1. EXPERIMENT NO: 4

2. TITLE: **BACKPROPAGATION ALGORITHM**

3. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python

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

5. THEORY:
   - Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples.
   - Algorithms such as BACKPROPAGATION gradient descent to tune network parameters to best fit a training set of input-output pairs.
   - ANN learning is robust to errors in the training data and has been successfully applied to problems such as interpreting visual scenes, speech recognition, and learning robot control strategies.

Backpropogation algorithm

1. Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.
2. Initialize each wi to some small random value (e.g., between -.05 and .05).
3. Until the termination condition is met, do

   For each training example $<(x_1,\ldots x_n),t>$, do

   // Propagate the input forward through the network:

   a. Input the instance $(x_1, ..,x_n)$ to the n/w & compute the n/w outputs o_k for every unit

   // Propagate the errors backward through the network:

   b. For each output unit k, calculate its error term δ_k ; $\delta_k = o_k(1-o_k)(t_k-o_k)$

   c. For each hidden unit h, calculate its error term δ_h; $\delta_h=o_h(1-o_h) \sum_k w_{h,k} \delta_k$

   d. For each network weight $w_{i,j}$ do; $w_{i,j}=w_{i,j}+\Delta w_{i,j}$ where $\Delta w_{i,j}= \eta \delta_j x_{i,j}$

6. PROCEDURE / PROGRAMME :

```
import numpy as np  # numpy is commonly used to process number array

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # Features ( Hrs Slept, Hrs Studied)
y = np.array(([92], [86], [89]), dtype=float)       # Labels(Marks obtained)

X = X/np.amax(X,axis=0) # Normalize
y = y/100

def sigmoid(x):
   return 1/(1 + np.exp(-x))
def sigmoid_grad(x):
   return x * (1 - x)

# Variable initialization
epoch=1000              #Setting training iterations
eta =0.2                #Setting learning rate (eta)
input_neurons  = 2      #number of features in data set
hidden_neurons = 3      #number of hidden layers neurons
output_neurons = 1      #number of neurons at output layer
```

```
# Weight and bias - Random initialization
wh=np.random.uniform(size=(input_neurons,hidden_neurons))      # 2x3
bh=np.random.uniform(size=(1,hidden_neurons))                  # 1x3
wout=np.random.uniform(size=(hidden_neurons,output_neurons))  # 1x1
bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
  #Forward Propogation
    h_ip=np.dot(X,wh) + bh          # Dot product + bias
    h_act = sigmoid(h_ip)           # Activation function
    o_ip=np.dot(h_act,wout) + bout
    output = sigmoid(o_ip)

  #Backpropagation
    # Error at Output layer
    Eo = y-output                          # Error at o/p
    outgrad = sigmoid_grad(output)
    d_output = Eo* outgrad                 # Errj=Oj(1-Oj)(Tj-Oj)

    # Error at Hidden later
    Eh = d_output.dot(wout.T)              # .T means transpose
    hiddengrad = sigmoid_grad(h_act)       # How much hidden layer wts contributed to error
    d_hidden = Eh * hiddengrad
    wout += h_act.T.dot(d_output) *eta     # Dotproduct of nextlayererror and currentlayerop
    wh += X.T.dot(d_hidden) *eta

print("Normalized Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

7. RESULTS & CONCLUSIONS:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
[[0.89427812]
 [0.88503667]
 [0.89099058]]
```

8. LEARNING OUTCOMES :

- The student will be able to build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

9. APPLICATION AREAS:

- Speech recognition, Character recognition, Human Face recognition

10. REMARKS: