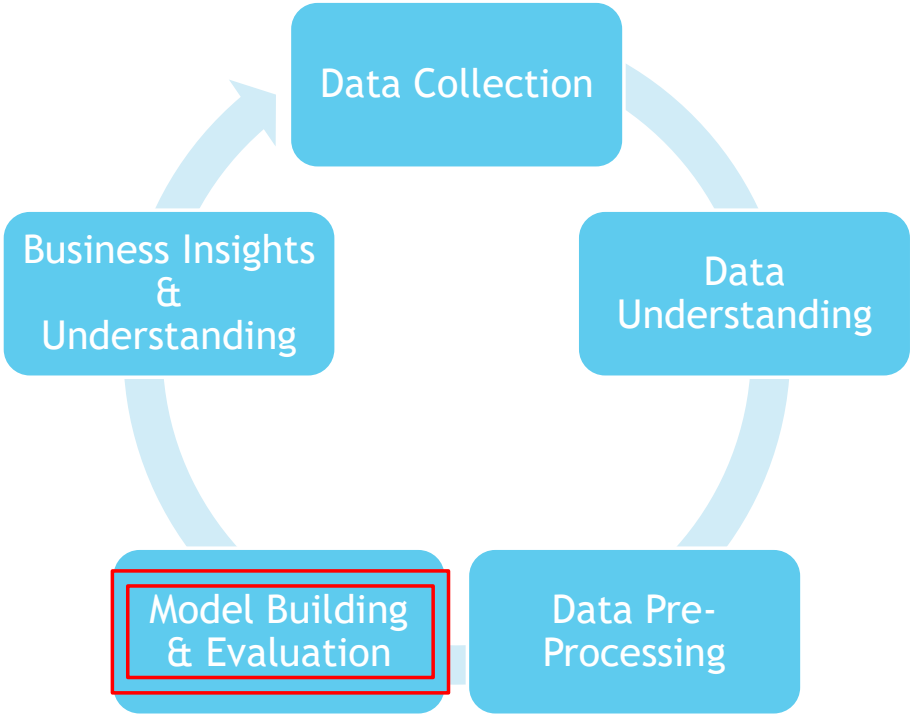
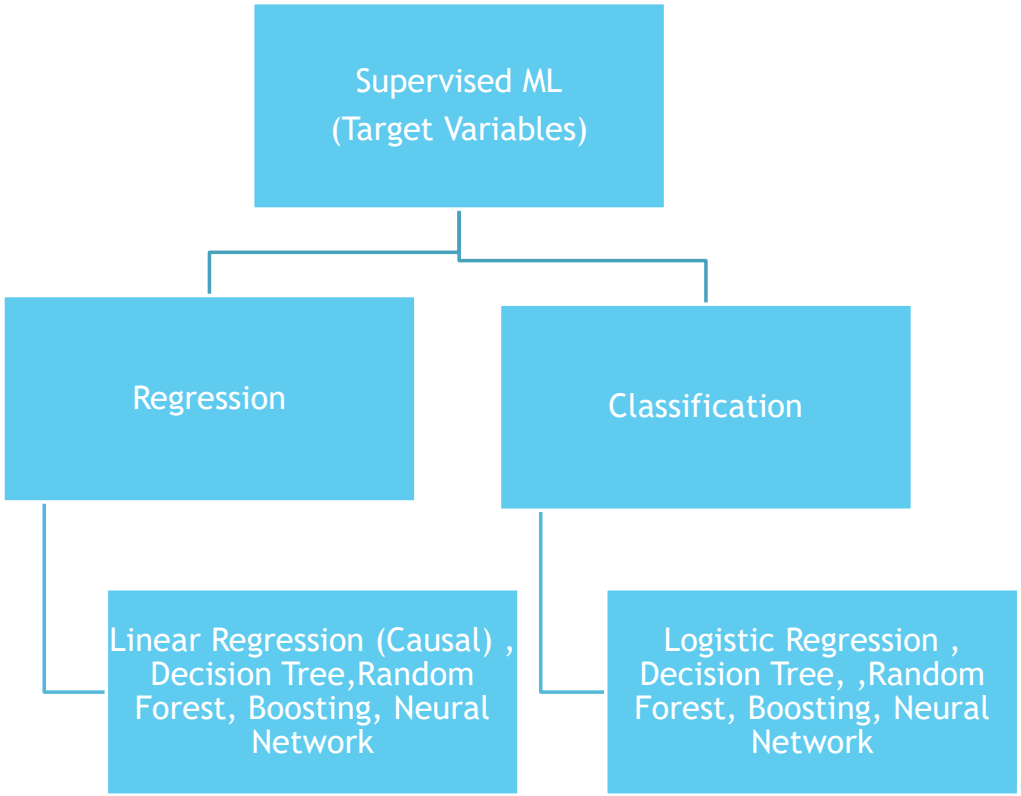
Neural Networks

DataDNA

Typical Data Science Cycle

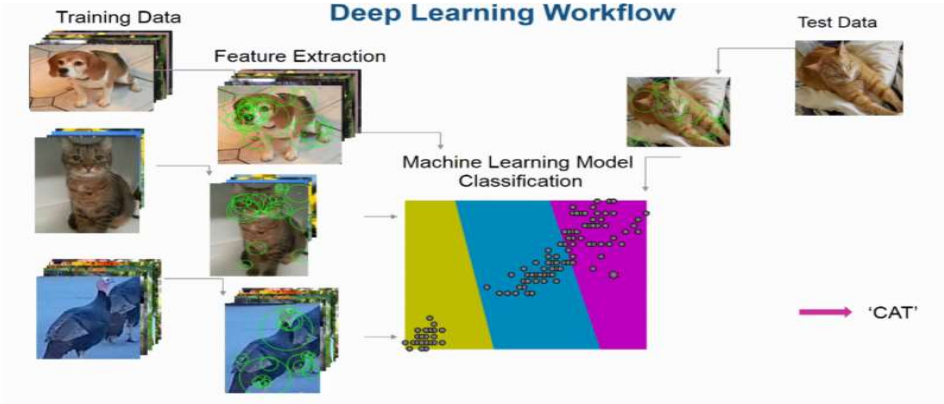


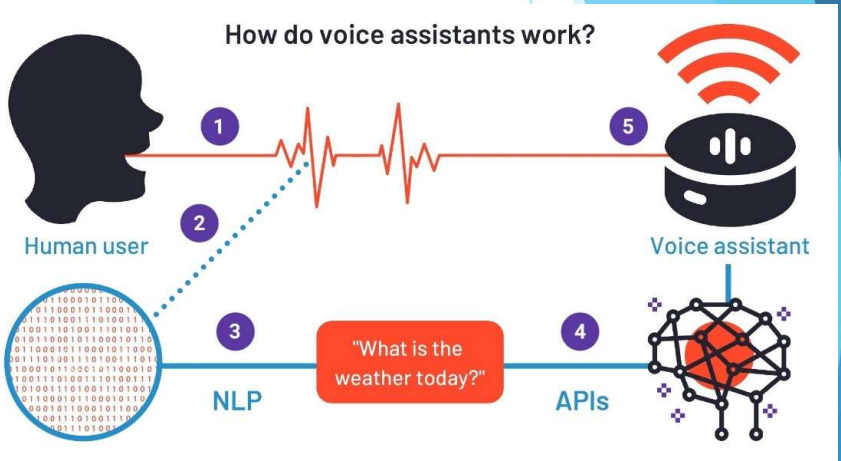
Machine Learning

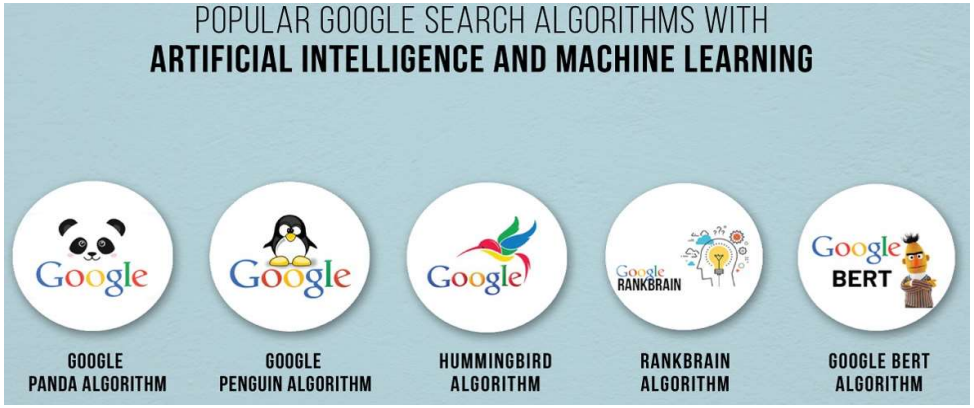


DataDNA

Deep Learning

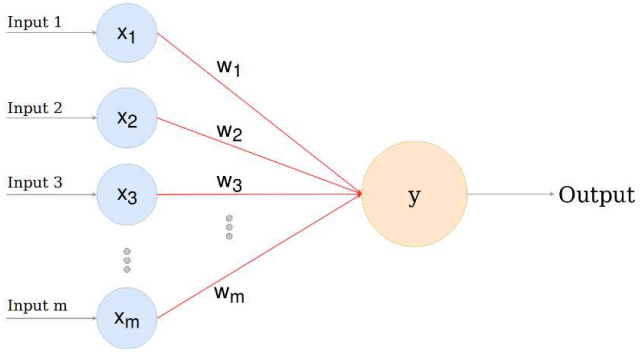


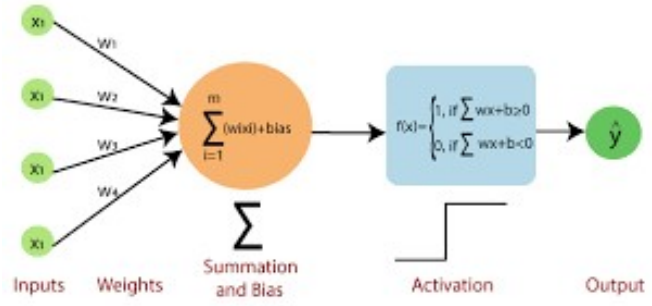




Perceptron

- A single-layer perceptron is the basic unit of a neural network.
- Architecture consists of a layer of input nodes fully connected to a single layer of output neuron
- It produces a binary output based on a threshold.
- A perceptron consists of input values, weights and a bias, a weighted sum and activation function

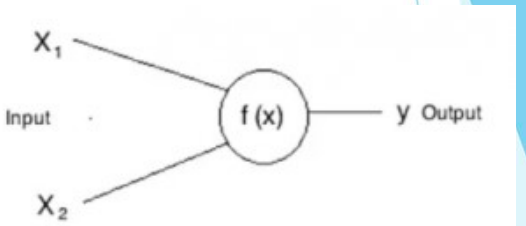




Features of Perceptron

Neuron

Each neuron is a mathematical operation that takes it's inputs, multiplies them by their weights and then passes the sum through the activation function for output.



Bias

- Bias is just like an intercept (constant) added in a linear equation.
- It is an additional parameter in the Neural Network which is used to adjust the output along with the weighted sum of the inputs to the neuron.
- It increases the flexibility of the model to fit given data.
- Bias is equal to negative of threshold require to activate (trigger) neuron

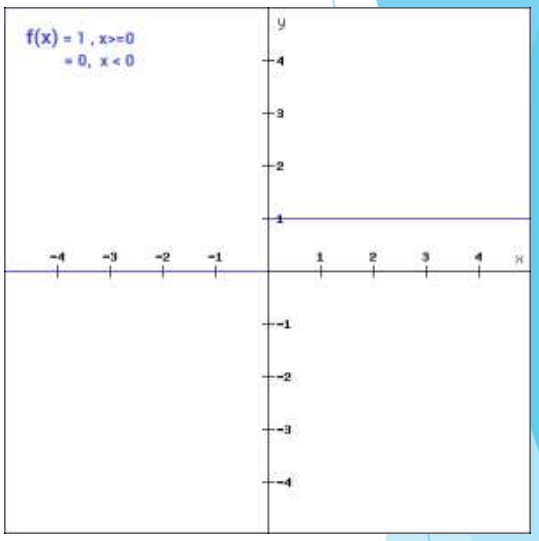
Weights

- Weights are the coefficients of the variables in the equations equation
- Negative weights reduce the value of an output and vice-versa for positive weights

output = sum (weights * inputs) + bias

How Perceptron Works

- Multiply all the inputs by their weights w and add them together referred as weighted sum: ∑ wj xj
- Add bias to it ∑ wj*xj +bias to create a linear combination of weight and bias
- Apply the activation function, in other words, determine whether the linear combination is greater than a threshold value.
- Perceptron uses a threshold (Binary step function) as an activation function
- If sum is greater than a threshold value, where -threshold is equivalent to bias, assign 1 ,if less then threshold then assign 0 as an output.



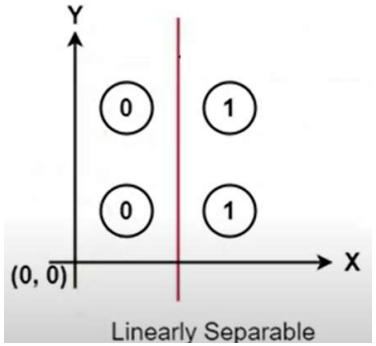
$$f(x) \begin{cases} 1 & \text{if } w.x + b \geq 0 \\ 0 & \text{if } w.x + b < 0 \end{cases}$$

bias = - threshold

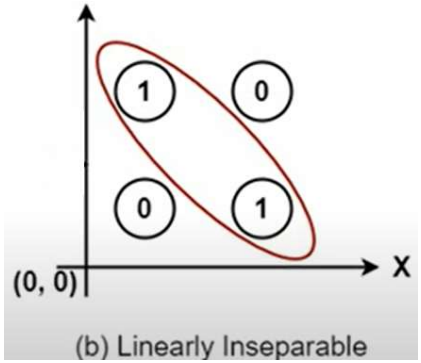
$\sum_j w_j x_j$

Neural Network (Deep Learning)

- A Perceptron has limitation i.e. simple perceptron can only classify linearly separable data sets.



Linearly Separable

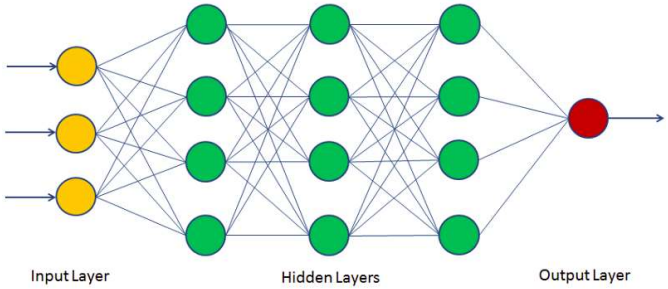


(b) Linearly Inseparable

- When layers of perceptron are combined together, they form a multilayer architecture, and this gives the required complexity of the neural network processing.

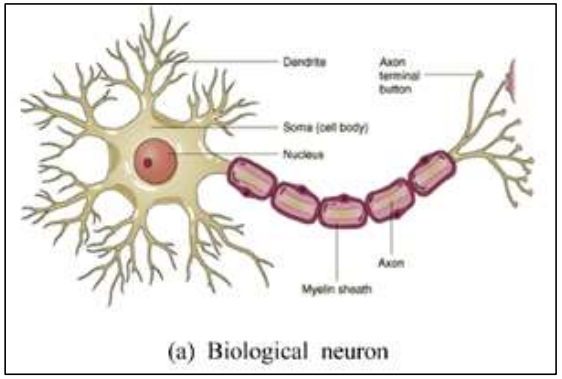
Neural Network (Deep Learning)

- Neural Network is a network of neurons and other components that strengthen predictive power of model on complex data

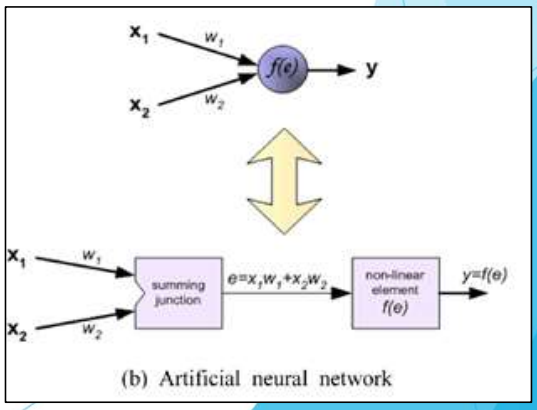


- An Artificial neural network is a very powerful machine learning mechanism which basically mimics how a human brain learns.

Direction of signal is along the Axon from Nucleus to synapse



(a) Biological neuron



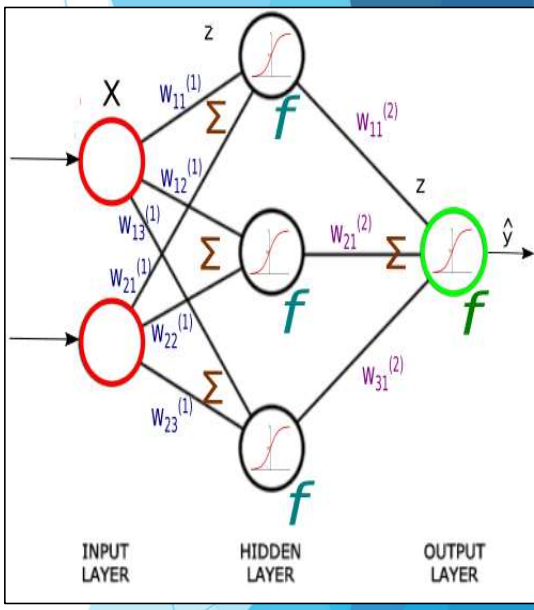
(b) Artificial neural network

- Neural Networks support both classification & regression problems.

Neural Network Components

- **Layers :** A neural network is made up of vertically stacked components called Layers.
  - Input Layer
  - Output Layer
  - Hidden Layer

Each input node is connected to each neuron of next hidden layer. Each neuron of hidden layer is connected to each neuron of next hidden layer and so on. Each neuron of last hidden layer is connected to output neuron.

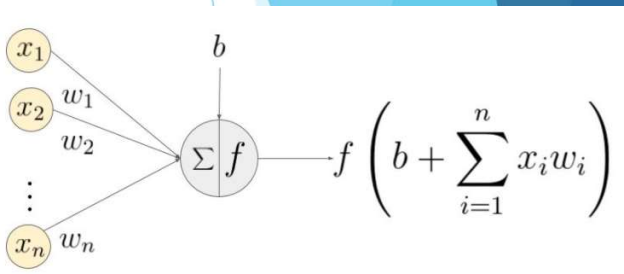


- **Weights:** Connection between nodes and neurons are assigned with individual weights that get updated in training process.

- **Bias:** Each neuron has a bias . The threshold (or *bias*) of a neuron is usually modelled as a weight which input is always 1. This bias is also learnable in training process

Features of Neural Network

Activation functions: Sometimes called a transfer function.



- Activation functions bring non-linearity in neural network.
- These are mathematical equations that determine the output of a neuron.

Activation function = f(∑(weights * inputs) + bias)

- Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between -1 and 1.
- NOTE: If we do not apply activation function then only linear transformation will be performed. Activation function introduced non-linear transformation to learn the complex patterns from the data
- We will discuss types of activation function after working of NN

Artificial Neural Network

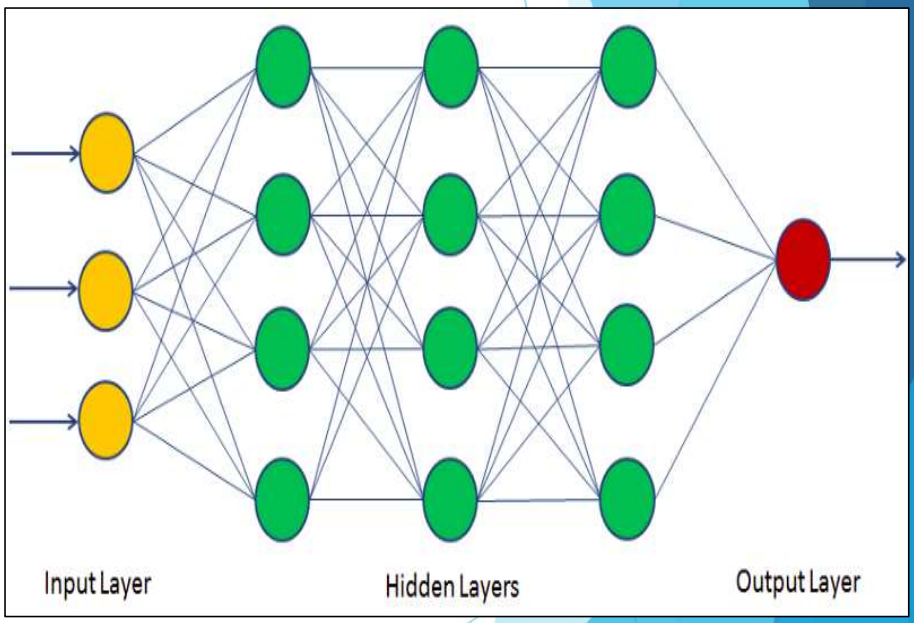
There are many types of neural networks available or that might be in the development stage. They can be classified depending on their structure, data flow, neurons used, their density, and layers

1. Perceptron
2. FeedForward Neural Network (FFNN)
3. Multi-Layer Perceptron (MLP)
4. RNN (Recurrent Neural Network)
5. CNN (Convolutional Neural Network)
6. Radial Basis Function Neural Network
7. LSTM (Long Short Term Memory)
8. Sequence to Sequence Models
9. Modular Neural Network

- We have already seen **perceptron**.
- Because of the limitation of backpropagations we do not use **FFNN** in deep learning. This neural network may or may not have the hidden layers
- Hence, we will focus upon **MLP** to start our deep learning journey.

Multi-Layer Perceptron (MLP)

- The term MLP is used ambiguously, sometimes loosely to mean any feedforward ANN
- Every single node is connected to all neurons in the next layer which makes it a fully connected neural network.
- Input and output layers are present having multiple hidden Layers i.e. at least three or more layers in total.
- It has a bi-directional propagation i.e. forward propagation and backward propagation.
- Used for deep learning [due to the presence of dense fully connected layers and back propagation]



How MLP works

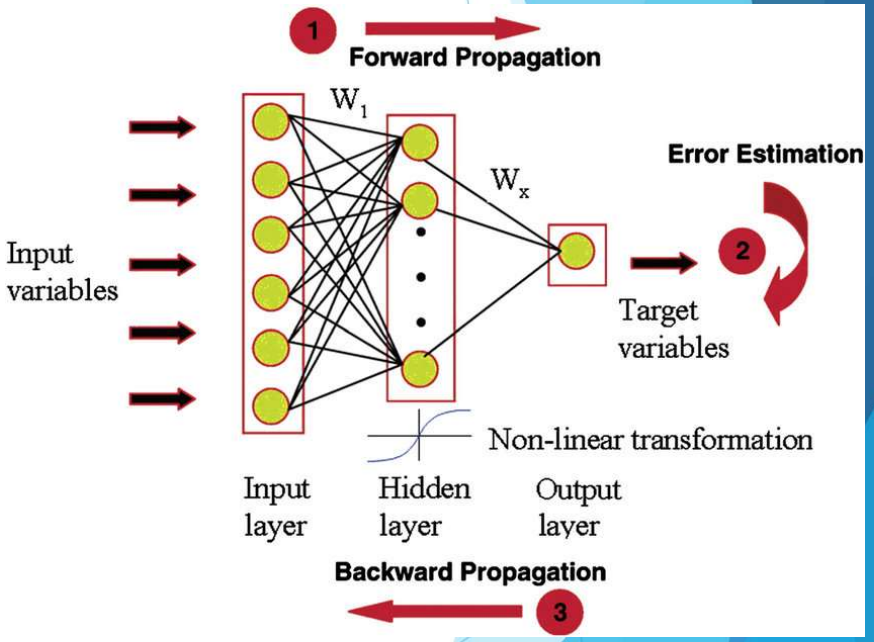
1. Define input variables (independent variables)

2. Randomly assign weights to all linkages between nodes and neurons

3. For first hidden layer apply activation function at each neuron using linearly combined input and calculate output for each neuron of first hidden layer

   output = f(∑ wj xj +bias)

4. Output from the activation function passed to the next hidden layer and the same process is repeated till output layer. This forward movement of information is known as the **forward propagation.**



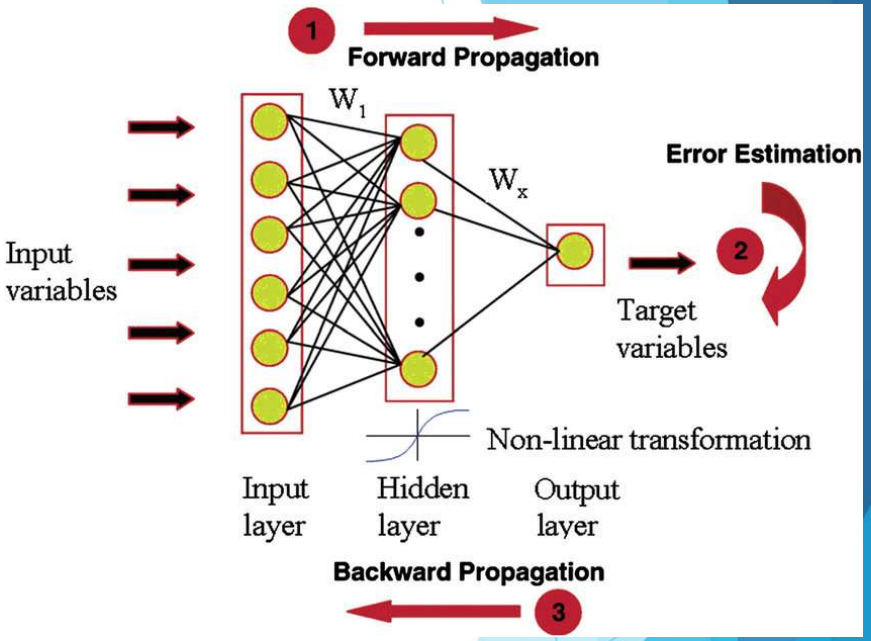
How MLP works

5. Using the final output from the forward propagation, error is calculated.

- Errors are passed back to all layers . This process is known as **back-propagation**. Purpose of this process is to update weights and biases of the network connections.
- Considering one layer at a time, the weights are updated according to the amount that they contributed to the error.

6. Forward and Back propagation processes are repeated for all of the examples in your training data. This complete round for the entire training dataset is called **an epoch**.

- A network may be trained for tens, hundreds or many thousands of epochs.
- After Each epoch ,weights/bias get updated to reduce errors (losses). This optimization is achieved using various **optimizers**

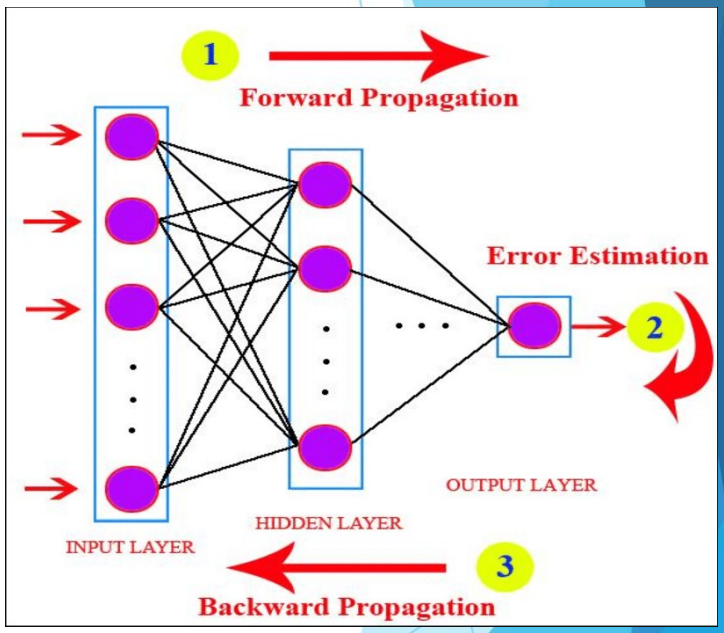


How MLP works

- We can pass one example at a time or we can process multiple observations in a group called **batch**. Errors are saved at the time when all data finishes their forward and backward phase in a batch. Finally weights can be updated for network after each batch.

- Size of batch normally is taken small to increase efficiency.

- The amount that weights are updated is controlled by a configuration parameters called the **learning rate**. Often small rates are used such as 0.1 or 0.01 or smaller

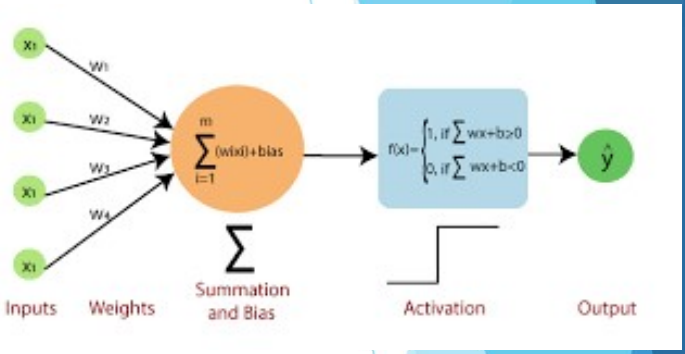


Features of Neural Network

Types of Activation functions

1. **Binary Step Activation Function:** Only two outputs (0 and 1)

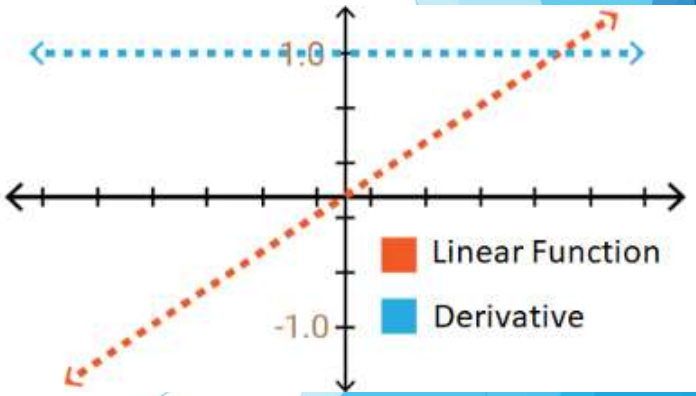
f(x) = 1 if x > 0 else 0 if x < 0



2. **Linear Activation Function**

Y = mx+c

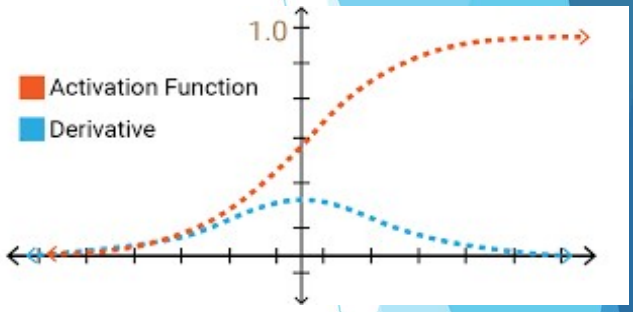
- Ranges = -infinity to +infinity
- Since gradient is constant (i.e. slope) weights and bias will be updated during the backprop but the updating factor (gradient) would be the same
- If we apply this AF to all layers then the last layer will simply become a linear function of the first layer



Features of Neural Network

3. **Sigmoid Activation Function:** It basically is a probabilistic approach towards decision making

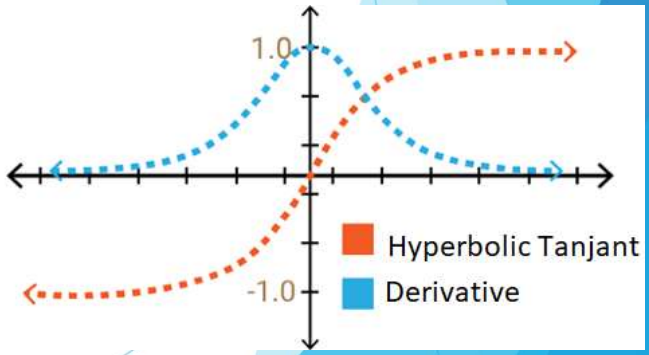
$$f(x) = \frac{1}{1 + e^{-x}}$$



- Ranges in between 0 to 1
- Derivative/gradient ranges between 0-0.25
- Suffers vanishing gradient problem i.e. As the gradient value approaches zero, the network does not really learn

4. **Hyperbolic Tangent Activation Function(Tanh):**

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



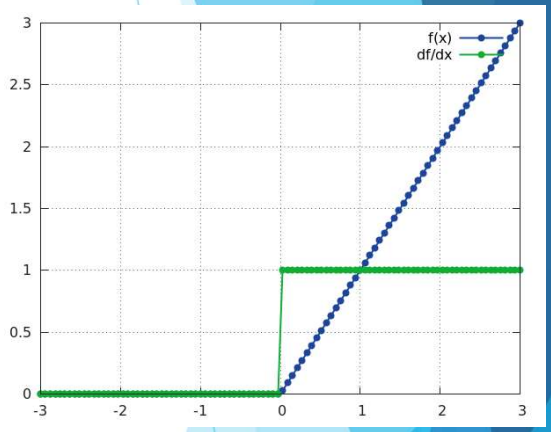
- Range -1 to 1 and gradient lies between 0-1
- Suffers vanishing gradient problem
- It is preferred over sigmoid since output is zero centric and optimization becomes easier during backpropagation process

Features of Neural Network

5. **ReLU (Rectified Linear unit) Activation function :**

Rectified linear unit or ReLU is most widely used non-linear activation function.

$$f(x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$$



- Defined by (f(x) = max(0,x))
- Range 0 to infinity
- Gradient is 0 for x<0 and 1 for x>1
- Solves vanishing gradient problem because many of the neurons have gradient = 1
- Computationally efficient and fast convergence because

1. use of non-exponential function
2. It does not activate all the neurons at the same time i.e. if the output of the linear transformation is less than 0 then neuron will be deactivated in other words 0 will be passed to next layer

Features of Neural Network

6. **Softmax Activation function :**

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for j} = 1,\ldots,K$$

- σ = softmax
- $\vec{z}$ = input vector
- e^z = standard exponential function for input vector
- K = number of classes in the multi-class classifier
- e^z = standard exponential function for output vector
- e^z = standard exponential function for output vector

- Softmax can be described as the combination of multiple sigmoidal function. it is very useful to handle multi-class classification problems.

- Softmax function returns the probability for a datapoint belonging to each individual class.
  Sum of all values will be 1

- Range 0 to infinity defined by (f(x) = max(0,x))

Features of Neural Network

Choose a Hidden Layer Activation Function

- Sigmoid and tanh should be avoided in hidden layers to avoid vanishing gradient problem
- ReLU function is most widely used today in hidden layers . it should be used only in hidden layers.
- We can start with ReLU function and switch to other activation function if performance is not good using ReLU

For further advanced NN architecture following is the general practice for hidden layers:

1. ANN (MLP): ReLU activation function.
2. Convolutional Neural Network (CNN): ReLU activation function.
3. Recurrent Neural Network: Tanh and/or Sigmoid activation function.

Choose an Output Layer Activation Function

- Regression Problem : One Node , Linear activation
- Binary Classification Problem : One Node, Sigmoid Activation Function
- Multiclass Classification Problem : One node per class Softmax Activation function

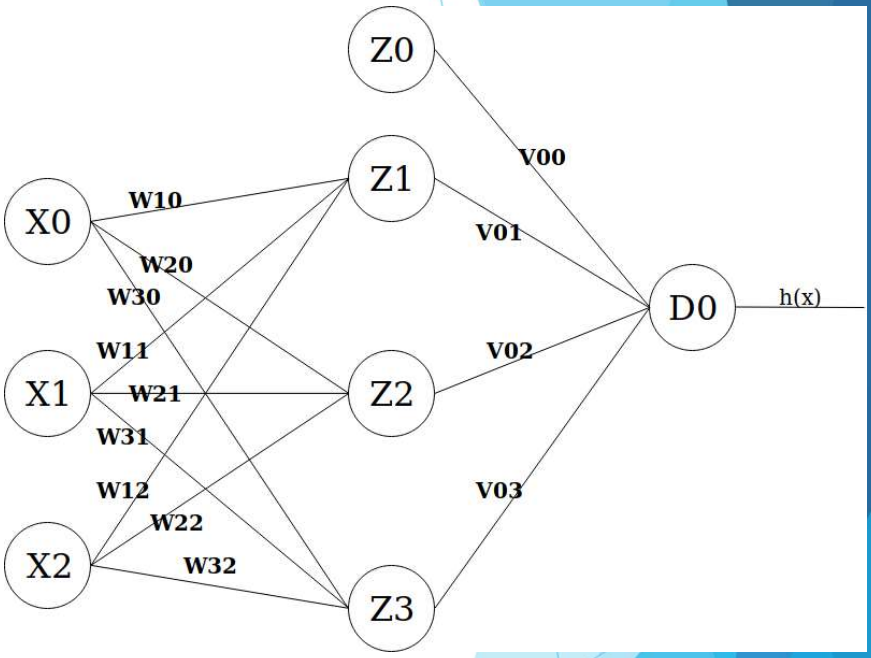
Update of Weights

Backpropagation, short for “backward propagation of errors”, is a mechanism used to update the **weights** using optimizer such as gradient descent.

$$w_{new} = w_{old} + \Delta w$$

$$w_{new} = w_{old} - \eta * \frac{\partial E}{\partial w_{old}}$$

η = learning rate
E = Error (loss)
W = weights



- It calculates the gradient of the error function with respect to weights. This calculation proceeds backwards through the network.

Intuition : if derivative $\frac{\partial E}{\partial w_{old}}$ is positive (i.e. effect of weighs on error is positive), then equation reduces weight by $\eta * \frac{\partial E}{\partial w_{old}}$

Update of Weights

Let's say we have 1 input layer with 3 input ,
1 hidden layer with 3 neuron and 1 output neuron.

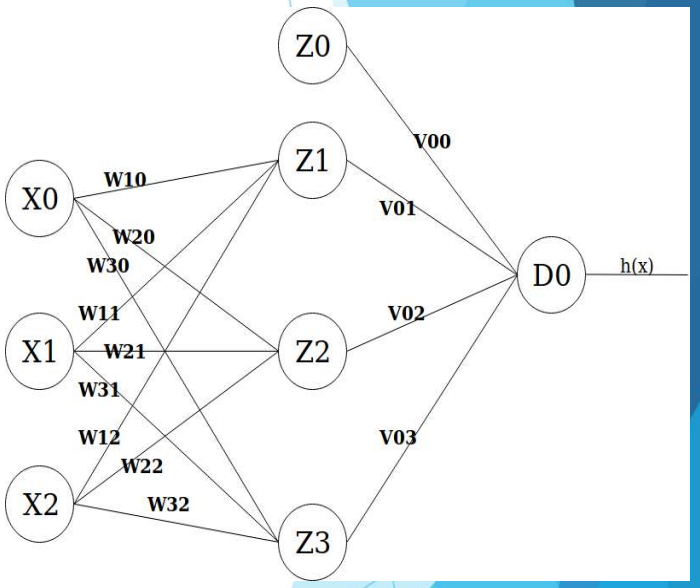
X0/X1/X2 are input for first hidden layer.
Z1/Z2/Z3 are output of first hidden layer and input for next layer (output)

h(x) is final output which is f(∑ Vj*Zj +bias) at last node

$$w_{v01\_new} = w_{v01\_old} - \eta * \frac{\partial E}{\partial w_{v01\_old}}$$ same for v02 and v03

same process will be applied to all weights in previous layers such as

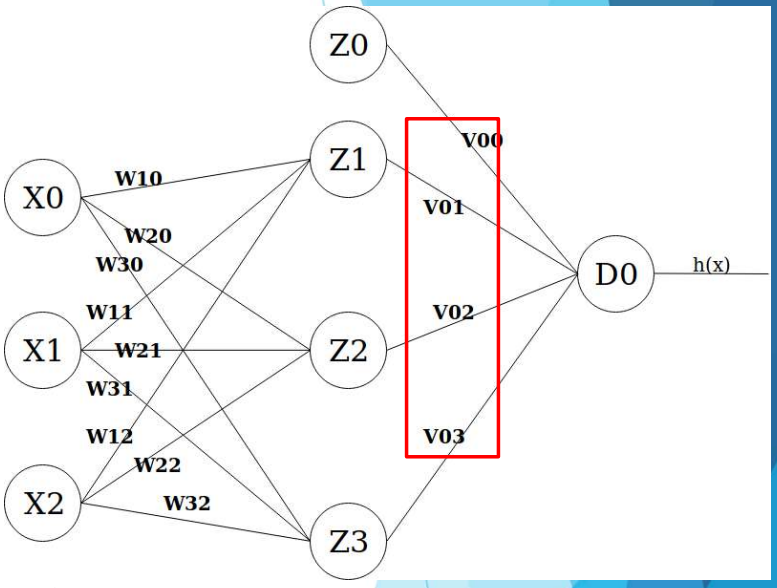
$$w_{w10\_new} = w_{w10\_old} - \eta * \frac{\partial E}{\partial w_{w10\_old}}$$



Update of Weights

$$w_{v01} = w_{v01} - \eta * \frac{\partial E}{\partial w_{v01}}$$

The derivation of the error function is further evaluated by applying the chain rule as following :-



For weight in last layer

$$\frac{\partial E}{\partial w_{v01}} = \frac{\partial E}{\partial h(x)} * \frac{\partial h(x)}{\partial w_{v01}}$$

$$\frac{\partial E}{\partial w_{v01}} = \frac{\partial E}{\partial h(x)} * \frac{\partial h(x)}{\partial (\sum Vj*Zj+bias)} * \frac{\partial (\sum Vj*Zj+bias)}{\partial w_{v01}}$$

Derivative of activation function at output neuron D0

Linear Combination I/P

If activation function is sigmoid then this derivative will shrink between 0-0.25 , will reduce $\frac{\partial E}{\partial w_{v01}}$. For subsequent hidden layers resultant gradient will become much smaller (because of chain effect) so that weights will not be updated in initial layers. This problem is called vanishing gradient.

Vanishing Gradient Problem

if the derivatives are small then the gradient will decrease exponentially as we propagate through the model until it eventually vanishes, and this is the ***vanishing gradient*** problem.

For weights in first layer $\frac{\partial E}{\partial w_{w10}} = \frac{\partial E}{\partial h(x)} * \frac{\partial h(x)}{\partial w_{w10}}$

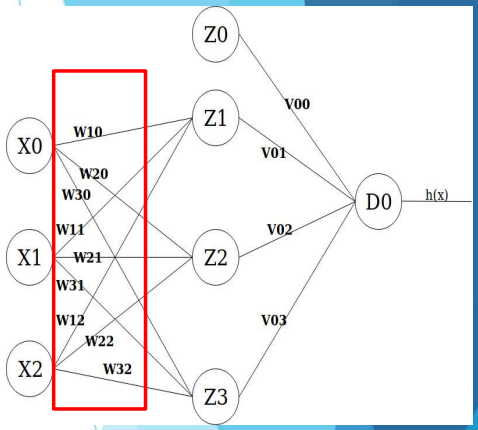
$$\frac{\partial E}{\partial w_{w10}} = \frac{\partial E}{\partial h(x)} * \frac{\partial h(x)}{\partial Z1} * \frac{\partial Z1}{\partial w_{w10}}$$

Derivative of activation function at output neuron D0

Derivative of activation function at output neuron Z1

$$\frac{\partial h(x)}{\partial Z} = \frac{\partial h(x)}{\partial(\sum Vj*Zj+bias)} * \frac{\partial(\sum Vj*Zj+bias)}{\partial Z1}$$

$$\frac{\partial Z1}{\partial w_{w10}} = \frac{\partial Z1}{\partial(\sum wj*xj+bias)} * \frac{\partial(\sum wj*xj+bias)}{\partial w_{w10}}$$



- It can cause slow training of network.
- We can avoid this problem by reducing number of layers or using ReLU in hidden layers.

Disadvantages of Neural Network

- Good predictive performance but considered a “black box” prediction machine, with no insight into relationships between predictors and outcome

- Heavy computational requirements if there are many variables (additional variables dramatically increase the number of weights to calculate)

DataDNA

Hyper parameter Tuning

hidden_layer_sizes tuple, length = n_layers - 2, default=(100,)
The I'th element represents the number of neurons in the I'th hidden layer.

activation{'identity', 'logistic', 'tanh', 'relu'}, default='relu'
Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max(0, x)

solver{'lbfgs', 'sgd', 'adam'}, default='adam'
The solver for weight optimization.

- 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- 'sgd' refers to stochastic gradient descent.
- 'adam' refers to a stochastic gradient-based optimizer

Hyper parameter Tuning

Alpha float, default=0.0001
L2 penalty (regularization term) parameter.

batch_size int, default='auto'
Size of minibatches for stochastic optimizers. When set to "auto", batch_size=min(200, n_samples)

learning_rate{'constant', 'invscaling', 'adaptive'}, default='constant'
Learning rate schedule for weight updates.

- 'constant' is a constant learning rate given by 'learning_rate_init'.
- 'invscaling' gradually decreases the learning rate at each time step 't' using an inverse scaling exponent of 'power_t'. effective_learning_rate = learning_rate_init / pow(t, power_t)
- 'adaptive' keeps the learning rate constant to 'learning_rate_init' as long as training loss keeps decreasing.

Only used when solver='sgd'.

DataDNA

Game

- Game

https://playground.tensorflow.org/#activation=sigmoid&batchSize=30&dataset=gauss&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.87984&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

Performance Validation

Performance validation of Neural Network

Customer	City	NoOfGamesPlayed	Channel	FavoriteGame	Actual	
1059	1	36	Favorite	Uniform	1	Training Data
1060	1	298	Favorite	Uniform	0	
1061	1	27	Uniform	Uniform	1	
1062	1	156	Favorite	Uniform	0	
1063	1	217	Favorite	Uniform	0	
1064	1	51	Favorite	Uniform	0	
1065	1	30	Favorite	Uniform	1	
1066	2	74	Favorite	Uniform	0	
1067	2	37	Favorite	Uniform	0	
1068	2	110	Favorite	Uniform	0	
1069	2	140	Favorite	Uniform	0	
1070	2	60	Favorite	Uniform	1	
1071	2	75	Favorite	Uniform	0	Test Data
1072	2	116	Favorite	Uniform	0	
1073	2	171	Favorite	Uniform	1	
1074	2	67	Favorite	Uniform	0	
1075	2	16	Favorite	Uniform	1	
1076	2	121	Uniform	Uniform	0	
1077	2	78	Favorite	Uniform	1	

Modelling on Training Data

ML Model

Apply on Test Data

Prediction

Customer	City	NoOfGamesPlayed	Channel	FavoriteGame	Actual	Prediction	
1071	2	75	Favorite	Uniform	0	1	Test Data
1072	2	116	Favorite	Uniform	0	1	
1073	2	171	Favorite	Uniform	1	0	
1074	2	67	Favorite	Uniform	0	1	
1075	2	16	Favorite	Uniform	1	1	
1076	2	121	Uniform	Uniform	0	0	
1077	2	78	Favorite	Uniform	1	1	

Model Evaluation Metrics

The various metrics used to evaluate the results of the prediction are :

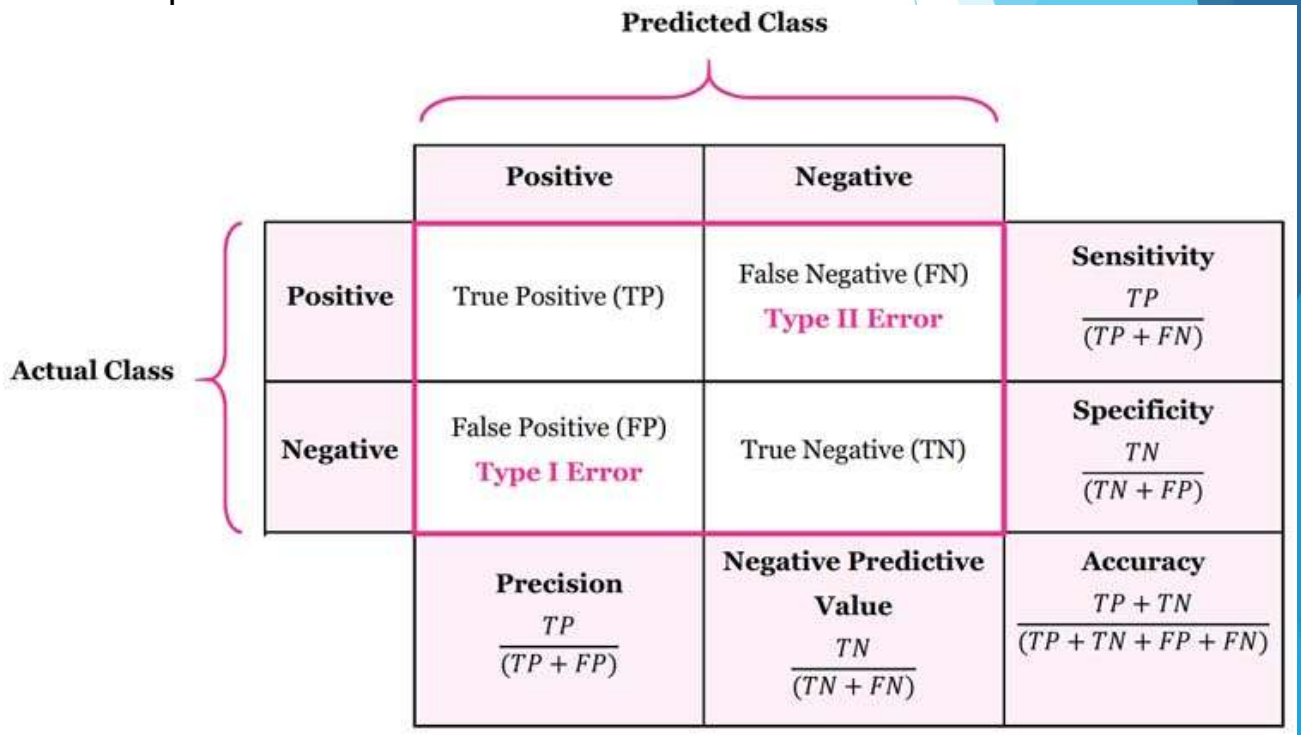
1.Accuracy

2.Precision

3.Recall

4.Area Under ROC Curve

5.MAPE/ MAD/ RMSE (For Regressors)



		Predicted Class: Positive	Predicted Class: Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$