ARTIFICIAL NETURAL NETWORKS

- **ANN is one of the deep learning algorithms that stimulates the workings of neurons in the human brain.**

STRUCTURE OF ANN

The artificial neural networks consists of the input layer, hidden layers, output layer. The hidden layer can be more than one in number. Each layer consists of n number of neurons. Each layer will be having an activation function associated with each of the neurons. The activation function is the function that is responsible for introducing non-linearity in the relationship.

Artificial neural networks consists of two phases,

Forward propagation

Backward propagation

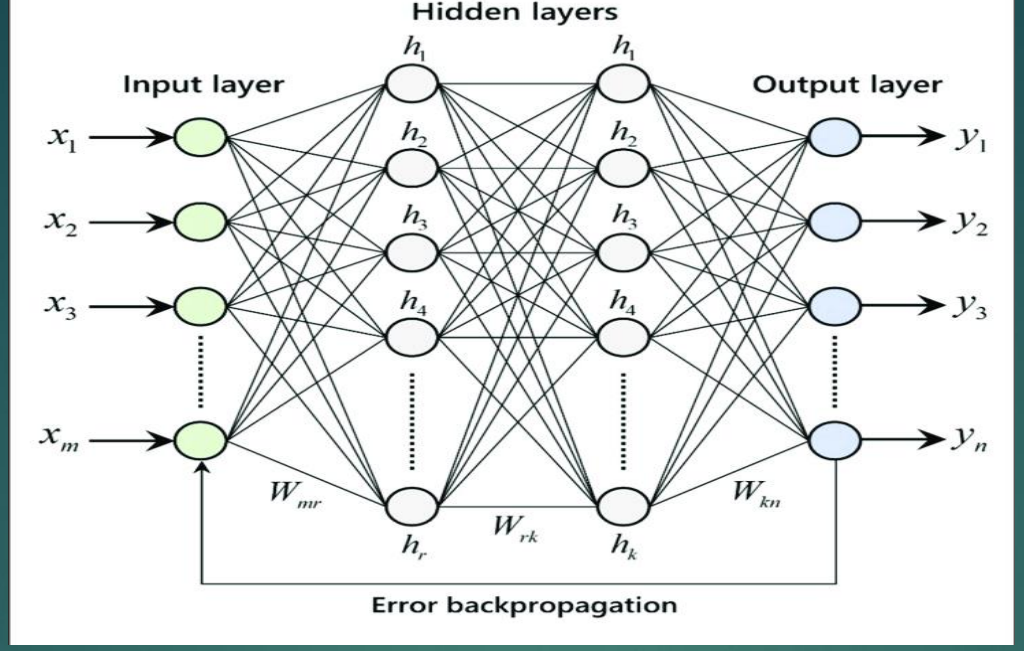
Forward propagation is the process of multiplying weights with each feature and adding them. The bias is also added to the result. Backward propagation is the process of updating the weights in the model. Backward propagation requires an optimization function and a loss function.

x - input layer

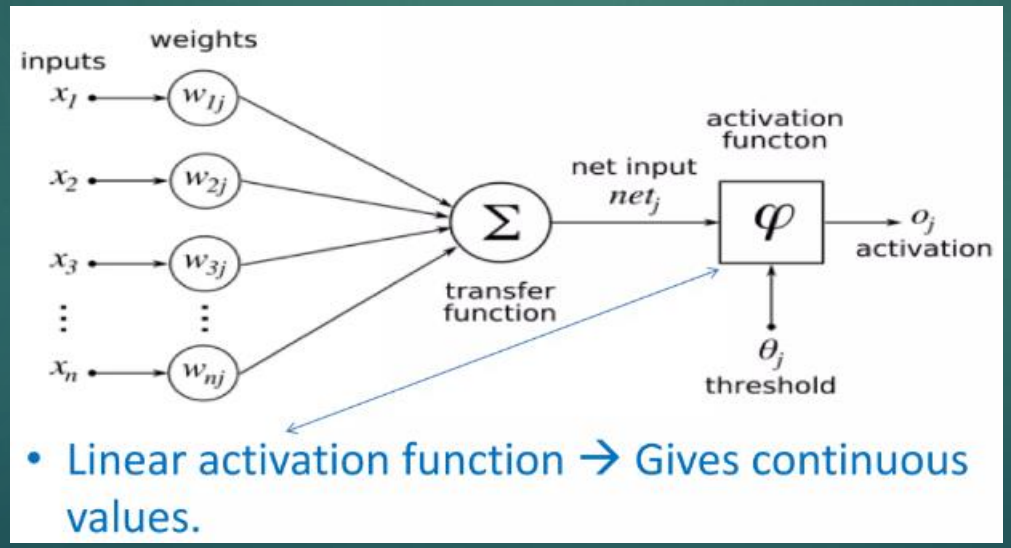
y - output layer

h – hidden layers

w – weights



Ann regression



Backpropagation formula:

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(X, \theta^t)}{\partial \theta}$$

w_{ij}^k : weight for node j in layer l_k for incoming node i
b_i^k : bias for node i in layer l_k
a_i^k : product sum plus bias (activation) for node i in layer l_k
o_i^k : output for node i in layer l_k
r_k : number of nodes in layer l_k

---

g : activation function for the hidden layer nodes
g_o : activation function for the output layer nodes

The error function in classic backpropagation is the mean squared error

$$E(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

$$w_{0i}^k = b_i^k.$$

$$a_i^k = b_i^k + \sum_{j=1}^{r_{k-1}} w_{ji}^k o_j^{k-1} = \sum_{j=0}^{r_{k-1}} w_{ji}^k o_j^{k-1},$$

$$E(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

$$\frac{\partial E(X, \theta)}{\partial w_{ij}^k} = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial}{\partial w_{ij}^k} \left(\frac{1}{2} (\hat{y}_d - y_d)^2 \right) = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial E_d}{\partial w_{ij}^k}.$$

Error Function Derivatives

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ij}^k},$$

$$\delta_j^k \equiv \frac{\partial E}{\partial a_j^k}.$$

$$\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k}\left(\sum_{l=0}^{r_{k-1}} w_{lj}^k o_l^{k-1}\right) = o_i^{k-1}.$$

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1}.$$

The Output Layer

$$E = \frac{1}{2}\left(\hat{y} - y\right)^2 = \frac{1}{2}\left(g_o(a_1^m) - y\right)^2,$$

$$\delta_1^m = \left(g_0(a_1^m) - y\right) g_o'(a_1^m) = \left(\hat{y} - y\right) g_o'(a_1^m)$$

$$\frac{\partial E}{\partial w_{i1}^m} = \delta_1^m o_i^{m-1} = \left(\hat{y} - y\right) g_o'(a_1^m)\, o_i^{m-1}.$$

The Hidden Layers

$$\delta_j^k = \frac{\partial E}{\partial a_j^k} = \sum_{l=1}^{r^{k+1}} \frac{\partial E}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_j^k},$$

$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{\partial a_j^k}.$$

$$a_l^{k+1} = \sum_{j=1}^{r^k} w_{jl}^{k+1} g(a_j^k),$$

$$\frac{\partial a_l^{k+1}}{\partial a_j^k} = w_{jl}^{k+1} g'(a_j^k).$$

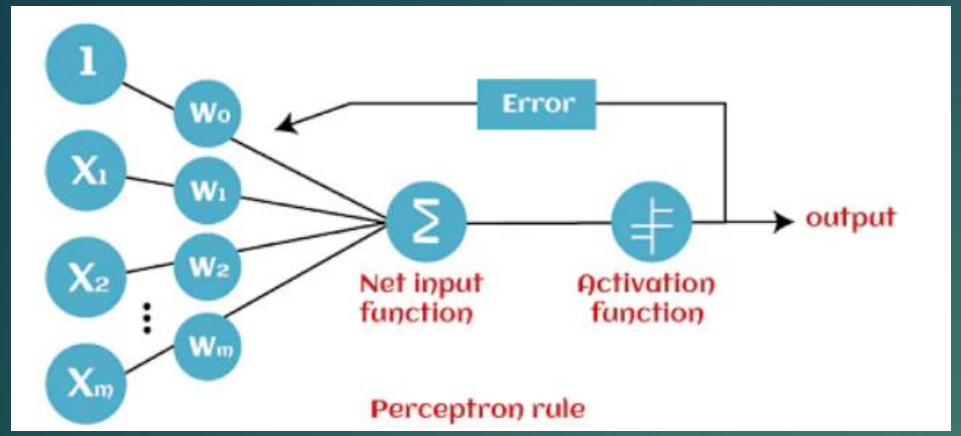
$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{jl}^{k+1} g'(a_j^k) = g'(a_j^k) \sum_{l=1}^{r^{k+1}} w_{jl}^{k+1} \delta_l^{k+1}.$$

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1} = g'(a_j^k) o_i^{k-1} \sum_{l=1}^{r^{k+1}} w_{jl}^{k+1} \delta_l^{k+1}.$$

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

Types of Perceptron model:

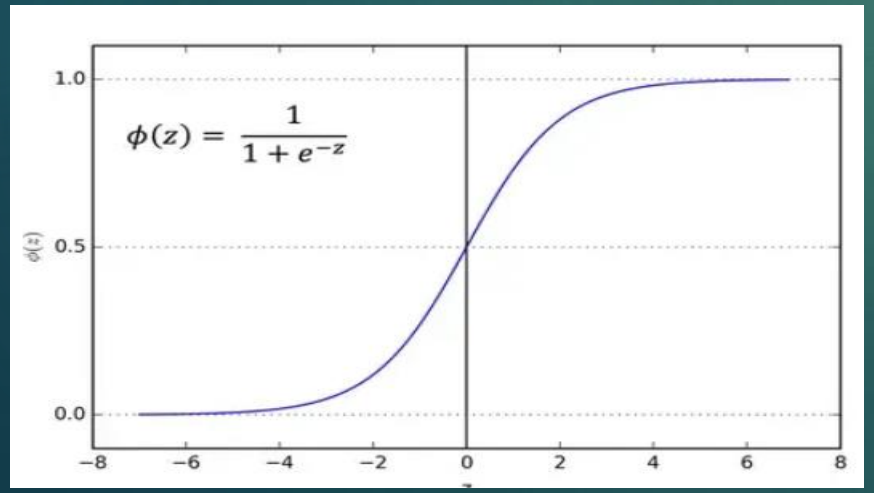
- Single Layer Perceptron model: One of the easiest ANN(Artificial Neural Networks) types consists of a feed-forward network and includes a threshold transfer inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes. A Single-layer perceptron can learn only linearly separable patterns.
- Multi-Layered Perceptron model: It is mainly similar to a single-layer perceptron model but has more hidden layers.



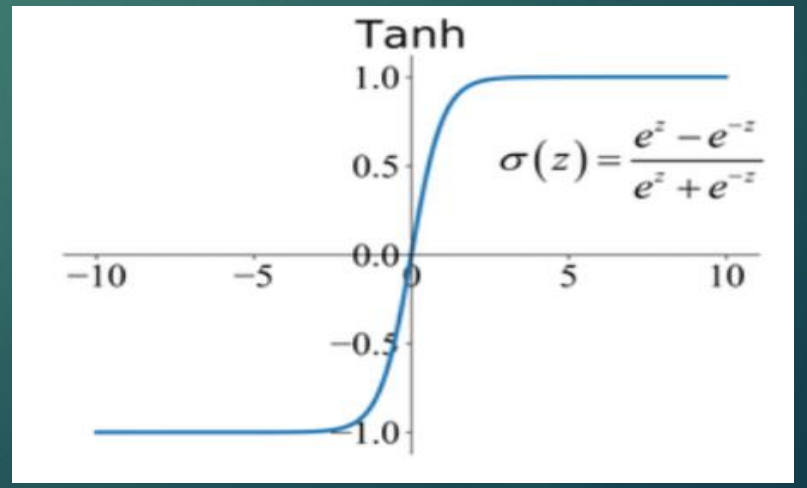
ACTIVATION FUNCTION: An activation function is a function used in artificial neural networks which outputs a small value for small inputs, and a larger value if its inputs exceed a threshold.

TYPES:

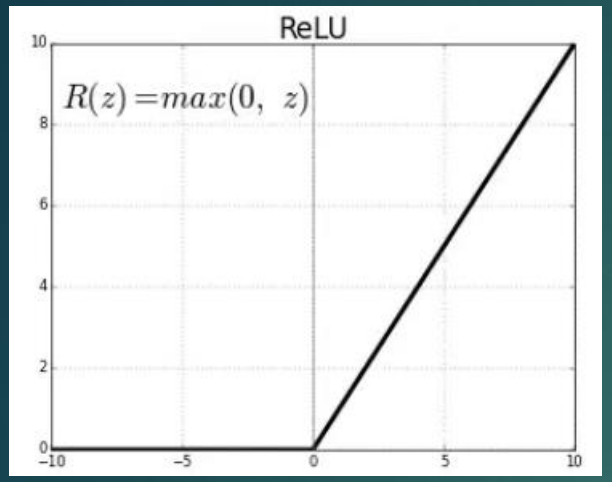
1. **Sigmoid or Logistic Activation Function**



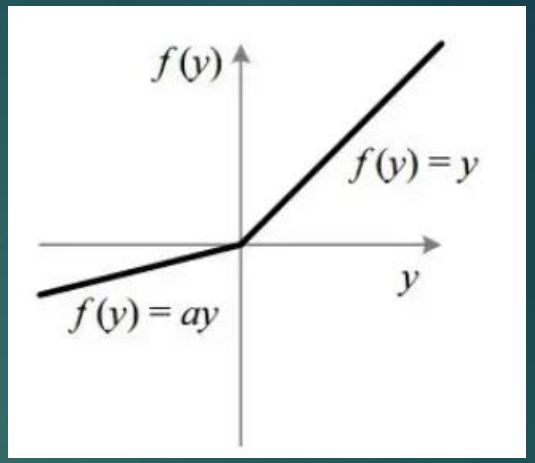
2. **Tanh or hyperbolic tangent Activation Function**



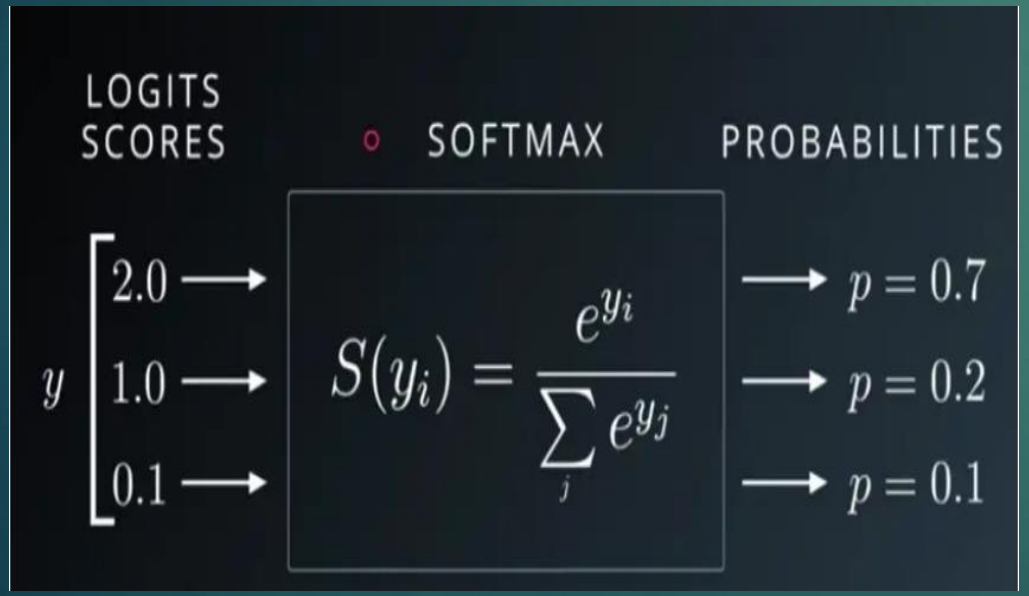
3. ReLU (Rectified Linear Unit) Activation Function



4. Leaky ReLU



5. Softmax



OPTIMIZATION

- Optimizers are algorithms or methods used to change the attributes of the neural network such as **weights** and **learning rate** to reduce the losses. Optimizers are used to solve optimization problems by minimizing the function.

TYPES:

1. Gradient Descent

Gradient descent is an optimization algorithm that's used when training a machine learning model. It's based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum.

WHAT IS GRADIENT DESCENT? Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent is simply used to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible.

2. Stochastic Gradient Descent

SGD algorithm is an extension of the Gradient Descent and it overcomes some of the disadvantages of the GD algorithm. In the SGD algorithm derivative is computed taking one point at a time.

3. Mini Batch Stochastic Gradient Descent (MB-SGD)

MB-SGD algorithm is an extension of the SGD algorithm and it overcomes the problem of large time complexity in the case of the SGD algorithm. MB-SGD algorithm takes a batch of points or subset of points from the dataset to compute derivate.

4. Adaptive Gradient Descent(AdaGrad)

It performs smaller updates for parameters associated with frequently occurring features, and larger updates for parameters associated with infrequently occurring features.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i}$$

5. RMSprop

RMSprop divides the learning rate by an exponentially decaying average of squared gradients.

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

6. Adaptive Moment Estimation (Adam)

Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. Adam computes adaptive learning rates for each parameter.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

➢ Vanishing Gradient:

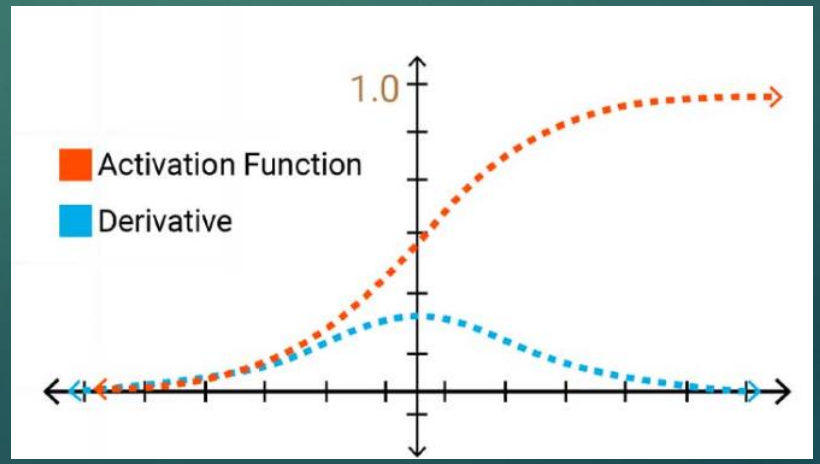
The use of sigmoid function restricted the training of deep neural networks because it caused the vanishing gradient problem. This caused the neural network to learn at a slower pace or in some cases no learning at all.

Method to overcome the problem

The vanishing gradient problem is caused by the derivative of the activation function used to create the neural network. The simplest solution to the problem is to replace the activation function of the network. Instead of sigmoid, use an activation function such as ReLU.

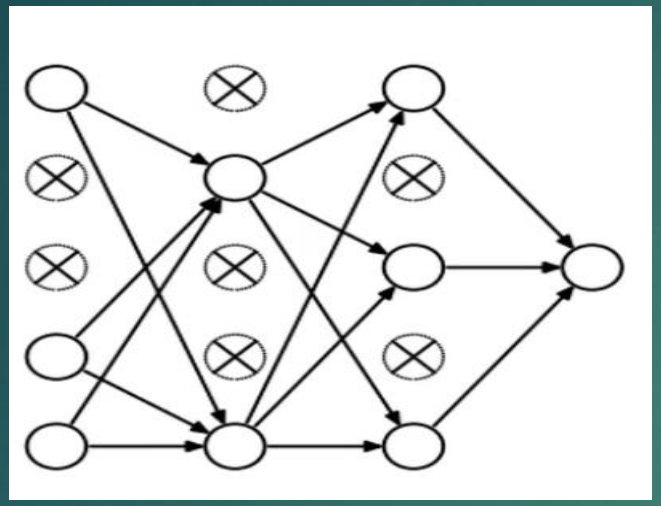
Rectified Linear Units (ReLU) are activation functions that generate a positive linear output when they are applied to positive input values. If the input is negative, the function will return zero.

If the ReLU function is used for activation in a neural network in place of a sigmoid function, the value of the partial derivative of the loss function will be having values of 0 or 1 which prevents the gradient from vanishing. The use of ReLU function thus prevents the gradient from vanishing.



- Dropout

In machine learning, “dropout” refers to the practice of disregarding certain nodes in a layer at random during training. A dropout is a regularization approach that prevents overfitting by ensuring that no units are codependent with one another.



REFERENCE

1. https://www.analyticsvidhya.com/blog/2021/08/a-walk-through-of-regression-analysis-using-artificial-neural-networks-in-tensorflow/
2. https://brilliant.org/wiki/backpropagation/#:~:text=%E2%88%82ajk%E2%80%8B,1%20%CE%B4%20l%20k%20%2B%201%20.
3. https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron
4. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
5. https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html