

Lab Experiment 7 & 8: Backpropagation

AIM: Program to Implement Backpropagation Neural Network

Theoretical Description/ Algorithm: In machine learning, backpropagation is an effective algorithm used to train artificial neural networks, especially in feed-forward neural networks. Backpropagation is an iterative algorithm, that helps to minimize the cost function by determining which weights and biases should be adjusted. During every epoch, the model learns by adapting the weights and biases to minimize the loss by moving down toward the gradient of the error. Thus, it involves the two most popular optimization algorithms, such as gradient descent or stochastic gradient descent

Backpropagation Algorithm:

1. Initialization:

- Initialize weights with small random values.
- Set the learning rate (α), a small constant that controls the size of weight updates.

2. Forward Propagation:

- For each training example:
 - Pass the input through the network to generate the output.
 - Compute the output of each neuron in the hidden and output layers using the activation function.
 - Calculate the loss by comparing the predicted output with the actual output.

3. Backpropagation of Errors:

- **Step 1: Compute Output Layer Error:**
 - Calculate the error at the output neurons.
 - For each output neuron j , the error δ_j is:

$$\delta_j = (y_j - \hat{y}_j) \cdot f'(z_j)$$

where y_j is the actual output, \hat{y}_j is the predicted output, z_j is the weighted sum of inputs to neuron j , and $f'(z_j)$ is the derivative of the activation function (often sigmoid or ReLU).

- **Step 2: Compute Hidden Layer Error:**
 - For each neuron in the hidden layers, the error is propagated back from the output layer. The error δ_h for hidden neuron h is:

$$\delta_h = \sum_j \delta_j \cdot w_{jh} \cdot f'(z_h)$$

Where w_{jh} is the weight from hidden neuron h to output neuron j , and $f'(z_h)$ is the derivative of the activation function for hidden neuron h .

4. Weight Updates:

- After calculating the errors for all neurons:
 - For each weight w_{ij} (connecting neuron i to neuron j), update the weight using gradient descent:

$$w_{ij} = w_{ij} - \alpha \cdot \delta_j \cdot a_i$$

where α is the learning rate, δ_j is the error term for the neuron j , and a_i is the activation of neuron i .

5. Repeat:

- Repeat the forward propagation, backpropagation, and weight update steps for each training example in the dataset (typically for multiple epochs) until the loss converges or stops improving.

Source Code:

```
import numpy as np

# Input and output samples
input_samples = np.array([[0, 1, 0],
                           [1, 0, 1],
                           [0, 0, 1],
                           [1, 1, 0]])
output_samples = np.array([[0],[1],[0],[1]])

# Initialize weights randomly
weights = np.random.random((3, 1))

# Print the initial weights
print("Initial Weights:")
print(weights)

# Training the model for 2000 iterations
for iteration in range(2000):
    # Calculate the weighted sum with bias
    sum = np.dot(input_samples, weights) + 0.02 # 0.02 is the bias value

    # Sigmoid activation function
    def activation(x):
        return 1 / (1 + np.exp(-x))

    ypred = activation(sum)

    # Calculate error
    error = output_samples - ypred

    # Gradient function for sigmoid derivative
    def gradient(x):
        return x * (1 - x)

    adjustment = error * gradient(ypred) # alpha * dy/dx
    weights += np.dot(input_samples.T, adjustment)
```

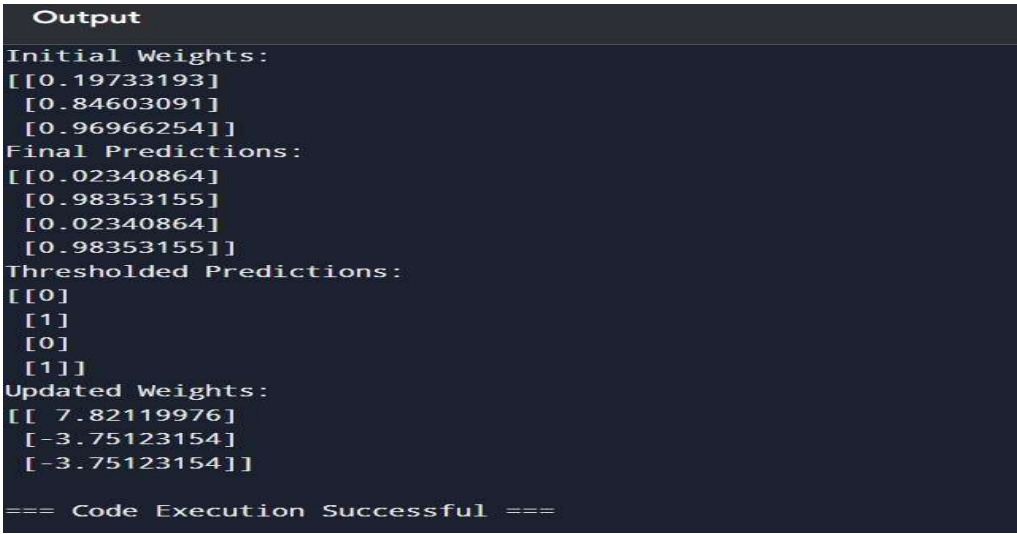
```
# Final predictions after training
print("Final Predictions:")
print(ypred)

# Apply threshold to predictions
thresholded_predictions = (ypred >= 0.5).astype(int)

# Print the thresholded predictions
print("Thresholded Predictions:")
print(thresholded_predictions)

# Print the updated weights after training
print("Updated Weights:")
print(weights)
```

OUTPUT:



```
Output
Initial Weights:
[[0.19733193]
 [0.84603091]
 [0.96966254]]
Final Predictions:
[[0.02340864]
 [0.98353155]
 [0.02340864]
 [0.98353155]]
Thresholded Predictions:
[[0]
 [1]
 [0]
 [1]]
Updated Weights:
[[ 7.82119976]
 [-3.75123154]
 [-3.75123154]]

=== Code Execution Successful ===
```