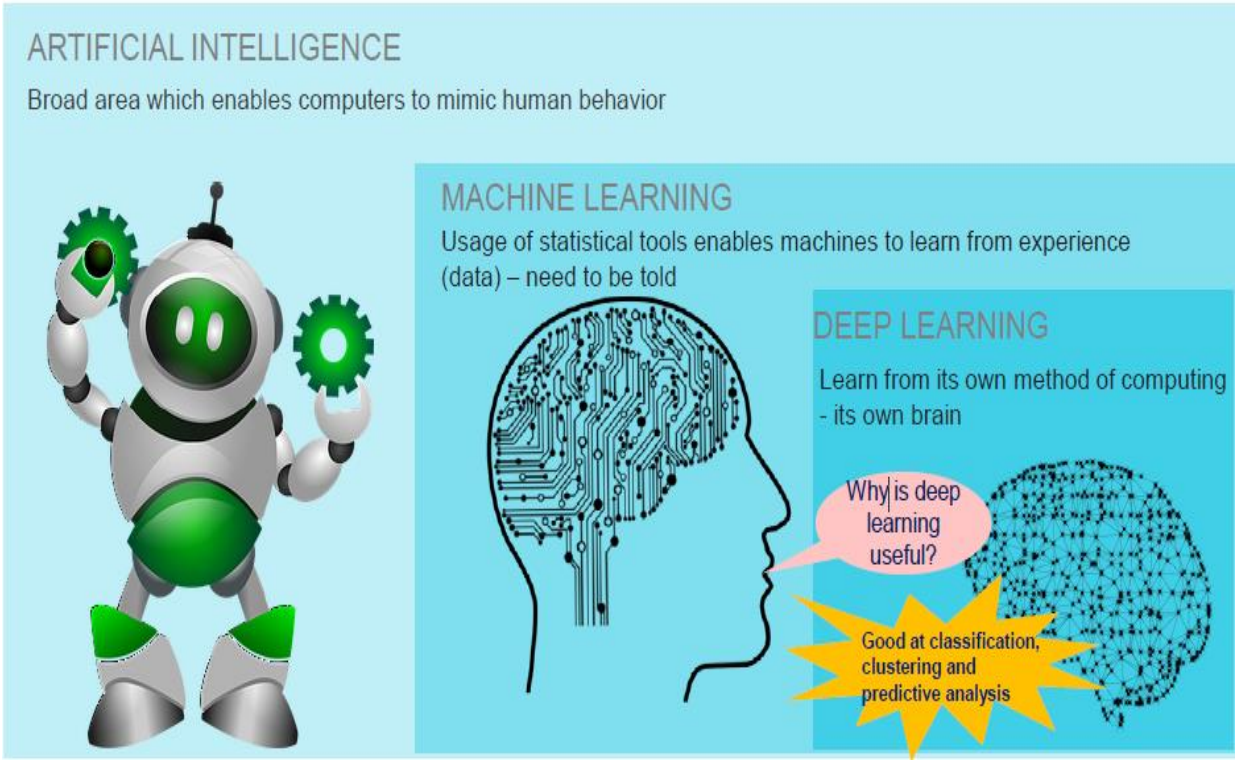
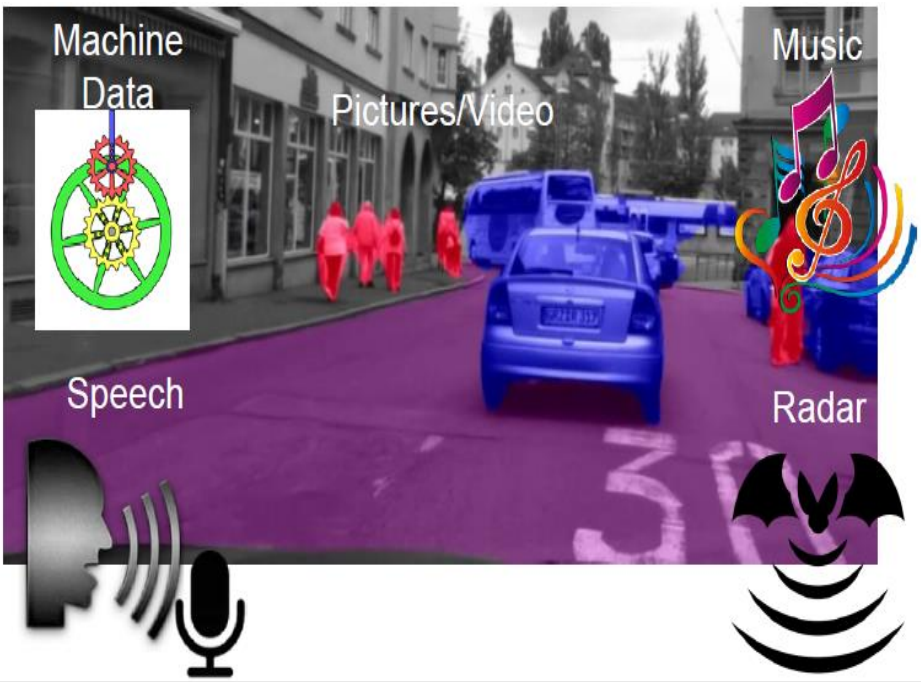
UNIT-I

Introduction to Deep Learning and Neural Networks: What is Deep Learning? , Design Issues, Applications and Types of DL Models. Artificial Neural Network- Basic Architecture, Non linear model of a neuron, Network Architectures- Single Layer Feed Forward, Multi Layer Feed Forward and Recurrent Networks. Training a Neural Network – Activation Functions: Step, Linear, Sigmoid, Tanh, ReLU, Leaky ReLU and Softmax. Back Propagation, loss and cost functions - Mean Absolute Error, Mean Squared Error, Binary Cross Entropy and Categorical Cross Entropy. Optimizers- Gradient Descent, Stochastic Gradient Descent, Mini-Batch Gradient Descent, Stochastic Gradient with Mumentm, Adagrad, AdaDelta, Adam.

Introduction to Deep Learning

What is deep learning?





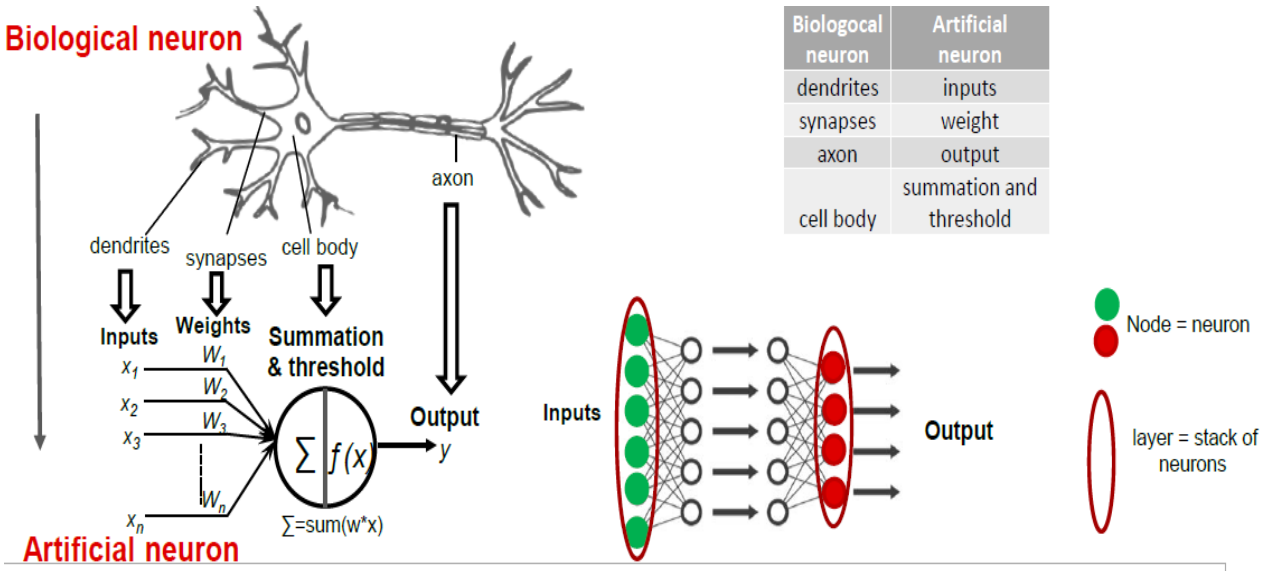
…any type of data you want to classify, cluster or predict

Deep learning is way of classifying, clustering, and predicting things by using a neural network that has been trained on vast amounts of data.

Deep Learning is a technique to mimic human brain.

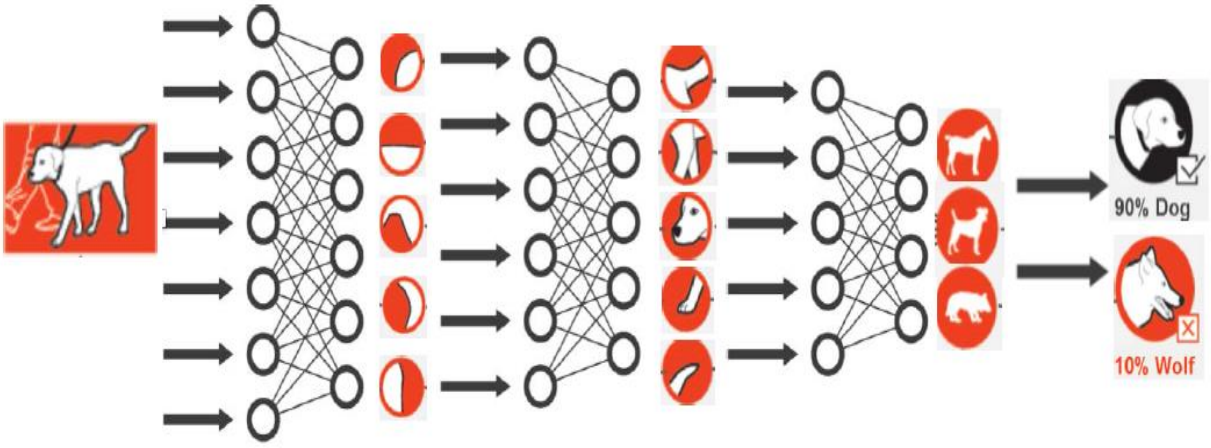
"Deep learning is a subset of machine learning technique that teaches computers to do what comes naturally to humans: learn by example.". i.e. Mimic the human being, makes the machine to learn, the way the human.

- **Deep learning has its roots in neural networks.**
- **Neural networks are sets of algorithms, modeled loosely after the human brain, that are designed to recognize patterns.**



Biologocal neuron	Artificial neuron
dendrites	inputs
synapses	weight
axon	output
cell body	summation and threshold

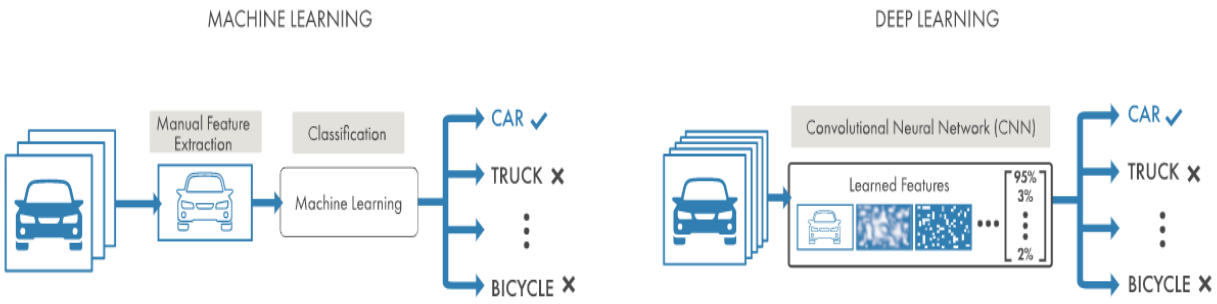
- **Deep learning creates many layers of neurons, attempting to learn structured representation of big data, layer by layer.**



- **A key advantage of deep learning networks is that they often continue to improve as the size of your data increases.**

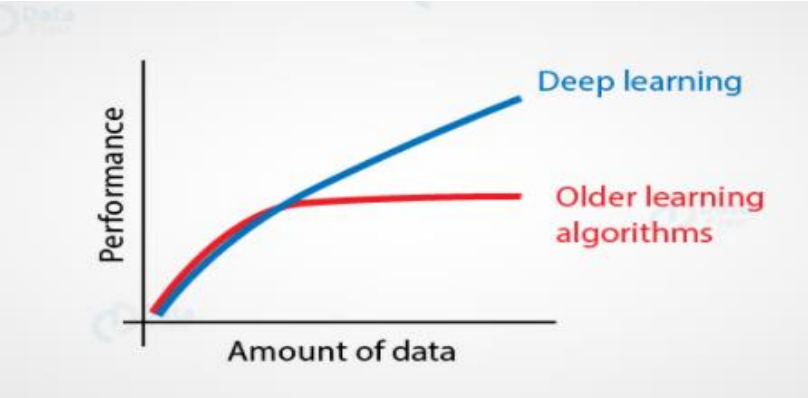
Design Challenges or Design Issues of Deep Learning

- **In machine learning, we manually choose features and a classifier to sort images. With deep learning, feature extraction and modeling steps are automatic.**



- **A successful deep learning application requires a very large amount of data (thousands of images) to train the model, as well as GPUs, or graphics processing units, to rapidly process your data.Having a high-performance GPU means the model will take less time to analyze all those images.**

Data dependencies: when the data is small, deep learning algorithms don't perform well. This is the only reason Deep Learning algorithms need a large amount of data to understand it perfectly.



1. **Hardware dependencies: Generally, deep learning depends on high-end machines. Thus, deep learning requirement includes GPUs. That is integral part of it's working. Also, they do a large amount of matrix multiplication operations.**

2. **Feature engineering: domain knowledge is put into the creation of feature extractors. Also, to reduce the complexity of the data. Further, make patterns more visible to learn algorithm working. Although, it's very difficult to process. Hence, it's time consuming and expertise.**

3. **Execution time: deep learning takes more time as compared to machine learning to train. The main reason behind its long time is that so many parameters in deep learning algorithm.**

Why Deep Learning:

1. **Exponential growth of data from social media, you tube, smart phones etc. With this complex use cases like recommendation system, face detection and so on can be implemented using deep learning models.**
   The accuracy is more with deep learning model.
2. **Technology upgradation in terms of Software and Hardware.**
   The data centers like Cloud and NVIDIA provides huge GPUs for processing.
3. **Feature Extraction (Feature Engineering) is part of deep learning model unlike machine learning models.**
4. **Solve complex problemstatements like image classification, object detection, NLP tasks, Chatbots etc.**

Deep learning: A few example uses

Industrial
- Object detection and localization
- Sorting
- Robotics
- Quality control and inspection
- AR (camera pose and location)
- Packaging

Retail
- Analytics
- Warehouse management
- Theft prevention
- Intelligent bar code scanners
- Monitoring and distribution control (shelf replenishment, etc.)

Entertainment/Gaming
- Gesture recognition
- User identification
- Emotional feedback
- Experience monitoring
- Advanced analytics

Smart Homes
- Vacuum cleaners
- Automatic lawn mowers
- Intrusion and Hazard detection
- Smart lights, ovens, refrigerators, etc.

Drones
- Obstacle avoidance
- Path planning
- Flight control with radar and camera sensors

Agriculture
- Autonomous tractors and combines
- Fruit harvesting
- Weed control

Smart Cities & Infrastructure
- Parking
- Traffic monitoring
- Security monitoring
- Road inspection

Food Industry
- Sorting
- Quality control

Mission Critical
- Perimeter surveillance
- Target acquisition
- Fire-and-forget guidance
- Autonomous vehicles

Anywhere you want to classify data …

Industrial Factory & Automation

- Improving pick and place
- Predictive maintenance/failure

Agriculture

Optimize crop watering and harvesting

Retail

- Improve automated checkout
- Track shoppers and provide incentives

Deep learning applications

- Self Driving Cars
- News Aggregation and Fraud News Detection
- Natural Language Processing
- Virtual Assistants
- Entertainment
- Visual Recognition
- Fraud Detection
- Healthcare
- Personalisations
- Automatic Machine Translation
- Automatic Handwriting Generation
- Automatic Game Playing
- Language Translations
- Pixel Restoration
- Photo Descriptions
- Demographic and Election Predictions
- Voice Controlled Assistance
- Automatic Image Caption Generation
- Automatic Machine Translation

Types of Deep Learning Models

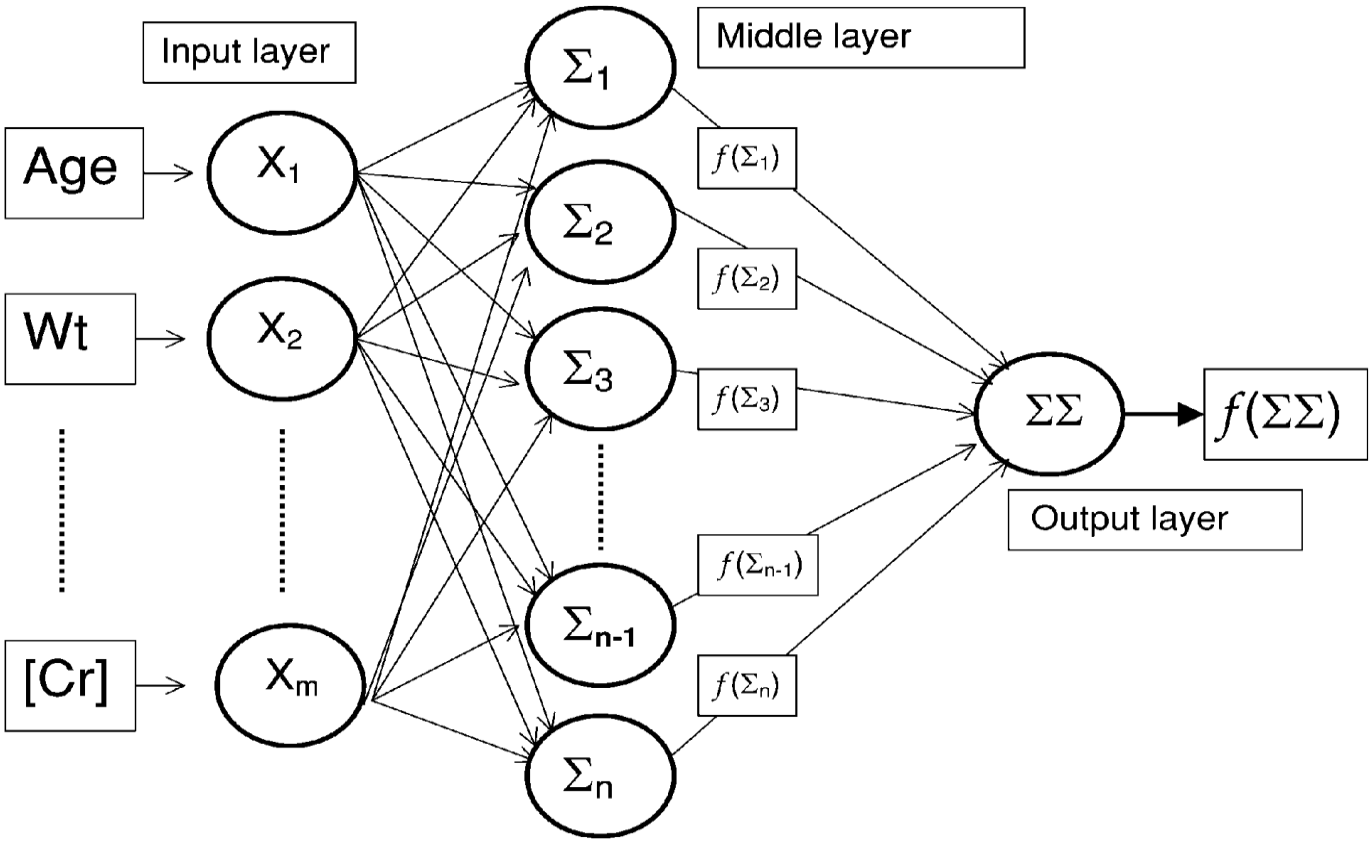
1. **ANN – Artificial Neural Network**
    - Works on tabular kind of input data
    - Every task of ML model can be solved using ANN
    - Applications solved using ANN are classification, pattern recognition, pattern completion, data mining (discovering hidden data) etc.
2. **CNN - Convolution Neural Network**
    - Works on Images or Video as input data
    - Video is a combination of multiple frames or images with sound.
    - Applications solved using CNN are
        - Image classification

- Object Detection
- Object Segmentation
- Object Tracking
- Face recognition

3. **RNN – Recurrent Neural Network**
   - Works with input data as text or time series data (Tracking daily, hourly, or weekly weather data, Tracking changes in application performance, Medical devices to visualize vitals in real time, Tracking network logsetc) or sequential data.
   - Applications solved using RNN can be
     - NLP tasks like
       - Prediction problems
       - Machine Translation
       - Speech Recognition
       - Generating Image Descriptions
       - Video Tagging
       - Text Summarization
       - Call Center Analysis
     - Text to Speech conversion

Artificial Neural Network

Basic Architecture of ANN:



Input Layer: **This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information (features) to the hidden layer.**

Hidden Layer: **Nodes of this layer are not exposed to the outer world, they are the part of the abstraction provided by any neural network. Hidden layer performs all sort of computation on the features entered through the input layer and transfer the result to the output layer.**

Output Layer: **This layer brings up the information learned by the network to the outer world.**

- ANNs consist of artificial neurons, each artificial neuron has a processing node ('body') represented by circles in the figure as well as connections from ('dendrites') and connections to ('axons') other neurons which are represented as arrows in the figure.
- In a commonly used ANN architecture, the multilayer perceptron, the neurons are arranged in layers. An ordered set (a vector) of predictor variables is presented to the input layer.
- Each neuron of the input layer distributes its value to all of the neurons in the middle layer (hidden layer).
- Along each connection between input and middle neurons there is a connection weight so that the middle neuron receives the product of the value from the input neuron and the connection weight.
- Each neuron in the middle layer takes the sum of its weighted inputs and then applies a non-linear (usually activation) function to the sum. The result of the function then becomes the output from that particular middle neuron.
- Each middle neuron is connected to the output neuron. Along each connection between a middle neuron and the output neuron there is a connection weight.
- In the final step, the output neuron takes the weighted sum of its inputs and applies the non-linear function to the weighted sum.
- The result of this function becomes the output for the entire ANN.
- If multiple output classification, then output layer has more neurons.

Non Linear Model of Neuron:

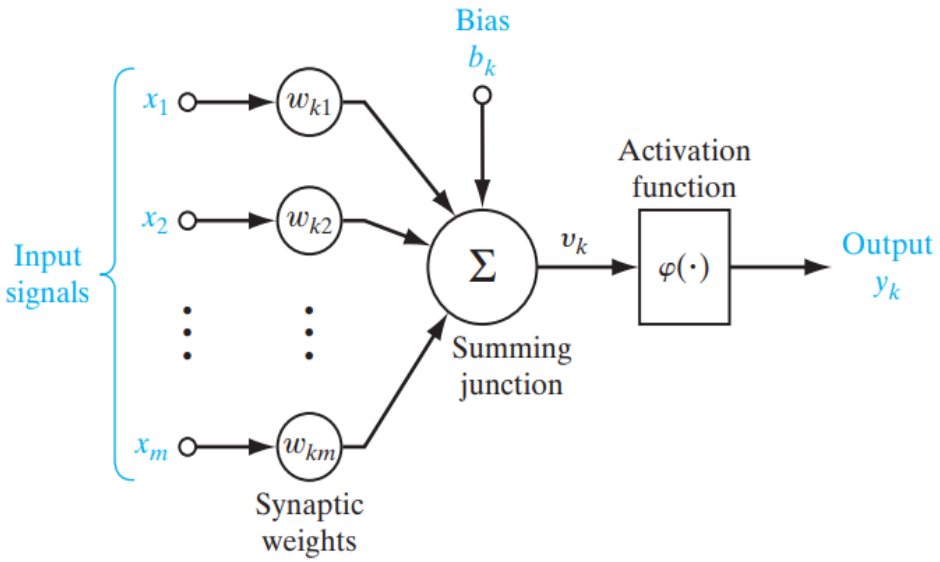
A neuron is an information processing unit that is fundamental to the operation of a neural network.

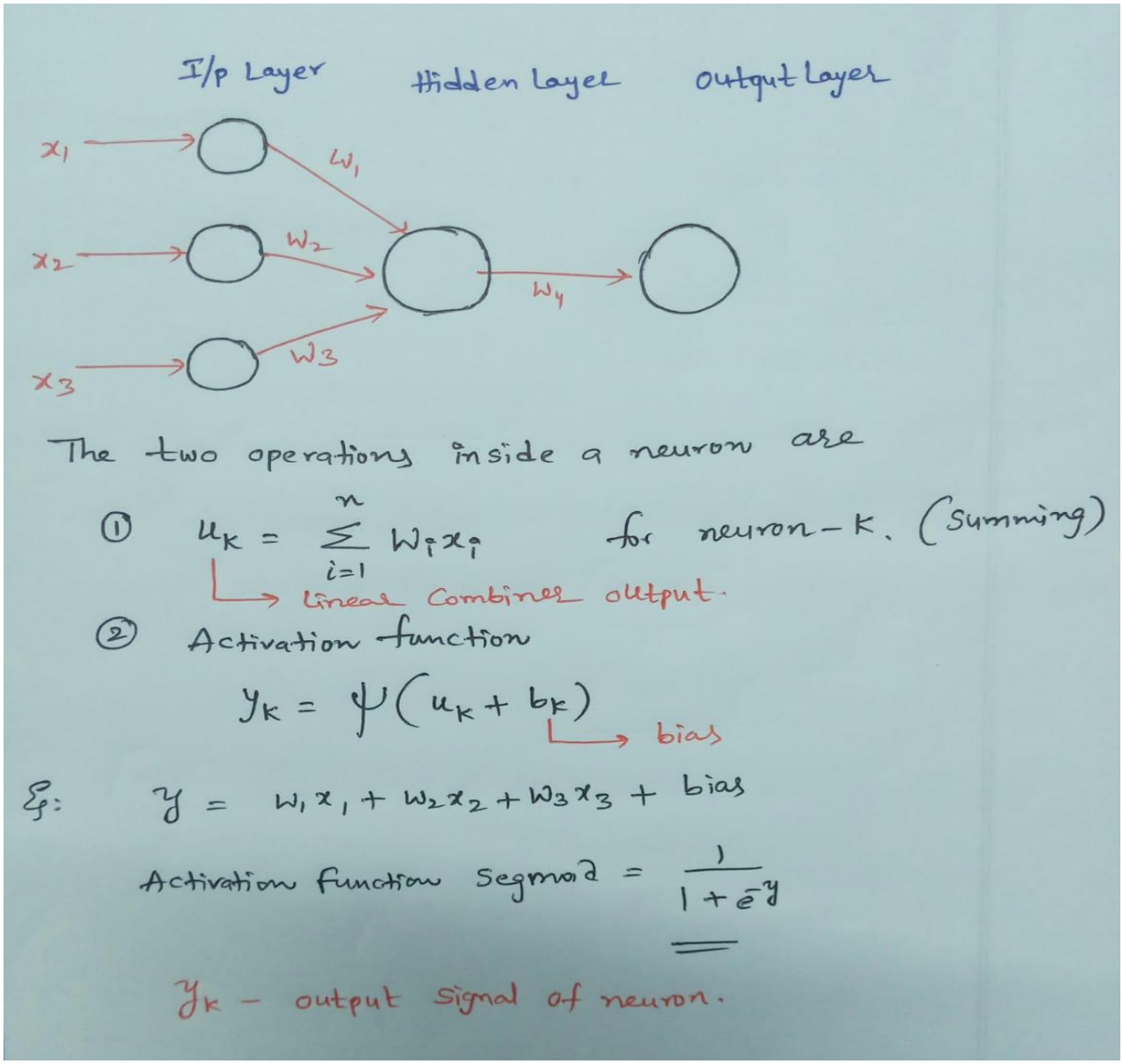
The following model of a neuron which form the basis for designing artificial neural networks.

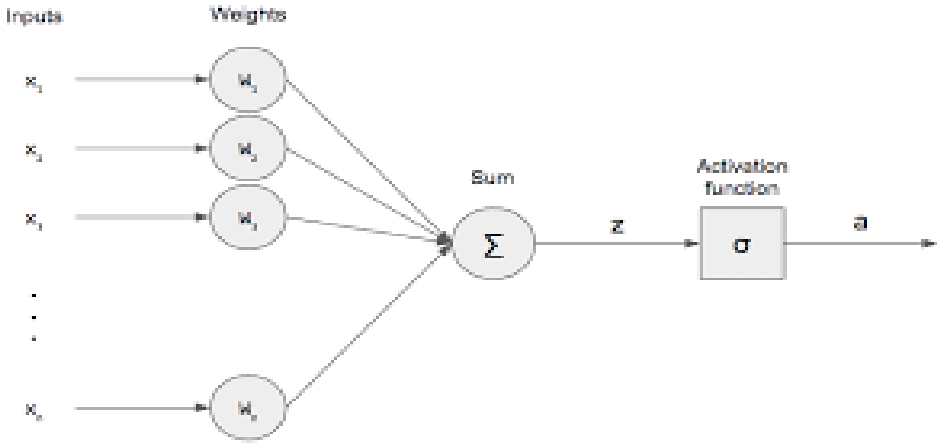
The three basic elements of a neural model are

1. **A set of synapses or connecting links, each which is characterized by a weight or strength of its own.**
2. **An adder for summing the input signals weighted by the respective synaptic of the neuron; the operations described here constitute a linear combiner.**
3. **An activation function for limiting the amplitude of the output of a neuron, also called squashing function. The activation function is used to activate a neuron.**
   The normalized range of output of a neuron is [0,1] or [-1 1].

   The neural model also includes an externally applied bias.The bias has the effect of increasing or lowering the net input of theactivation function. (depending on positive or negative bias).




I/p Layer
Hidden Layer
Output Layer
x_1
x_2
x_3
w_1
w_2
w_3
w_4
The two operations inside a neuron are
① $u_k = \sum_{i=1}^{n} w_i x_i$ for neuron-k. (Summing)
Linear Combiner output.
② Activation function
$y_k = \psi(u_k + b_k)$
bias
Eg: $y = w_1 x_1 + w_2 x_2 + w_3 x_3 + \text{bias}$
Activation function Segmoid = $\frac{1}{1+e^{-y}}$
y_k - output signal of neuron.

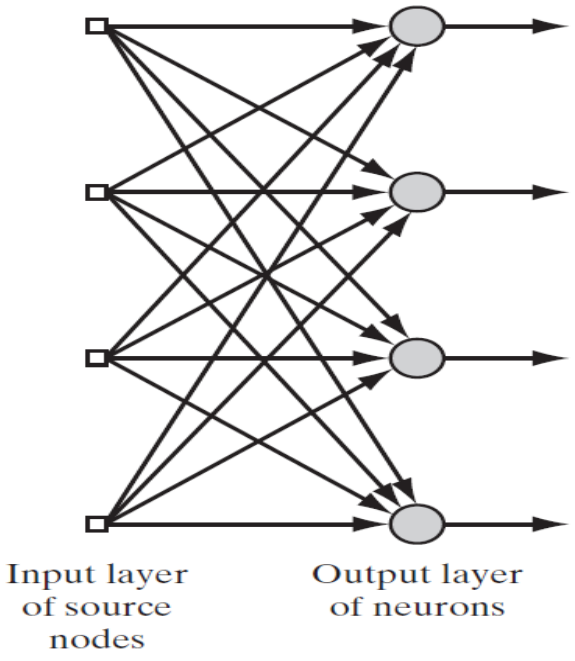

Inputs
Weights
Sum
Activation function
z
a
σ

Neural Network Architectures

The way in which the neurons of a neural network are structured is linkedwith the learning algorithm used to train the network. The various structures are

1. Single-Layer Feed forward Networks

- **In a layered neural network, the neurons are organized in the form of layers.**
- **In the simplest form of a layered network, we have an input layer of source nodes that projects directly onto an output layer of neurons (computation nodes).**
- **This network is strictly of a feedforward type.**
- **The case of four nodes in both the input and output layers is shown below is called a single-layer network.**

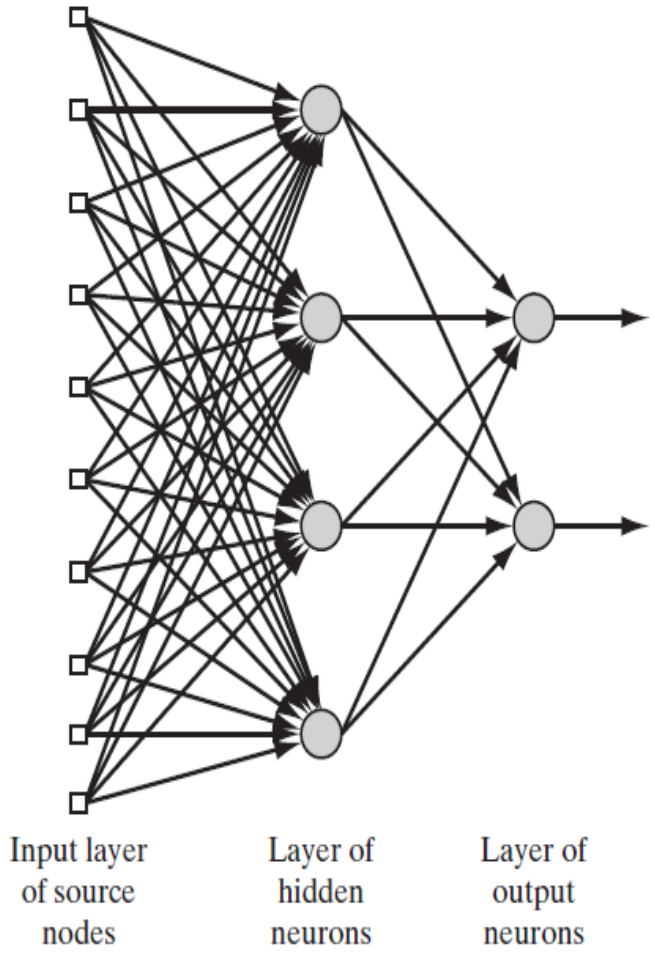


- **The designation "single-layer" referring to the output layer of computation nodes (neurons).**
- **We do not count the input layer of source nodes because no computation is performed there.**

2. Multilayer Feed forward Networks

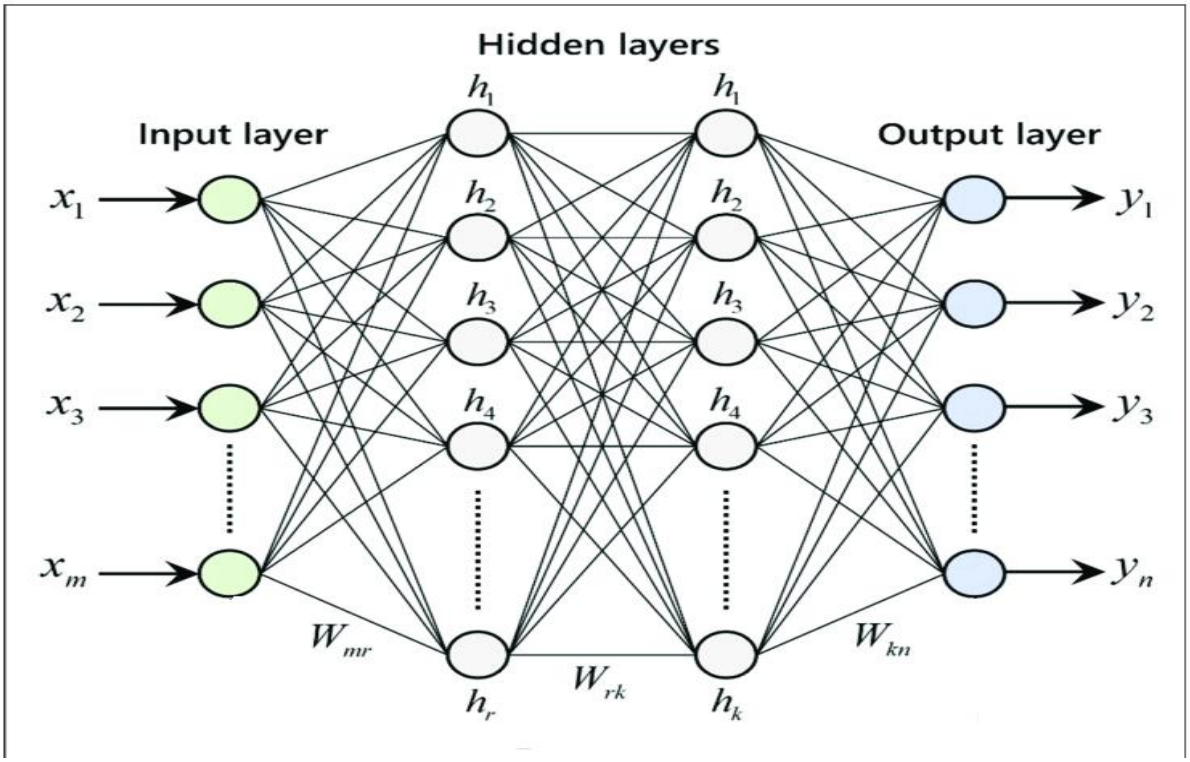
- The presence of one or more hidden layers, whose computation nodes are correspondingly called hidden neurons or hidden units;
- The function of hidden neurons is to intervene between the external input and the network output in some useful manner.
- By adding one or more hidden layers, the network is enabled to extract higher-order statistics from its input.
- Due to the extra set of synaptic connections and the extra dimension of neural interactions the learning process is more accurate.
- The source nodes in the input layer of the network supply respective elements of the activation pattern (input vector), which constitute the input signals applied to the neurons (computation nodes) in the second layer (i.e., the first hidden layer). The outputsignals of the second layer are used as inputs to the third layer, and so on for the rest of the network.
- Typically, the neurons in each layer of the network have as their inputs the output signals of the preceding layer only. The set of output signals of the neurons in the output (final) layer of the network constitutes the overall response of the network to the activation pattern supplied by the source nodes in the input (first) layer.
- The architectural graph of a multilayer feedforward neural network for the case of a single hidden layer is shown below. It is referred to as a 10–4–2 network because it has 10 source nodes, 4 hidden neurons, and 2 output neurons.
- In general, a feedforward network with m source nodes, h1 neurons in the first hidden layer, h2 neurons in the second hidden layer, and q neurons in the output layer is referred to as an m–h1–h2–q network.

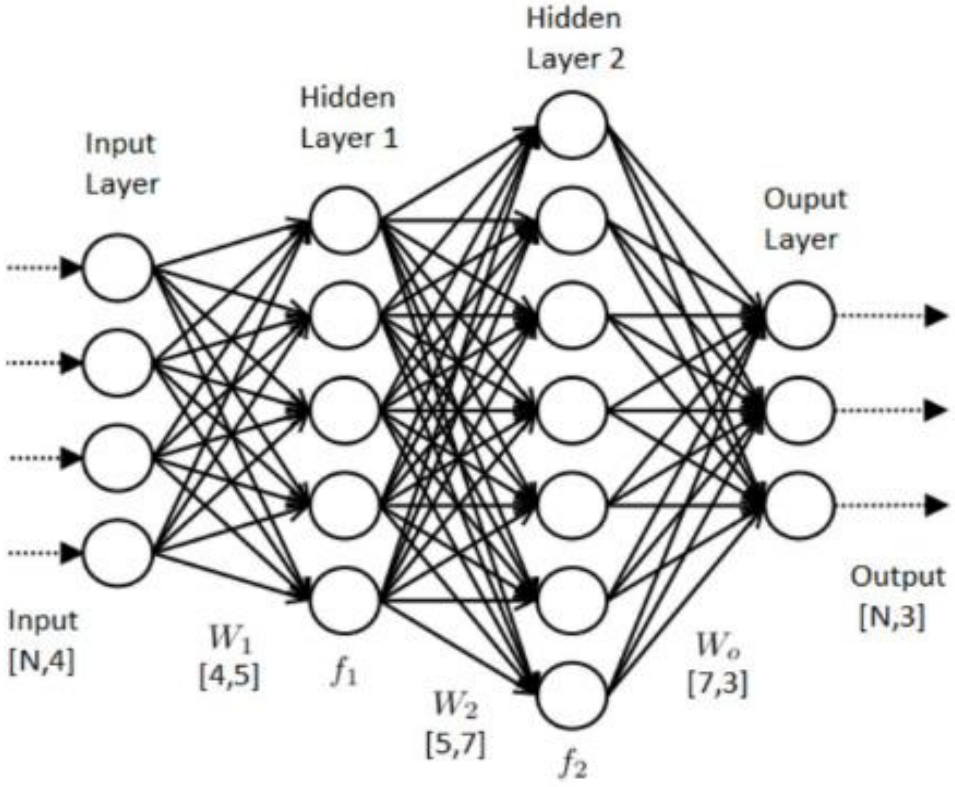
Fully connected feedforward network with one hidden layer and one output layer.



The neural network in the above figure is said to be fully connected in the sense that everynode in each layer of the network is connected to every other node in the adjacent forwardlayer.

If, however, some of the communication links (synaptic connections) aremissing from the network, we say that the network is partially connected.


Hidden layers
Input layer
Output layer
x_1
x_2
x_3
x_m
h_1
h_2
h_3
h_4
h_r
h_1
h_2
h_3
h_4
h_k
y_1
y_2
y_3
y_n
W_{mr}
W_{rk}
W_{kn}

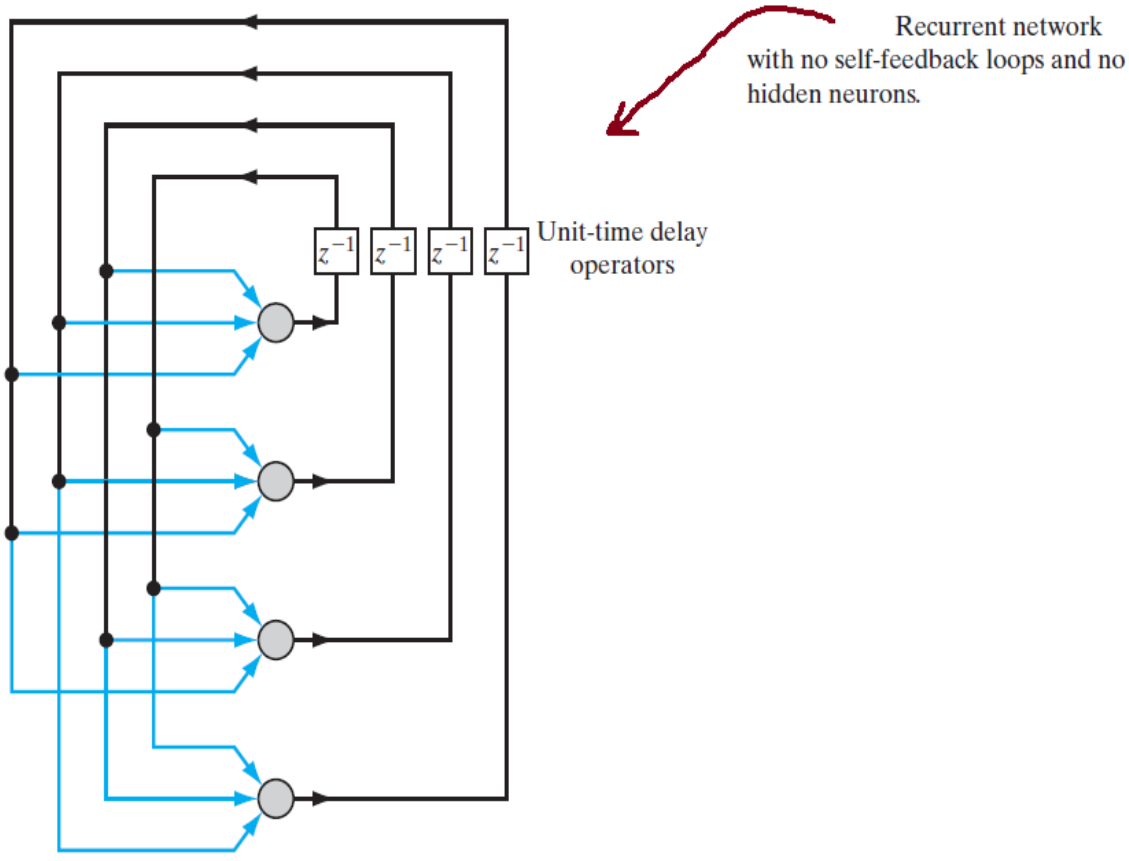

Hidden
Layer 2
Hidden
Layer 1
Input
Layer
Ouput
Layer
Input
[N,4]
W_1
[4,5]
f_1
W_2
[5,7]
f_2
W_o
[7,3]
Output
[N,3]

3. Recurrent Networks

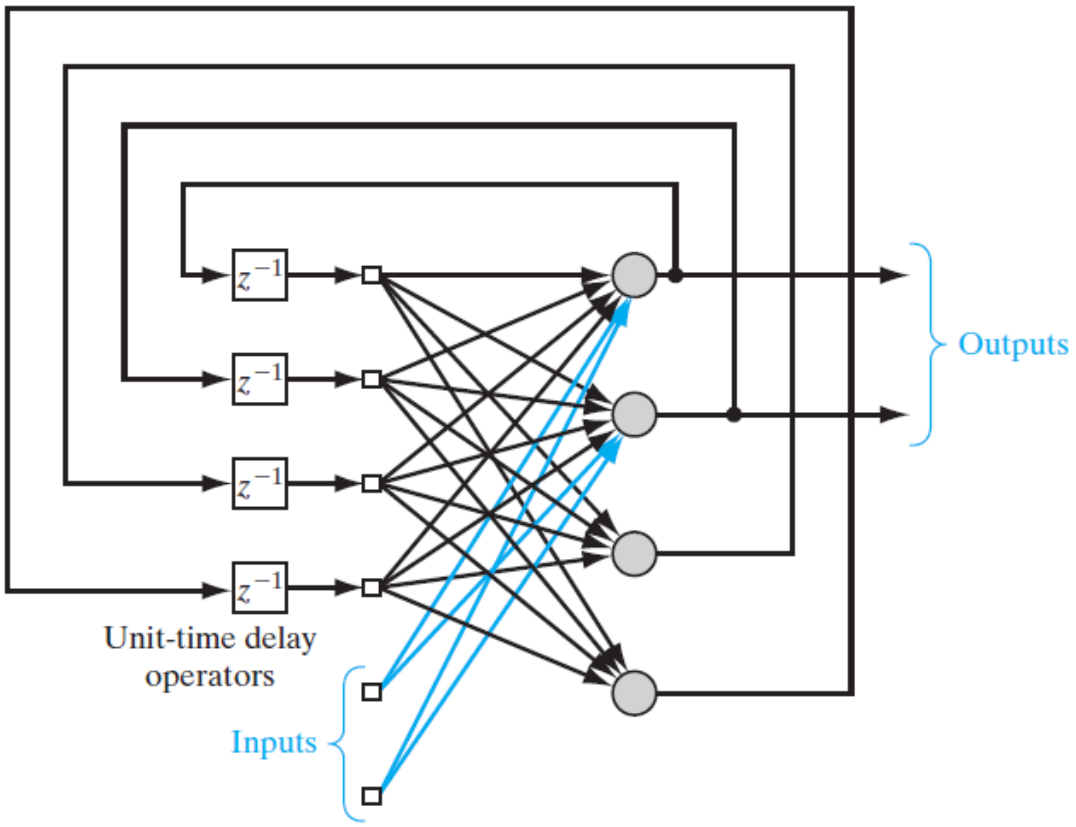
Arecurrent neural network distinguishes itself from a feedforward neural network in thatit has at least one feedback loop.

For example, a recurrent network may consist of a singlelayer of neurons with each neuron feeding its output signal back to the inputs of all the other neurons.

In the below structure there are no self-feedback loops in the network; self-feedback refersto a situation where the output of a neuron is fed back into its own input.The network also has no hidden neurons.



The recurrent networks with hidden neurons.



Recurrent network with hidden neurons.

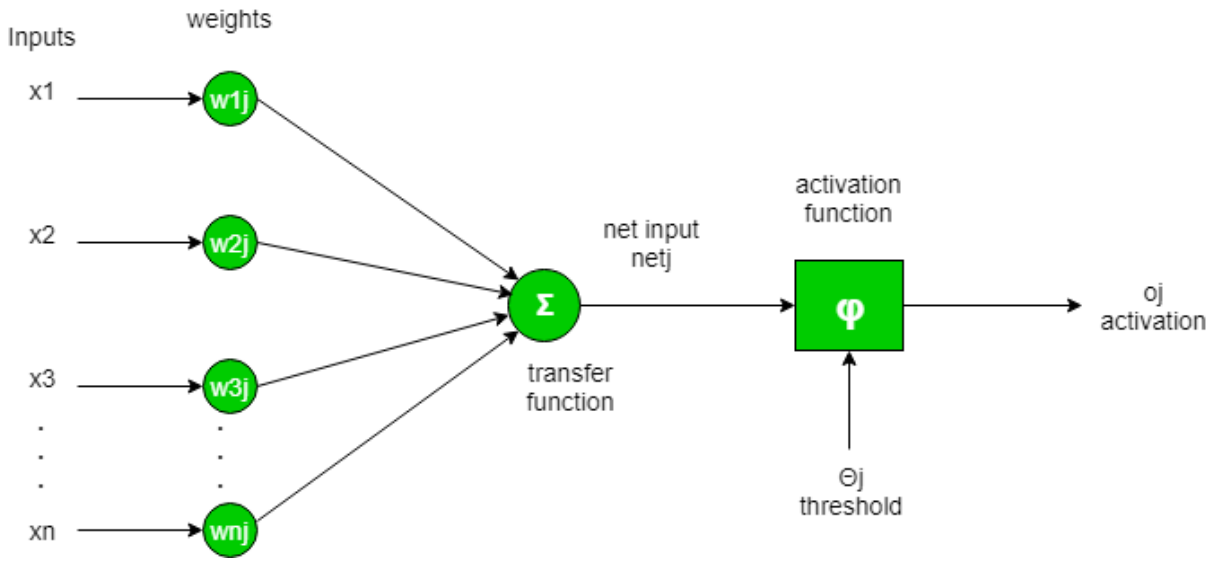
The feedback connections originate from the hidden neurons as wellas from the output neurons.

The presence of feedback loopshas a profound impact on the learning capability of the network and on its performance.

Moreover, the feedback loops involve the use of particular branches composedof unit-time delay elements (denoted by z-1), which result in a nonlinear dynamicbehavior, assuming that the neural network contains nonlinear units.

Activation Functions

An artificial neuron calculates the 'weighted sum' of its inputs and adds a bias, as shown in the figure below by the net input.



Mathematically,

$$\text{net input} = \sum_{(\text{weight} * \text{input})+\text{bias}}$$

- **Now the value of net input can be any anything from -inf to +inf.**
- **The neuron doesn't really know how to bound to value and thus is not able to decide the firing pattern.**
- **The activation function is an important part of an artificial neural network. They basically decide whether a neuron should be activated or not.Thus it bounds the value of the net input.**
- **The activation function is a non-linear transformation that we do over the input before sending it to the next layer of neurons or finalizing it as output.**
- **Activation function is also called "Transfer Function".**

Different types of activation functions are used in Deep Learning.

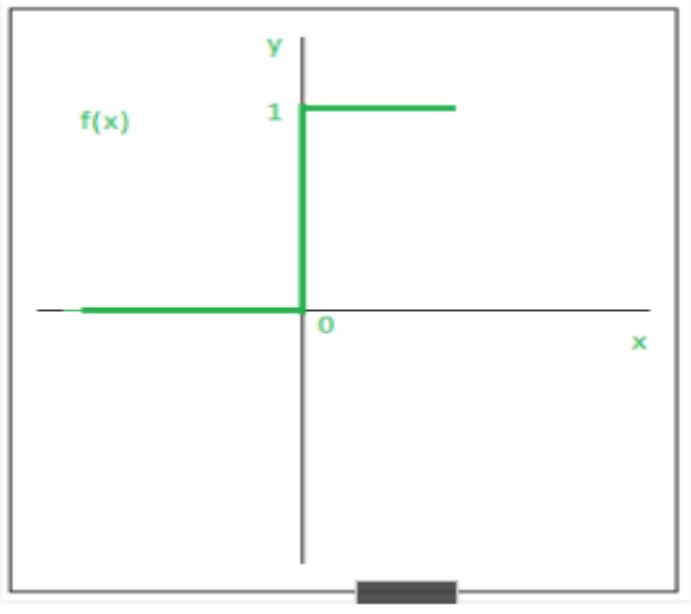
1. **Step Function:**

   Step Function is one of the simplest kind of activation functions. In this, we consider a threshold value and if the value of net input say y is greater than the threshold then the neuron is activated.

   Mathematically,

$$f(x) = 1, \text{if } x >= 0$$
$$f(x) = 0, \text{if } x < 0$$

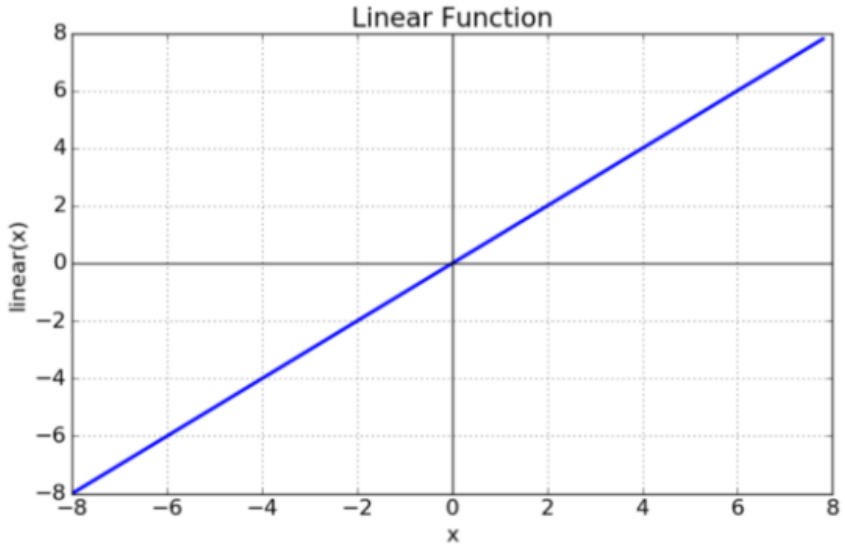
   The graphical representation of step function is



2. Linear Function:

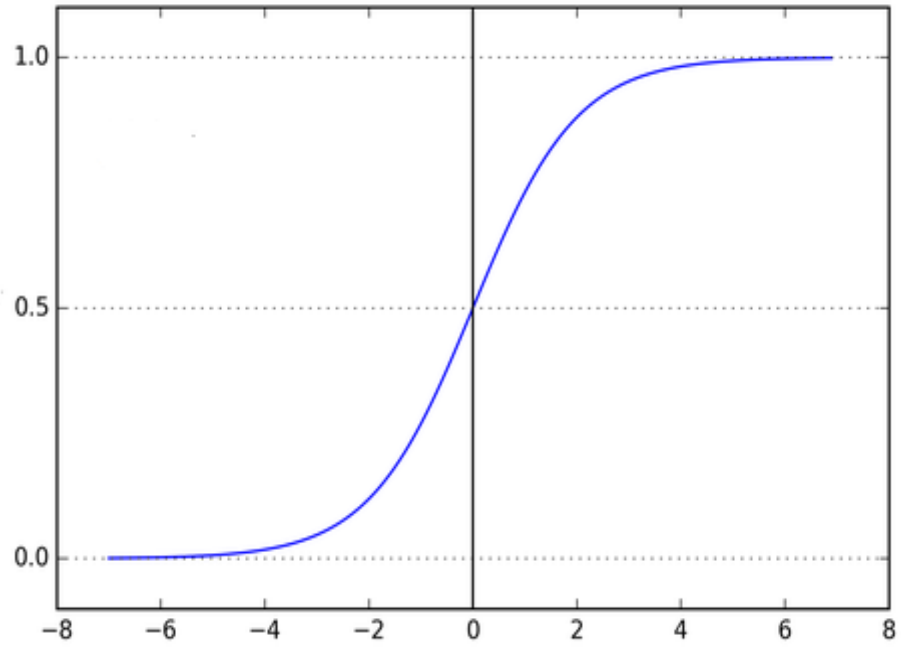
- **Linear function has the equation similar to as of a straight line i.e. y = ax.**
- **No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.**
- **Range of output is -inf to +inf**
- **Linear activation function is used at just one place i.e. output layer.**
- **If we will differentiate linear function to bring non-linearity, result will no more depend on input "x" and function will become constant, it won't introduce any ground-breaking behavior to our algorithm.**
- **Y = cx, derivative with respect to x is c. That means, the gradient has no relationship with x. It is a constant gradient and the descent is going to be on constant gradient.**

- **Eg: Calculation of price of a house is a regression problem. House price may have any big/small value, so we can apply linear activation at output layer.**



3. Sigmoid Function:

- Sigmoid function is a widely used activation function.
- It is a function which is plotted as 'S' shaped graph.
- Equation:$A = 1/ (1 + e^{-x})$
- This is a smooth function and is continuously differentiable. when I have multiple neurons having sigmoid function as their activation function, the output is nonlinear as well.
- Non-linear activation function.
- The function ranges value from 0 to 1.
- Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.
- If the two outputs category



4. TanhFunction:

- **Tanh function also knows as Tangent Hyperbolic function.**
- **It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.**
- **The function is**

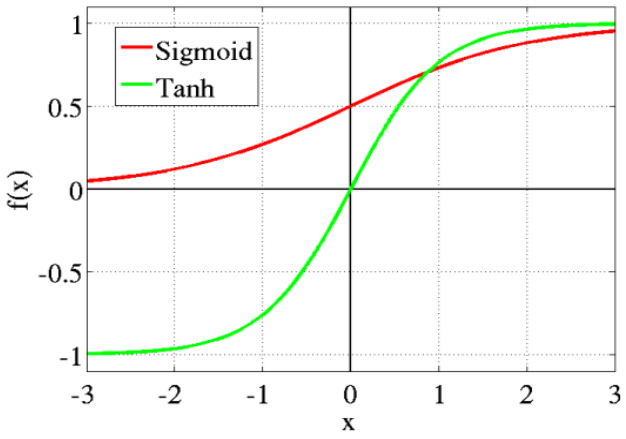
$$f(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

(or)

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

(Or)

- **tanh(x) = 2 * sigmoid(2x) – 1**
- **The range of values is from -1 to +1.**
- **non-linear activation function**



- **The negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph. The function is differentiable.**

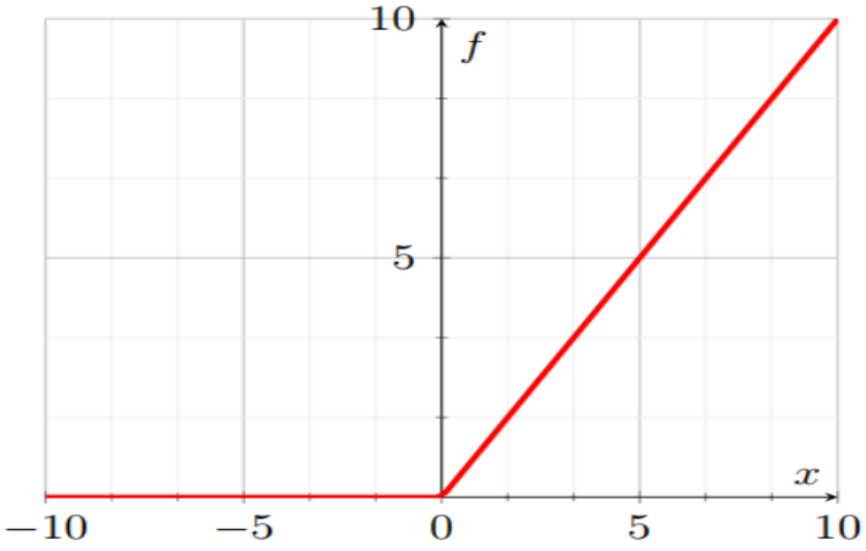
- **The tanh function is mainly used classification between two classes.**

5. RELU Function:

- **Stands for Rectified linear unit.**
- **It is the most widely used activation function.**
- **Equation: A(x) = max (0, x). It gives an output x if x is positive and 0 otherwise.**

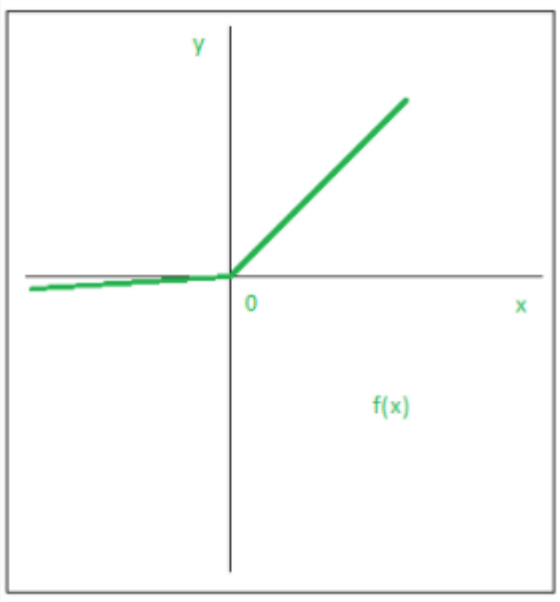
  If x is –ve => max(0, -ve) =0

  If x is +ve => max(0,+ve)=+ve
- **The value range is [0, inf).**
- **non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.**
- **ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.**
- **Used in Hidden layers (middle).**
- **RELU learns much faster than sigmoid and Tanh function.**



6. Leaky ReLU:

- **Leaky ReLU function is nothing but an improved version of the ReLUfunction.Instead of defining the Relu function as 0 for x less than 0, we define it as a small linear component of x. It can be defined as:**
- **f(x) = ax, x<0**
  f(x) = x, otherwise.



7. Softmax Function:

- **The softmax function is also a type of sigmoid function and used to handle classification problems.**
- **It is used if output has more than two.**
- **Usually used when trying to handle multiple classes. The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.**
- **non-linear activation function**
- **The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.**

- **output for the Softmax function is the ratio of the exponential of the parameter and the sum of exponential parameter.**

Formula

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

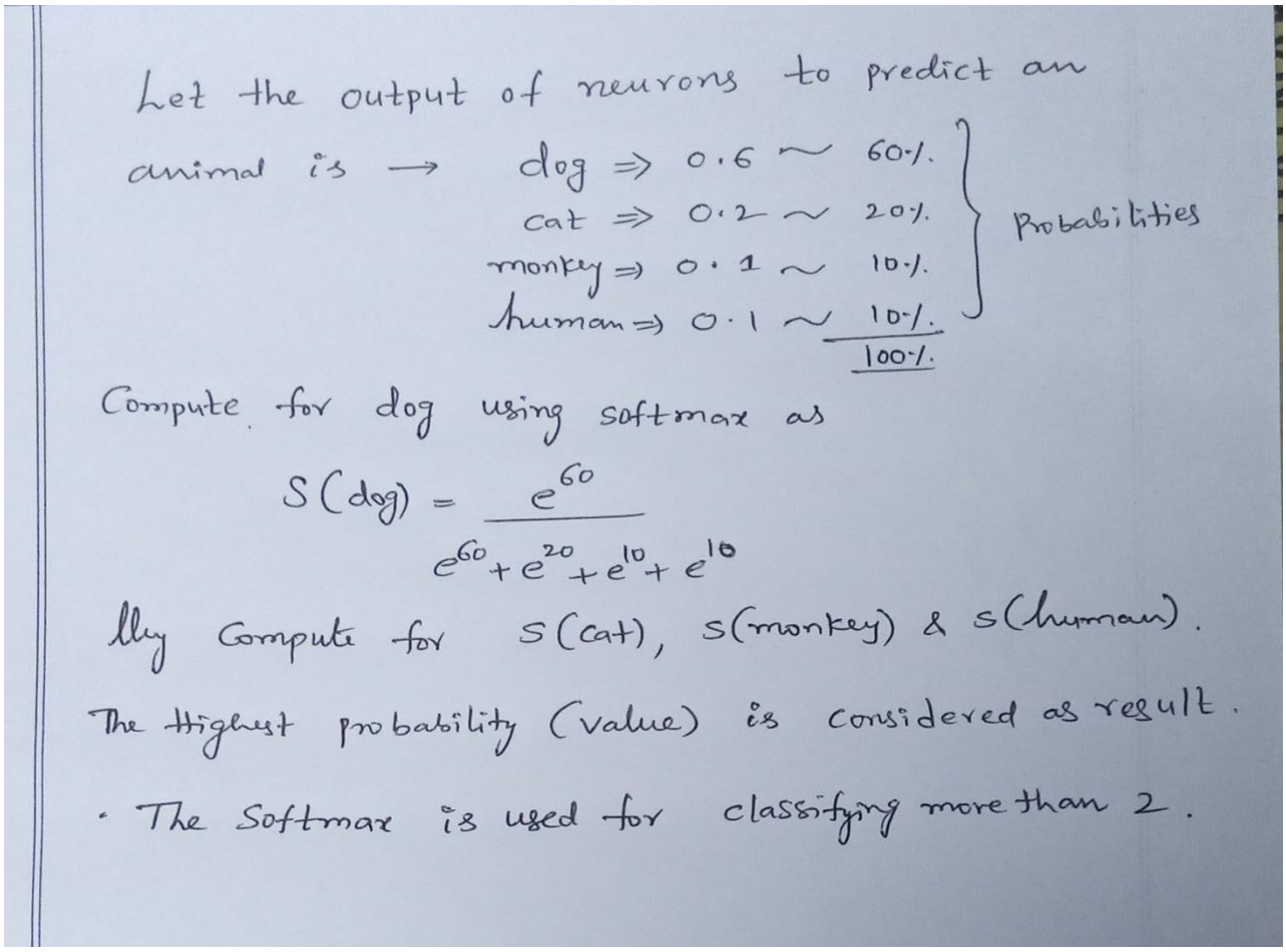
σ = softmax

$\vec{z}$ = input vector

e^{z_i} = standard exponential function for input vector

K = number of classes in the multi-class classifier

e^{z_j} = standard exponential function for output vector



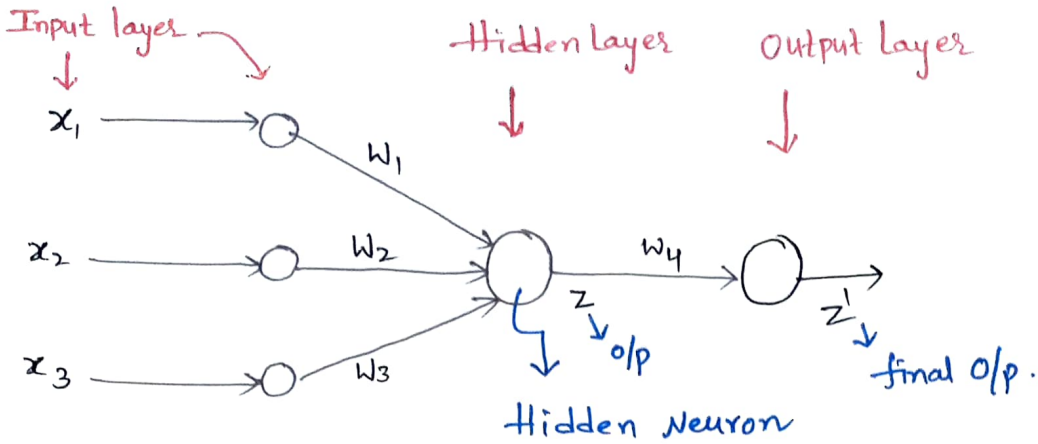
CHOOSING THE RIGHT ACTIVATION FUNCTION

- **If we don't know what activation function to use, then simply use RELU as it is a general activation function and is used in most cases these days.**
- **If output is for binary classification then, sigmoid function is very choice for output layer.**
- **If output is other than binary classification then, softmax function is choice for output layer.**
- **For all hidden layers RELU is choice of activation function.**

"The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks."

Training Artificial Neural Network

How Neural Network Works



In a neural network, at each neuron two operations are performed.

Step-1 : Summation

$$\sum_{i=1}^{n} w_i x_i + \text{bias}$$, Where n – no. of incoming edges at a neuron. i.e. Synapses.

In the above example diagram

$$y = w_1x_1 + w_2x_2 + w_3x_3 + \text{bias}.$$

Step-2 : Activation function.

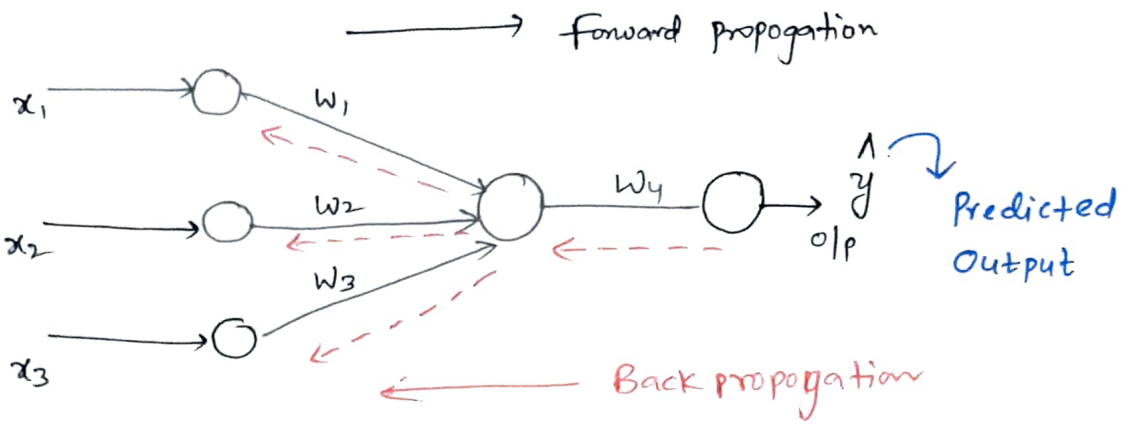
$\text{Act}\left(\sum_{i=1}^{n} w_i x_i + \text{bias}\right) \Rightarrow$ denoted by $\psi(y)$

i.e. $z = \psi(y)$.

The final o/p $z' = \psi\left((z * w_4) + \text{bias}\right)$

Training Neural Network (Back Propogation) (1).

Consider the following neural network for a dataset consisting of 3 input features and an output.



At each neuron the two operations Summation & activation function are applied.

① $y = \sum_{i=1}^{n} w_i x_i + \text{bias}$

② $z = \psi(y)$.

The dataset consist of records of various students and their output value. If output value is 1, the student is pass. If the output value is 0, then the student is fail.

Eg:

Play	Study	Sleep	o/p (y)
2 hrs	4 hrs	8 hrs	1
—	—	—	—
—	—	—	—
—	—	—	—

The predicted output ($\hat{y}$) is compared with actual output (y) by computing a loss function. (cost function)

$\text{loss} = (y - \hat{y})^2$ ⇒ for one record

$\text{loss} = \sum_{i=1}^{n} (y - \hat{y})^2$ ⇒ for n records in dataset.

- The difference between actual and predicted output must be minimized to reach same value using neural network.
- To minimize loss "back propogation" is used with an optimizer.
- During backpropogation the weights are updated and finds the $\hat{y}$ (predicted o/p) again in next iteration.

Backpropogation :- It is the essence of neural network training.

- It is the method of fine tuning the weights of neural network based on the error rate obtained in previous epoch (i.e. iteration).

Proper tuning of weights allows to reduce error rates and make the model reliable by increasing its generalization.

- 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.
- The weights are updated as $\Rightarrow W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$

Here η – is Learning rate, a small value (~ 0.001)

Eg: In the above neural network

$$W_{4new} = W_4 - \eta \frac{\partial L}{\partial W_4}$$

P.T.O.

Learning Rate : It is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

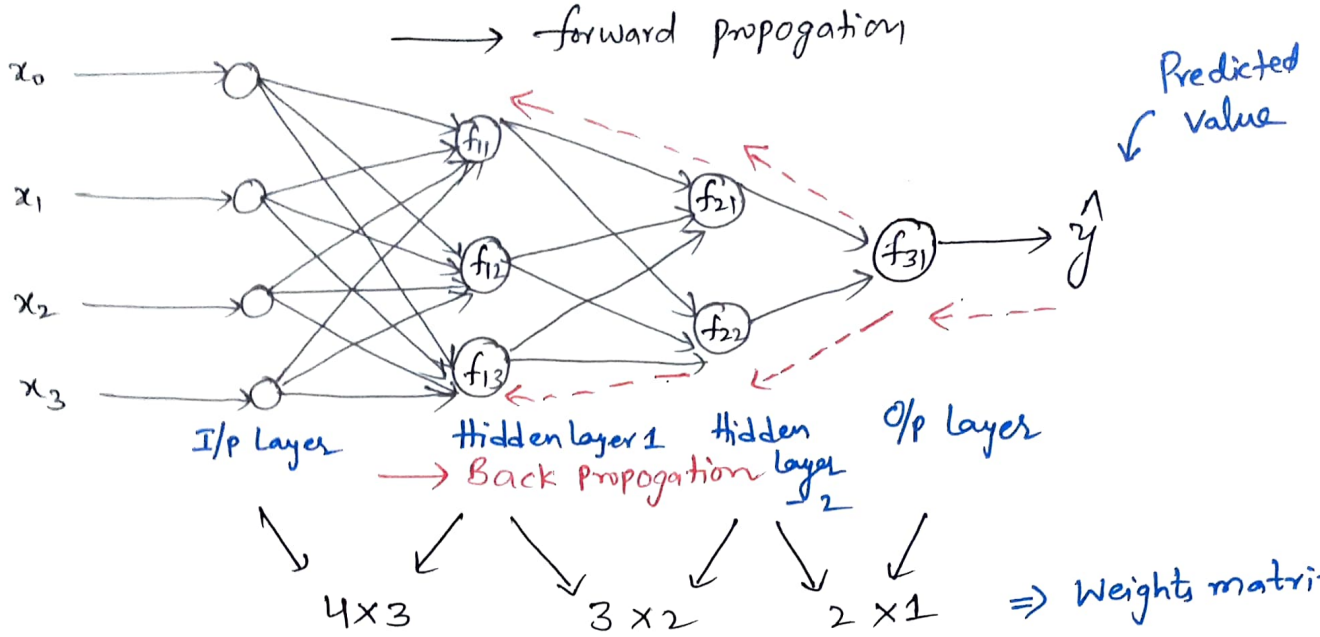
- In setting learning rate there must be tradeoff between the rate of Convergence and overshooting.
- It is commenly reffered as ~~eg~~ "gain".

✡ One forward propogation & backpropogation combinely called as one iteration (or) epoch.

✡ The epochs are continued untill the 'loss' is very minimum.

✡ The learning process of neurons in each iteration is in Correct path or not will be known with 'loss' value Computed after each iteration. If 'loss' is minimizing then the training is in right path and the value selected for (η) learning rate is appropriate.

Training MultiLayer Neural Network



After each forward propogation, the loss function is used to Compare Actual and predicted value. To reduce the loss value, an optimizer is used in backpropogation to update weights.

$$W_{new} = W_{old} - \eta \frac{dL}{dW}$$

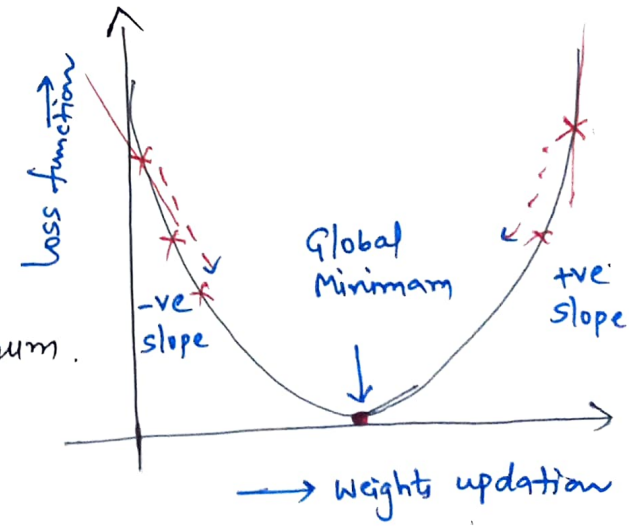
η → learning rate.

$\frac{dL}{dW}$ → derivative of loss w.r.to the weight.

The Commonly used optimizer is "Gradient descent".

In weights updation, the derivative is to find the slope.

- learning rate decides the rate of decrease/increase in loss.
- The weights updation is repeated until we reach global minimum.



if $\frac{\partial L}{\partial w}$ is -ve $\Rightarrow$ $w_{new} = w_{old} - \eta * (-ve\ value)$

(-ve slope) = Adds weight to old weight

if $\frac{\partial L}{\partial w}$ is +ve $\Rightarrow$ $w_{new} = w_{old} - \eta * (+ve\ value)$

(+ve slope) = Subtracts weight to old weight

The Epochs are continued untill we Reach Global minimum.

Loss and Cost Functions

A cost function is a measure of "how good" a neural network did with respect to its given training sample and the expected output.

A cost function is a single value it rates how good the neural network did as a whole.

A cost function depends on variables such as weights and biases.

A cost function is the average loss over the entire training dataset. It is the error representation. It shows how our model is predicting compared to the actual values.

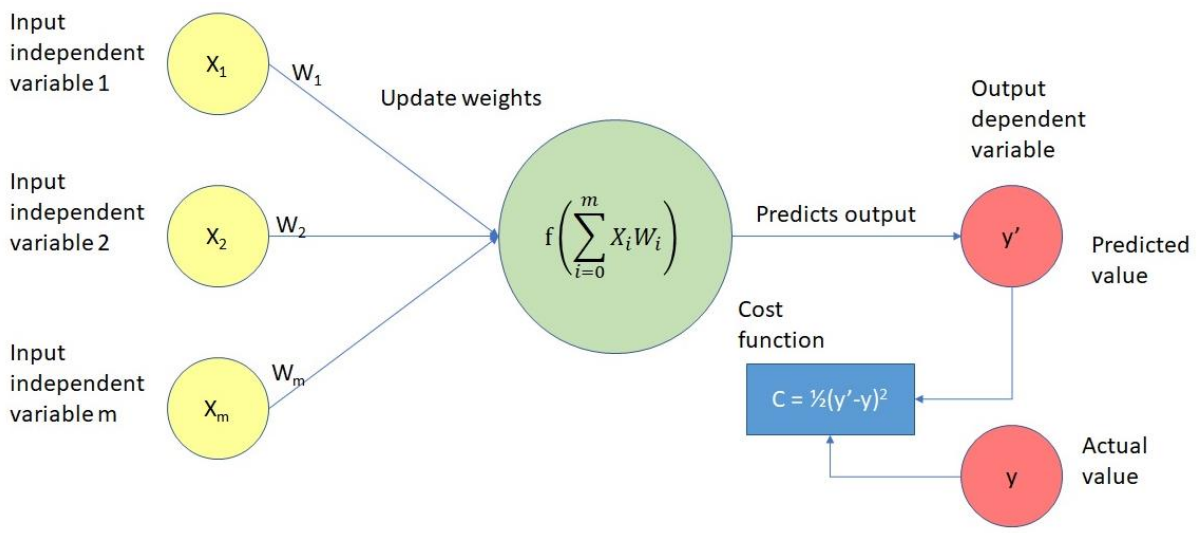
The optimization strategies aim at minimizing the cost function.

Lesser the cost function value then more will be the accuracy.

Loss Function- Loss function or error function is for a single training example or dataset.

Cost Function- Cost function is the average loss over the entire training dataset.

A cost function is selected based on the problem selected for training.



The various Cost functions are

1. Mean Absolute Error:

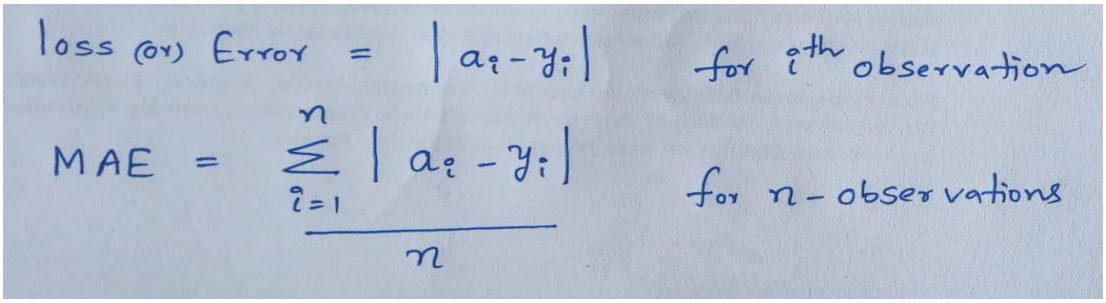
It is a measure of errors between paired observations.

Example: a_i -> Predicted output

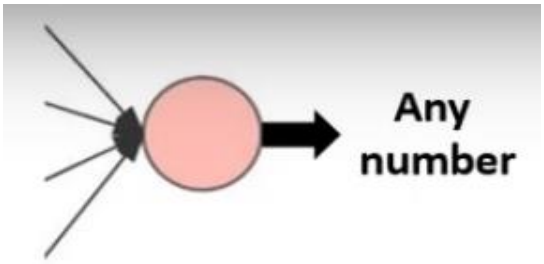
y_i -> Actual or Observed Output

n -> number of observations / data inputs

For a_i vs y_i include comparisons of predicted and observed values.



- **Used for Regression or Linear Regression Models. Here the output can take any number. Eg: House Price Prediction.**



2. **Mean Squared Error:** **Measures the average of the squares of the errors. I.e. The average squared differences between the estimated and the actual values. Also called Mean Squared Deviation.**

Used for Regression or Linear Regression Models. Here the output can take any number.

Eg: House Price Prediction.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}|a_i - y_i|^2 \quad \text{for } n \text{ observations}$$

(or)

$$MSE = \frac{1}{2n}\sum_{i=1}^{n}|a_i - y_i|^2 \quad \text{for } n \text{ observations}$$

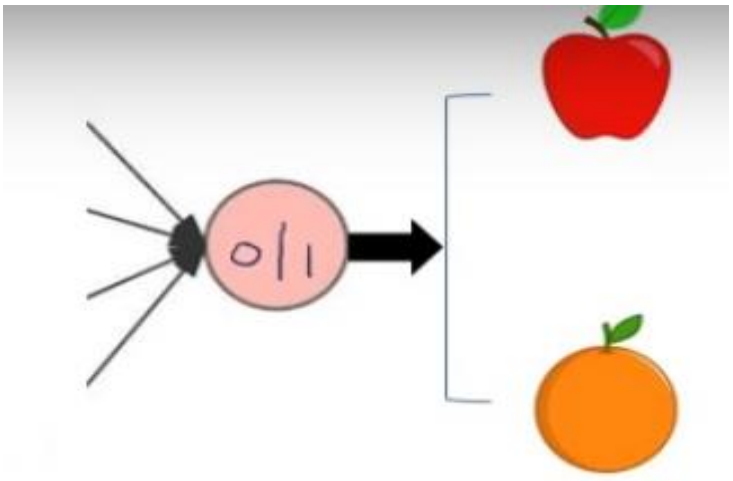
$$\text{Loss or Error} = |a_i - y_i|^2 \quad \text{for } i^{th} \text{ observation}$$

3. Binary Cross Entropy:

Compares each of the predicted probabilities to actual class output which can be either 0 or 1.Alo called log loss.

It is the negative average of the log of corrected predicted probabilities.

Used for binary classification.



$$loss = -\left[y_i \log(a_i) + (1-y_i)\log(1-a_i)\right]$$

for i^{th} observation.

$$Cost = \frac{-1}{n}\sum\left[y_i \log(a_i) + (1-y_i)\log(1-a_i)\right]$$

for n observations.

If y_i=0 => error = -log(1-a_i)

When ai approaches to 1, then high error,

When ai close to 0, then less error

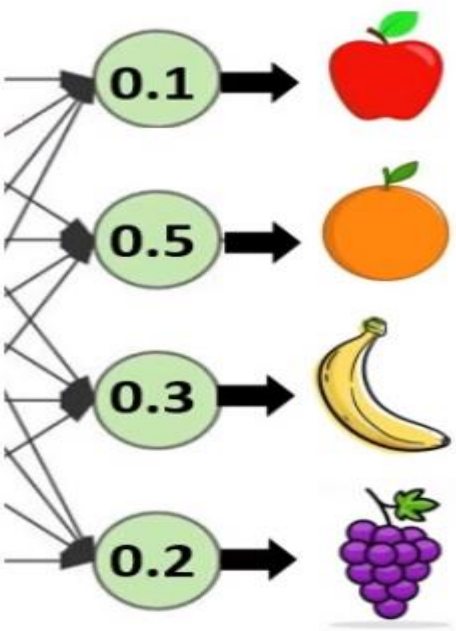
If yi=1=> error =-log(ai)

When ai approaches to 1 , then less error,

When ai approaches to 0, then high error.

4. Categorical Cross Entropy:

It is used for multi class classification.



In the above example, for each observation one-hot representation is used for a_i and y_i.

i.e. $y_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $a_i = \begin{bmatrix} 0.1 \\ 0.5 \\ 0.3 \\ 0.2 \end{bmatrix}$ for 4 output neurons.

$$\text{loss} = -\left[y_{i1} \cdot \log(a_{i1}) + y_{i2} \log(a_{i2}) + y_{i3} \log(a_{i3}) + y_{i4} \log(a_{i4}) \right]$$

$$= -\left[0 \times \log(0.1) + 1 \times \log(0.5) + 0 \times \log(0.3) + 0 \times \log(0.2) \right]$$

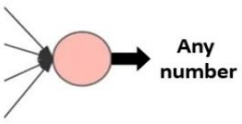
$$= -\log(0.5).$$

For each observation y_i has 1 in one-hot representation for one neuron.

For n observations with m-output neurons

$$\text{Cost} = \frac{-1}{n} \sum_{i=0}^{n} \sum_{j=0}^{m} \left(y_{ij} * \log(a_{ij}) \right).$$

1.) Regression

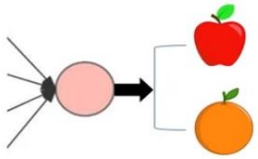


$$\text{Cost} = \frac{1}{m} \sum_{i=1}^{m} |a_i - y_i|$$

or

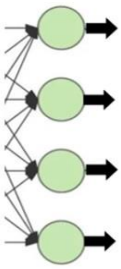
$$\text{Cost} = \frac{1}{2m} \sum_{i=1}^{m} [a_i - y_i]^2$$

2.) Binary Classification



$$\text{Cost} = -\frac{1}{m} \sum_{i=1}^{m} [y_i * log(a_i) + (1 - y_i) * log(1 - a_i)]$$

3.) Multi-class classification



$$\text{cost} = -\sum_{j=0}^{M} \sum_{i=0}^{N} (y_{ij} * log(a_{ij}))$$

OPTIMIZERS

Optimization is a technique which speedup the training / learning of a model in deep learning.

Optimizers are algorithms or methods used to minimize an error function (loss function) or to maximize the efficiency of production.

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.

Optimizers are mathematical functions which are dependent on model's learnable parameters i.e Weights & Biases.

The change of weights or learning rates of neural network to reduce the losses is defined by the optimizers.

Optimization algorithms or strategies are responsible for reducing the losses and to provide the most accurate results possible.

Optimizers commonly used are - Gradient Descent, Stochastic Gradient Descent, Mini-Batch Gradient Descent, Adagrad, AdaDelta, Adam.

The various Types of optimizers are

1. Gradient Descent:

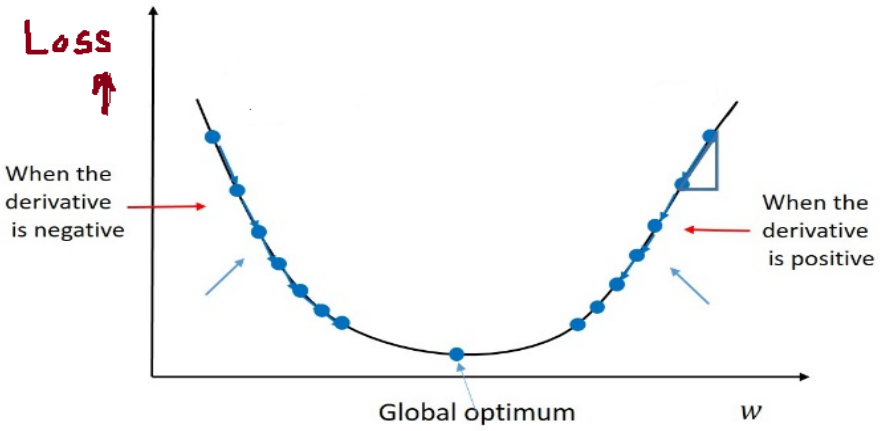
- **Gradient Descent is the most basic and mostly used optimization algorithm.**
- **Gradient descent is an optimization algorithm based on a convex function and adjust its parameters iteratively to minimize a given function to its local minimum.**
- **It is used heavily in linear regression and classification algorithms.**
- **Backpropagation in neural networks also uses a gradient descent algorithm.**

- **Gradient Descent iteratively reduces a loss function by moving in the direction opposite to that of steepest ascent.**
- **Gradient descent is a first-order optimization algorithm which is dependent on the first order derivative of a loss function. It calculates that which way the weights should be altered so that the function can reach minima.**
- **It is dependent on the derivatives of the loss function for finding minima.**
- **All the data points (records) are used at a time (cost function).**

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$

Here η — is learning rate

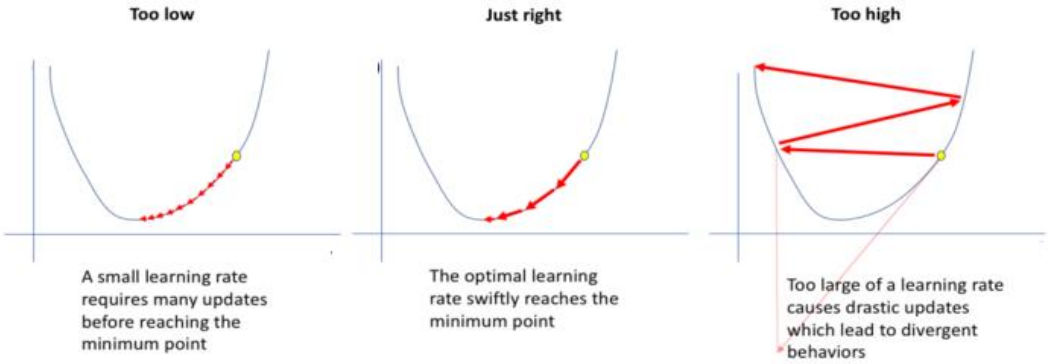
Also the symbol 'α' is used for learning rate.



if $\frac{\partial L}{\partial w}$ is -ve $\Rightarrow$ $w_{new} = w_{old} - \eta * (-ve\ value)$
(-ve slope) $= Adds\ weight\ to\ old\ weight$

if $\frac{\partial L}{\partial w}$ is +ve $\Rightarrow$ $w_{new} = w_{old} - \eta * (+ve\ value)$
(+ve slope) $= Subtracts\ weight\ to\ old\ weight$

- **Uses the data of the entire training set to calculate the gradient of the cost function to the parameters which requires large amount of memory and slows down the process.**
- **How big/small the steps are gradient descent takes into the direction of the local minimum are determined by the learning rate, which figures out how fast or slow we will move towards the optimal weights.**



Advantages:

- **Easy to understand**
- **Easy to implement**
- **Easy for Computation**

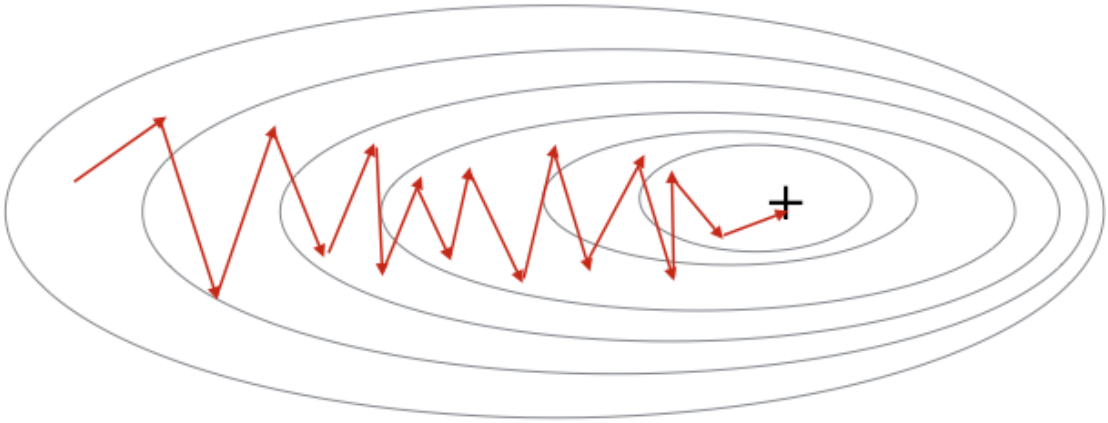
Disadvantages:

- **Because this method calculates the gradient for the entire data set in one update, the calculation is very slow. So, if the dataset is too large then this may take years to converge to the minima.**

- **It requires large memory and it is computationally expensive.**

2. Stochastic Gradient Descent (SGD)

- **It is a variant of Gradient Descent. It updates the model parameters one by one.**
- **In this, the model parameters are altered after computation of loss on each training example.**
- **Here only one data point (record) is considered at a time for weight updation.**
- **Linear regression uses SGD optimizer.**
- **If the model has 10K dataset SGD will update the model parameters 10k times instead of one time as in Gradient Descent.**
- **As the model parameters are frequently updated, parameters have high variance and fluctuations in loss functions at different intensities.**



Advantages:

1. Frequent updates of model parameter

2. Requires less memory as no need to store values of loss functions.

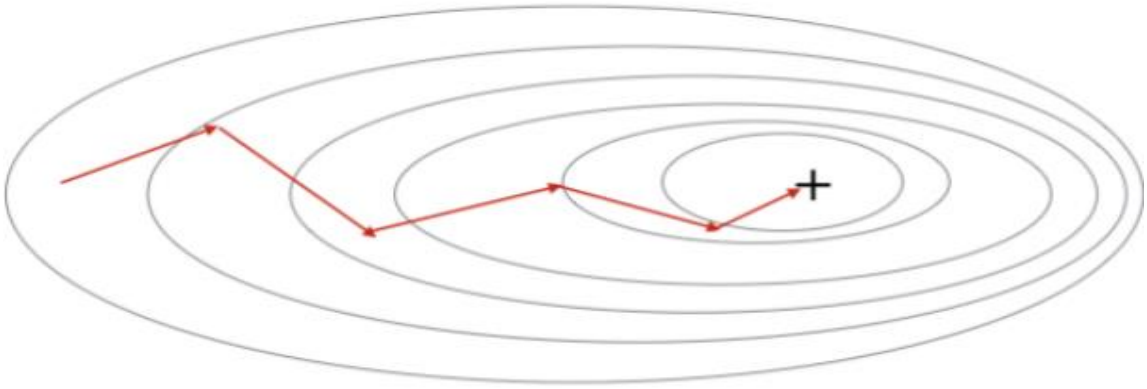
3. Allows the use of large data sets as it has to update only one record at a time.

Disadvantages:

1. The frequent can also result in noisy gradients which may cause the error to increase instead of decreasing it.
2. High Variance.
3. Frequent updates are computationally expensive.

3. Mini-Batch Gradient Descent

- It's best among all the variations of gradient descent algorithms. Most of the neural networks uses this model.
- It is an improvement on both SGD and standard gradient descent.
- Here K data points (records) are used at a time for weight updating.
- It updates the model parameters after every batch. So, the dataset is divided into various batches and after every batch, the parameters are updated.
- It simply splits the training dataset into small batches and performs an update of weights for each of those batches.
- This creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.
- It can reduce the variance when the parameters are updated, and the convergence is more stable.
- It splits the data set into batches in between 50 to 256 records, chosen at random.



Advantages:

- **It leads to more stable convergence.**
- **More efficient gradient calculations.**
- **Requires less amount of memory.**
- **Frequently updates the model parameters and also has less variance.**

Disadvantages:

- **Mini-batch gradient descent does not guarantee good convergence,**
- **If the learning rate is too small, the convergence rate will be slow. If it is too large, the loss function will oscillate or even deviate at the minimum value.**

4. SGD with Momentum:

Momentum was invented for reducing high variance in SGD and softens the convergence.

It accelerates the convergence towards the relevant direction and reduces the fluctuation to the irrelevant direction.

Momentum simulates the inertia of an object when it is moving, that is, the direction of the previous update is retained to a

certain extent during the update, while the current update gradient is used to fine-tune the final update direction.

One more hyper parameter is used in this method known as momentum symbolized by 'γ'.

$$V(t) = \gamma V(t-1) - \eta \frac{\partial L}{\partial W_{old}}$$

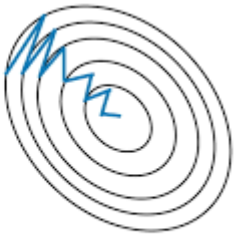
$$W_{new} = W_{old} - V(t)$$

The momentum term 'γ' is usually set to 0.9 or a similar value.

It is s a stochastic optimization method that adds a momentum term to regular stochastic gradient descent.

In this way, we can increase the stability to a certain extent, so that we can learn faster.

Stochastic Gradient Descent **without** Momentum



Stochastic Gradient Descent **with** Momentum

Advantages:

- Momentum helps to reduce the noise. Reduces the oscillations and high variance of the parameters.
- Converges faster than gradient descent.
- Exponential Weighted Average is used to smoothen the curve.

Disadvantages:

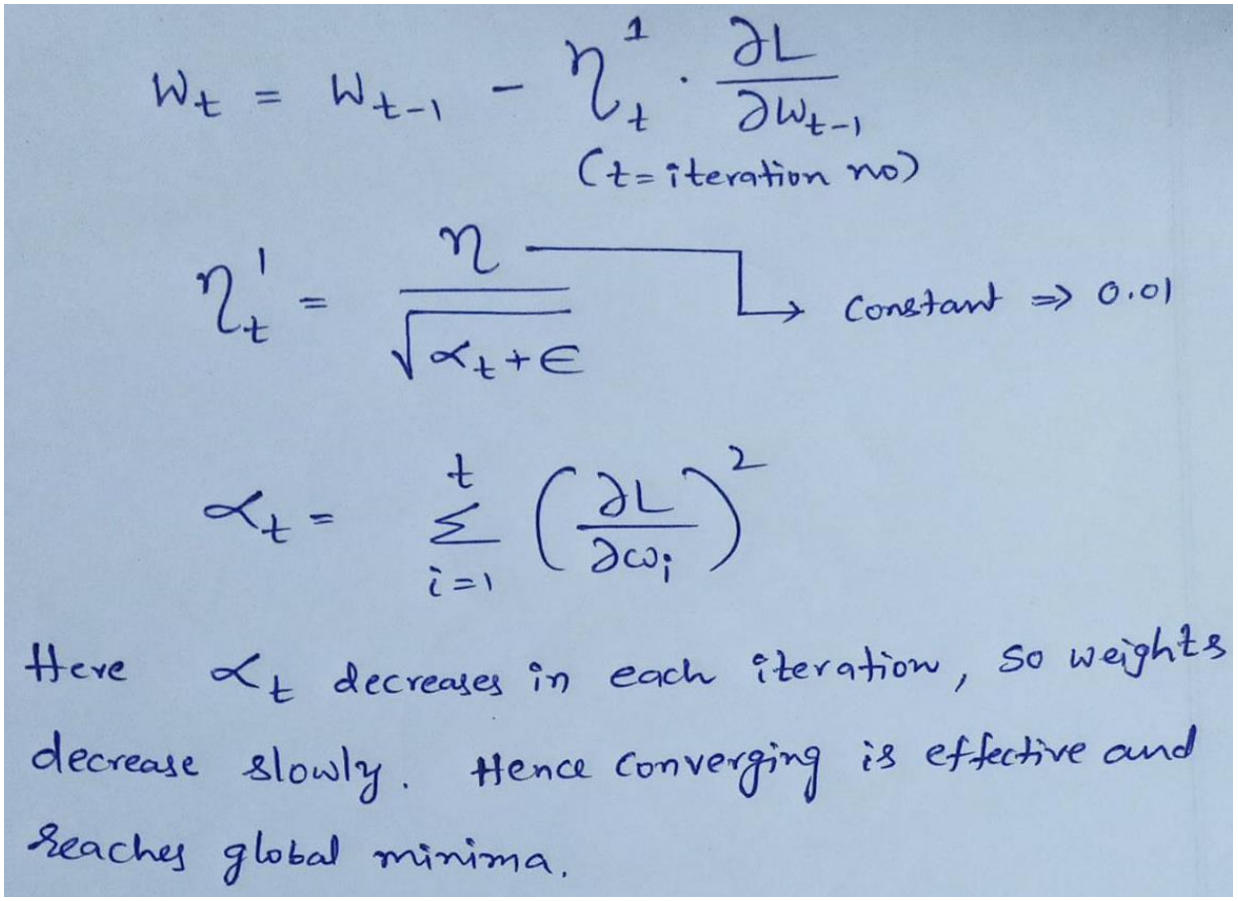
- **Extra hyper parameter is added, which needs to be selected manually and accurately.**

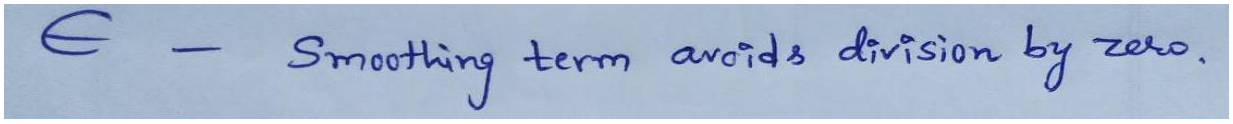
All types of Gradient Descent have some challenges:

- **Choosing an optimum value of the learning rate. If the learning rate is too small than gradient descent may take ages to converge.**
- **Have a constant learning rate for all the parameters. There may be some parameters which we may not want to change at the same rate.**
- **May get trapped at local minima.**

5. Adagrad: (Adaptive Gradient Descent)

- **In all the above optimizers GD, SGD, Mini Batch GD and SGD with Momentum uses the constant learning rate for all parameters and for each cycle.**
- **Adagard optimizer changes the learning rate. It changes the learning rate 'η' for each parameter and at every time step 't'.**
- **It works on the derivative of an error function.**
- **The intuition behind AdaGrad is we can use different Learning Rates for each and every neuron for each and every hidden layer based on different iterations.**





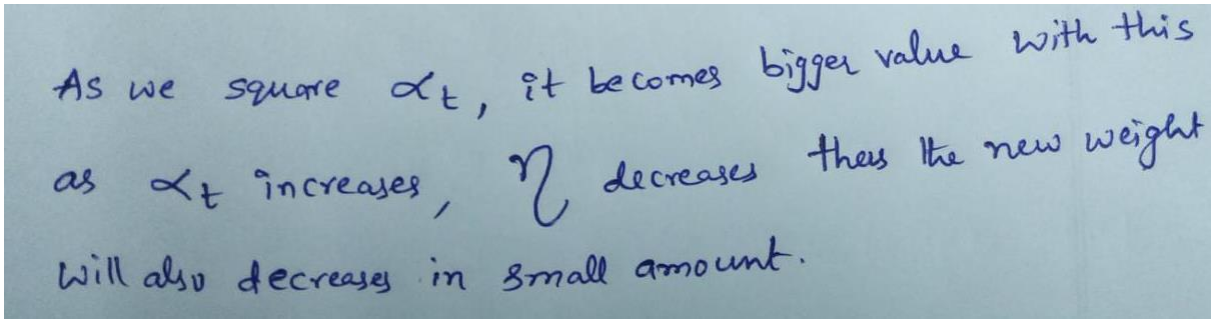
Advantages:

1. **Learning rate changes for each training parameter.**
2. **Don't need to manually tune the learning rate.**
3. **Able to train on sparse data (contains 1s and 0s). (high percentage of data having zeros rather than no actual values)**

Disadvantages:

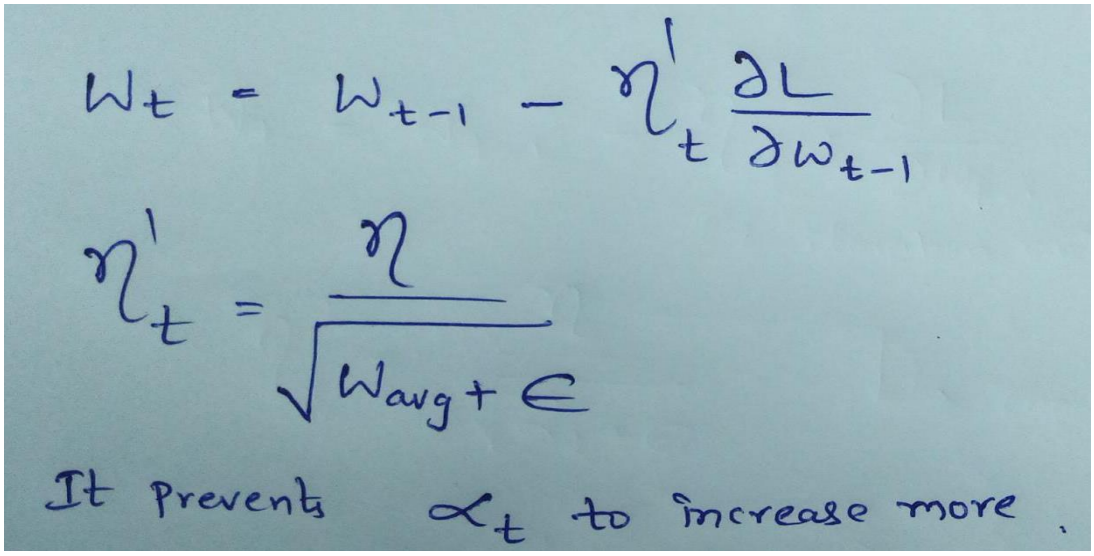
1. **Computationally expensive as a need to calculate the second order derivative.**
2. **The learning rate is always decreasing, results in slow training.**
3. **If the neural network is deep the learning rate becomes very small number, which will cause dead neuron problem.**

4. **Need to set the default learning rate first.**



6. AdaDelta:

- **Adadelta is an extension of Adagrad and it also tries to reduce Adagrad's aggressive, monotonically reducing the learning rate and remove decaying learning rate problem.**
- **In Adadelta we do not need to set the default learning rate as we take the ratio of the running average of the previous time steps to the current gradient.**



Advantages:

1. **Now the learning rate does not decay and the training does not stop.**

Disadvantages:

1. **Computationally expensive.**

7. Adam (Adaptive Moment Estimation)

- **Adam optimizer is one of the most popular and famous gradient descent optimization algorithms.**
- **It is a method that computes adaptive learning rates for each parameter.**
- **Works with momentums of first and second order. The intuition behind the Adam is that we don't want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for a careful search.**
- **Updates both bias and weights.**

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{S_{dw_t} - \varepsilon}} * V_{dw_t}$$

$$b_t = b_{t-1} - \frac{\eta}{\sqrt{S_{db_t} - \varepsilon}} * V_{db_t}$$

8. RMSprop Optimizer

The RMSprop optimizer is similar to the gradient descent algorithm with momentum.

The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster.

The following equations show how the gradients are calculated for the RMSprop and gradient descent with momentum. The value of momentum is denoted by beta and is usually set to 0.9.

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db$$

$$W = W - \alpha \cdot v_{dw}$$

$$b = b - \alpha \cdot v_{db}$$

Gradient descent with momenttum

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

RMSprop optimizer