Experiment 04 : Write a Program for Error Back Propagation Algorithm (EBPA) Learning.

Learning Objective : Write a Program for Error Back Propagation Algorithm (EBPA) Learning.

Tools : Python

Theory :

The Error Back Propagation Algorithm (EBPA) is a supervised learning algorithm used in artificial neural networks (ANNs) for training multi-layer feedforward networks. It is an iterative algorithm that adjusts the weights of the connections between the neurons in the network to minimize the difference between the predicted output and the actual output of the network.

The EBPA learning algorithm consists of two main phases: forward propagation and backward propagation. In the forward propagation phase, the input data is fed into the network, and the output is computed by passing the input through the network's layers.

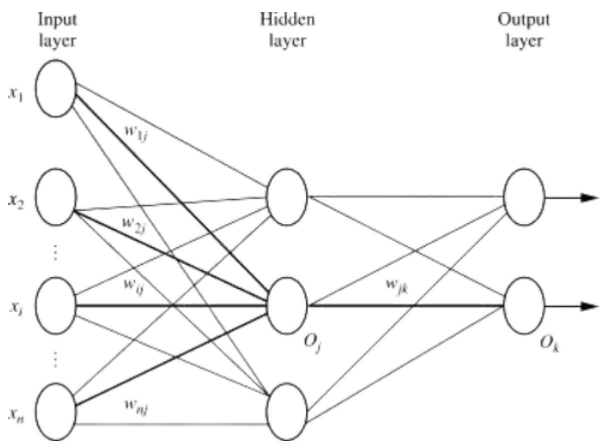
In the backward propagation phase, the error between the predicted output and the actual output is computed, and the algorithm adjusts the weights of the connections between the neurons in the network to minimize this error. The algorithm does this by propagating the error backwards through the network, layer by layer, and adjusting the weights of the connections between the neurons based on the error gradient.

The algorithm uses a loss function to measure the error between the predicted output and the actual output. The most commonly used loss function is the mean squared error (MSE) function. The error gradient is computed by taking the derivative of the loss function with respect to the weights of the connections.

The weights of the connections are then updated using the gradient descent algorithm, which adjusts the weights in the direction of the steepest descent of the error function. The learning rate, which determines the step size of the weight update, is a hyperparameter that needs to be set during the training process.

The training process continues iteratively until the error between the predicted output and the actual output is below a certain threshold or until a predetermined number of iterations is reached. At the end of the training process, the weights of the connections between the neurons in the network are optimized to produce accurate predictions on new input data.

EBPA learning is a powerful technique for training neural networks and has been used in many applications, including image recognition, speech recognition, and natural language processing. However, the algorithm is prone to overfitting and can be computationally expensive for large networks. Therefore, regularization techniques and optimization algorithms have been developed to improve the performance of the algorithm.



Step 1: Inputs X, arrive through the preconnected path.

Step 2: The input is modeled using true weights W. Weights are usually chosen randomly.

Step 3: Calculate the output of each neuron from the input layer to the hidden layer to the output layer.

Step 4: Calculate the error in the outputs

Backpropagation Error= Actual Output – Desired Output

Step 5: From the output layer, go back to the hidden layer to adjust the weights to reduce the error.

Step 6: Repeat the process until the desired output is achieved.

- x = inputs training vector $x=(x_1,x_2,\ldots\ldots\ldots\ldots x_n)$.
- t = target vector $t=(t_1,t_2\ldots\ldots\ldots\ldots t_n)$.
- δ_k = error at output unit.
- δ_j = error at hidden layer.
- α = learning rate.
- V_{0j} = bias of hidden unit j.

Training Algorithm :

- Step 1: Initialize weight to small random values.
- Step 2: While the steps stopping condition is to be false do step 3 to 10.
- Step 3: For each training pair do step 4 to 9 (Feed-Forward).
- Step 4: Each input unit receives the signal unit and transmits the signal x_i signal to all the units.
- Step 5 : Each hidden unit Zj (z=1 to a) sums its weighted input signal to calculate its net input

$z_{inj} = v_{0j} + \Sigma x_i v_{ij}$ (i=1 to n)

Applying activation function $z_j = f(z_{inj})$ and sends this signals to all units in the layer about i.e output units

For each output l=unit y_k = (k=1 to m) sums its weighted input signals.

$y_{ink} = w_{0k} + \Sigma\, z_i w_{jk}$ (j=1 to a)

and applies its activation function to calculate the output signals.

$y_k = f(y_{ink})$

Backpropagation Error :

- Step 6: Each output unit y_k (k=1 to n) receives a target pattern corresponding to an input pattern then error is calculated as:

  $\delta_k = (t_k - y_k) + y_{ink}$

- Step 7: Each hidden unit Z_j (j=1 to a) sums its input from all units in the layer above

  $\delta_{inj} = \Sigma\, \delta_j\, w_{jk}$

  The error information term is calculated as :

  $\delta_j = \delta_{inj} + z_{inj}$

Updation of weight and bias :

- Step 8: Each output unit y_k (k=1 to m) updates its bias and weight (j=1 to a). The weight correction term is given by :

  $\Delta\, w_{jk} = \alpha\, \delta_k\, z_j$

  and the bias correction term is given by $\Delta w_k = \alpha\, \delta_k$.

  therefore $w_{jk(new)} = w_{jk(old)} + \Delta\, w_{jk}$

  $w_{0k(new)} = w_{ok(old)} + \Delta\, w_{ok}$

  for each hidden unit z_j (j=1 to a) update its bias and weights (i=0 to n) the weight connection term

  $\Delta\, v_{ij} = \alpha\, \delta_j\, x_i$

  and the bias connection on term

  $\Delta\, v_{0j} = \alpha\, \delta_j$

  Therefore $v_{ij(new)} = v_{ij(old)} + \Delta v_{ij}$

  $v_{0j(new)} = v_{0j(old)} + \Delta v_{0j}$

- Step 9: Test the stopping condition. The stopping condition can be the minimization of error, number of epochs.

Implementation :

[2]:
```
import numpy as np
```

[3]:
```
# Define the activation function (sigmoid function)
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

[4]:
```
# Define the derivative of the activation function
def sigmoid_derivative(x):
    return x * (1 - x)
```

[5]:
```
# Define the Error Back Propagation Algorithm
def backpropagation(X, y, num_epochs, learning_rate):
    # Initialize random weights
    np.random.seed(1)
    weights_1 = 2 * np.random.random((3, 4)) - 1
    weights_2 = 2 * np.random.random((4, 1)) - 1

    # Training loop
    for epoch in range(num_epochs):
        # Forward propagation
        layer_1 = sigmoid(np.dot(X, weights_1))
        layer_2 = sigmoid(np.dot(layer_1, weights_2))

        # Back propagation
        error = y - layer_2
        delta_layer_2 = error * sigmoid_derivative(layer_2)
        delta_layer_1 = delta_layer_2.dot(weights_2.T) *␣
 ↪sigmoid_derivative(layer_1)

        # Update weights
        weights_2 += learning_rate * layer_1.T.dot(delta_layer_2)
        weights_1 += learning_rate * X.T.dot(delta_layer_1)

    return layer_2
```

[6]:
```
# Test the backpropagation algorithm
X = np.array([[0, 0, 1],
              [0, 1, 1],
              [1, 0, 1],
              [1, 1, 1]])

y = np.array([[0],
              [1],
              [1],
              [0]])
```

[7]:
```
num_epochs = 10000
learning_rate = 0.1

output = backpropagation(X, y, num_epochs, learning_rate)
print(output)
```

```
[[0.03499461]
 [0.95478633]
 [0.96201778]
 [0.05337798]]
```

<u>Result and Discussion</u> :

Learning Outcomes : Students should have the ability to

LO 4.1: Ability to understand how error back propagation works.
LO 4.2: Ability to implement the algorithm for error back propagation.

Course Outcomes :

CO : Ability to implement the algorithm for error back propagation.

Conclusion :

Viva Questions :

Q1. What is error backpropagation, and how is it used in training artificial neural networks?
Q2. What is the role of the loss function in the error backpropagation algorithm, and how is it chosen?

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	Total
Marks Obtained				