**U19EC046 | ML | LAB 10**

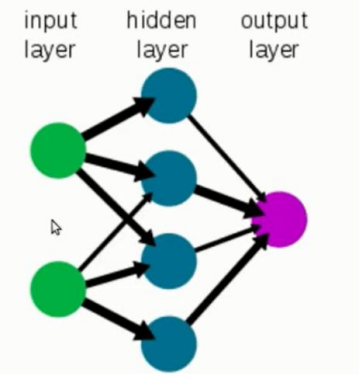
**AIM**

Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

**THEORY**

*What is Artificial Neural Networks?*

A neural network is a group of connected I/O units where each connection has a weight associated with its computer programs. It helps you to build predictive models from large databases. This model builds upon the human nervous system. It helps you to conduct image understanding, human learning, computer speech, etc.



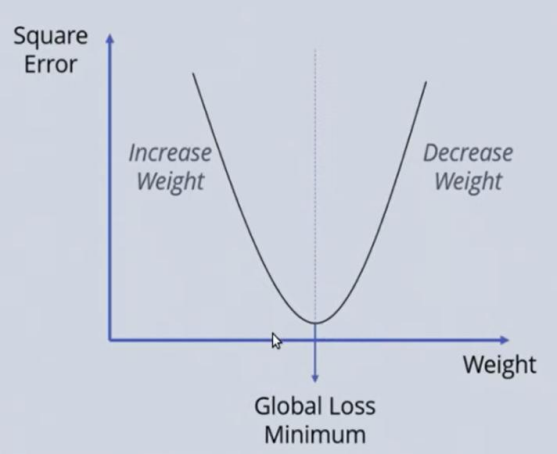
*What is Backpropagation?*

Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.

Backpropagation in neural network is a short form for “backward propagation of errors.” 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.

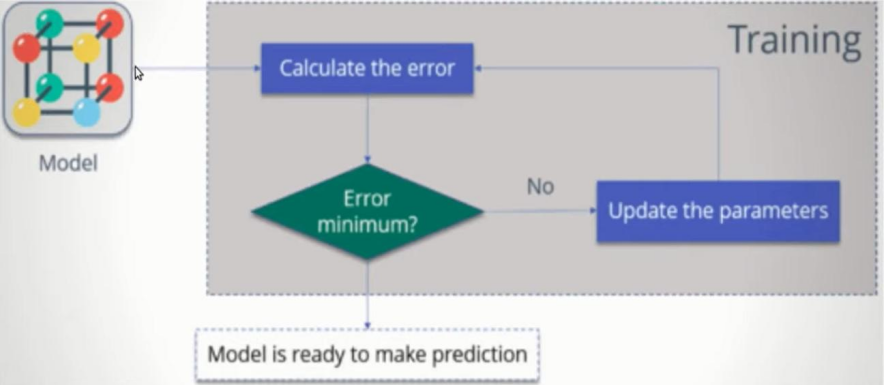
*How Backpropagation Algorithm Works*

The Back propagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.



*Consider the following Back propagation neural network example diagram to understand:*

1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs
5. ErrorB= Actual Output – Desired Output
6. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.
7. Keep repeating the process until the desired output is achieved

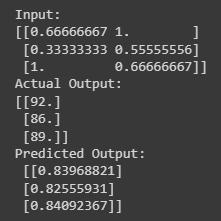


**CODE**

|  |
| --- |
| **import numpy as np**  **X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)**  **y = np.array(([92], [86], [89]), dtype=float)**  **X = X/np.amax(X,axis=0) # maximum of X array longitudinally y = y/100**  **#Sigmoid Function**  **def sigmoid (x):**  **return (1/(1 + np.exp(-x)))**  **#Derivative of Sigmoid Function**  **def derivatives\_sigmoid(x):**  **return x \* (1 - x)**  **#Variable initialization**  **epoch=7000 #Setting training iterations**  **lr=0.1 #Setting learning rate**  **inputlayer\_neurons = 2 #number of features in data set**  **hiddenlayer\_neurons = 3 #number of hidden layers neurons**  **output\_neurons = 1 #number of neurons at output layer**  **#weight and bias initialization**  **wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))**  **bh=np.random.uniform(size=(1,hiddenlayer\_neurons))**  **wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))**  **bout=np.random.uniform(size=(1,output\_neurons))**  **# draws a random range of numbers uniformly of dim x\*y**  **#Forward Propagation**  **for i in range(epoch):**  **hinp1=np.dot(X,wh)**  **hinp=hinp1 + bh**  **hlayer\_act = sigmoid(hinp)**  **outinp1=np.dot(hlayer\_act,wout)**  **outinp= outinp1+ bout**  **output = sigmoid(outinp)**    **#Backpropagation**  **EO = y-output**  **outgrad = derivatives\_sigmoid(output)**  **d\_output = EO\* outgrad**  **EH = d\_output.dot(wout.T)**  **hiddengrad = derivatives\_sigmoid(hlayer\_act)**  **#how much hidden layer wts contributed to error**  **d\_hiddenlayer = EH \* hiddengrad 'true', ':', 'true',**  **wout += hlayer\_act.T.dot(d\_output) \*lr**  **# dotproduct of nextlayererror and currentlayerop**  **bout += np.sum(d\_output, axis=0,keepdims=True) \*lr**  **wh += X.T.dot(d\_hiddenlayer) \*lr**  **#bh += np.sum(d\_hiddenlayer, axis=0,keepdims=True) \*lr**  **print("Input: \n" + str(X))**  **print("Actual Output: \n" + str(y))**  **print("Predicted Output: \n" ,output)** |

**OUTPUT**

1.



**CONCLUSION**

In this experiment we have implemented ANN with backpropagation in python.