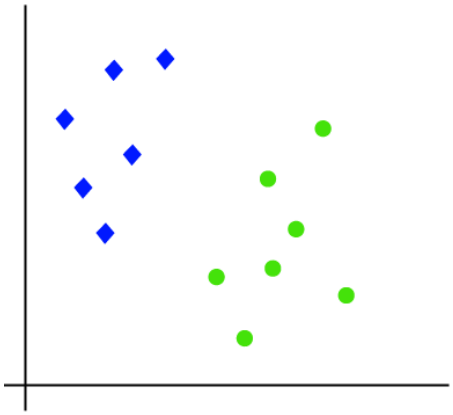
The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

Types of SVM:

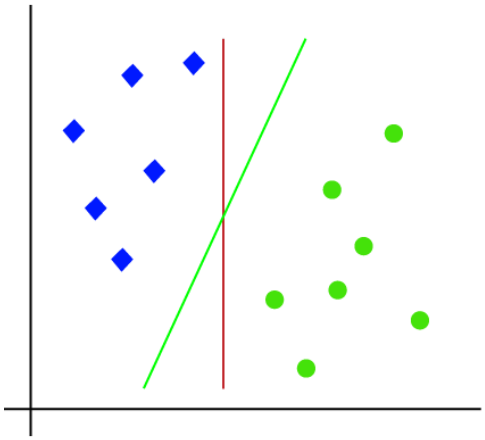
- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and the classifier used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear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

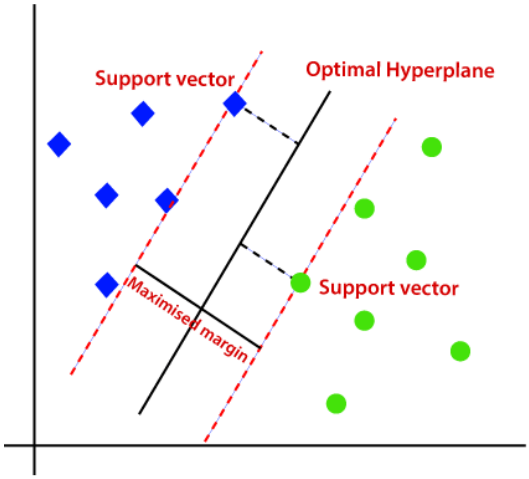
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



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:

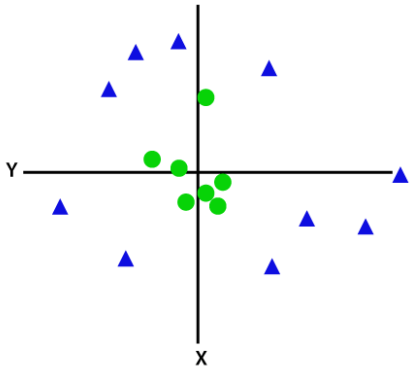


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane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

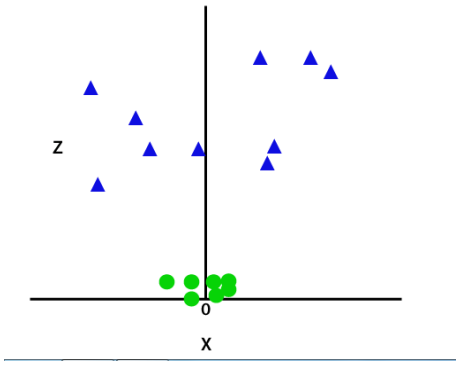
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



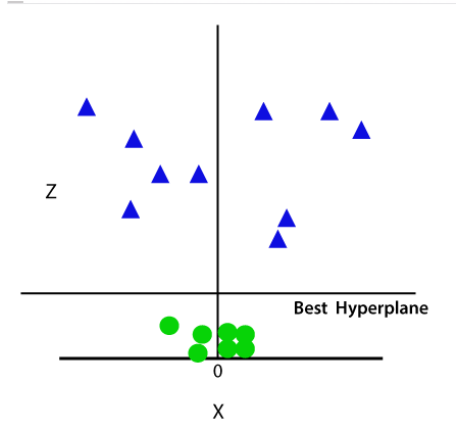
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$z=x^2 +y^2$

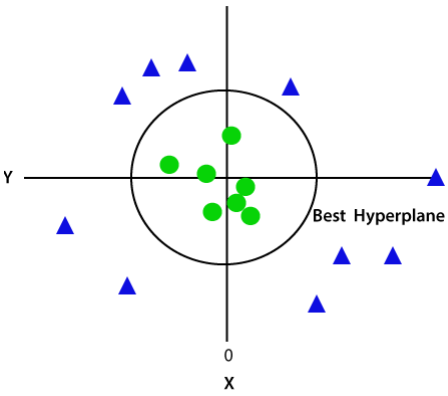
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



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

Perceptrons:

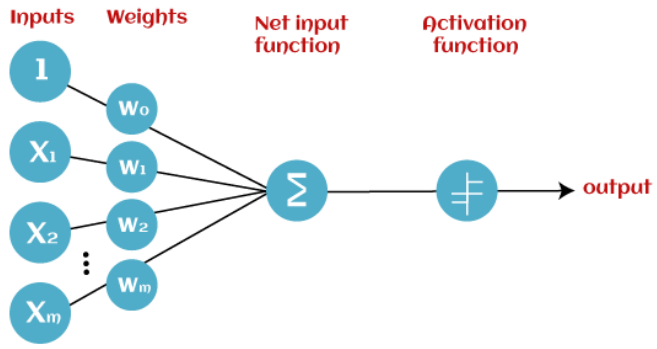
A Perceptron is a neural network unit that does certain computations to detect features or business intelligence in the input data. Perceptron is a linear Machine Learning algorithm used for supervised learning for various binary classifiers.

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.**

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:

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.

Weight and Bias:

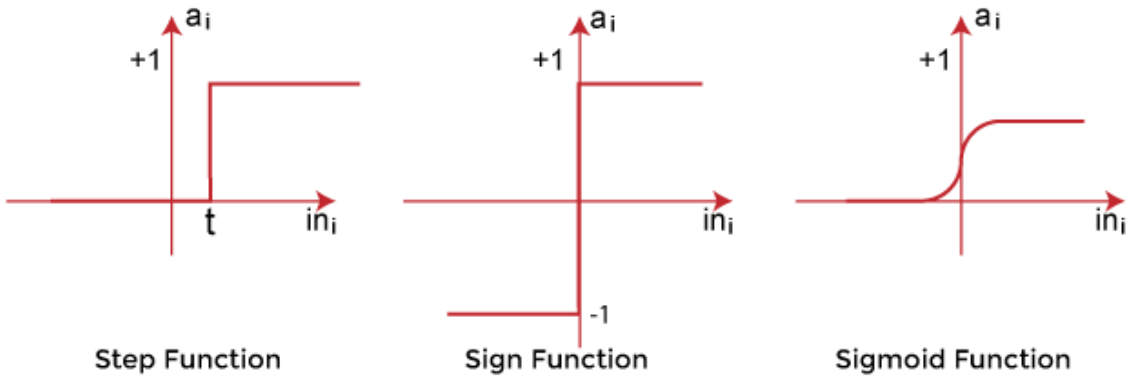
Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept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

Working of Perceptron:

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.

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

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:

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

Types of Perceptron Models

Based on the layers, Perceptron models are divided into two types. These are as follows:

1. Single-layer Perceptron Model
2. Multi-layer Perceptron model

Single Layer Perceptron Model:

It is one of the easiest Artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with inconstantly allocated input for weight parameters. Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.

If the outcome is same as pre-determined or threshold value, then the performance of this model is stated as satisfied, and weight demand does not change. However, this model consists of a few discrepancies triggered when multiple weight inputs values are fed into the model. Hence, to find desired output and minimize errors, some changes should be necessary for the weights input.

Multi-Layered Perceptron Model:

Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.

The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

- **Forward Stage:** Activation functions start from the input layer in the forward stage and terminate on the output layer.
- **Backward Stage:** In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

Hence, a multi-layered perceptron model has considered as multiple artificial neural networks having various layers in which activation function does not remain linear, similar to a single layer perceptron model. Instead of linear, activation function can be executed as sigmoid, TanH, ReLU, etc., for deployment.

A multi-layer perceptron model has greater processing power and can process linear and non-linear patterns. Further, it can also implement logic gates such as AND, OR, XOR, NAND, NOT, XNOR, NOR.

Advantages of Multi-Layer Perceptron:

- A multi-layered perceptron model can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

Disadvantages of Multi-Layer Perceptron:

- In Multi-layer perceptron, computations are difficult and time-consuming.
- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

Limitations of Perceptron Model

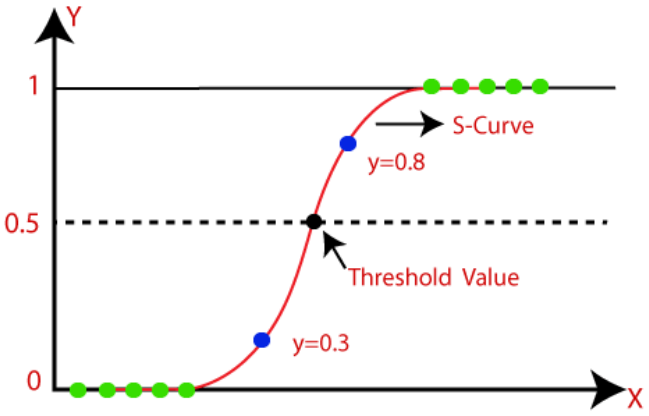
- The output of a perceptron can only be a binary number (0 or 1) due to the hard limit transfer function.
- Perceptron can only be used to classify the linearly separable sets of input vectors. If input vectors are non-linear, it is not easy to classify them properly.

Logistic regression:

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or

False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.

- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}$$; 0 for y= 0, and infinity for y=1

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

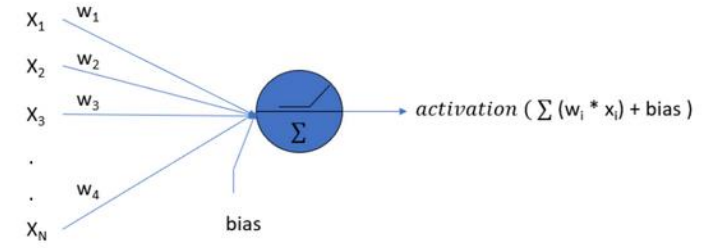
Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Intro to Neural Nets:

Neural networks are used to mimic the basic functioning of the human brain and are inspired by how the human brain interprets information. It is used to solve various real-time tasks because of its ability to perform computations quickly and its fast responses.



A single neuron shown with X_i inputs with their respective weights W_i and a bias term and applied activation function

An Artificial Neural Network model contains various components that are inspired by the biological nervous system. Artificial Neural Network has a huge number of interconnected processing elements, also known as Nodes. These nodes are connected with other nodes using a connection link. The connection link contains weights, these weights contain the information about the input signal. Each iteration and input in turn leads to updating of these weights. After inputting all the data instances from the training data set, the final weights of the Neural Network along with its architecture are known as the Trained Neural Network. This process is called Training of Neural Networks. This trained neural network is used to solve specific problems as defined in the problem statement.

Types of tasks that can be solved using an artificial neural network include Classification problems, Pattern Matching, Data Clustering, etc.

Real-life applications of neural networks include Air Traffic Control, Optical Character Recognition as used by some scanning apps like Google Lens, Voice Recognition, etc.

Types of Neural Networks

ANN– It is also known as an artificial neural network. It is a feed-forward neural network because the inputs are sent in the forward direction. It can also contain hidden layers which can make the model even denser. They have a fixed length as specified by the programmer. It is used for Textual Data or Tabular Data. A widely used real-life application is Facial Recognition. It is comparatively less powerful than CNN and RNN.

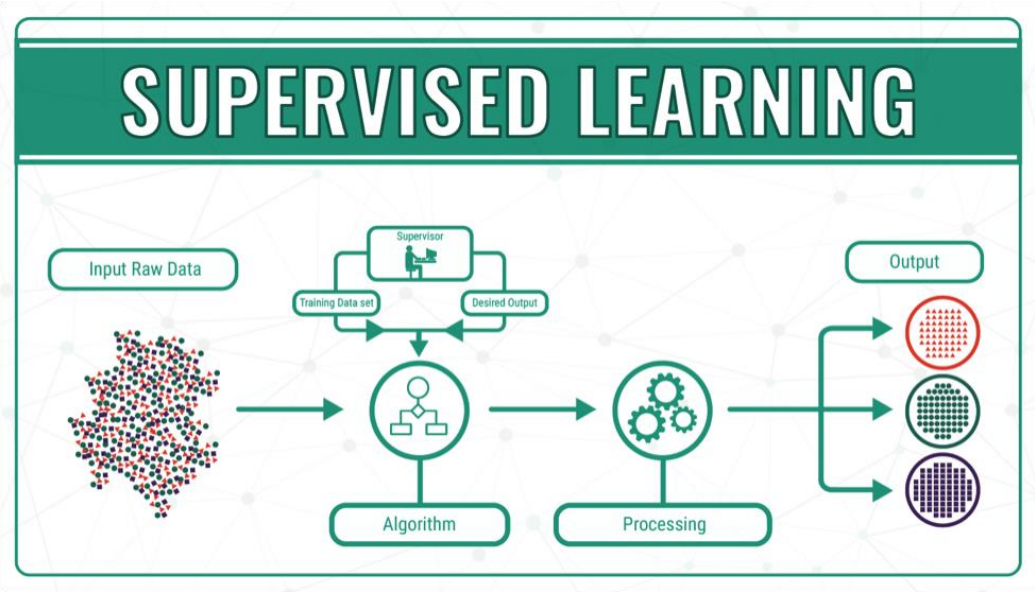
CNN– It is also known as Convolutional Neural Networks. It is mainly used for Image Data. It is used for Computer Vision. Some of the real-life applications are object detection in autonomous vehicles. It contains a combination of convolutional layers and neurons. It is more powerful than both ANN and RNN.

RNN-It is also known as Recurrent Neural Networks. It is used to process and interpret time series data. In this type of model, the output from a processing node is fed back into nodes in the same or previous layers. The most known types of RNN are **LSTM** (Long Short Term Memory) Networks

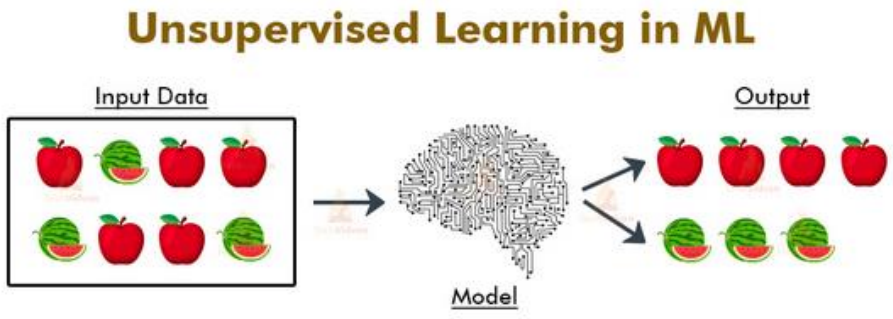
There are 3 types of **learnings in Neural networks**, namely

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

Supervised Learning: There are input training pairs that contain a set of input and the desired output. Here the output from the model is compared with the desired output and an error is calculated, this error signal is sent back into the network for adjusting the weights. This adjustment is done till no more adjustments can be made and the output of the model matches the desired output. In this, there is feedback from the environment to the model.



Unsupervised Learning: In this type of learning, there is no feedback from the environment, there is no desired output and the model learns on its own. During the training phase, the inputs are formed into classes that define the similarity of the members. Each class contains similar input patterns. On inputting a new pattern, it can predict to which class that input belongs based on similarity with other patterns. If there is no such class, a new class is formed.

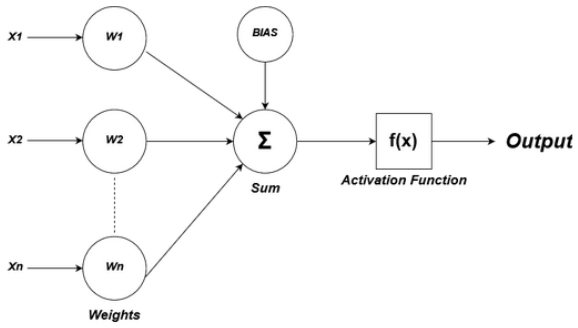


Reinforcement Learning: It gets the best of both worlds, that is, the best of both Supervised learning and Unsupervised learning. It is like learning with a critique. Here there is no exact feedback from the environment, rather there is critique feedback. The critique tells how close our

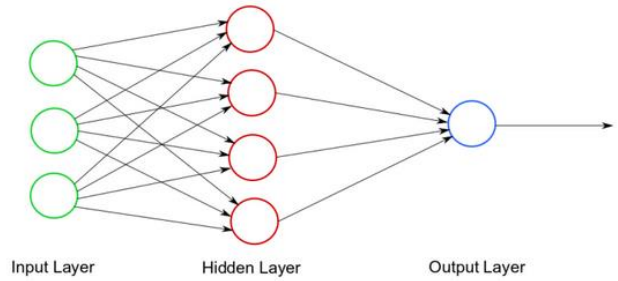
solution is. Hence the model learns on its own based on the critique information. It is similar to supervised learning in that it receives feedback from the environment, but it is different in that it does not receive the desired output information, rather it receives critique information.

Working of Neural Network:

An artificial neuron can be thought of as a simple or multiple linear regression model with an activation function at the end. A neuron from layer i will take the output of all the neurons from the later i-1 as inputs calculate the weighted sum and add bias to it. After this is sent to an activation function as we saw in the below diagram.



The first neuron from the first layer is connected to all the inputs from the previous layer, Similarly, the second neuron from the first hidden layer will also be connected to all the inputs from the previous layer and so on for all the neurons in the first hidden layer.

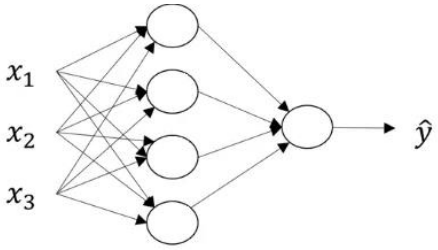


For neurons in the second hidden layer (outputs of the previously hidden layer) are considered as inputs and each of these neurons are connected to previous neurons, likewise. This whole process is called **Forward propagation**.

After the prediction of the output it is then compared to the actual output. We then calculate the loss and try to minimize it. The loss should be minimized for this minimization we undergo another concept which is known as **Back Propagation**. First, the loss is calculated then weights and biases are adjusted in such a way that they try to minimize the loss. Weights and biases are updated with the help of another algorithm called gradient descent.

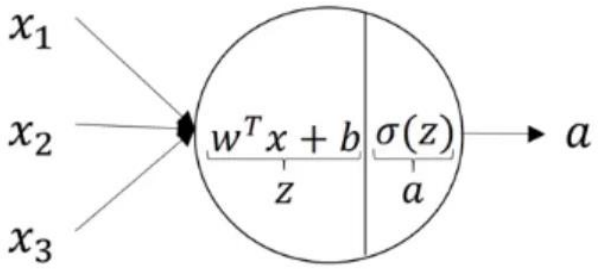
What a shallow network computes:

Shallow neural networks consist of only 1 or 2 hidden layers. The figure below shows a shallow neural network with 1 hidden layer, 1 input layer and 1 output layer.



The Neuron

The neuron is the atomic unit of a neural network. Given an input, it provides the output and passes that output as an input to the subsequent layer. A neuron can be thought of as a combination of 2 parts:



1. The first part computes the output ***Z***, using the inputs and the weights.
2. The second part performs the activation on **Z** to give out the final output **A** of the neuron.

The Hidden Layer

The hidden layer comprises of various neurons, each of which performs the above 2 calculations. The 4 neurons present in the hidden layer of our shallow neural network compute the following:

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, a_1^{[1]} = \sigma\left(z_1^{[1]}\right)$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, a_2^{[1]} = \sigma\left(z_2^{[1]}\right)$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, a_3^{[1]} = \sigma\left(z_3^{[1]}\right)$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, a_4^{[1]} = \sigma\left(z_4^{[1]}\right)$$

In the above equations,

1. The superscript number ***[i]*** denotes the layer number and the subscript number ***j*** denotes the neuron number in a particular layer.
2. ***X*** is the input vector consisting of 3 features.
3. ***W[i]j*** is the weight associated with neuron ***j*** present in the layer ***i***.
4. ***b[i]j*** is the bias associated with neuron ***j*** present in the layer ***i***.
5. ***Z[i]j*** is the intermediate output associated with neuron ***j*** present in the layer ***i***.
6. ***A[i]j*** is the final output associated with neuron ***j*** present in the layer ***i***.
7. ***Sigma*** is the sigmoid activation function. Mathematically it is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

As we can see, the above 4 equations seem redundant. Therefore we will vectorize them as:

$$Z^{[1]} = X^{[1]T}X + b^{[1]}$$

$$A^{[1]} = \sigma\left(Z^{[1]}\right)$$

1. The first equation computes all the intermediate outputs ***Z*** in single matrix multiplication.
2. The second equation computes all the activations **A** in single matrix multiplication.

The Shallow Neural Network

A neural network is built using various hidden layers. Now that we know the computations that occur in a particular layer, let us understand how the whole neural network computes the output for a given input ***X***. These can also be called the ***forward-propagation*** equations.

$$Z^{[1]} = W^{[1]T}X + b^{[1]}$$

$$A^{[1]} = \sigma\left(Z^{[1]}\right)$$

$$Z^{[2]} = W^{[2]T}A^{[1]} + b^{[2]}$$

$$\hat{y} = A^{[2]} = \sigma\left(Z^{[2]}\right)$$

1. The first equation calculates the intermediate output ***Z[1]*** of the first hidden layer.
2. The second equation calculates the final output ***A[1]*** of the first hidden layer.
3. The third equation calculates the intermediate output ***Z[2]*** of the output layer.
4. The fourth equation calculates the final output ***A[2]*** of the output layer which is also the final output of the whole neural network.

Activation Functions

The neural network is basically a set of mathematical equations and weights. To make the network robust, so that it performs well in different scenarios, we leverage activations functions. These activations functions introduce non-linear properties in the neural networks. Without the activation functions, our shallow neural network can be represented as:

$$Z^{[1]} = W^{[1]T}X + b^{[1]}$$

$$\hat{y} = Z^{[2]} = W^{[2]T}Z^{[1]} + b^{[2]}$$

If we substitute the value of Z[1] from equation 1 into equation 2, then we get the following equations:

$$Z^{[1]} = W^{[1]T}X + b^{[1]}$$
$$\hat{y} = Z^{[2]} = W^{[2]T}W^{[1]T}X + W^{[2]T}b^{[1]} + b^{[2]}$$
$$\hat{y} = Z^{[2]} = W_{new}X + b_{new}$$

As you can see, the output will become a linear combination of a new Weight Matrix ***W***, Input ***X*** and a new Bias ***b***, which means that there remains no significance of the neurons present in the hidden layer and the weights present in the hidden layer. Therefore, to introduce non-linearity in the network, we use the activation functions.

There are many activation functions that can be used. These include ***Sigmoid***, ***Tanh***, ***ReLU***, ***Leaky ReLU*** and many others. It is not mandatory to use a particular activation function for all layers. You can select an activation function for a particular layer and a different activation for another layer and so on.

Weight Initialization

Weight Matrix ***W*** of a Neural Network is randomly initialized. One may wonder, why can't initialize ***W*** with 0's or any particular value. Let ***W1***, the weight matrix of layer 1 and ***W2***, the weight matrix of layer 2 be initialized with 0 or any other value. Now, if the weight matrices are the same, the activations of neurons in the hidden layer would be the same. Moreover, the derivatives of the activations would be the same. Therefore, the neurons in that hidden layer would be modifying the weights in a similar fashion i.e. there would be no significance of having more than 1 neuron in a particular hidden layer. Instead, we want that each neuron in the hidden layer to be unique, have different weight and work as a unique function. Therefore, we initialize the weights randomly. The best method of initialization is ***Xavier's Initialization***. Mathematically it is defined as:

$$W^{[l]} \sim \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}}\right)$$

$$b^{[l]} = 0$$

It states that Weight Matrix ***W*** of a particular layer ***l*** are picked randomly from a *normal distribution* with ***mean μ= 0*** and ***variance sigma² = multiplicative inverse of the number of neurons in layer l−1***. The bias ***b*** of all layers is initialized with 0.

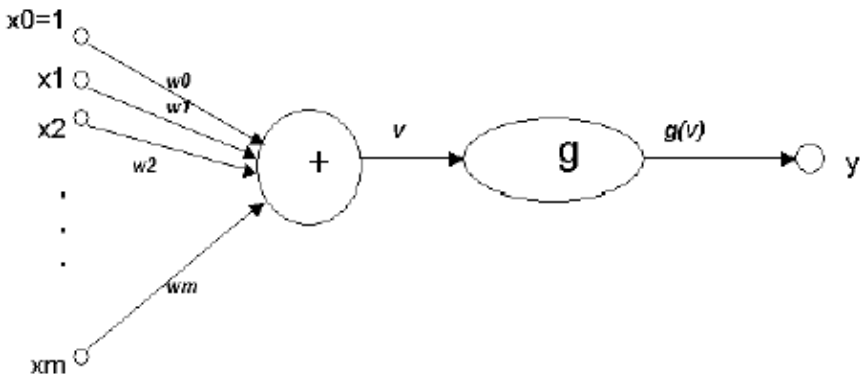
Training a network:

Artificial neural networks are relatively crude electronic networks of "neurons" based on the neural structure of the brain. They process records one at a time, and "learn" by comparing their classification of the record (which, at the outset, is largely arbitrary) with the known actual classification of the record. The errors from the initial classification of the first record is fed back into the network, and used to modify the networks algorithm the second time around, and so on for many iterations.

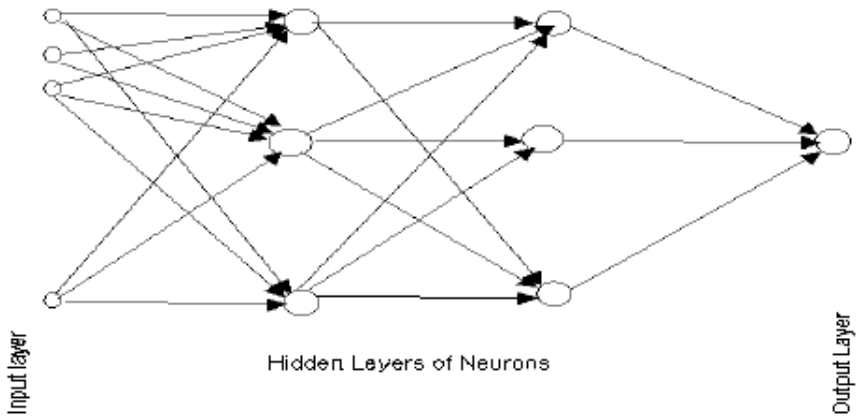
A neuron in an artificial neural network is

1. A set of input values (xi) and associated weights (wi)

2. A function (g) that sums the weights and maps the results to an output (y).



Neurons are organized into layers.



The input layer is composed not of full neurons, but rather consists simply of the values in a data record, that constitutes inputs to the next layer of neurons. The next layer is called a hidden layer; there may be several hidden layers. The final layer is the output layer, where there is one node for each class. A single sweep forward through the network results in the assignment of a value to each output node, and the record is assigned to whichever class's node had the highest value.

Training an Artificial Neural Network

In the training phase, the correct class for each record is known (this is termed supervised training), and the output nodes can therefore be assigned "correct" values -- "1" for the node corresponding to the correct class, and "0" for the others. It is thus possible to compare the network's calculated values for the output nodes to these "correct" values, and calculate an error term for each node (the "Delta" rule). These error terms are then used to adjust the weights in the hidden layers so that, hopefully, the next time around the output values will be closer to the "correct" values.

The Iterative Learning Process:

A key feature of neural networks is an iterative learning process in which data cases (rows) are presented to the network one at a time, and the weights associated with the input values are adjusted each time. After all cases are presented, the process often starts over again. During this learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of input samples. Neural network learning is also referred to as "connectionist learning," due to connections between the units. Advantages of neural networks include their high tolerance to noisy data, as well as their ability to classify patterns on which they have not been trained.

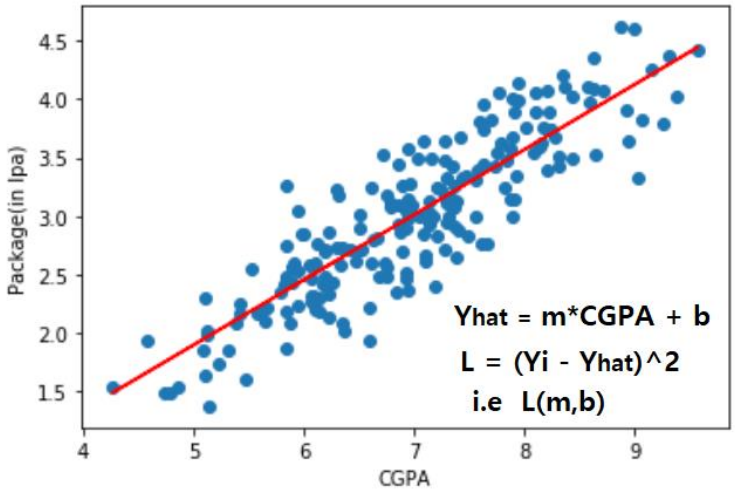
Once a network has been structured for a particular application, that network is ready to be trained. To start this process, the initial weights are chosen randomly. Then the training, or learning, begins.

The network processes the records in the training data one at a time, using the weights and functions in the hidden layers, then compares the resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights for application to the next record to be processed. This process occurs over and over as the weights are continually tweaked. During the training of a network the same set of data is processed many times as the connection weights are continually refined.

LOSS FUNCTIONS:

In simple terms, the Loss function is a method of evaluating how well your algorithm is modeling your dataset. It is a mathematical function of the parameters of the machine learning algorithm. If the value of the loss function is lower then it’s a good model otherwise, we have to change the parameter of the model and minimize the loss.

In simple linear regression, prediction is calculated using slope(m) and intercept(b). the loss function for this is the (Yi – Yihat)^2 i.e loss function is the function of slope and intercept.



Loss function vs Cost function

Loss Function:

A loss function/error function is for a single training example/input.

Cost Function:

A cost function, on the other hand, is the average loss over the entire training dataset.

Loss function in Deep Learning

1. Regression

1. MSE(Mean Squared Error)
2. MAE(Mean Absolute Error)
3. Hubber loss

2. Classification

1. Binary cross-entropy
2. Categorical cross-entropy

Regression Loss

1. Mean Squared Error

The Mean Squared Error (MSE) is the simplest and most common loss function. To calculate the MSE, you take the difference between the actual value and model prediction, square it, and average it across the whole dataset.

$$MSE = \frac{1}{N}\sum_{i}^{N}(Yi - \hat{Y}i)^2$$

Advantage

- Easy to interpret.
- Always differential because of the square.
- Only one local minima.

Disadvantage

- Error unit in the square. because the unit in the square is not understood properly.
- Not robust to outlier

2. Mean Absolute Error

The Mean Absolute Error (MAE) is also the simplest loss function. To calculate the MAE, you take the difference between the actual value and model prediction and average it across the whole dataset.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|Yi - \hat{Y}i|$$

A**dvantage**

- 1. Intuitive and easy
- 2. Error Unit Same as the output column.
- 3. Robust to outlier

Disadvantage

- Graph, not differential. we can not use gradient descent directly, then we can subgradient calculation.

3. Huber Loss

In statistics, the Huber loss is a loss function used in robust regression, that is less sensitive to outliers in data than the squared error loss.

$$Huber = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(y_i - \hat{y}_i)^2 \qquad |y_i - \hat{y}_i| \le \delta$$

$$Huber = \frac{1}{n}\sum_{i=1}^{n}\delta\left(|y_i - \hat{y}_i| - \frac{1}{2}\delta\right) \qquad |y_i - \hat{y}_i| > \delta$$

- n – the number of data points.

- y – the actual value of the data point. Also known as true value.
- $\hat{y}$ – the predicted value of the data point. This value is returned by the model.
- δ – defines the point where the Huber loss function transitions from a quadratic to linear.

Advantage

- Robust to outlier
- It lies between MAE and MSE.

Disadvantage

- Its main disadvantage is the associated complexity. In order to maximize model accuracy, the hyperparameter δ will also need to be optimized which increases the training requirements.

Classification Loss :

1. Binary Cross Entropy/log loss

It is used in binary classification problems like two classes. example a person has covid or not or my article gets popular or not.Binary cross entropy compares each of the predicted probabilities to the actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value.

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$$

- yi – actual values
- yihat – Neural Network prediction

Advantage

- A cost function is a differential.

Disadvantage

- Multiple local minima
- Not intuitive

2. Categorical Cross entropy

- Categorical Cross entropy is used for Multiclass classification.
- Categorical Cross entropy is also used in softmax regression.

loss function = -sum up to k(yjlagyjhat) where k is classes

$$\text{Loss} = -\sum_{j=1}^{K} y_j \log(\hat{y}_j)$$ where k is number of classes in the data

cost function = -1/n(sum upto n(sum j to k (yijloghijhat))

$$\text{Cost} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}[y_{ij}\log(\hat{y}_{ij})]$$

where

- k is classes,
- y = actual value
- yhat – Neural Network prediction

Note – In multi-class classification at the last neuron use the softmax activation function.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

if problem statement have 3 classes

softmax activation – f(z) = ez1/(ez1+ez2+ez3)

When to use categorical cross-entropy and sparse categorical cross-entropy?

If target column has One hot encode to classes like 0 0 1, 0 1 0, 1 0 0 then use categorical cross-entropy. and if the target column has Numerical encoding to classes like 1,2,3,4….n then use sparse categorical cross-entropy. sparse categorical cross-entropy faster than categorical cross-entropy.

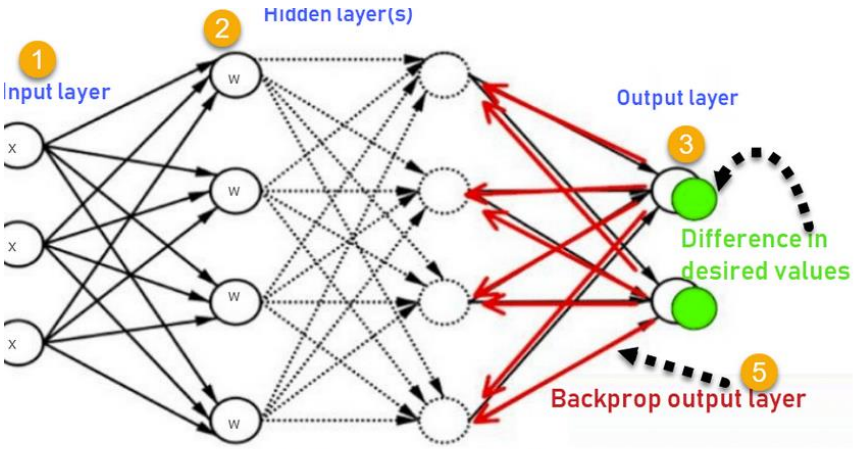
Back Propagation:

Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the iteration. Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its

generalization. Backpropagation in neural network is a short form for “backward propagation of errors.” It is a standard method of training artificial neural networks. This method helps calculate the gradient of a loss function with respect to all the weights in the network.

Working of Backpropagation Algorithm:

The Back propagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.



How Backpropagation Algorithm Works

1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs
5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

$Error_B$= Actual Output – Desired Output

Advantages of Backpropagation

- Backpropagation is fast, simple and easy to program
- It has no parameters to tune apart from the numbers of input
- It is a flexible method as it does not require prior knowledge about the network
- It is a standard method that generally works well
- It does not need any special mention of the features of the function to be learned.

Types of Backpropagation Networks

Two Types of Backpropagation Networks are:

- Static Back-propagation
- Recurrent Backpropagation

Static back-propagation:

It is one kind of backpropagation network which produces a mapping of a static input for static output. It is useful to solve static classification issues like optical character recognition.

Recurrent Backpropagation:

Recurrent Back propagation in data mining is fed forward until a fixed value is achieved. After that, the error is computed and propagated backward.

The main difference between both of these methods is: that the mapping is rapid in static back-propagation while it is nonstatic in recurrent backpropagation.

Gradient Descent:

- Gradient Descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems.
- The general idea is to tweak parameters iteratively in order to minimize the cost function.
- An important parameter of Gradient Descent (GD) is the size of the steps, determined by the learning rate hyperparameters. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time, and if it is too high we may jump the optimal value.

Types of Gradient Descent: :

1. Batch Gradient Descent
2. Stochastic Gradient Descent
3. Mini-batch Gradient Descent

Stochastic Gradient Descent:

The word '*stochastic*'means a system or process linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration.

In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, the problem arises when our dataset gets big.

Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for

completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima are reached. Hence, it becomes computationally very expensive to perform.

This problem is solved by Stochastic Gradient Descent. In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration. The sample is randomly shuffled and selected for performing the iteration.

SGD algorithm:

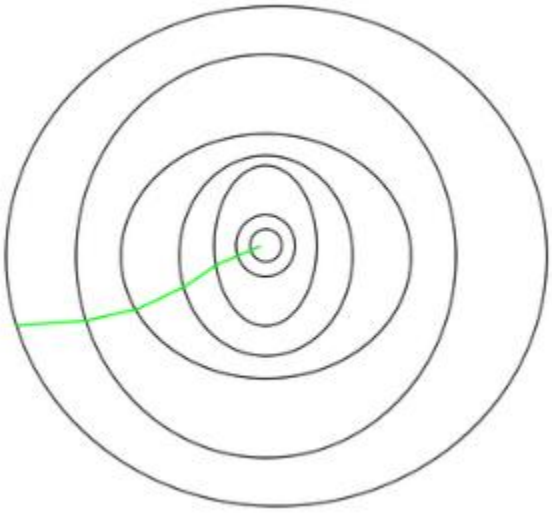
for i in range (m) :

$$\theta_j = \theta_j - \alpha \left(\hat{y}^i - y^i\right) x_j^i$$

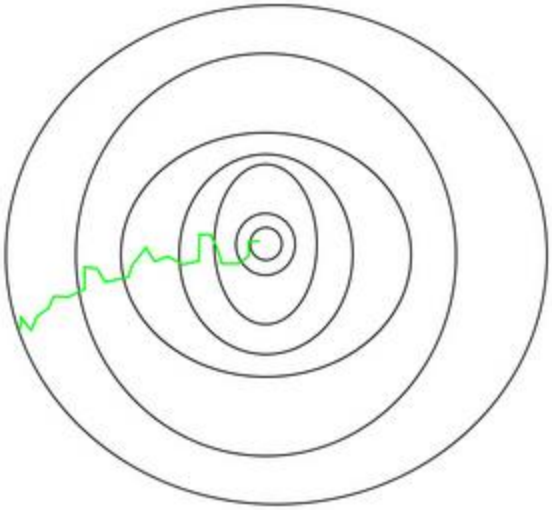
So, in SGD, we find out the gradient of the cost function of a single example at each iteration instead of the sum of the gradient of the cost function of all the examples.

In SGD, since only one sample from the dataset is chosen at random for each iteration, the path taken by the algorithm to reach the minima is usually noisier than your typical Gradient Descent algorithm. But that doesn't matter all that much because the path taken by the algorithm does not matter, as long as we reach the minima and with a significantly shorter training time.

The path is taken by Batch Gradient Descent as shown below as follows:



A path has been taken by Stochastic Gradient Descent –



One thing to be noted is that, as SGD is generally noisier than typical Gradient Descent, it usually took a higher number of iterations to reach the minima, because of its randomness in its descent. Even though it requires a higher number of iterations to reach the minima than typical Gradient Descent, it is still computationally much less expensive than typical Gradient Descent. Hence, in most scenarios, SGD is preferred over Batch Gradient Descent for optimizing a learning algorithm.

Advantages of Stochastic gradient descent:

In Stochastic gradient descent (SGD), learning happens on every example, and it consists of a few advantages over other gradient descent.

- It is easier to allocate in desired memory.
- It is relatively fast to compute than batch gradient descent.
- It is more efficient for large datasets.

Neural networks as universal function approximates:

In the mathematical theory of artificial neural networks, the universal approximation theorem states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions under mild assumptions on the activation function.

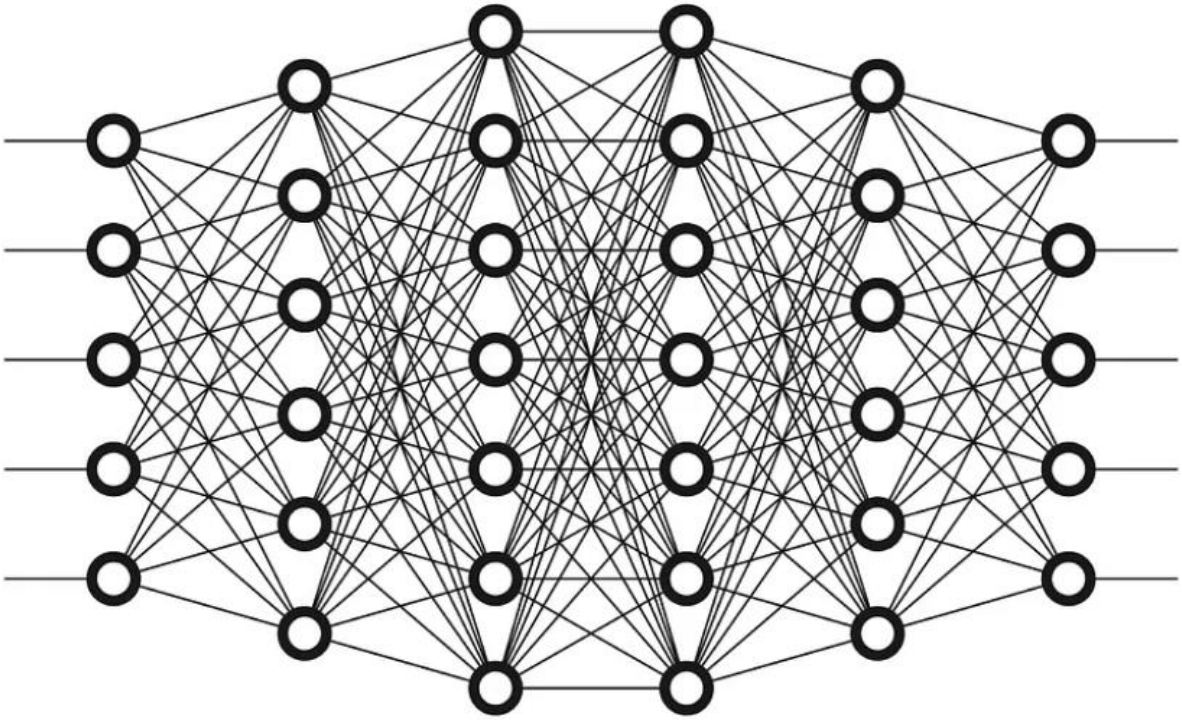
Suppose we have a mathematical function :

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

$y = x + 2x^2$

if x = { 1 , 2 , 3 , 0 } then y = { 3 , 10 , 21 , 0 }

We substituted the values of x in the equation and got the corresponding y values.



Suppose we train a neural net on the x values and set the y values as labels. If we train the network to a good extent, it will predict almost accurate values of x. It will do this without knowing the equation. Also, the equation was not hard-coded in the algorithm like,

```
x                                                   =                                                   4
```

```
y = x + ( 2 * x^2 )
```

The above equation or function was very easy and less complex. But, suppose, we make an equation which establishes a relationship between a newspaper article and its category ex.[0] (ex. 0 = politics and 1 = sports).

Text Classification

In this case, we don’t know the equation, so the neural network approximates or estimates or gives us a rough idea about the function to us. In short, they map the function.

We could estimate a function between a cell image and whether it's cancerous or not. Hence, we try to express every aspect as a mathematical function, which is to be approximated by a neural network.

A neural network consists of weights, biases, and non-linearities like sigmoid, hyperbolic tangent or ReLU. For a linear function, a network can contain linear activations like y = x. But, most functions in the world are non-linear hence we require non-linear activation functions. The values flow through the weights, get added to the bias and are activated by the activation functions.