

Unit-5

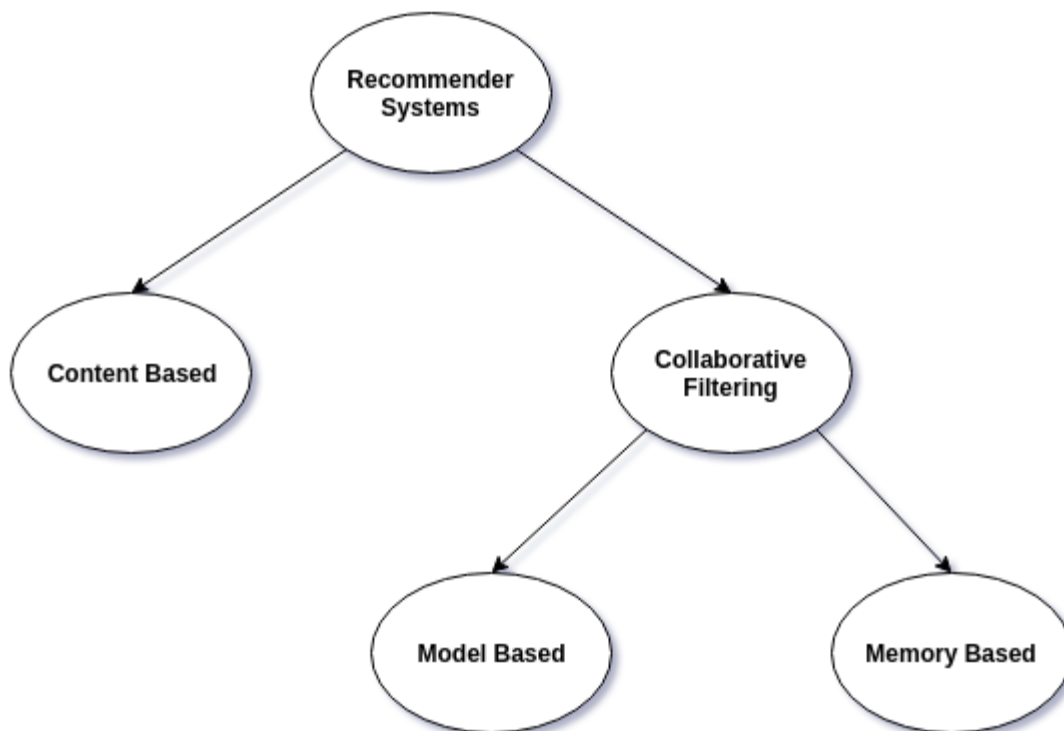
Recommended System

Introduction: recommender systems or recommendation system encompass a class of techniques and algorithms which are able to suggest “relevant” items to users. Ideally, the suggested items are as relevant to the user as possible, so that the user can engage with those items: YouTube videos, news articles, online products, and so on.

Items are ranked according to their relevancy, and the most relevant ones are shown to the user. The relevancy is something that the recommender system must determine and is mainly based on historical data. If you’ve recently watched YouTube videos about elephants, then YouTube is going to start showing you a lot of elephant videos with similar titles and themes.

Recommender systems are generally divided into two main categories:

1. Collaborative filtering
2. Content-based systems



1. **Collaborative filtering** methods for recommender systems are methods that are solely based on the past interactions between users and the target items. Thus, the input to a collaborative filtering system will be all historical data of user interactions with target items. This data is typically stored in a matrix where the rows are the users, and the columns are the items.

The core idea behind such systems is that the historical data of the users should be enough to make a prediction. I.e we don't need anything more than that historical data, no extra push from the user, no presently trending information.


























				
				
				
				
				
				

Figure Illustrates, how collaborative filtering works for predicting a user's rating of 4 things: an image, a book, a video, and a video game. Based on the users' historical data, the likes and dislikes of each item, the system tries to predict how the user would rate a new item which they haven't rated yet. The predictions themselves are based the past ratings of other users, whose ratings and therefore supposed preferences, are similar to the active user. In this case, the system made the prediction/recommendation that the active user won't like the video.

Beyond this, collaborative filtering methods are further divided into two sub-groups: **memory-based and model-based methods**.

Memory-based methods are the most simplistic as they use no model whatsoever. They assume that predictions can be made on pure "memory" of past data and usually just employ a simple distance-measurement approach, like nearest neighbour.

Model-based approaches, on the other hand, always assume some kind of underlying model and basically try to make sure that whatever predictions come out will fit the model well.

Example of Model-based collaborative filtering: Here is a step-by-step worked out example for four users and three items. We will consider the following sample data of preference of four users for three items:

ID	user	item	rating
241	u1	m1	2
222	u1	m3	3
276	u2	m1	5
273	u2	m2	2
200	u3	m1	3
229	u3	m2	3
231	u3	m3	1
239	u4	m2	2
286	u4	m3	2

Step 1: Write the user-item ratings data in a matrix form. The above table gets rewritten as follows:

	m1	m2	m3
u1	2	?	3
u2	5	2	?
u3	3	3	1
u4	?	2	2

Here rating of user u1 for item m3 is 3. There is no rating for item m2 by user u1. And no rating also for item m3 by user u2.

Step 2: We will now create an item-to-item similarity matrix. The idea is to calculate how similar an item is to another item. There are a number of ways of calculating this. We will use cosine similarity measure. To calculate similarity between items m1 and m2, for example, look at all those users who have rated both these items. In our case, both m1 and m2 have been rated by users u2 and u3. We create two item-vectors, v1 for item m1 and v2 for item m2, in the user-space of (u2,u3) and then find the cosine of angle between these vectors. A zero angle or overlapping vectors with cosine value of 1 means total similarity (or per user, across all items, there is same rating) and an angle of 90 degrees would mean cosine of 0 or no similarity. Thus, the two item-vectors would be,

$$v1 = 5 u2 + 3 u3$$

$$v2 = 3 u2 + 3 u3$$

The cosine similarity between the two vectors, v1 and v2, would then be:

$$\cos(v1,v2) = (5*3 + 3*3)/\sqrt{[(25 + 9)*(9+9)]} = 0.76$$

Similarly, to calculate similarity between m1 and m3, we consider only users u1 and u3 who have rated both these items. The two item vectors, v1 for item m1 and v3 for item m3, in the user-space would be as follows:

$$v1 = 2 u1 + 3 u3$$

$$v3 = 3 u1 + 1 u3$$

The cosine similarity measure between v1 and v3 is:

$$\cos(v1,v3) = (2*3 + 3*1)/\sqrt{[(4 + 9)*(9+1)]} = 0.78$$

We can similarly calculate similarity between items m2 and m3 using ratings given to both by users u3 and u4. The two item-vectors v3 and v4 would be:

$$v2 = 3 u3 + 2 u4$$

$$v3 = 1 u3 + 2 u4$$

And cosine similarity between them is:

$$\cos(v2,v3) = (3*1 + 2*2)/\sqrt{[(9 + 4)*(1 + 4)]} = 0.86$$

We now have the complete item-to-item similarity matrix as follows:

	m1	m2	m3
m1	1	0.76	0.78
m2	0.76	1	0.86
m3	0.78	0.86	1

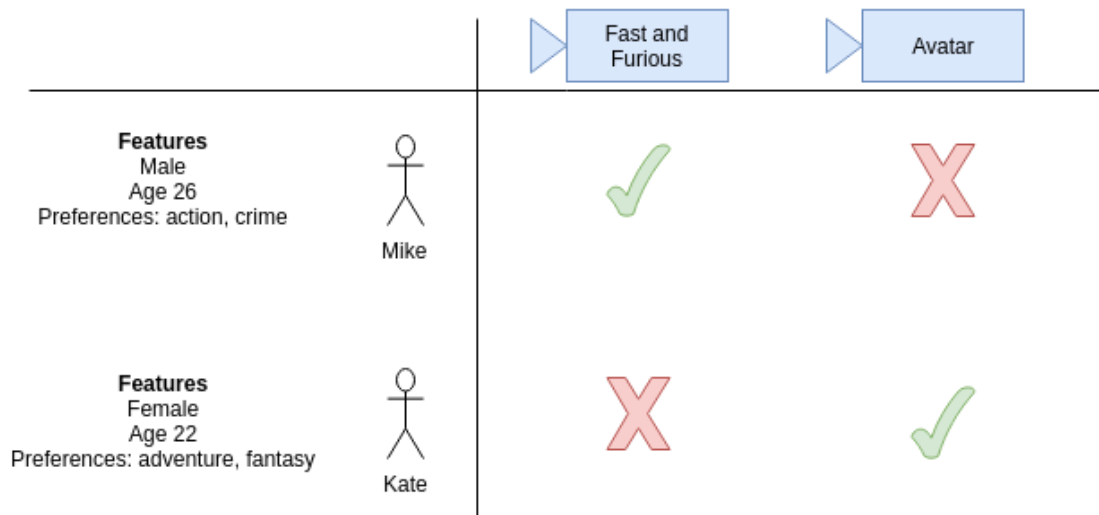
Step 3: For each user, we next predict his ratings for items that he had not rated. We will calculate rating for user u1 in the case of item m2 (target item). To calculate this, we weigh the just-calculated similarity-measure between the target item and other items that user has already rated. The weighing factor is the ratings given by the user to items already rated by him. We further scale this weighted sum with the sum of similarity-measures so that the calculated rating remains within a predefined limit. Thus, the predicted rating for item m2 for user u1 would be calculated using similarity measures between (m2, m1) and (m2, m3) weighted by the respective ratings for m1 and m3:

$$\text{rating} = (2 * 0.76 + 3 * 0.86)/(0.76+0.86) = 2.53$$

2. Content-based Systems

In contrast to collaborative filtering, content-based approaches will use additional information about the user and / or items to make predictions.

A content-based system might consider the age, sex, occupation, and other personal user factors when making the predictions. It's much easier to predict that the person wouldn't like the video if we knew it was about skateboarding, but the user's age is 87.



That's why when you sign up for many online websites and services, they ask you to (optionally) give your date of birth, gender, and ethnicity! It's just more data for their system to make better predictions.

Thus, content-based methods are more similar to classical machine learning, in the sense that we will build features based on user and item data and use that to help us make predictions. Our system input is then the features of the user and the features of the item. Our system output is the prediction of whether or not the user would like or dislike the item.

Example: Suppose, we search for "*IoT and analytics*" on Google and the top 5 links that appear have some frequency count of certain words as shown below:

Articles	Analytics	Data	Cloud	Smart	Insight
Article 1	21	24	0	2	2
Article 2	24	59	2	1	0
Article 3	40	115	8	10	19
Article 4	4	28	5	0	1
Article 5	8	48	4	3	4
Article 6	17	49	8	0	5
DF	5,000	50,000	10,000	5,00,000	7000

Among the corpus of documents / blogs which are used to search for the articles, 5000 contains the word **analytics**, 50,000 contain **data** and similar count goes for other words. Let us assume that the total corpus of docs is 1 million(10^6).

Term Frequency (TF)

Articles	Analytics	Data	Cloud	Smart	Insight	Length of Vector
Article 1	2.322219295	2.380211242	0	1.301029996	1.301029996	3.800456039
Article 2	2.380211242	2.770852012	1.301029996	1	0	4.004460697
Article 3	2.602059991	3.06069784	1.903089987	2	2.278753601	5.380804488
Article 4	1.602059991	2.447158031	1.698970004	0	1	3.527276247
Article 5	1.903089987	2.681241237	1.602059991	1.477121255	1.602059991	4.257450611
Article 6	2.230448921	2.69019608	1.903089987	0	1.698970004	4.326697114

As seen in the image above, for article 1, the term Analytics has a TF of $1 + \log_{10} 21 = 2.322$. In this way, TF is calculated for other attributes of each of the articles. These values make up the attribute vector for each of the articles.

Inverse Document Frequency (IDF)

IDF is calculated by taking the logarithmic inverse of the document frequency among the whole corpus of documents. So, if there are a total of 1 million documents returned by our search query and amongst those documents, 'smart' appears in 0.5 million documents. Thus, it's IDF score will be: $\log_{10} (10^6 / 500000) = 0.30$.

DF	5000	50000	10000	500000	7000
IDF	2.301029996	1.301029996	2	0.301029996	2.15490196
N =	10^6				

We can also see (image above), the most commonly occurring term 'smart' has been assigned the lowest weight by IDF. The length of these vectors are calculated as the **square root of sum of the squared values** of each attribute in the vector:

Articles	Analytics	Data	Cloud	Smart	Insight	Length of Vector
Article 1	2.322219295	2.380211242	0	1.301029996	1.3	$=\text{SQRT}(J2^2+K2^2+L2^2+M2^2+N2^2)$

After we have found out the TF-IDF weights and also vector lengths, let's normalize the vectors.

Articles	Analytics	Data	Cloud	Smart	Insight	Sum of Normalized Lengths
Article 1	0.61103701	0.626296217	0	0.342335231	0.342335231	1
Article 2	0.594389962	0.691941368	0.324895184	0.249721517	0	1
Article 3	0.483581962	0.568817887	0.353681311	0.371691632	0.423496822	1
Article 4	0.454191812	0.693781224	0.481666273	0	0.283504872	1
Article 5	0.447002246	0.62977624	0.376295614	0.34694971	0.376295614	1
Article 6	0.51550845	0.621766675	0.439848211	0	0.392671352	1

Each term vector is divided by the document vector length to get the normalized vector. So, for the term 'Analytics' in Article 1, the normalized vector is: $\frac{2.322}{3.800}$. Similarly all the terms in the document are normalized and as you can see (image above) each document vector now has a length of 1.

Now, that we have obtained the normalized vectors, let's calculate the cosine values to find out the similarity between articles. For this purpose, we will take only three articles and three attributes.

Articles	Analytics	Data	Cloud
Article 1	0.61103701	0.626296217	0
Article 2	0.594389962	0.691941368	0.324895184
Article 3	0.483581962	0.568817887	0.353681311
Article Vector	Cosine Values		
cos(A1,A2)	0.796554526		
cos(A2,A3)	0.795934246		
cos(A1,A3)	0.651734967		

Cos(A1,A2) is simply the sum of dot product of normalized term vectors from both the articles. The calculation is as follows:

$$\text{Cos}(A1,A2) = 0.611 \times 0.59 + 0.63 \times 0.69 + 0 \times 0.32 = .7965$$

As you can see the articles 1 and 2 are most similar and hence appear at the top two positions of search results. Also, if you remember we had said that Article 1 (M1) is more about analytics than cloud-Analytics has a weight of 0.611 whereas cloud has 0 after length normalization.

Content based recommenders have their own limitations. They are not good at capturing inter-dependencies or complex behaviours. For example: I might like articles on Machine Learning, only when they include practical application along with the theory, and not just theory. This type of information cannot be captured by these recommenders.

Artificial Neural Network:

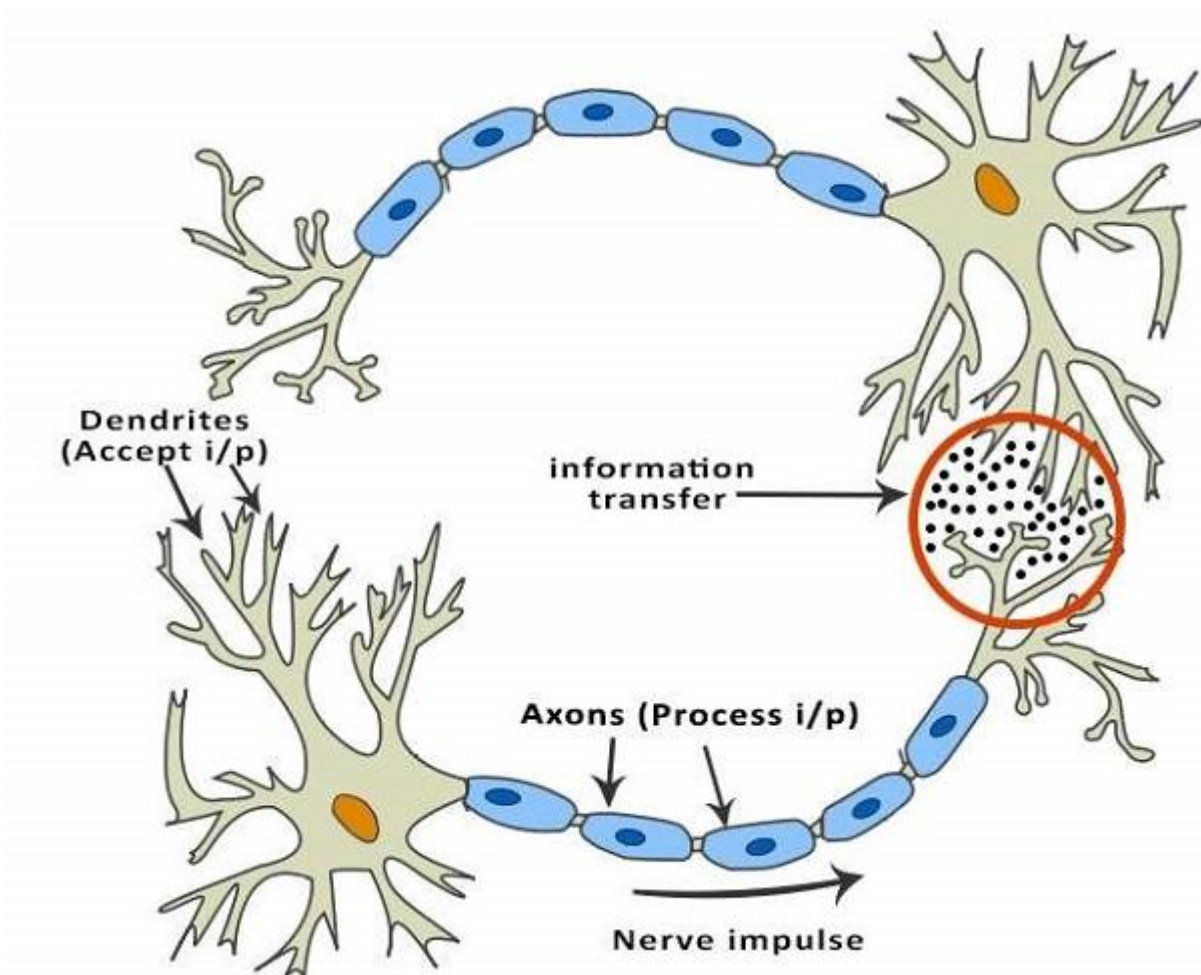
The inventor of the first neurocomputer, Dr. Robert Hecht-Nielsen, defines a neural network as –

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

Basic Structure of ANNs

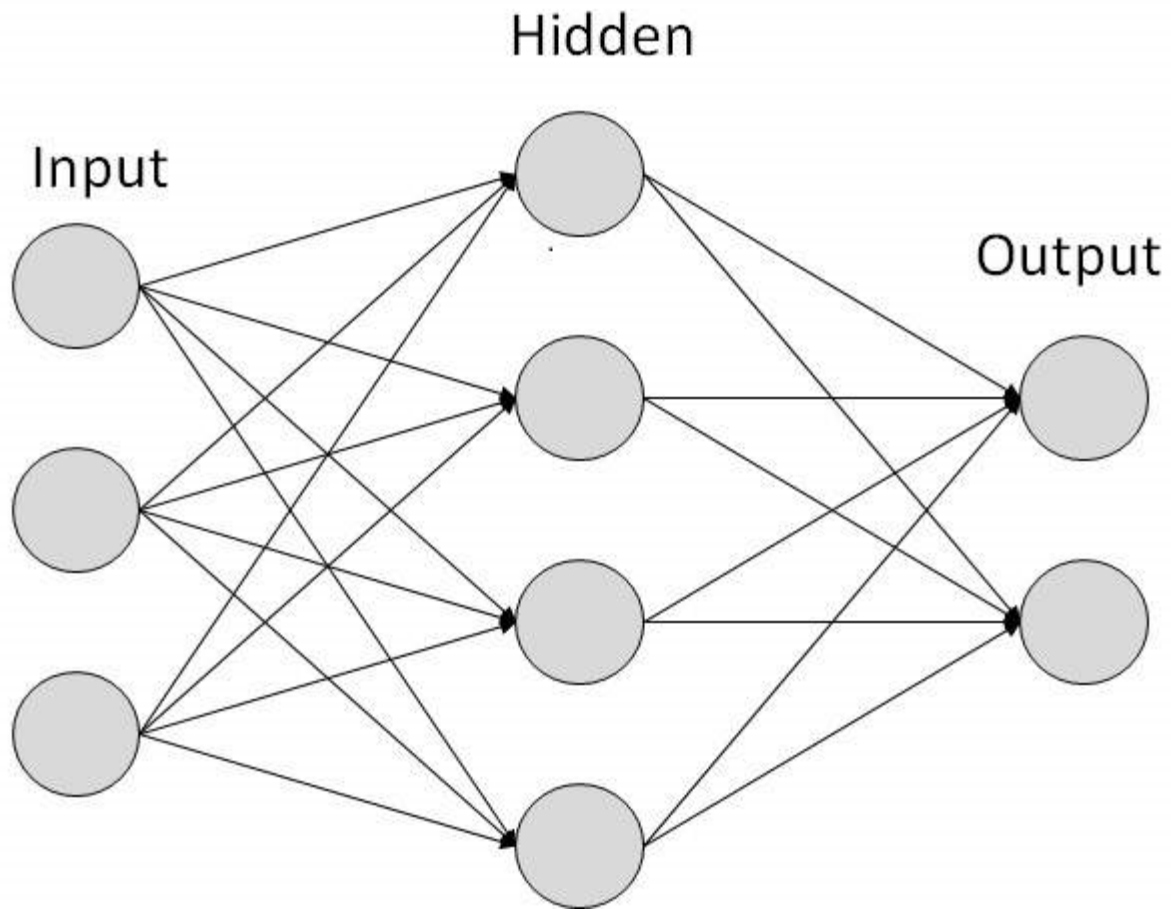
The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living **neurons** and **dendrites**.

The human brain is composed of 86 billion nerve cells called **neurons**. They are connected to other thousand cells by **Axons**. Stimuli from external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to other neuron to handle the issue or does not send it forward.



ANNs are composed of multiple **nodes**, which imitate biological **neurons** of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its **activation** or **node value**.

Each link is associated with **weight**. ANNs are capable of learning, which takes place by altering weight values. The following illustration shows a simple ANN –

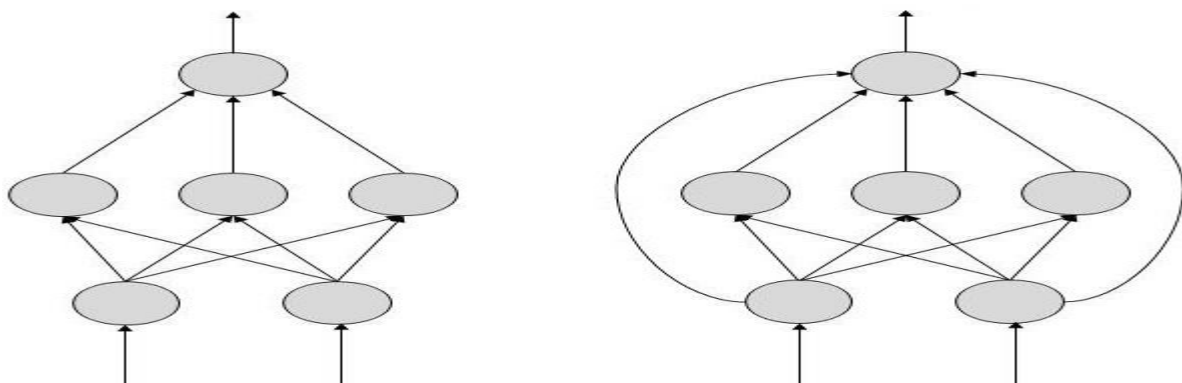


Types of Artificial Neural Networks

There are two Artificial Neural Network topologies – **FeedForward** and **Feedback**.

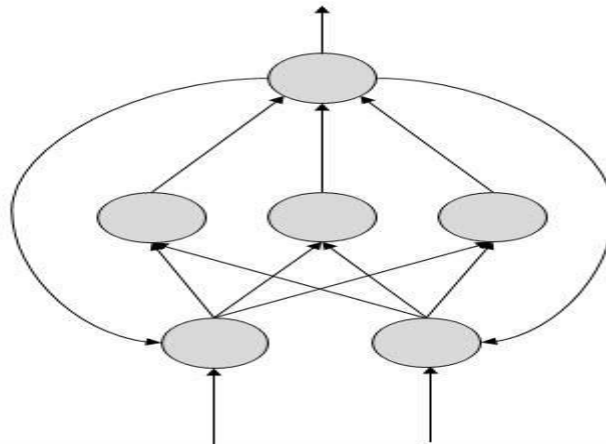
Feedforward ANN

In this ANN, the information flow is unidirectional. A unit sends information to other unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation/recognition/classification. They have fixed inputs and outputs.



FeedBack ANN

Here, feedback loops are allowed. They are used in content addressable memories.



Working of ANNs

In the topology diagrams shown, each arrow represents a connection between two neurons and indicates the pathway for the flow of information. Each connection has a weight, an integer number that controls the signal between the two neurons.

If the network generates a “good or desired” output, there is no need to adjust the weights. However, if the network generates a “poor or undesired” output or an error, then the system alters the weights in order to improve subsequent results.

Machine Learning in ANNs

ANNs are capable of learning and they need to be trained. There are several learning strategies –

- **Supervised Learning** – It involves a teacher that is scholar than the ANN itself. For example, the teacher feeds some example data about which the teacher already knows the answers.
For example, pattern recognizing. The ANN comes up with guesses while recognizing. Then the teacher provides the ANN with the answers. The network then compares it guesses with the teacher’s “correct” answers and makes adjustments according to errors.
- **Unsupervised Learning** – It is required when there is no example data set with known answers. For example, searching for a hidden pattern. In this case, clustering i.e. dividing a set of elements into groups according to some unknown pattern is carried out based on the existing data sets present.
- **Reinforcement Learning** – This strategy built on observation. The ANN makes a decision by observing its environment. If the observation is negative, the network adjusts its weights to be able to make a different required decision the next time.

Back Propagation Algorithm

It is the training or learning algorithm. It learns by example. If you submit to the algorithm the example of what you want the network to do, it changes the network’s

weights so that it can produce desired output for a particular input on finishing the training.

Back Propagation networks are ideal for simple Pattern Recognition and Mapping Tasks.

Bayesian Networks (BN)

These are the graphical structures used to represent the probabilistic relationship among a set of random variables. Bayesian networks are also called **Belief Networks** or **Bayes Nets**. BNs reason about uncertain domain.

In these networks, each node represents a random variable with specific propositions. For example, in a medical diagnosis domain, the node Cancer represents the proposition that a patient has cancer.

The edges connecting the nodes represent probabilistic dependencies among those random variables. If out of two nodes, one is affecting the other then they must be directly connected in the directions of the effect. The strength of the relationship between variables is quantified by the probability associated with each node.

There is an only constraint on the arcs in a BN that you cannot return to a node simply by following directed arcs. Hence the BNs are called Directed Acyclic Graphs (DAGs).

BNs are capable of handling multivalued variables simultaneously. The BN variables are composed of two dimensions –

- Range of prepositions
- Probability assigned to each of the prepositions.

Consider a finite set $X = \{X_1, X_2, \dots, X_n\}$ of discrete random variables, where each variable X_i may take values from a finite set, denoted by $Val(X_i)$. If there is a directed link from variable X_i to variable X_j , then variable X_i will be a parent of variable X_j showing direct dependencies between the variables.

The structure of BN is ideal for combining prior knowledge and observed data. BN can be used to learn the causal relationships and understand various problem domains and to predict future events, even in case of missing data.

Building a Bayesian Network

A knowledge engineer can build a Bayesian network. There are a number of steps the knowledge engineer needs to take while building it.

Example problem – Lung cancer. A patient has been suffering from breathlessness. He visits the doctor, suspecting he has lung cancer. The doctor knows that barring lung cancer, there are various other possible diseases the patient might have such as tuberculosis and bronchitis.

Gather Relevant Information of Problem

- Is the patient a smoker? If yes, then high chances of cancer and bronchitis.
- Is the patient exposed to air pollution? If yes, what sort of air pollution?
- Take an X-Ray positive X-ray would indicate either TB or lung cancer.

Identify Interesting Variables

The knowledge engineer tries to answer the questions –

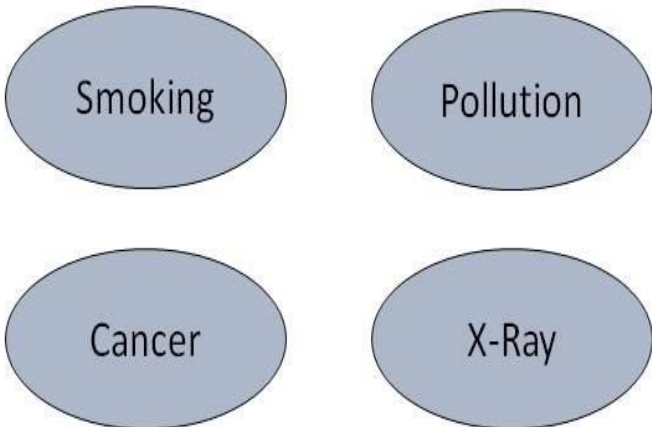
- Which nodes to represent?
- What values can they take? In which state can they be?

For now let us consider nodes, with only discrete values. The variable must take on exactly one of these values at a time.

Common types of discrete nodes are –

- **Boolean nodes** – They represent propositions, taking binary values TRUE (T) and FALSE (F).
- **Ordered values** – A node *Pollution* might represent and take values from {low, medium, high} describing degree of a patient's exposure to pollution.
- **Integral values** – A node called *Age* might represent patient's age with possible values from 1 to 120. Even at this early stage, modeling choices are being made.

Possible nodes and values for the lung cancer example –

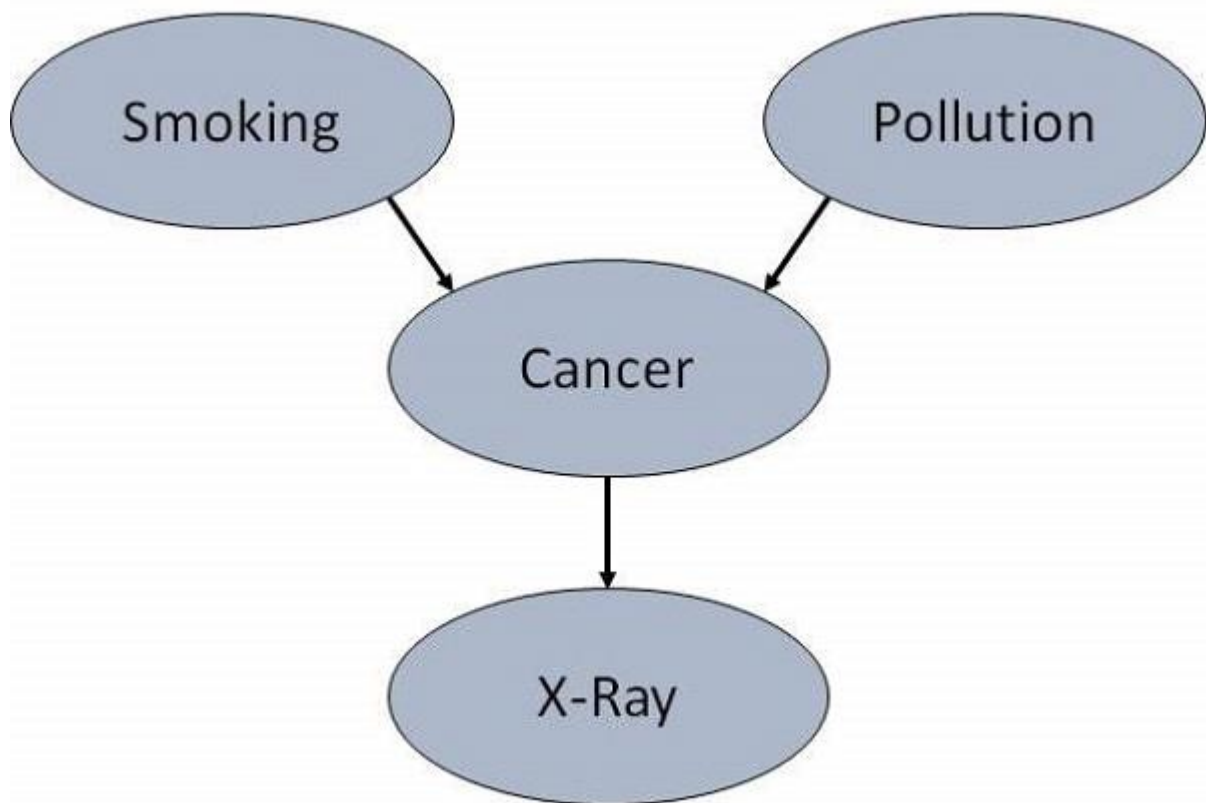
Node Name	Type	Value	Nodes Creation	
Polution	Binary	{LOW, HIGH, MEDIUM}		
Smoker	Boolean	{TRUE, FASLE}		
Lung-Cancer	Boolean	{TRUE, FASLE}		
X-Ray	Binary	{Positive, Negative}		

Create Arcs between Nodes

Topology of the network should capture qualitative relationships between variables.

For example, what causes a patient to have lung cancer? - Pollution and smoking. Then add arcs from node *Pollution* and node *Smoker* to node *Lung-Cancer*.

Similarly if patient has lung cancer, then X-ray result will be positive. Then add arcs from node *Lung-Cancer* to node *X-Ray*.



Specify Topology

Conventionally, BNs are laid out so that the arcs point from top to bottom. The set of parent nodes of a node X is given by $\text{Parents}(X)$.

The *Lung-Cancer* node has two parents (reasons or causes): *Pollution* and *Smoker*, while node *Smoker* is an **ancestor** of node *X-Ray*. Similarly, *X-Ray* is a child (consequence or effects) of node *Lung-Cancer* and **successor** of nodes *Smoker* and *Pollution*.

Conditional Probabilities

Now quantify the relationships between connected nodes: this is done by specifying a conditional probability distribution for each node. As only discrete variables are considered here, this takes the form of a **Conditional Probability Table (CPT)**.

First, for each node we need to look at all the possible combinations of values of those parent nodes. Each such combination is called an **instantiation** of the parent set. For each distinct instantiation of parent node values, we need to specify the probability that the child will take.

For example, the *Lung-Cancer* node's parents are *Pollution* and *Smoking*. They take the possible values = { (H,T), (H,F), (L,T), (L,F)}. The CPT specifies the probability of cancer for each of these cases as <0.05, 0.02, 0.03, 0.001> respectively.

Each node will have conditional probability associated as follows –

Smoking	
$P(S = T)$	
0.30	

Pollution	
$P(P = L)$	
0.90	

Lung-Cancer		
P	S	$P(C = T P, S)$
H	T	0.05
H	F	0.02
L	T	0.03
L	F	0.001

X-Ray	
C	$X = (Pos C)$
T	0.90
F	0.20

Applications of Neural Networks

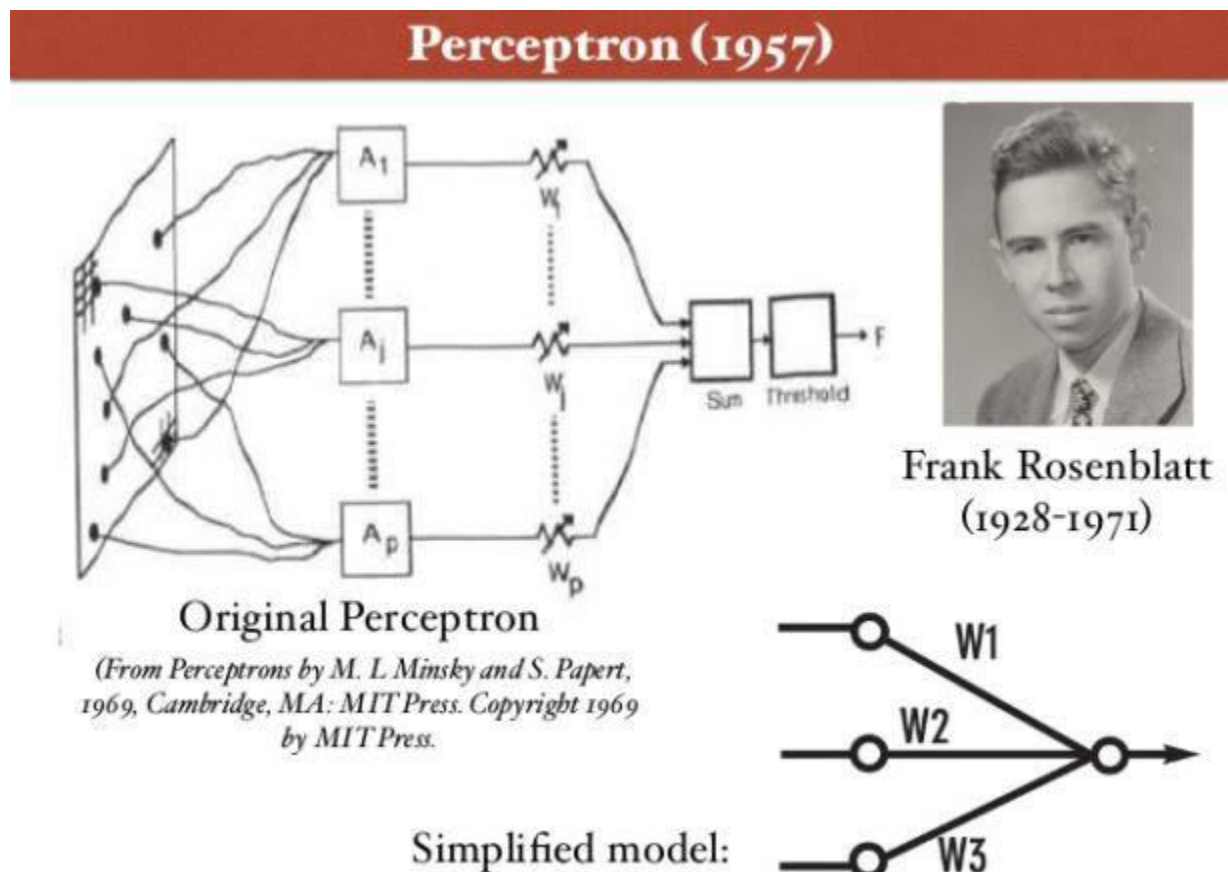
They can perform tasks that are easy for a human but difficult for a machine –

- **Aerospace** – Autopilot aircrafts, aircraft fault detection.
- **Automotive** – Automobile guidance systems.
- **Military** – Weapon orientation and steering, target tracking, object discrimination, facial recognition, signal/image identification.
- **Electronics** – Code sequence prediction, IC chip layout, chip failure analysis, machine vision, voice synthesis.
- **Financial** – Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, portfolio trading program, corporate financial analysis, currency value prediction, document readers, credit application evaluators.
- **Industrial** – Manufacturing process control, product design and analysis, quality inspection systems, welding quality analysis, paper quality prediction, chemical product design analysis, dynamic modelling of chemical process systems, machine maintenance analysis, project bidding, planning, and management.
- **Medical** – Cancer cell analysis, EEG and ECG analysis, prosthetic design, transplant time optimizer.
- **Speech** – Speech recognition, speech classification, text to speech conversion.

- **Telecommunications** – Image and data compression, automated information services, real-time spoken language translation.
- **Transportation** – Truck Brake system diagnosis, vehicle scheduling, routing systems.
- **Software** – Pattern Recognition in facial recognition, optical character recognition, etc.
- **Time Series Prediction** – ANNs are used to make predictions on stocks and natural calamities.
- **Signal Processing** – Neural networks can be trained to process an audio signal and filter it appropriately in the hearing aids.
- **Control** – ANNs are often used to make steering decisions of physical vehicles.
- **Anomaly Detection** – As ANNs are expert at recognizing patterns, they can also be trained to generate an output when something unusual occurs that misfits the pattern.

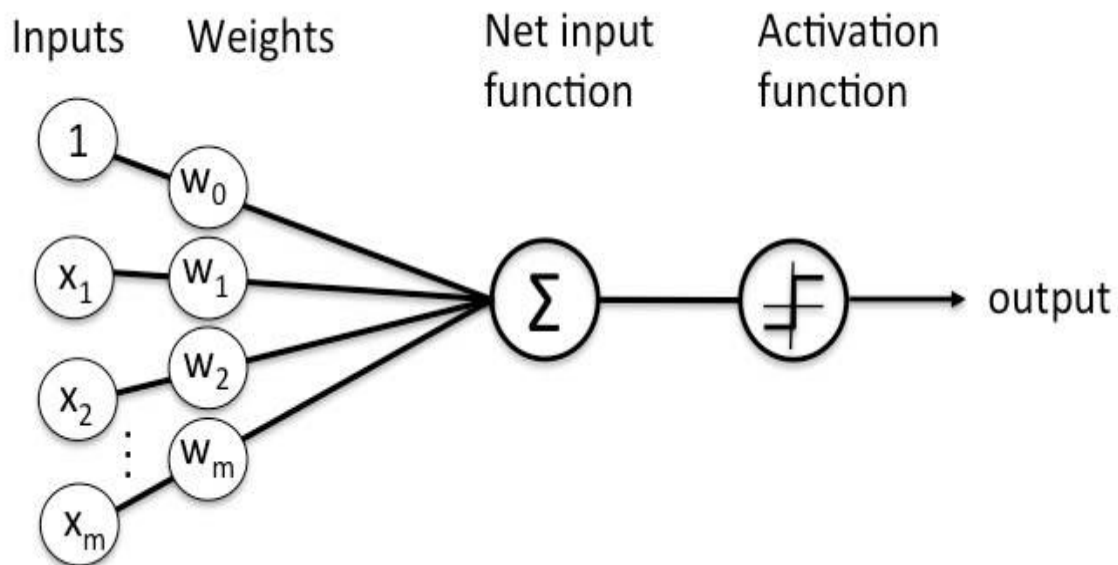
Perceptron

A perceptron is a neural network unit (an artificial neuron) that does certain computations to detect features or business intelligence in the input data.



Perceptron was introduced by Frank Rosenblatt in 1957. He proposed a Perceptron learning rule based on the original MCP neuron.

A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.



There are two types of Perceptrons: Single layer and Multilayer.

Single layer Perceptrons can learn only linearly separable patterns.

Multilayer Perceptrons or feedforward neural networks with two or more layers have the greater processing power.

The Perceptron algorithm learns the weights for the input signals in order to draw a linear decision boundary.

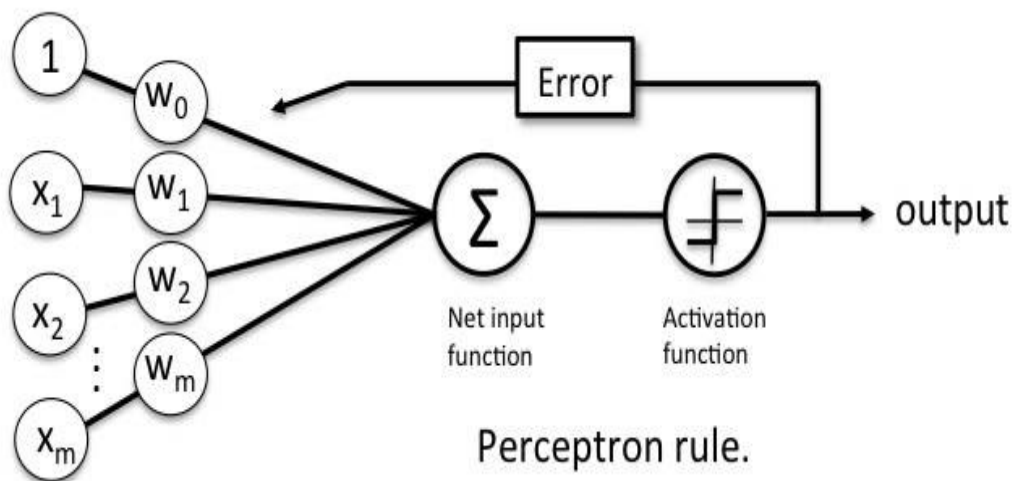
This enables you to distinguish between the two linearly separable classes +1 and -1.

Note: Supervised Learning is a type of Machine Learning used to learn models from labeled training data. It enables output prediction for future or unseen data.

Let us focus on the Perceptron Learning Rule in the next section.

Perceptron Learning Rule

Perceptron Learning Rule states that the algorithm would automatically learn the optimal weight coefficients. The input features are then multiplied with these weights to determine if a neuron fires or not.



The Perceptron receives multiple input signals, and if the sum of the input signals exceeds a certain threshold, it either outputs a signal or does not return an output. In the context

of supervised learning and classification, this can then be used to predict the class of a sample.

In the next section, let us focus on the perceptron function.

Perceptron Function

Perceptron is a function that maps its input “x,” which is multiplied with the learned weight coefficient; an output value “f(x)” is generated.

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

In the equation given above:

“w” = vector of real-valued weights

“b” = bias (an element that adjusts the boundary away from origin without any dependence on the input value)

“x” = vector of input x values

$$\sum_{i=1}^m w_i x_i$$

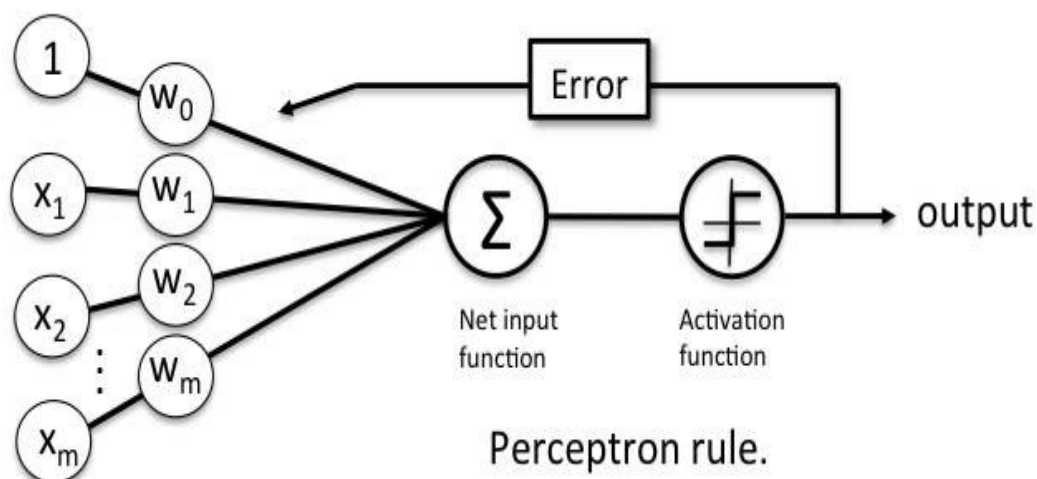
“m” = number of inputs to the Perceptron

The output can be represented as “1” or “0.” It can also be represented as “1” or “-1” depending on which activation function is used.

Let us learn the inputs of a perceptron in the next section.

Inputs of a Perceptron

A Perceptron accepts inputs, moderates them with certain weight values, then applies the transformation function to output the final result. The above below shows a Perceptron with a Boolean output.



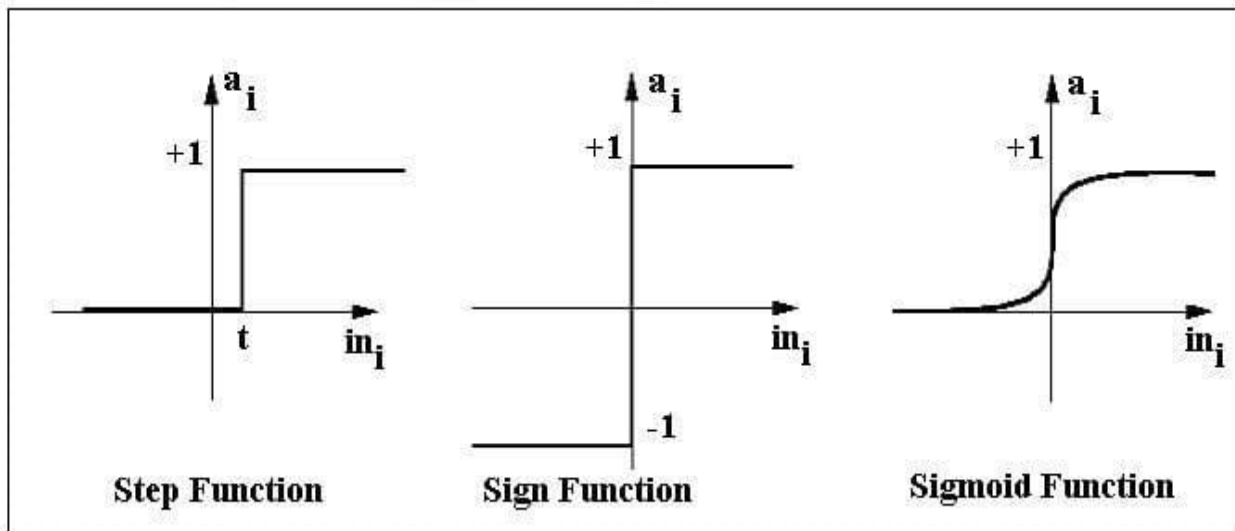
A Boolean output is based on inputs such as salaried, married, age, past credit profile, etc. It has only two values: Yes and No or True and False. The summation function “ Σ ” multiplies all inputs of “x” by weights “w” and then adds them up as follows:

$$w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$

In the next section, let us discuss the activation functions of perceptron.

Activation Functions of Perceptron

The activation function applies a step rule (convert the numerical output into +1 or -1) to check if the output of the weighting function is greater than zero or not.



For example:

If $\sum w_i x_i > 0 \Rightarrow$ then final output "o" = 1 (issue bank loan)

Else, final output "o" = -1 (deny bank loan)

Step function gets triggered above a certain value of the neuron output; else it outputs zero. Sign Function outputs +1 or -1 depending on whether neuron output is greater than zero or not. Sigmoid is the S-curve and outputs a value between 0 and 1.

Output of Perceptron

Perceptron with a Boolean output:

Inputs: $x_1 \dots x_n$

Output: $o(x_1 \dots x_n)$

$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

Weights: $w_i \Rightarrow$ contribution of input x_i to the Perceptron output;

$w_0 \Rightarrow$ bias or threshold

If $\sum w_i x_i > 0$, output is +1, else -1. The neuron gets triggered only when weighted input reaches a certain threshold value.

$$o(\vec{x}) = \text{sgn}(\vec{w} \cdot \vec{x})$$

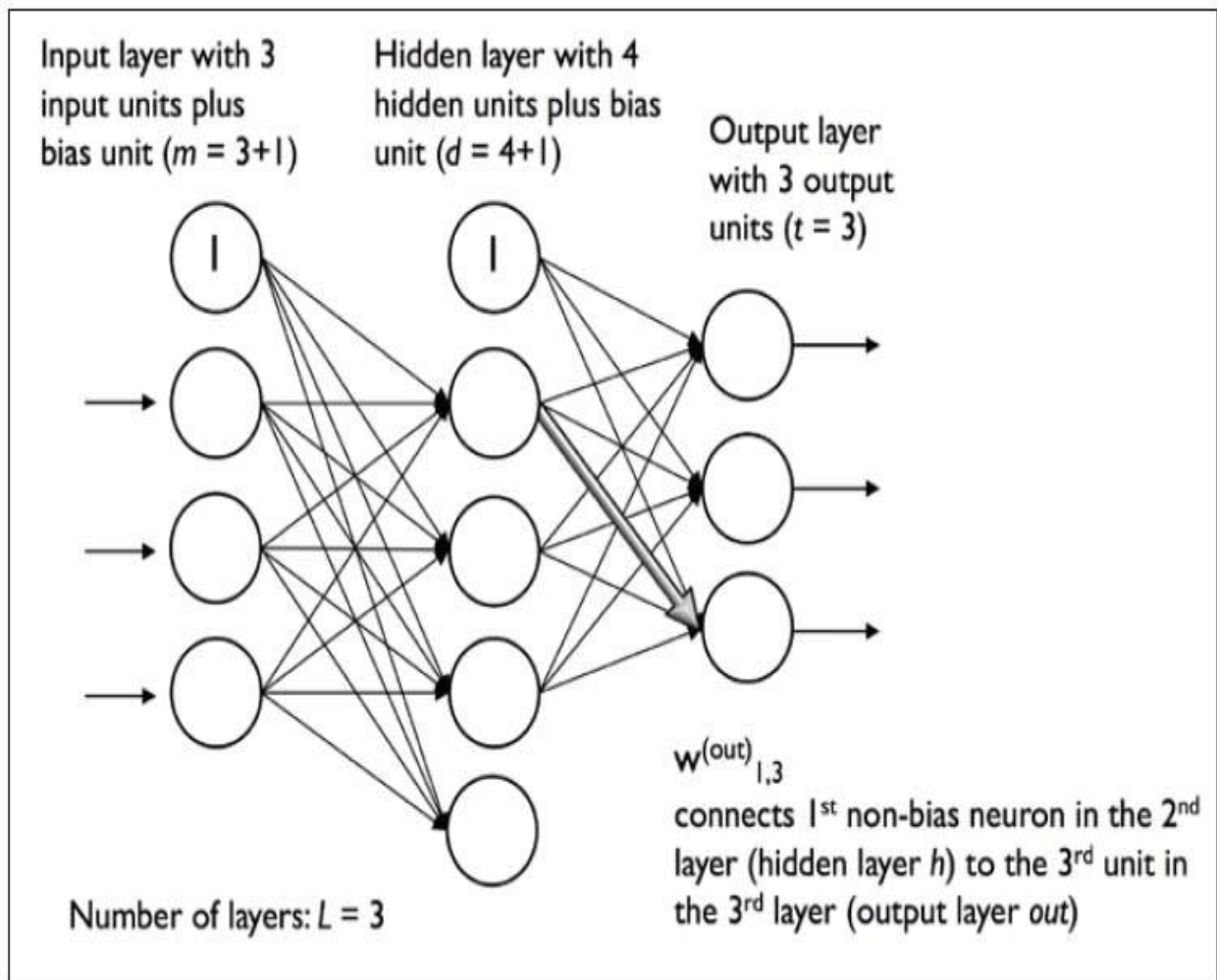
$$\text{sgn}(y) = \begin{cases} 1 & \text{if } y > 0 \\ -1 & \text{otherwise} \end{cases}$$

An output of +1 specifies that the neuron is triggered. An output of -1 specifies that the neuron did not get triggered.

"sgn" stands for sign function with output +1 or -1.

Multi-layer ANN

A fully connected multi-layer neural network is called a Multilayer Perceptron (MLP).



It has 3 layers including one hidden layer. If it has more than 1 hidden layer, it is called a deep ANN.

An MLP is a typical example of a feedforward artificial neural network.

In this figure, the i^{th} activation unit in the l^{th} layer is denoted as $a_l^{(i)}$.

The number of layers and the number of neurons are referred to as hyperparameters of a neural network, and these need tuning. Cross-validation techniques must be used to find ideal values for these.

The weight adjustment training is done via backpropagation. Deeper neural networks are better at processing data. However, deeper layers can lead to vanishing gradient problem. Special algorithms are required to solve this issue.

Notations

In the representation below:

$$a^{(in)} = \begin{bmatrix} a_0^{(in)} \\ a_1^{(in)} \\ \vdots \\ a_m^{(in)} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1^{(in)} \\ \vdots \\ x_m^{(in)} \end{bmatrix}$$

- $a_i^{(in)}$ refers to the i^{th} value in the input layer
- $a_i^{(h)}$ refers to the i^{th} unit in the hidden layer
- $a_i^{(out)}$ refers to the i^{th} unit in the output layer
- $a_0^{(in)}$ is simply the bias unit and is equal to 1; it will have the corresponding weight w_0
- The weight coefficient from layer l to layer $l+1$ is represented by $w_{k,l}^{(l)}$

A simplified view of the multilayer is presented here.

This image shows a fully connected three-layer neural network with 3 input neurons and 3 output neurons. A bias term is added to the input vector.

Forward Propagation

In the following topics, we discuss the forward propagation in detail.

MLP Learning Procedure

The MLP learning procedure is as follows :

- Starting with the input layer, propagate data forward to the output layer. This step is the forward propagation.
- Based on the output, calculate the error (the difference between the predicted and known outcome). The error needs to be minimized.
- Backpropagate the error. Find its derivative with respect to each weight in the network, and update the model.

Repeat the three steps given above over multiple epochs to learn ideal weights. Finally, the output is taken via a threshold function to obtain the predicted class labels.

Forward Propagation in MLP

In the first step, calculate the activation unit $a_l(h)$ of the hidden layer.

$$z_1^{(h)} = a_0^{(in)} w_{0,1}^{(h)} + a_1^{(in)} w_{1,1}^{(h)} + \dots + a_m^{(in)} w_{m,1}^{(h)}$$

$$a_1^{(h)} = \phi\left(z_1^{(h)}\right)$$

Activation unit is the result of applying an activation function ϕ to the z value. It must be differentiable to be able to learn weights using gradient descent.

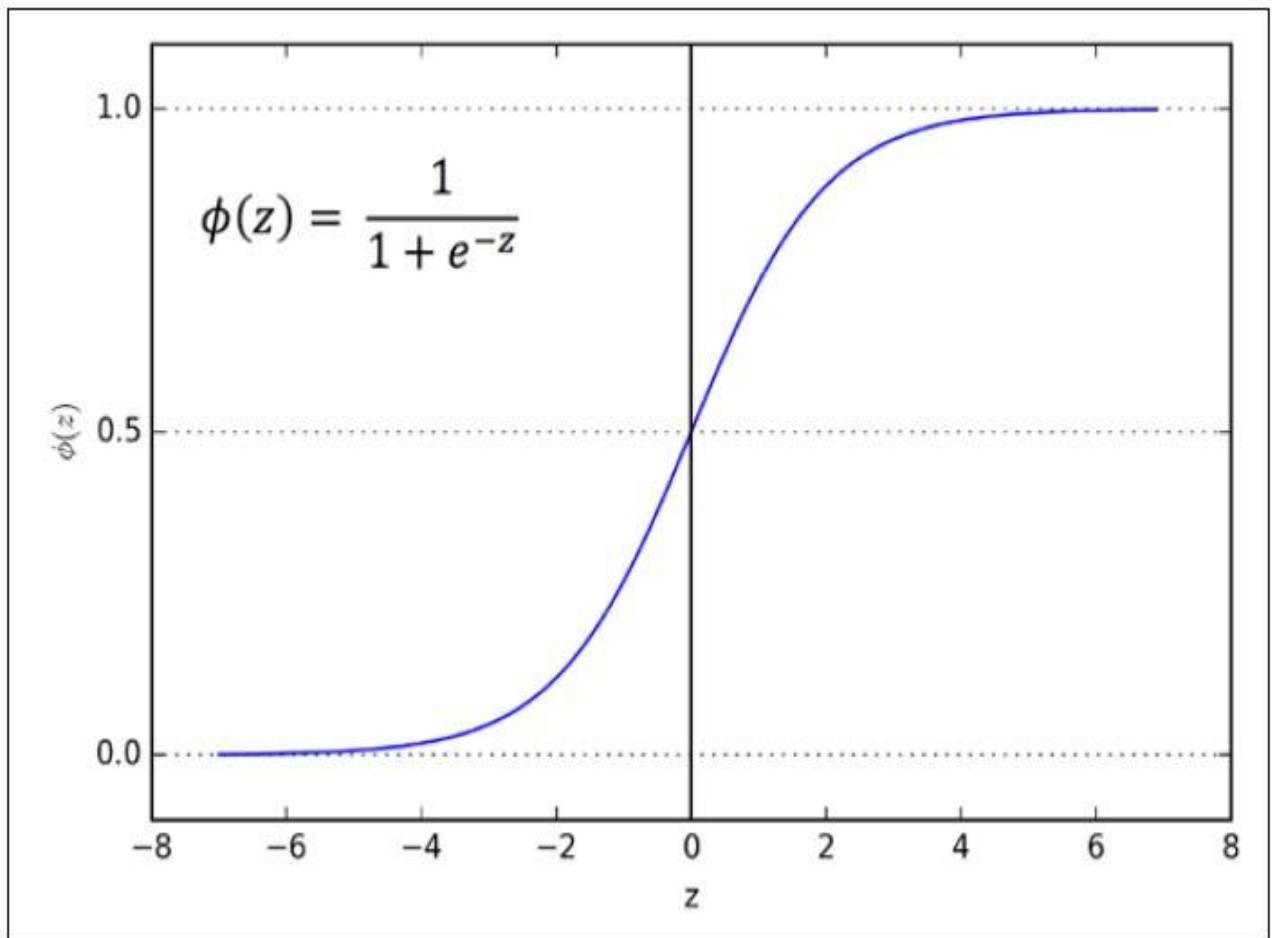
The activation function ϕ is often the sigmoid (logistic) function.

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

It allows nonlinearity needed to solve complex problems like image processing.

Sigmoid Curve

The sigmoid curve is an S-shaped curve.



Backpropagation?

The Backpropagation algorithm looks for the minimum value of the error function in weight space using a technique called the delta rule or gradient descent. The weights that minimize the error function is then considered to be a solution to the learning problem.

Let's understand how it works with an example:

You have a dataset, which has labels.

Consider the below table:



Input	Desired Output
0	0
1	2
2	4

Now the output of your model when 'W' value is 3:

Input	Desired Output	Model output (W=3)
0	0	0
1	2	3
2	4	6

Notice the difference between the actual output and the desired output:

Input	Desired Output	Model output (W=3)	Absolute Error	Square Error
0	0	0	0	0
1	2	3	1	1
2	4	6	2	4

Let's change the value of 'W'. Notice the error when 'W' = '4'

Input	Desired Output	Model output (W=3)	Absolute Error	Square Error	Model output (W=4)	Sq
0	0	0	0	0	0	

1	2	3	1	1	4	
2	4	6	2	4	8	

Now if you notice, when we increase the value of 'W' the error has increased. So, obviously there is no point in increasing the value of 'W' further. But, what happens if I decrease the value of 'W'? Consider the table below:

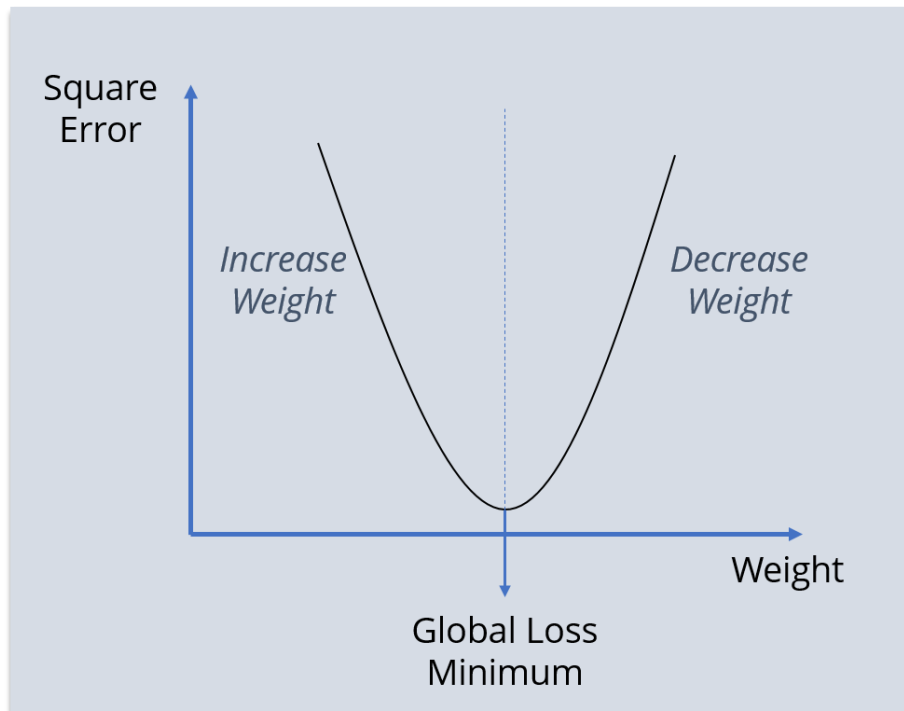
Input	Desired Output	Model output (W=3)	Absolute Error	Square Error	Model output (W=2)	Sq
0	0	0	0	0	0	
1	2	3	2	4	3	
2	4	6	2	4	4	

Now, what we did here:

- We first initialized some random value to 'W' and propagated forward.
- Then, we noticed that there is some error. To reduce that error, we propagated backwards and increased the value of 'W'.
- After that, also we noticed that the error has increased. We came to know that, we can't increase the 'W' value.
- So, we again propagated backwards and we decreased 'W' value.
- Now, we noticed that the error has reduced.

So, we are trying to get the value of weight such that the error becomes minimum. Basically, we need to figure out whether we need to increase or decrease the weight value. Once we know that, we keep on updating the weight value in that direction until error becomes minimum. You might reach a point, where if you further update the weight, the error will increase. At that time you need to stop, and that is your final weight value.

Consider the graph below:



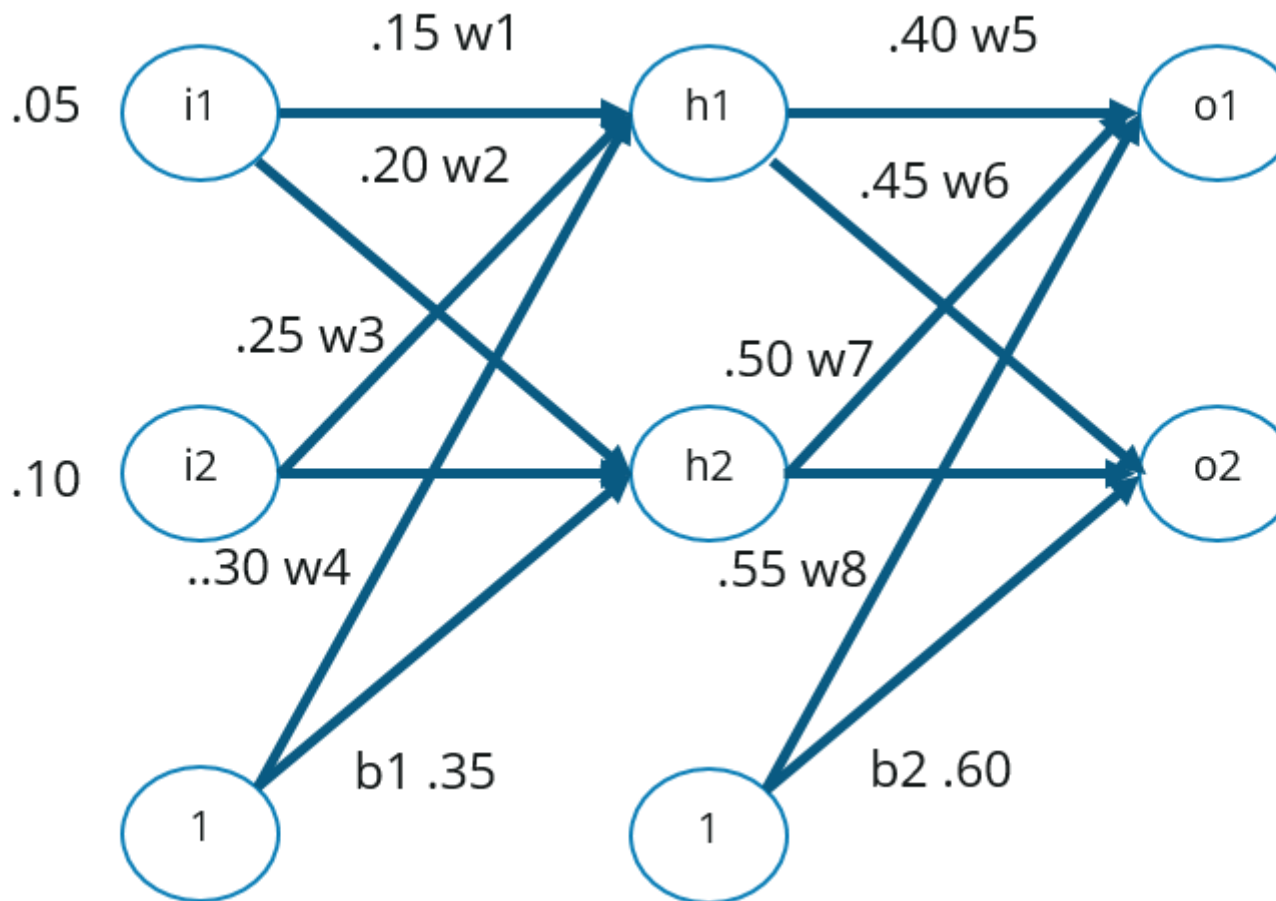
We need to reach the 'Global Loss Minimum'.

This is nothing but Backpropagation.

Let's now understand the math behind Backpropagation.

How Backpropagation Works?

Consider the below Neural Network:



The above network contains the following:

- two inputs
- two hidden neurons
- two output neurons
- two biases

Below are the steps involved in Backpropagation:

- Step – 1: Forward Propagation
- Step – 2: Backward Propagation
- Step – 3: Putting all the values together and calculating the updated weight value

Step – 1: Forward Propagation

We will start by propagating forward.

Net Input For h1:

$$\text{net } h1 = w1*i1 + w2*i2 + b1*1$$

$$\text{net } h1 = 0.15*0.05 + 0.2*0.1 + 0.35*1 = 0.3775$$

Output Of h1:

$$\text{out } h1 = 1/1+e^{-\text{net } h1}$$

$$1/1+e^{.3775} = 0.593269992$$

Output Of h2:

$$\text{out } h2 = 0.596884378$$

We will repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

Output For o1:

$$\text{net } o1 = w5*\text{out } h1 + w6*\text{out } h2 + b2*1$$

$$0.4*0.593269992 + 0.45*0.596884378 + 0.6*1 = 1.105905967$$

$$\text{Out } o1 = 1/1+e^{-\text{net } o1}$$

$$1/1+e^{-1.105905967} = 0.75136507$$

Output For o2:

$$\text{Out } o2 = 0.772928465$$

Now, let's see what is the value of the error:

Error For o1:

$$E_{o1} = \frac{1}{2}(\text{target} - \text{output})^2$$

$$\frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

Error For o2:

$$E_{o2} = 0.023560026$$

Total Error:

$$E_{\text{total}} = E_{o1} + E_{o2}$$

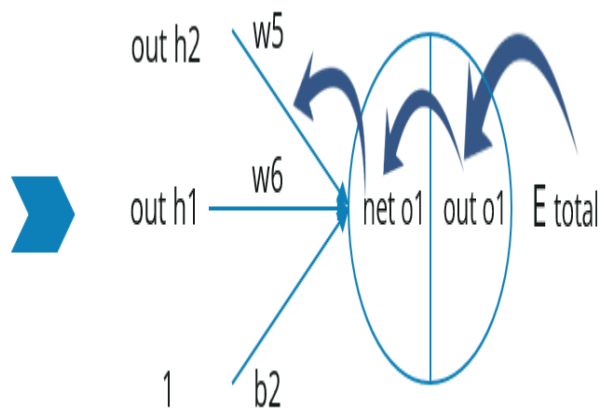
$$0.274811083 + 0.023560026 = 0.298371109$$

Step – 2: Backward Propagation

Now, we will propagate backwards. This way we will try to reduce the error by changing the values of weights and biases.

Consider W5, we will calculate the rate of change of error w.r.t change in weight W5.

$$\frac{\delta E_{\text{total}}}{\delta w_5} = \frac{\delta E_{\text{total}}}{\delta \text{out } o1} * \frac{\delta \text{out } o1}{\delta \text{net } o1} * \frac{\delta \text{net } o1}{\delta w_5}$$



Since we are propagating backwards, first thing we need to do is, calculate the change in total errors w.r.t the output O1 and O2.

$$E_{total} = 1/2(\text{target } o1 - \text{out } o1)^2 + 1/2(\text{target } o2 - \text{out } o2)^2$$

$$\frac{\delta E_{total}}{\delta \text{out } o1} = -(\text{target } o1 - \text{out } o1) = -(0.01 - 0.75136507) = 0.74136507$$

Now, we will propagate further backwards and calculate the change in output O1 w.r.t to its total net input.

$$\text{out } o1 = 1/1 + e^{-\text{net } o1}$$

$$\frac{\delta \text{out } o1}{\delta \text{net } o1} = \text{out } o1 (1 - \text{out } o1) = 0.75136507 (1 - 0.75136507) = 0.186815602$$

Let's see now how much does the total net input of O1 changes w.r.t W5?

$$\text{net } o1 = w5 * \text{out } h1 + w6 * \text{out } h2 + b2 * 1$$

$$\frac{\delta \text{net } o1}{\delta w5} = 1 * \text{out } h1 * w5^{(1-1)} + 0 + 0 = 0.593269992$$

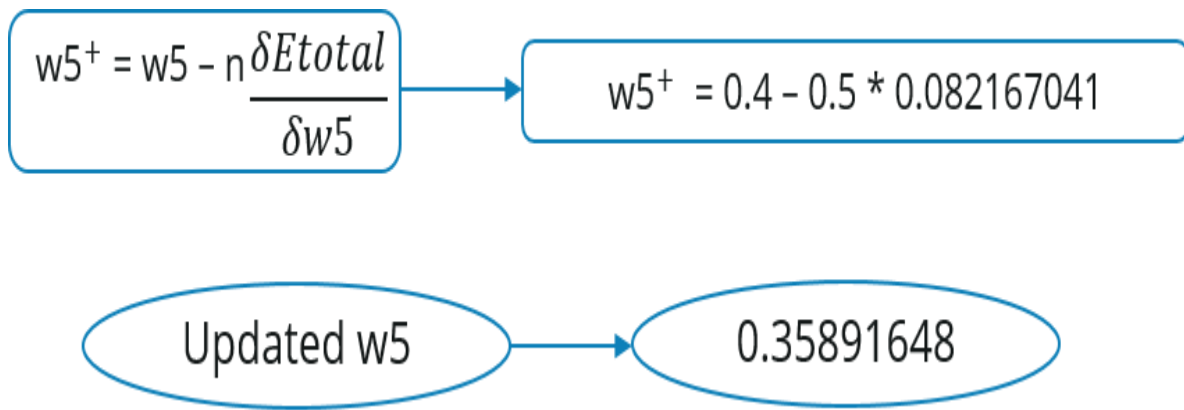
Step – 3: Putting all the values together and calculating the updated weight value

Now, let's put all the values together:

$$\frac{\delta E_{total}}{\delta w5} = \frac{\delta E_{total}}{\delta \text{out } o1} * \frac{\delta \text{out } o1}{\delta \text{net } o1} * \frac{\delta \text{net } o1}{\delta w5}$$

0.082167041

Let's calculate the updated value of W5:



- Similarly, we can calculate the other weight values as well.
- After that we will again propagate forward and calculate the output. Again, we will calculate the error.
- If the error is minimum we will stop right there, else we will again propagate backwards and update the weight values.
- This process will keep on repeating until error becomes minimum.

Conclusion:

Well, if I have to conclude Backpropagation, the best option is to write pseudo code for the same.

Backpropagation Algorithm:

```
initialize network weights (often small random values)

do
    forEach training example named ex
        prediction = neural-net-output(network, ex) // forward pass
        actual = teacher-output(ex)
        compute error (prediction - actual) at the output units
        compute  $\Delta w_{\{h\}}$  for all weights from
        hidden layer to output layer // backward pass
        compute  $\Delta w_{\{i\}}$  for all weights from
        input layer to hidden layer // backward pass continued
```

```
        update network weights // input layer not modified by error estimate
```

```
    until all examples classified correctly or another stopping criterion satisfied
```

```
    return the network
```

Deep Learning

What is Deep Learning?

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. In deep learning, we don't need to explicitly program everything. The concept of deep learning is not new. It has been around for a couple of years now. It's on hype nowadays because earlier we did not have that much processing power and a lot of data. As in the last 20 years, the processing power increases exponentially, deep learning and machine learning came in the picture. A formal definition of deep learning is- neurons

Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

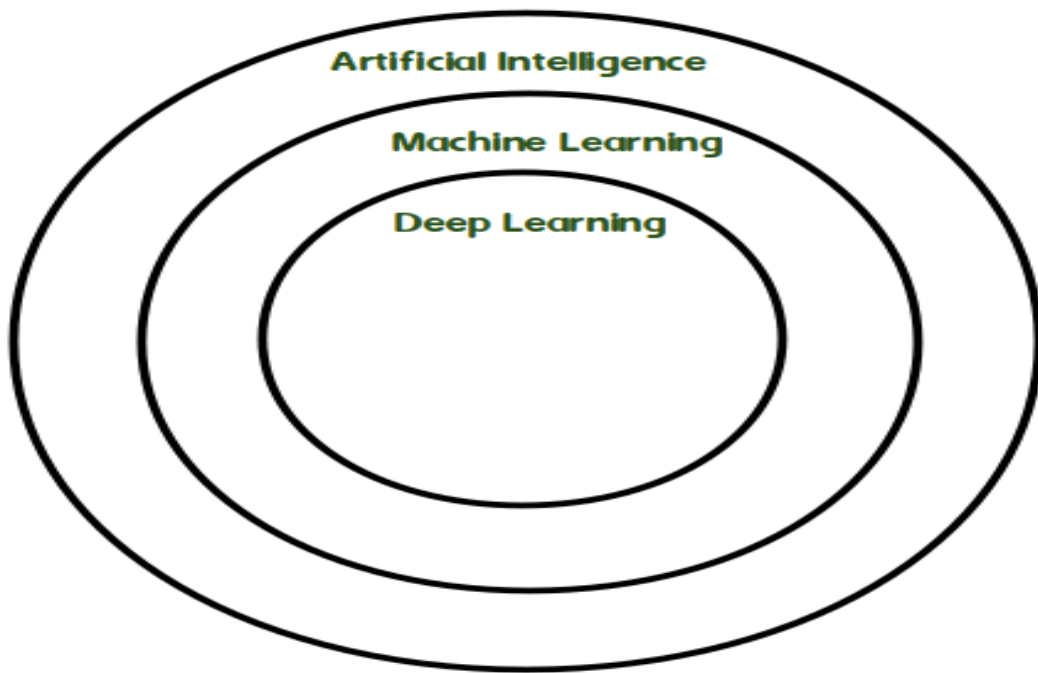
In human brain approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousands of their neighbours. The question here is how do we recreate these neurons in a computer. So, we create an artificial structure called an artificial neural net where we have nodes or neurons. We have some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.

Architectures :

1. **Deep Neural Network** – It is a neural network with a certain level of complexity (having multiple hidden layers in between input and output layers). They are capable of modeling and processing non-linear relationships.
2. **Deep Belief Network(DBN)** – It is a class of Deep Neural Network. It is multi-layer belief networks.

Steps for performing DBN :

- a. Learn a layer of features from visible units using Contrastive Divergence algorithm.
 - b. Treat activations of previously trained features as visible units and then learn features of features.
 - c. Finally, the whole DBN is trained when the learning for the final hidden layer is achieved.
3. **Recurrent (perform same task for every element of a sequence) Neural Network** – Allows for parallel and sequential computation. Similar to the human brain (large feedback network of connected neurons). They are able to remember important things about the input they received and hence enables them to be more precise.

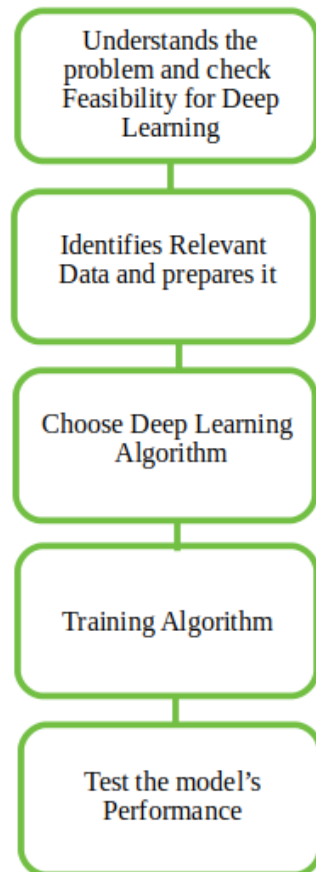


Difference between Machine Learning and Deep Learning :

MACHINE LEARNING	DEEP LEARNING
Works on small amount of Dataset for accuracy.	Works on Large amount of Dataset.
Dependent on Low-end Machine.	Heavily dependent on High-end Machine.
Divides the tasks into sub-tasks, solves them individually and finally combine the results.	Solves problem end to end.
Takes less time to train.	Takes longer time to train.
Testing time may increase.	Less time to test the data.

Working :

First, we need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not). Second, we need to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Third, Choose the Deep Learning Algorithm appropriately. Fourth, Algorithm should be used while training the dataset. Fifth, Final testing should be done on the dataset.



Tools used :

Anaconda, Jupyter, Pycharm, etc.

Languages used :

R, Python, Matlab, CPP, Java, Julia, Lisp, Java Script, etc.

Real Life Examples :

1. How to recognize square from other shapes?
2. ...a) Check the four lines!
3. ...b) Is it a closed figure?
4. ...c) Does the sides are perpendicular from each other?
5. ...d) Does all sides are equal?
- 6.
7. So, Deep Learning is a complex task of identifying the shape and broken down into simpler
8. tasks at a larger side.
- 9.
- 10.
11. Recognizing an Animal! (Is it a Cat or Dog?)

12. Defining facial features which are important for classification and system will then identify this automatically.
13. (Whereas Machine Learning will manually give out those features for classification)

Limitations :

1. Learning through observations only.
2. The issue of biases.

Advantages :

1. *Best in-class performance on problems.*
2. *Reduces need for feature engineering.*
3. *Eliminates unnecessary costs.*
4. *Identifies defects easily that are difficult to detect.*

Disadvantages :

1. *Large amount of data required.*
2. *Computationally expensive to train.*
3. *No strong theoretical foundation.*

Applications :

1. **Automatic Text Generation** – Corpus of text is learned and from this model new text is generated, word-by-word or character-by-character. Then this model is capable of learning how to spell, punctuate, form sentences, or it may even capture the style.
2. **Healthcare** – Helps in diagnosing various diseases and treating it.
3. **Automatic Machine Translation** – Certain words, sentences or phrases in one language is transformed into another language (Deep Learning is achieving top results in the areas of text, images).
4. **Image Recognition** – Recognizes and identifies peoples and objects in images as well as to understand content and context. This area is already being used in Gaming, Retail, Tourism, etc.
5. **Predicting Earthquakes** – Teaches a computer to perform viscoelastic computations which are used in predicting earthquakes.