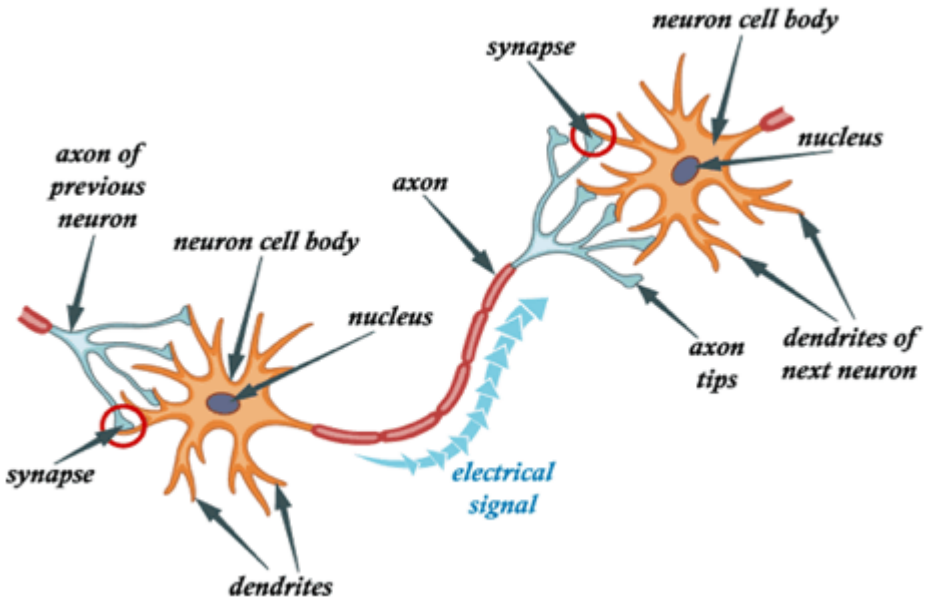
Deep Learning is the human brain embedded in machines. It is one of the most robust branches of Machine Learning. Also, known as **Deep Structured Learning** and **Hierarchical Learning**, it can be thought of as the simplest way to automate predictive analysis.

At a basic level, deep learning models are multilayer neural networks.

Now let's understand What is neural network ?

Neural networks form the base of deep learning, which is a subfield of **machine learning**, where the structure of the human brain inspires the algorithms.

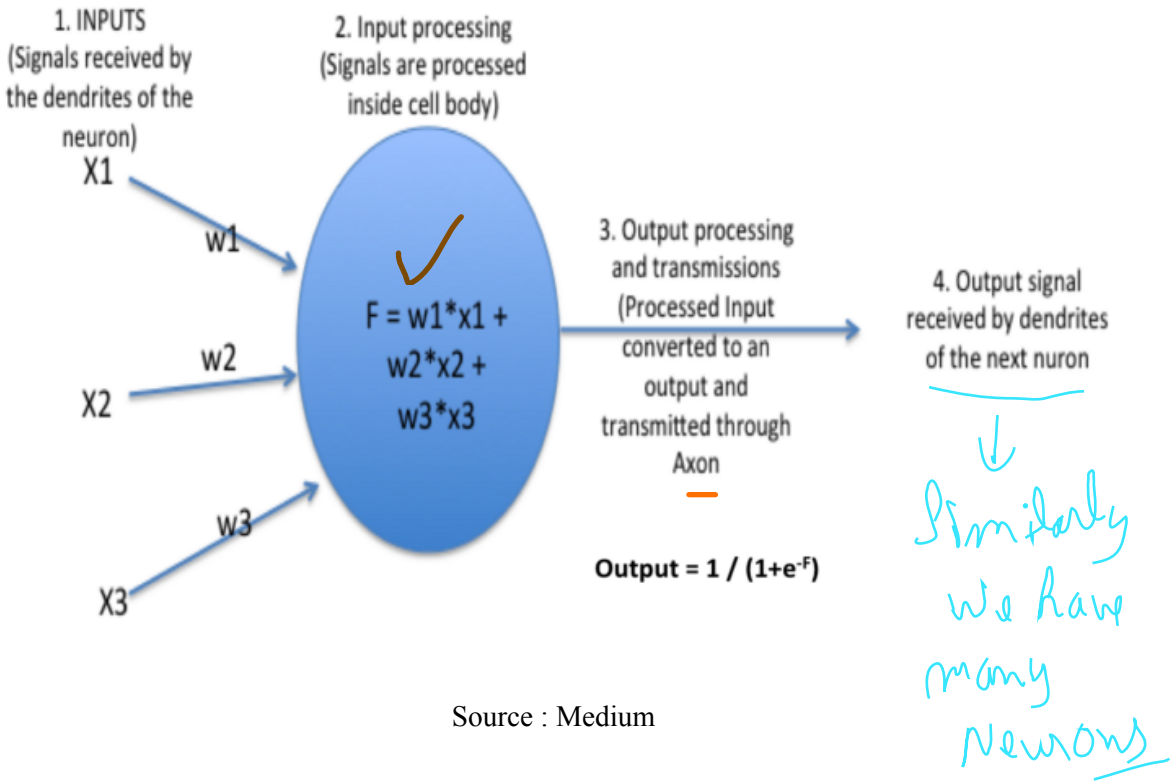
Neural nets are inspired by our brains. Let's understand how ?



Step 1: External signal received by dendrites	**Step 2**: External signal processed in the neuron cell body	**Step 3**: Processed signal converted to an output signal and transmitted through the Axon	**Step 4**: Output signal received by the dendrites of the next neuron through the synapse

Source : Medium

Let's now understand how Neural Networks works.

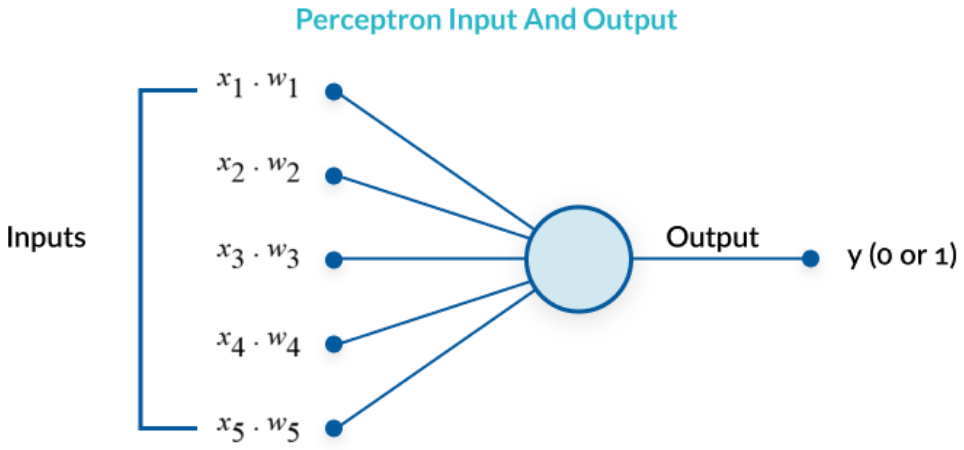


Source : Medium

As you can see from the above, an ANN is a very simplistic representation of how a brain neuron works.

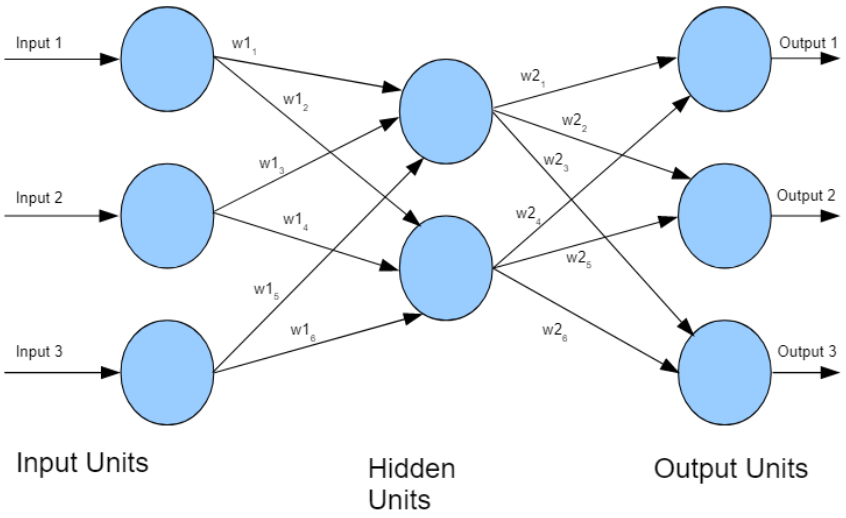
Perceptron :

The *perceptron* model, proposed by Minsky-Papert, is a neural network without any hidden layer i.e it is a single layer neural network. A perceptron only has an input layer and an output layer.



Multilayer Perceptron (Artificial Neural Network)

A multilayer perceptron (MLP) is a group of perceptrons, organized in multiple layers, that can accurately answer complex questions. Each perceptron in the first layer (on the left) sends signals to all the perceptrons in the second layer, and so on. An MLP contains an input layer, at least one hidden layer, and an output layer.



The Multi layer perceptron consists of :

- Input Layer
- Hidden Layers
- Activation Function
- Output Layer

Let's discuss each one of them in detail .

Input Layer : **Input units** are designed to receive various forms of information from the outside world that the network will attempt to learn about, recognize, or otherwise process and are together referred to as the "Input Layer". No computation is performed in any of the Input nodes they just pass on the information to the hidden nodes.

Hidden Layers : The Hidden nodes have no direct connection with the outside world (hence the name "hidden"). They perform computations and transfer information from the input nodes to

the output nodes. A collection of hidden nodes forms a "Hidden Layer".

Activation Function : 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.

Output layer : The Output nodes are collectively referred to as the "Output Layer" and are responsible for computations and transferring information from the network to the outside world.

Working of the Neural Network

1. Input units are passed i.e data is passed to the hidden layer . We can have any number of hidden layers .
2. Each hidden layer consists of neurons . All the inputs are connected to each neuron .
3. After passing on the inputs , all the computation is performed in the hidden layer .

   Computation performed in hidden layers are done in two steps which are as follows :

   - First of all , all the inputs are multiplied with their weights . Weight is the gradient or coefficient of each variable . It shows the strength of the particular input . After assigning the weights , a bias variable is added . Bias is a constant that helps the model to fit in the best way possible

     Z1 = W1*In1 + W2*In2 + W3*In3 + W4*In4 + W5*In5 + Bias

   - Then in the second step , **Activation function** is applied to the linear equation Z1 . The activation function is a nonlinear transformation that is applied to the input before sending it to the next layer of neurons . The importance of activation function is to inculcate non linearity in the model .
     There are several activation functions which will be discussed a little later .

4. The whole process described in point 3 is performed in each hidden layer . After passing through every hidden layer , we move to the last layer i.e our output layer .

This whole process explained above is known as Forward Propagation .

5. After getting the predictions from the output layer , the error is calculated i.e the difference between the actual and the predicted output . If the error is large , then the steps are taken to minimize the error and for the same purpose **Back Propagation** is performed.

   Now let's understand Back Propagation in detail .

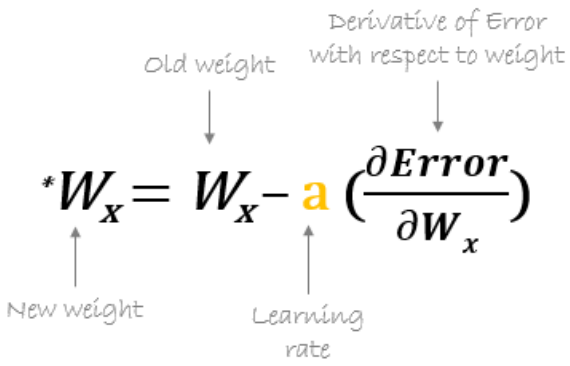
   What is Back Propagation and How it works ?
   As discussed previously, we need to minimize the error to get the optimal predictions . For this purpose Backpropagation can discover the optimal weights which helps in minimizing

the error .

There are various **optimizers** to reduce the error . Optimizers are the methods to change the attributes of a neural network such as weights to reduce the error .

Back Propagation with Gradient Descent

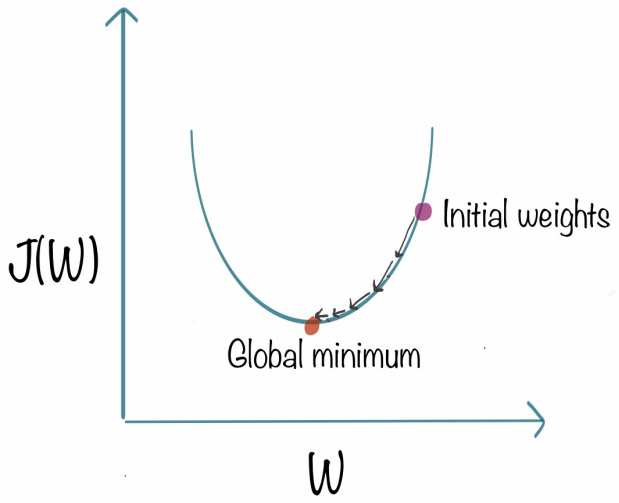
Gradient Descent is one of the optimizers to update weights and reduce error .



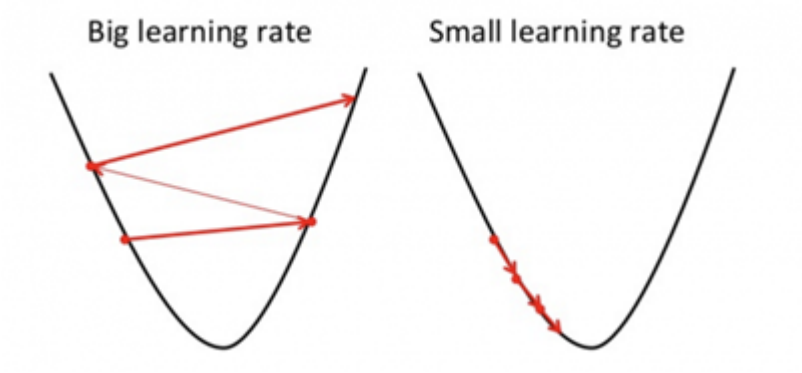
In the above equation , L.H.S is the new weight i.e updated weight and on the RHS is the old weight minus derivative of error multiplied by learning rate .

This is how Gradient Descent works for optimizing loss

- First , weights are initialized randomly i.r random value of weight and intercepts are assigned to the model
- Then the gradient will be calculated i.e derivative of error w.r.t weights
- After that weights are updated using the above formula .
- This process continues till we reach global minima and loss is minimized .



A point to note here is that , learning rate i.e a in our weight updation equation should be chosen wisely . Learning rate is the amount of change or step size taken towards reaching global minima. It should not be very small as it will take time to converge as well as it should not be very large that it doesn't reach global minima at all .



Some drawbacks of Gradient Descent are :

- It's slow as it trains on the whole dataset before updating weights and if the dataset is large it is not advised to use this optimizer .
- It will occupy lots of memory if the dataset is huge .

There are various other better optimizers to minimize losses . We will discuss all of them in sometime .
First , let's learn about Activation functions .

Activation Functions

Activation functions are attached to each neuron and are mathematical equations which determine whether a neuron should be activated or not based on whether the neuron's input is relevant for the model's prediction or not .

Various Types of Activation Functions

1. Sigmoid Activation Function :

The sigmoid AF is a S shaped curve and its equation is

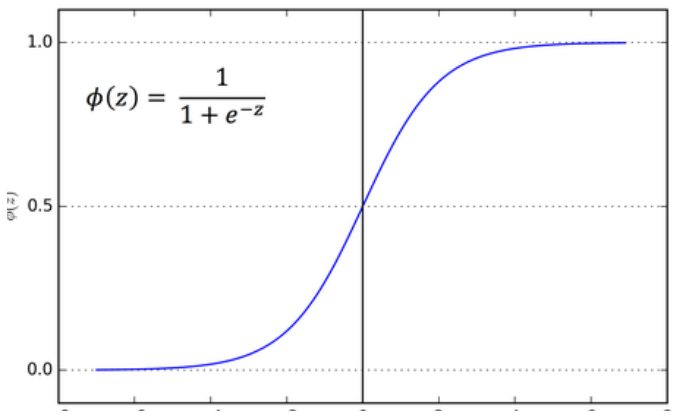
$$f(X) = \frac{1}{1 + e^{-X}}$$

Where,

$f(X)$ = *output value between* 0 *and* 1.

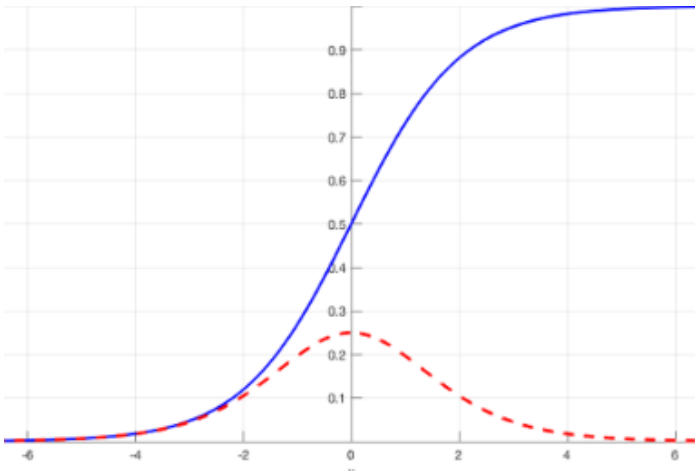
e = *mathematical constant* (2.71828).

X = *input values.*



The value of the Sigmoid AF lies between 0 to 1 , that is why this is most used for the problems where we need to predict probabilities .

The derivative of this function is $f'(x) = f(x)\,(1 - f(x))$ and it lies between 0 to 0.25 .
Below graph blue curve represents Sigmoid AF and red curve represents the derivative of Sigmoid AF.

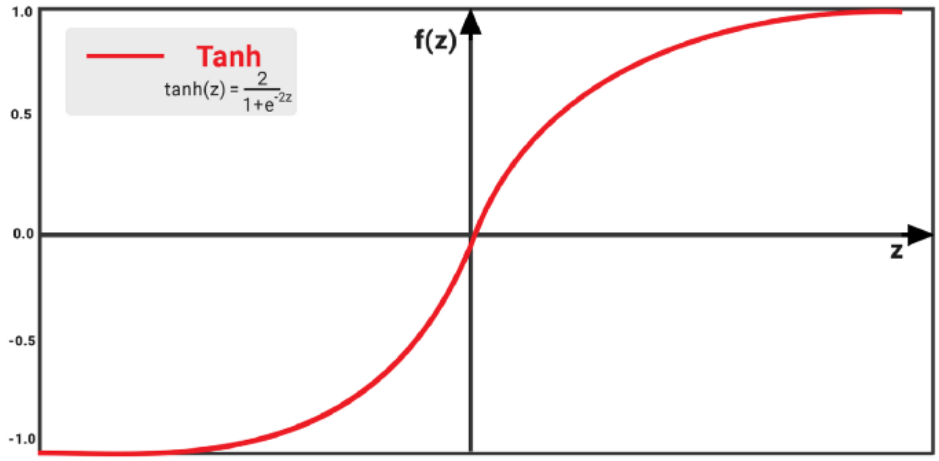


Disadvantage of Sigmoid AF :

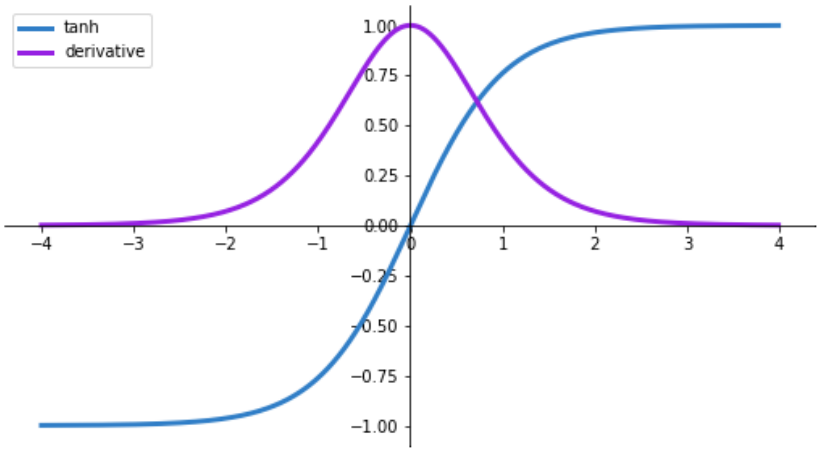
- Vanishing Gradient problem : In this problem , the difference between the old weights and new weights becomes so small that new weight does not change any further . For this reason , Sigmoid Activation Function is not used in Hidden layers and mostly used in the output layer .

2. TanH / Hyperbolic Tangent Activation Function

TanH AF is an improved version of Sigmoid Activation Function . The value of TanH AF lies between -1 and 1 and it is also S shaped .



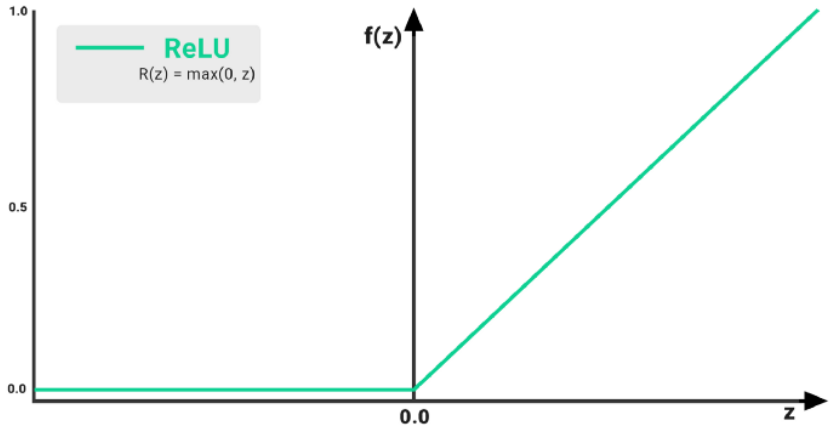
The derivative of this function is f ’(x) = 1 - f(x)² and lies between 0 to 1 .
Below graph represents the TanH and it’s derivative curve .



The disadvantage of this function is the same as the Sigmoid function . It also deals with the Vanishing Gradient Problem .

3. Rectified Linear Unit Function (ReLU)

ReLU function is the most used Activation function . The equation of ReLU is **f(z)=max(0,z)** which implies that if the function receives any negative or zero input , the output will be zero and if the input is any positive value then output will also be that same value .

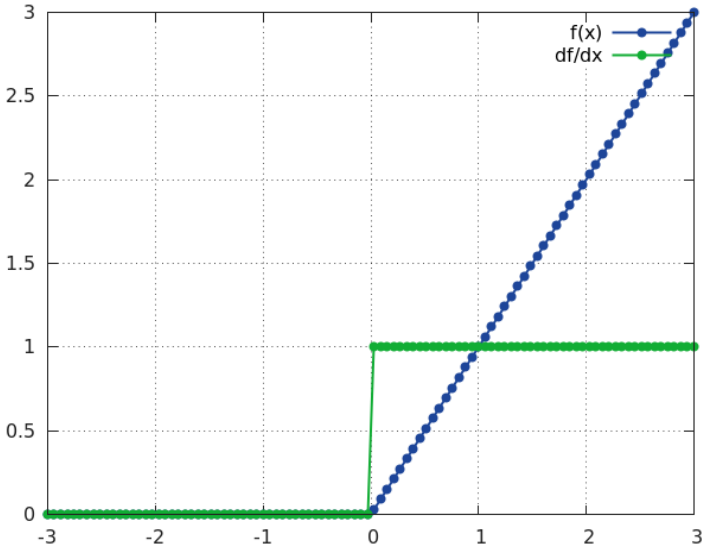


The derivative of this function is 0 if input value is 0 and 1 if input value is positive .
Value of derivative :

0 if z<0

1 if z>0

Below graph represents the ReLU activation function and it's derivative curve .



Advantages :

- It resolves the problem of Vanishing gradient descent as its derivative value is always constant i.e 0 or 1 .
- It is less computationally inexpensive than Sigmoid and TanH as the mathematical calculations in ReLU are much more simpler and can be used while training deep neural networks .

Disadvantage :

- It suffers from the problem of dead neurons or dying ReLU due to the derivative value of 0 . Derivative value 0 leads to the overall gradient value to 0 which leads to failure of back propagation process and the function fails to update weights .

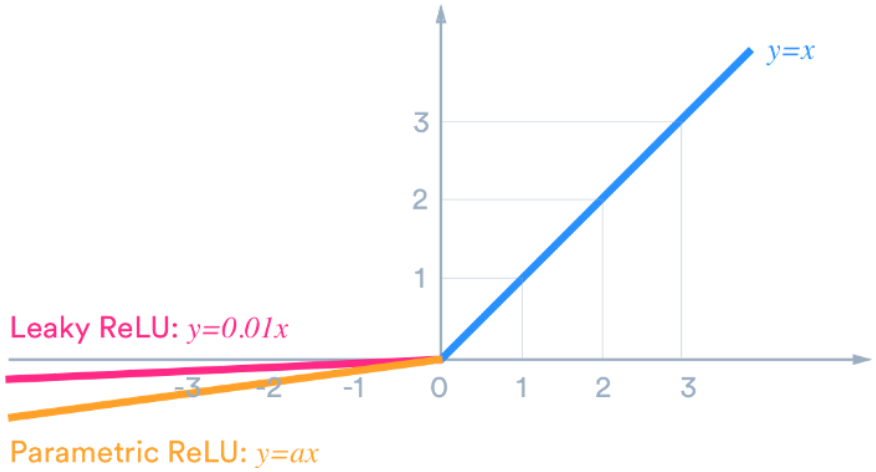
To solve the dying ReLU problem , we have a modified version named as Leaky ReLU .

4. Leaky ReLU

The equation of Leaky ReLU is :

y = 0.01x if x <0
x if x>0

To solve the problem of dying ReLU , this function multiplies a constant value of 0.01 to the negative input and adds a small slope for negative values instead of making it zero .



The derivative of Leaky ReLU lies between

0.01 if x<0
1 if x>0

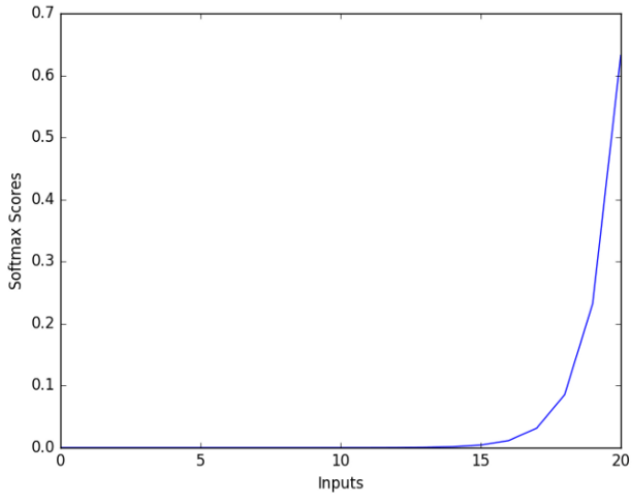
This implies the derivative value will never be zero and the dead neuron problem will be solved .

5. Soft max

Soft max Activation Function can be used for multi classification problems as it not only maps probability values to 0 or 1 but also considers other probability values such that the sum of all probabilities is 1.

The equation of Soft max function is :

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$



What are drop outs in deep learning ?

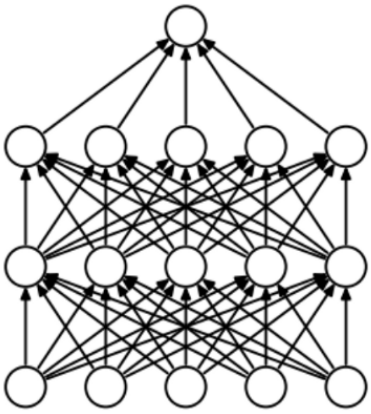
As we use L1 and L2 regularization techniques to deal with the problem of overfitting in Machine Learning , in the same way **Dropout is a regularization technique** in Deep Learning to solve overfitting problems .

There are various neurons and weights involved in the Deep Neural Network which usually leads to the problem of Overfitting . To solve this problem , we have to control the depth of the Neural Network as we do for Decision Trees in Machine Learning and we use the Dropout technique for the same .

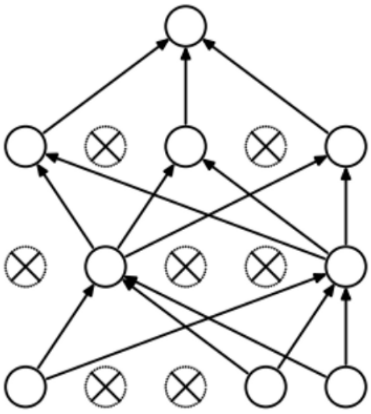
How does it work ?

- In forward propagation we select a **p value** (which is also known as drop out ratio and lies between 0 to 1), then based on the p value our model randomly deactivates some neurons from the first layer and pass the activated neurons to the hidden layer .
- Again a p value is selected in the hidden layer and randomly some neurons are deactivated and activated neurons are passed onto the next layer .
- This process continues till the output is achieved .
- In the testing phase , all the activations are used but they are reduced by the factor p i.e drop out value .

In this technique , Backward Propagation will be the same i.e the weights are updated for the neurons which are activated .



(a) Standard Neural Net

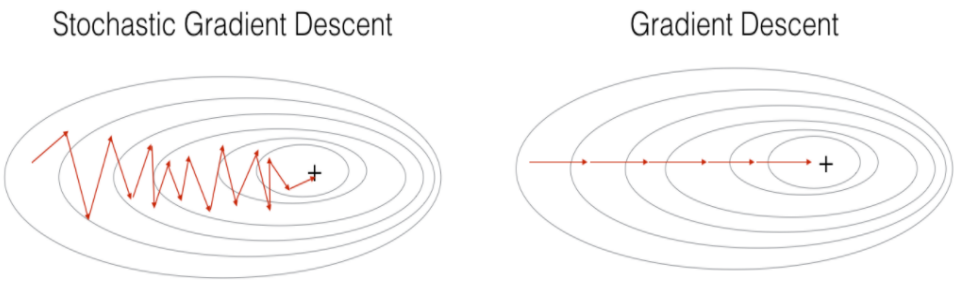


(b) After applying dropout.

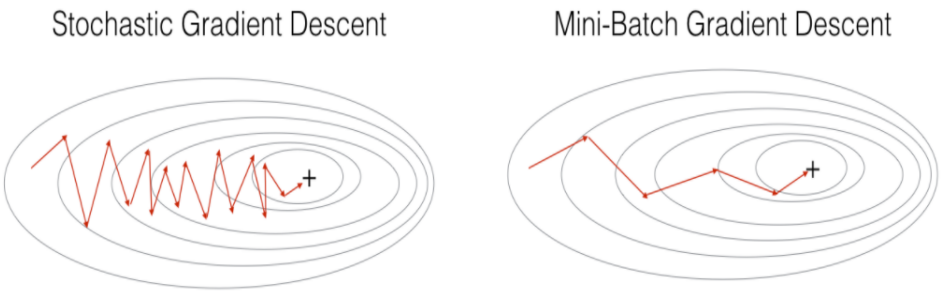
Optimizers for minimizing loss

Other than Gradient Descent optimizer which we discussed above we have many other optimizers that help to minimize losses much efficiently . Let's discuss them in detail .

1. Stochastic Gradient Descent (SGD)
   It is one of the alternatives to the Gradient Descent Optimizer . In this method instead of whole data , a simple datapoint is taken to and gradient is calculated for each data point and weights are updated with every iteration . It updates the model's coefficients more frequently than Gradient Descent.
   For example , if our dataset consists of 15000 rows , SGD will update the coefficients 1500 times unlike Gradient Descent which does it one time .



2. Mini Batch Gradient Descent
   It is another alternative to Gradient Descent . In this optimizer , our dataset is divided into batches and weights are updated after every batch .
   Suppose , our dataset has 1500 data points and it is divided into 5 batches , each batch having 300 data points . So weights will be updated 5 times equal to the number of the batches our dataset is divided into .



3. AdaGrad (Adaptive Gradient Algorithm)
   This optimizer is a little different from other optimizers in the sense that AdaGrad optimizer use different learning rate i.e it proposed different learning rates for each and every neuron and hidden layer based on each iteration in the weight updation process . AdaGrad denotes different learning rates to Dense and Sparse features . Dense features are those in which most of the values are non-zero and Sparse features are those in which most of the feature values are zero .

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} \cdot g_t$$

G_t is sum of the squares of the past gradients w.r.t. to all parameters θ

Above is the weight updation equation where ,

θ_{t+1} - is the new weight

θ_t - old weight

$\frac{\eta}{\sqrt{G_t + \varepsilon}}$ - learning rate for different time intervals

g_t - derivative of error w.r.t weight

Advantages :

- It is well suited for sparse data
- We don't have to manually train the learning rate

Disadvantages :

- In the denominator, the sum of the squares of the past gradients are accumulating . Each term is a positive so it keeps on growing and the learning becomes small to the point that algorithms can no longer learn.
  This disadvantage is resolved by other optimizers such as AdaDelta , Adam

4. AdaDelta
   This optimizer is the improved version of AdaGrad which tries to remove the problem of reducing learning rate . It is done by reducing the previous accumulated gradients to a fixed size say w , instead of considering all the past gradients .

$$\mathbb{E}[g^2]_t = \gamma \mathbb{E}[g^2]_{t-1} + (1-\gamma) g_t^2,$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\mathbb{E}[g^2]_t + \epsilon}} \cdot g_t.$$

In AdaDelta , **an exponentially moving average** of gradients is used rather than the sum of the gradients.
Advantage of the optimizer is that it solves the problem of decaying learning rate and disadvantage is that it is computationally expensive.

5. RMSProp
   RMSProp is Root Mean Square Propagation and it helps to solve the problem of AdaGrad by using a **moving average of squared gradient** . RMSProp also automatically adjusts the learning rate and chooses a different learning rate for each parameter .

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{(1-\gamma) g^2{}_{t-1} + \gamma g_t + \varepsilon}} \cdot g_t$$

6. Adam-

Adam is generally considered as the best optimizer if someone wants to train the neural network in less time with more efficiency. Adam converges faster than any other optimizers and the hyper parameter requires very less tuning. It requires little memory space as well as works well with the large datasets and large parameters.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \rightarrow (\text{mean}),$$

$$v_t = \beta_1 v_{t-1} + (1 - \beta_2) g_t^2 \rightarrow (\text{uncentered variance}),$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t},$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t},$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t,$$

where $\beta 1 = 0.9, \beta 2 = 0.999, \varepsilon = 10{-}8.$

Case Study of ANN

- Import Data -

  The dataset is imported from excel file. To import this we are using Pandas Library.

```
#Used Panda Library to Load data into a dataframe from the Excel File
import pandas as pd
features = pd.read_excel('D:\\SLIIT\\4th Year\\ML\\Assignment\\default of credit card clients Data Set\\default of credit card c

#Display first 5 rows of the loaded dataset
features.head(5)
```

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X15	X16	X17	X18	X19	X20	X21	X22	X23	Y
1	20000	2	2	1	24	2	2	-1	-1	-2	...	0	0	0	0	689	0	0	0	0	1
2	120000	2	2	2	26	-1	2	0	0	0	...	3272	3455	3261	0	1000	1000	1000	0	2000	1
3	90000	2	2	2	34	0	0	0	0	0	...	14331	14948	15549	1518	1500	1000	1000	1000	5000	0
4	50000	2	2	1	37	0	0	0	0	0	...	28314	28959	29547	2000	2019	1200	1100	1069	1000	0
5	50000	1	2	1	57	-1	0	-1	0	0	...	20940	19146	19131	2000	36681	10000	9000	689	679	0

5 rows × 24 columns

- Count of target labels -

We use this to check the target labels that are present in the Dataset. This is classification task so the target labels will be in the form of 0 and 1. This shows that the dataset is imbalance and biased towards the negative class.

```
#Display the Count of each target label and it's proportion
label_count = features.Y.value_counts()
print('Class 0:', label_count[0])
print('Class 1:', label_count[1])
print('Proportion:', round(label_count[0] / label_count[1], 2), ': 1')
```

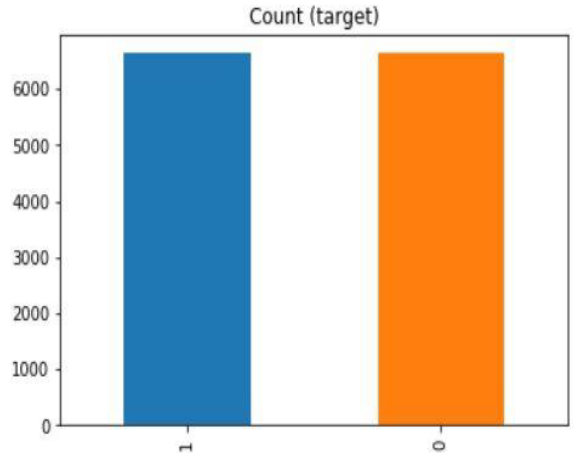
```
Class 0: 23364
Class 1: 6636
Proportion: 3.52 : 1
```

- Apply random under sampling -

To handle such imbalance datasets we need to use some technique which is called under sampling where we reduce the majority class labels and make the dataset balanced.

```
#Random undersampling to remove samples from majority class and equate the proporion of target labels

# count classes
count_class_0, count_class_1 = features.Y.value_counts()

# Divide by class and random undersampling
df_class_0 = features[features['Y'] == 0]
df_class_1 = features[features['Y'] == 1]

df_class_0_under = df_class_0.sample(count_class_1)
df_test_under = pd.concat([df_class_0_under, df_class_1], axis=0)

print('Random Under Sampling')
print(df_test_under.Y.value_counts())

df_test_under.Y.value_counts().plot(kind='bar', title='Count (target)');
```

```
Random Under Sampling
1    6636
0    6636
Name: Y, dtype: int64
```



- Split Labels and Features from the Dataset -

To perform Train and Test split we need to drop the labels from the original dataset and store it into another variable ‘y’. Remaining dataset is stored in the ‘X’ variable.

```
#Defining the feature sets into x variable and the series of corresponding labels into the y variable
y = df_test_under['Y']
x = df_test_under.drop(columns = ['Y'])
```

- Split the dataset into Train and Test Set -

We need to train our model and avoid overfitting and to do that we split our dataset into train and test that will be further used to evaluate the trained model.

```
#Splitting the datset into training set and test set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)
```

Generally, we have splitted our dataset into a 20% test set and 80% training set for further processing.

- Apply normalization to features -

Here we are importing the normalizing our dataset so that range for all the data points should lie in common range value which ultimately increases the training speed of our model.

```
#Standard scalar normalization to normalize the feature set
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

- Importing the classifier to train the Dataset -

  Here, we are importing the MLP classifier to use on this dataset and used fit function on the training data so that we can evaluate our model on the test dataset.

```
#Importing the Classifier for training and making predictions
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier(hidden_layer_sizes=(50,50,50,50),activation ='logistic', max_iter=3000)
classifier.fit(x_train, y_train.values.ravel())
predictions = classifier.predict(x_test)
```

After training the test set is used to predict the result.

- Performance Evaluation -

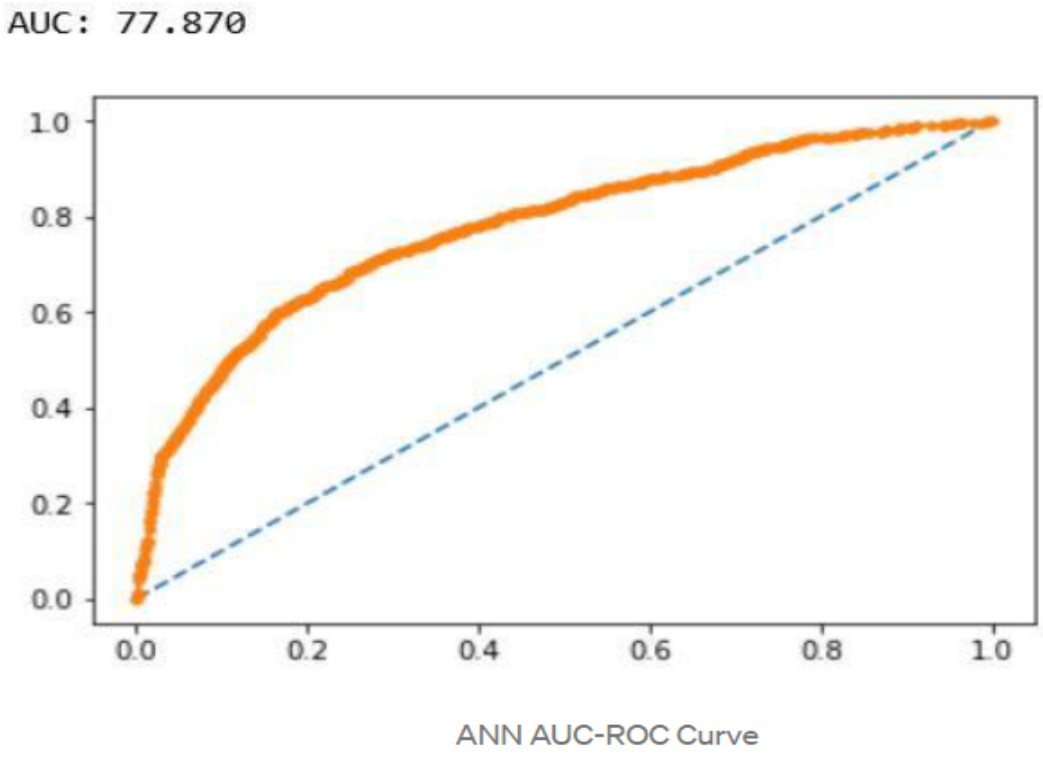
Precision - This gives the score of a class for a negative sample that shouldn't be labeled as positive.

Recall - This gives the score of a class for a sample that is identified correctly by the classifier.

F1-score - This gives the accuracy of a class for classifying the samples belong to that class compared to other classes.

- AUC ROC Curve -

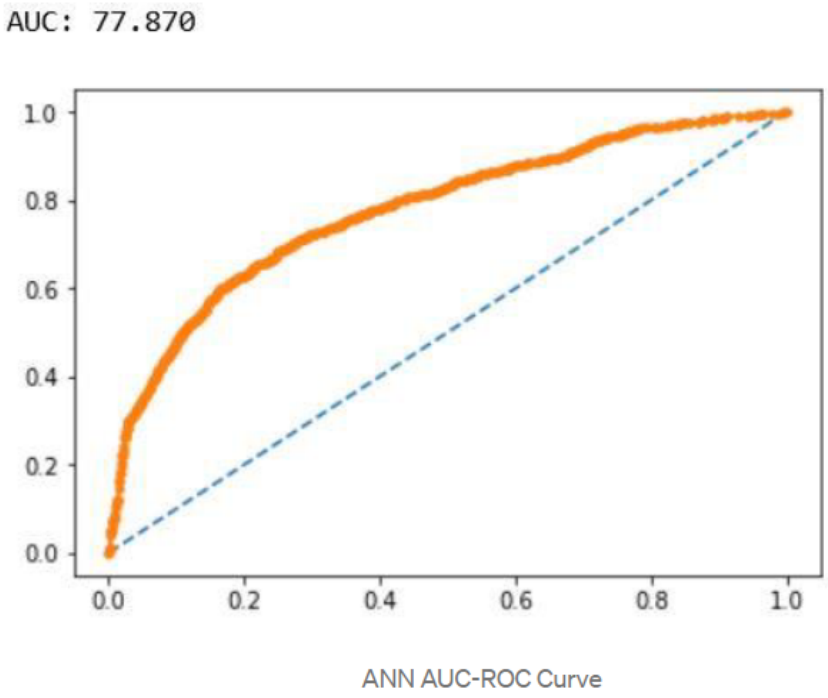
Higher the AUC, better the model prediction and better the model is. AUC value of ANN is 77.87%



ANN AUC-ROC Curve

ROC is a probability curve and AUC gives the degree of measure of separability.

- Confusion Matrix - Confusion Matrix is performance metrics in the form of table which gives information about the number of correct classification and incorrect classification i.e. misclassification.



ANN AUC-ROC Curve