Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Experiment 7: Analyse the forward pass and backward pass of back propagation algorithm.

Aim/Objective: To analyze and implement the forward pass and backward pass of the backpropagation algorithm, a key component in training neural networks. The experiment aims to provide an in-depth understanding of the computations involved in both passes, emphasizing the flow of information during training.

Description: In this lab experiment, the goal is to delve into the mechanics of the backpropagation algorithm, breaking it down into the forward pass and backward pass. The forward pass involves the computation of the model's predictions, while the backward pass computes gradients with respect to the model parameters. The experiment includes coding the algorithm from scratch, gaining insights into the role of each step-in training neural networks.

Pre-Requisites: Basic knowledge of Neural Networks, Calculus and Chain Rule, Python Programming, Machine Learning Basics, Linear Algebra Basics.

Pre-Lab:

1. What is the primary objective of the backpropagation algorithm in the context of training neural networks?

Minimize error by adjusting weights using gradient descent.

2. Briefly describe the computations involved in the forward pass of the backpropagation algorithm. What is the output of the forward pass?

Computes weighted sums, applies activation functions, and produces predicted outputs.

3. What is the purpose of the backward pass in the backpropagation algorithm? How does it contribute to updating the model parameters?

Calculates gradients, updates weights to reduce error, and improves model performance.

4. Explain the role of the chain rule in backpropagation.

Propagates gradients layer by layer to update weights efficiently.

In-Lab:

Program 1: Analyze the forward pass and backward pass of back propagation algorithm for the network using your own initiate, forward, backward and loss functions. Only use NumPy , matplot libraries only. It should simulate keras methods.

Procedure/Program:

```
import numpy as np
import matplotlib.pyplot as plt


class SimpleNN:

    def __init__(self, input_size, hidden_size, output_size, lr=0.5):
        self.lr = lr
        self.W1, self.b1 = np.random.randn(input_size, hidden_size) * 0.01, np.zeros((1,
hidden_size))
        self.W2, self.b2 = np.random.randn(hidden_size, output_size) * 0.01, np.zeros((1,
output_size))

    def sigmoid(self, Z):
        return 1 / (1 + np.exp(-Z))

    def sigmoid_deriv(self, A):
        return A * (1 - A)

    def loss(self, Y, Y_pred):
        return np.mean((Y - Y_pred) ** 2)

    def forward(self, X):
        self.A1 = self.sigmoid(np.dot(X, self.W1) + self.b1)
        self.A2 = self.sigmoid(np.dot(self.A1, self.W2) + self.b2)
        return self.A2

    def backward(self, X, Y):
        dA2 = -(Y - self.A2) * self.sigmoid_deriv(self.A2)
        dW2, db2 = np.dot(self.A1.T, dA2), np.sum(dA2, axis=0, keepdims=True)
        dA1 = np.dot(dA2, self.W2.T) * self.sigmoid_deriv(self.A1)
        dW1, db1 = np.dot(X.T, dA1), np.sum(dA1, axis=0, keepdims=True)
```

```
        for param, dparam in zip([self.W1, self.b1, self.W2, self.b2], [dW1, db1, dW2, db2]):
            param -= self.lr * dparam

    def train(self, X, Y, epochs=5000):
        losses = [self.loss(Y, self.forward(X)) for _ in range(epochs) if not self.backward(X, Y)]

        plt.plot(losses)
        plt.xlabel("Epochs")
        plt.ylabel("Loss")
        plt.title("Loss Curve")
        plt.show()


np.random.seed(42)
X, Y = np.array([[0,0], [0,1], [1,0], [1,1]]), np.array([[0], [1], [1], [0]])

nn = SimpleNN(2, 2, 1)
nn.train(X, Y)

print(nn.forward(X))
```

- **Data and Results:**

Data: This dataset consists of XOR inputs and corresponding output labels.

Result: The neural network successfully learns to approximate XOR function.

- **Analysis and Inferences:**

Analysis: Loss decreases over epochs, improving network prediction accuracy.

Inferences: The trained model correctly classifies XOR inputs using backpropagation.

Sample VIVA-VOCE Questions (In-Lab):

1. How do different activation functions impact the computations in the forward pass and gradients in the backward pass of the backpropagation algorithm?

- Affect output range, non-linearity, and gradient flow.
- ReLU avoids vanishing gradients, while Sigmoid/Tanh may cause them.

2. Neural networks use non-linear activation functions. Why is non-linearity crucial for the success of backpropagation, and how does it help in capturing complex relationships in data?

- Prevents layers from collapsing into a single linear function.
- Helps capture complex patterns for better learning.

3. In the context of backpropagation, what role does the learning rate play during the parameter update step?

- Controls step size in weight updates.
- Too high: instability; too low: slow convergence.

4. How does backpropagation contribute to the potential issue of overfitting, and what regularization techniques can be employed during training to address this concern?

- Can lead to overfitting by memorizing training data.
- Regularization: L1/L2, Dropout, Batch Norm, Data Augmentation.

Post-Lab:

Program 2: How would the forward pass, backward pass, and weight updates change if you added another hidden layer to the network? Implement and describe the changes required in your code for this modification.

Procedure/Program:

```
import numpy as np

sigmoid = lambda x: 1 / (1 + np.exp(-x))
sigmoid_derivative = lambda x: x * (1 - x)

np.random.seed(42)

W1 = np.random.rand(3, 4)
W2 = np.random.rand(4, 4)
W3 = np.random.rand(4, 1)

def train(X, y, lr=0.01):
   global W1, W2, W3

   H1 = sigmoid(X @ W1)
   H2 = sigmoid(H1 @ W2)
   O = sigmoid(H2 @ W3)

   dO = (y - O) * sigmoid_derivative(O)
   dH2 = dO @ W3.T * sigmoid_derivative(H2)
   dH1 = dH2 @ W2.T * sigmoid_derivative(H1)

   W3 += H2.T @ dO * lr
   W2 += H1.T @ dH2 * lr
   W1 += X.T @ dH1 * lr

   return O
```

- **Data and Results:**

Data

The dataset contains input features and target values for training.

Result

The model successfully learns patterns and improves prediction accuracy.

- **Analysis and Inferences:**

Analysis

Weight updates refine network connections, optimizing learning efficiency progressively.

Inferences

Deeper networks capture complex relationships but require careful parameter tuning.

Evaluator Remark (if Any):	**Marks Secured ______ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.