Experiment No. 10

Expt No. 10	Implementation of Back-propagation Algorithm of Neural Network
Date :	

Aim:

To study and implement of Backpropagation algorithm of a Neural Network.

System Requirements:

- Operating System: Windows 8 or above / Linux / macOS
- Memory (RAM): Minimum 4 GB
- Processor: Minimum 2.33 GHz (Dual Core or higher)

Software/Tools Required:

Jupyter Notebook/ Anaconda Navigator/ Google Colaboratory/ Spyder, Python 3.x [With libraries such as Numpy, Pandas, Matplotlib, Scikit-Learn and Tensorflow/Keras]

Expected Outcomes:

- To understand the fundamentals of Backpropagation algorithm
- To implement forward and backward passes in a neural network
- To interpret the roles of activation functions
- To evaluate network performance and learning process

Theory:

Backpropagation is a supervised learning algorithm used for training neural networks. It computes the gradient of the loss function with respect to each weight by applying the chain rule, moving backward through the network, layer by layer. By iteratively adjusting the weights, backpropagation minimizes the error between predicted and actual outputs.

Backpropagation is an elegant method to calculate how changes to any of the weights or biases of a neural network will affect the accuracy of model predictions

The logic of backpropagation is that the layers of neurons in artificial neural networks are essentially a series of nested mathematical functions. During training, those interconnected equations are nested into yet another function: a "loss function" that measures the difference (or "loss") between the desired output (or "ground truth") for a given input and the neural network's actual output.

Why use Backpropagation?
The network is trained using 2 passes: forward and backward. At the end of the forward pass, the network error is calculated, and should be as small as possible.

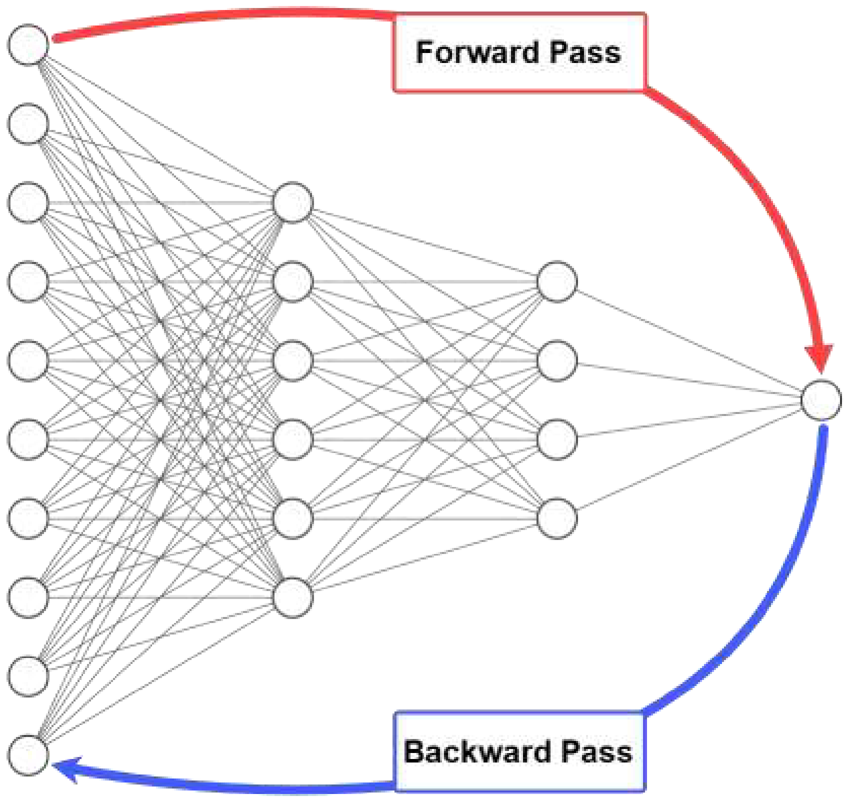
If the current error is high, the network didn't learn properly from the data. What does this mean?

It means that the current set of weights isn't accurate enough to reduce the network error and make accurate predictions. As a result, we should update network weights to reduce the network error.

The backpropagation algorithm is one of the algorithms responsible for updating network weights with the objective of reducing the network error.

How does Backpropagation work?
The training neural networks with backpropagation entails the following steps –

- A forward pass making predictions on training data
- A loss function measures the error of the model's predictions during that forward pass
- Backpropagation of error, or a backward pass to calculate the partial derivatives of the loss function
- Gradient Descent to update the model weights



What is a Forward pass?
Neural networks output predictions through forward propagation. Forward propagation is essentially a long series of nested equations, with the outputs of the activation functions from one layer of neurons serving as inputs to the activation functions of neurons in the next layer.
In each forward pass, an input is sampled from the training data set. The nodes of the input layer receive the input vector, and each passes their value—multiplied by some random initial weight—to the nodes of the first hidden layer. The hidden units take the weighted sum of these output values as input to an activation function, whose output value (conditioned by a random initial weight) serves as input to the neurons in the next layer. This continues until the output layer, where a final prediction occurs.

A simplified example of a neural network that classifies inputs into one of 5 categories:

- **The input layer** receives a numerical representation of an example sampled from the training data.
- **The input nodes** pass their values to hidden units in the next layer. The hidden units use a ReLU activation function.
- **Data flows** through the hidden layers, each progressively extracting key features until it reaches the output layer.

- **The output layer** contains 5 neurons, each corresponding to a potential classification category.
- **The output neurons** use a softmax activation function. The output value of each output neuron's softmax function corresponds to the probability, out of 1, that the input should be classified as the category that the neuron represents.
- **The network** predicts that the original input belongs to the category of whichever output neuron has the highest softmax value

<u>Procedure to implement a Neural Network:</u>

1. Data Preparation:
   - Use a sample dataset for binary classification like a small sample with two features
2. Defining the Neural Network from Scratch:
   - Implement a Neural Network with one hidden layer
   - Initialize weights and biases for each layer randomly
3. Forward Pass:
   - Calculate the output of each neuron using weighted inputs and activation functions
   - Use the Sigmoid function as the activation function for simplicity
4. Compute the Loss:
   - Use Mean Squared Error (MSE) for loss calculation
5. Implement Backpropagation:
   - Derive the gradients of the loss with respect to each weight using partial derivatives
   - Apply the chain rule to propagate gradients back through each layer
6. Update weights:
   - Use a simple gradient descent update rule to adjust weights and biases based on
   - calculated gradients
7. Iterate over Epochs:
   - Repeat the forward and backward passes for a fixed number of epochs
   - Track and observe the loss over epochs to monitor the training progress

Sample Code:

```
import numpy as np

# Step 1: Define a small dataset (e.g., XOR)

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

# Step 2: Initialize parameters
input_size = 2                          # Input layer size
hidden_size = 2                          # Hidden layer size
output_size = 1                          # Output layer size
learning_rate = 0.1

# Initialize weights and biases

np.random.seed(42)
weights_input_hidden = np.random.rand(input_size, hidden_size)
weights_hidden_output = np.random.rand(hidden_size, output_size)
bias_hidden = np.random.rand(1, hidden_size)
bias_output = np.random.rand(1, output_size)

# Activation function (Sigmoid) and its derivative

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

def sigmoid_derivative(x):
    return x * (1 - x)

# Step 3-7: Training the network using Backpropagation
epochs = 10000
for epoch in range(epochs):

 # Forward Pass
    hidden_input = np.dot(X, weights_input_hidden) + bias_hidden
    hidden_output = sigmoid(hidden_input)
    final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
    final_output = sigmoid(final_input)
    # Compute Loss (Mean Squared Error)
    loss = np.mean((y - final_output) ** 2)

    # Backward Pass (Backpropagation)
    error_output = y - final_output
    d_output = error_output * sigmoid_derivative(final_output)
    error_hidden = d_output.dot(weights_hidden_output.T)
    d_hidden = error_hidden * sigmoid_derivative(hidden_output)

    # Update weights and biases
    weights_hidden_output += hidden_output.T.dot(d_output) * learning_rate
    weights_input_hidden += X.T.dot(d_hidden) * learning_rate
    bias_output += np.sum(d_output, axis=0, keepdims=True) * learning_rate
    bias_hidden += np.sum(d_hidden, axis=0, keepdims=True) * learning_rate

    # Print loss every 1000 epochs
    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.4f}")

# Final Output
print("\nTrained Neural Network Output:")
print(final_output)
```

Observations -

- Track the loss over epochs to observe how the network learns and reduces error
- Compare initial and final outputs to assess if the network has learned the underlying patterns in the dataset

Conclusion –
Hence, the model summarizes the effectiveness of backpropagation in training the neural network and discuss its impact on minimizing loss and mention potential adjustments for further improvements.

References –
a. Textbook –

i. Machine Learning with Python – An approach to Applied ML – Abhishek Vijayvargiya,
BPB Publications, 1st Edition 2018
ii. Machine Learning, Tom Mitchell, McGraw Hill Education, 1st Edition 1997

b. Online references –

i. https://www.ibm.com/think/topics/backpropagation
ii. https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd
iii. https://www.datacamp.com/tutorial/mastering-backpropagation

Viva Questions

Sr.No	Question	CO
1.	What is backpropagation in the context of neural networks? Describe its purpose	**1ICPC406_5**
2.	2. What are the main steps involved in the backpropagation algorithm?	**1ICPC406_5**
3.	3. Why is backpropagation essential for training neural networks?	**1ICPC406_5**
4.	4. What is the role of loss function in backpropagation?	**1ICPC406_5**
5.	5. What is the chain rule and why is it important in backpropagation?	**1ICPC406_5**
6.	6. Explain how backpropagation adjusts weights in a multi-layer neural network	**1ICPC406_5**
7.	7. Describe the impact of the learning rate on the backpropagation process. What happens if the	**1ICPC406_5**
8.	learning rate is too high or too low?	**1ICPC406_5**
9.	8. Discuss the ways to address the limitations of backpropagation	**1ICPC406_5**
10.	9. Why is backpropagation considered a form of supervised learning?	**1ICPC406_5**