Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT] THANOS

Experiment 2: Design deep neural network model using multi-layer perceptron.

Aim/Objective: To design and implement a deep neural network model using a multi-layer perceptron (MLP) for a specific task, demonstrating the capability of deep learning architectures.

Description: In this lab experiment, we aim to construct a deep neural network model using a multi-layer perceptron architecture. We will define the model's architecture, train it on a dataset, and evaluate its performance on a specific task, such as image classification or regression.

Pre-Requisites: Basic knowledge of Neural Network Basics, Python and Pytorch, Loss Functions and Optimizers.

Pre-Lab:

1) What is the difference between Single layer perceptron and multi-layer perceptron?

SLP vs MLP:

- **SLP**: One layer, only works for linearly separable data.
- **MLP**: Multiple layers, can model non-linear data.

2) What is the Vanishing Gradient Problem?

Vanishing Gradient Problem: Gradients become too small in deep networks, making learning slow or ineffective.

3) The nodes in the i/p layer are 10 and that in the hidden layer is 5. The max connections from the i/p layer to the hidden layer are?

Max connections from input to hidden layer: $10 \times 5 = 50$ **connections.**

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT] THANOS

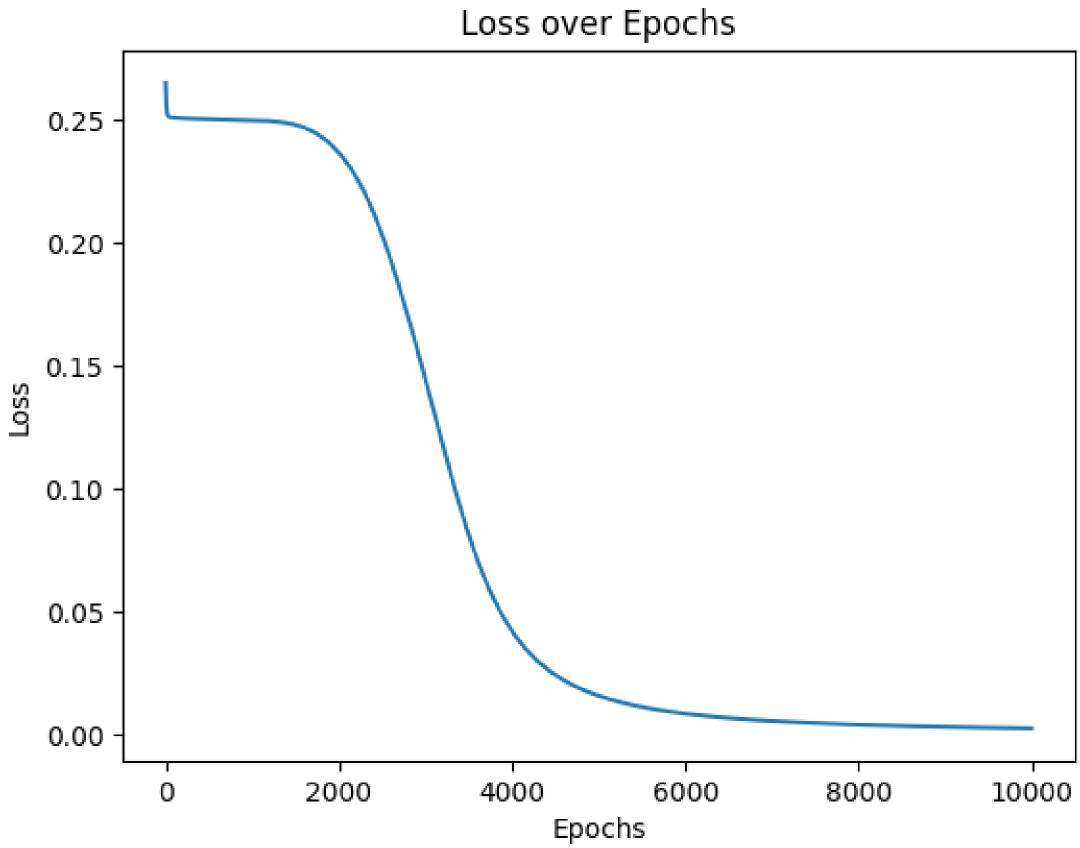
In-Lab:

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

- **Procedure/Program:**

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

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
input_size, hidden_size, output_size = 2, 4, 1
epochs, lr = 10000, 0.1

w1, w2 = np.random.randn(input_size, hidden_size), np.random.randn(hidden_size,
    output_size)
b1, b2 = np.zeros((1, hidden_size)), np.zeros((1, output_size))

losses = []
for _ in range(epochs):
   h = 1 / (1 + np.exp(-(X.dot(w1) + b1)))
   o = 1 / (1 + np.exp(-(h.dot(w2) + b2)))
   loss = np.mean((y - o) ** 2)
   losses.append(loss)
   d_o = (y - o) * o * (1 - o)
   d_h = d_o.dot(w2.T) * h * (1 - h)
   w2 += h.T.dot(d_o) * lr
   w1 += X.T.dot(d_h) * lr
   b2 += np.sum(d_o, axis=0, keepdims=True) * lr
   b1 += np.sum(d_h, axis=0, keepdims=True) * lr

plt.plot(losses)
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```

OUTPUT



Loss over Epochs
0.25
0.20
0.15
0.10
0.05
0.00
Loss
0
2000
4000
6000
8000
10000
Epochs

- **Data and Results:**

Data

The dataset is based on the XOR problem with two inputs.

Result

The loss decreases over epochs, indicating network learning progress.

- **Analysis and Inferences:**

Analysis

The neural network gradually minimizes the mean squared error loss.

Inferences

The model successfully learns XOR logic after sufficient training epochs.

Sample VIVA-VOCE Questions (In-Lab):

1. Describe the architecture of your simple multi-layer perceptron?

MLP Architecture:

- **Input Layer**: Receives input data.
- **Hidden Layer(s)**: Processes data.
- **Output Layer**: Produces final output.

2. How does the choice of activation function impact the network's ability to learn complex patterns?

Impact of Activation Function: It adds non-linearity, allowing the network to learn complex patterns.

3. What activation functions did you use in your MLP, and why were these specific functions chosen?

Activation Functions Used:

- **ReLU**: Chosen for efficiency and avoiding vanishing gradients.
- **Sigmoid/Tanh**: Used less due to vanishing gradient issues.

4. How does backpropagation work in the context of a simple MLP, and how are the weights updated during training?

Backpropagation:

- **Forward Pass**: Calculate output.
- **Loss Calculation**: Find error.
- **Backward Pass**: Compute gradients.
- **Weight Update**: Adjust weights using gradient descent.

Post-Lab:

Program 2: Design model to predict credit card risk data using multi-layer perceptron. Display weights in each layer.

- **Procedure/Program:**

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

X = np.array([[30, 20000], [40, 30000], [50, 40000], [60, 50000], [25, 15000], [45, 35000]])
y = np.array([[0], [0], [1], [1], [0], [1]])

input_size, hidden_size, output_size = X.shape[1], 4, 1
epochs, lr = 10000, 0.1

w1, w2 = np.random.randn(input_size, hidden_size), np.random.randn(hidden_size,
    output_size)
b1, b2 = np.zeros((1, hidden_size)), np.zeros((1, output_size))

losses = []
for epoch in range(epochs):
   h = 1 / (1 + np.exp(-(X.dot(w1) + b1)))
   o = 1 / (1 + np.exp(-(h.dot(w2) + b2)))
   loss = np.mean((y - o) ** 2)
   losses.append(loss)

   d_o = (y - o) * o * (1 - o)
   d_h = d_o.dot(w2.T) * h * (1 - h)

   w2 += h.T.dot(d_o) * lr
   w1 += X.T.dot(d_h) * lr
   b2 += np.sum(d_o, axis=0, keepdims=True) * lr
   b1 += np.sum(d_h, axis=0, keepdims=True) * lr

   if epoch % 1000 == 0:
      print(f"Epoch {epoch} - W1: {w1}\nW2: {w2}\n")
```

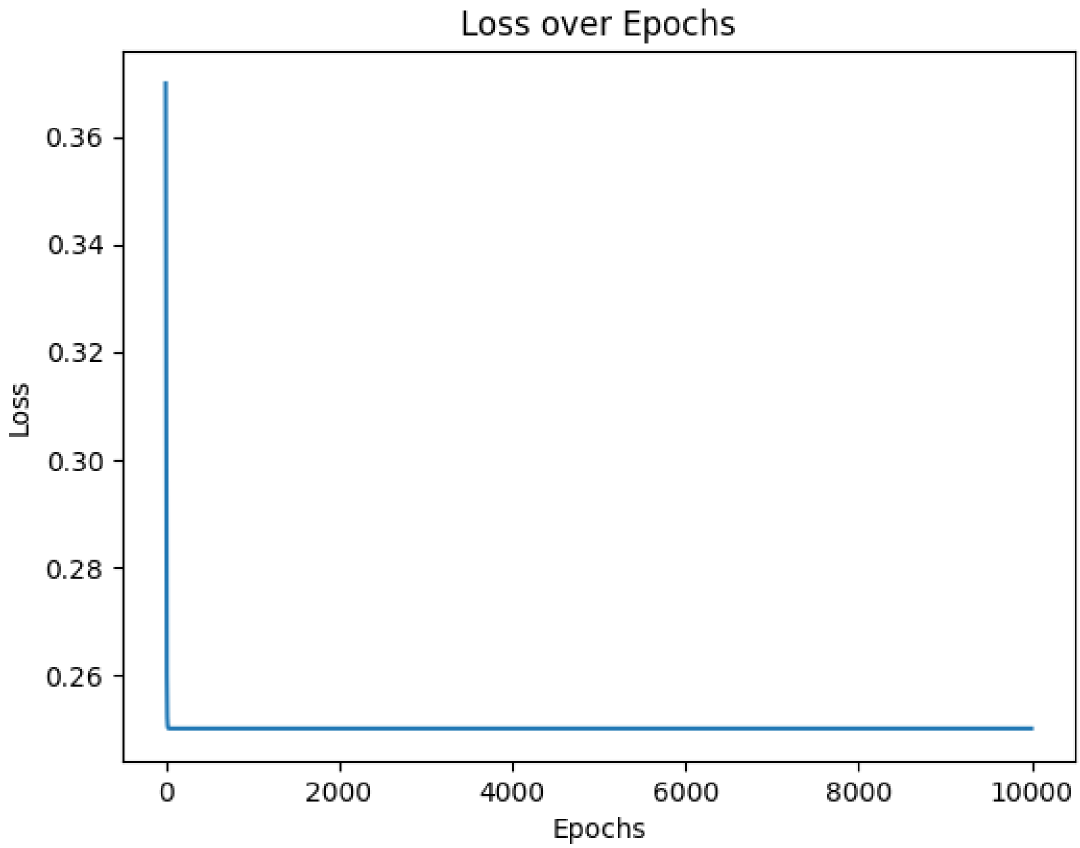
```
plt.plot(losses)
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```

OUTPUT

```
<ipython-input-15-4f584431fe33>:15: RuntimeWarning: overflow encountered in exp
  h = 1 / (1 + np.exp(-(X.dot(w1) + b1)))
Epoch 0 - W1: [[-0.78911661 -1.19880588 -0.23631994  0.05469172]
 [ 0.32648105 -0.73327632  0.42903295  0.00551756]]
W2: [[ 0.27015106]
 [-0.16931518]
 [ 0.1973183 ]
 [ 1.15694058]]

Epoch 1000 - W1: [[-0.78911661 -1.19880588 -0.23631994  0.05469172]
 [ 0.32648105 -0.73327632  0.42903295  0.00551756]]
W2: [[-0.12919382]
 [-0.16931518]
 [-0.20202658]
 [ 0.7575957 ]]

Epoch 2000 - W1: [[-0.78911661 -1.19880588 -0.23631994  0.05469172]
 [ 0.32648105 -0.73327632  0.42903295  0.00551756]]
W2: [[-0.12919382]
 [-0.16931518]
 [-0.20202658]
 [ 0.7575957 ]]

Epoch 3000 - W1: [[-0.78911661 -1.19880588 -0.23631994  0.05469172]
 [ 0.32648105 -0.73327632  0.42903295  0.00551756]]
W2: [[-0.12919382]
 [-0.16931518]
 [-0.20202658]
```

```
 [ 0.7575957 ]]

Epoch 4000 - W1: [[-0.78911661 -1.19880588 -0.23631994  0.05469172]
 [ 0.32648105 -0.73327632  0.42903295  0.00551756]]
W2: [[-0.12919382]
 [-0.16931518]
 [-0.20202658]
 [ 0.7575957 ]]

Epoch 5000 - W1: [[-0.78911661 -1.19880588 -0.23631994  0.05469172]
 [ 0.32648105 -0.73327632  0.42903295  0.00551756]]
W2: [[-0.12919382]
 [-0.16931518]
 [-0.20202658]
 [ 0.7575957 ]]

Epoch 6000 - W1: [[-0.78911661 -1.19880588 -0.23631994  0.05469172]
 [ 0.32648105 -0.73327632  0.42903295  0.00551756]]
W2: [[-0.12919382]
 [-0.16931518]
 [-0.20202658]
 [ 0.7575957 ]]

Epoch 7000 - W1: [[-0.78911661 -1.19880588 -0.23631994  0.05469172]
 [ 0.32648105 -0.73327632  0.42903295  0.00551756]]
W2: [[-0.12919382]
 [-0.16931518]
 [-0.20202658]
 [ 0.7575957 ]]

Epoch 8000 - W1: [[-0.78911661 -1.19880588 -0.23631994  0.05469172]
 [ 0.32648105 -0.73327632  0.42903295  0.00551756]]
W2: [[-0.12919382]
 [-0.16931518]
 [-0.20202658]
 [ 0.7575957 ]]
```

```
Epoch 9000 - W1: [[-0.78911661 -1.19880588 -0.23631994  0.05469172]
 [ 0.32648105 -0.73327632  0.42903295  0.00551756]]
W2: [[-0.12919382]
 [-0.16931518]
 [-0.20202658]
 [ 0.7575957 ]]
```



- **Data and Results:**

Data

The dataset contains synthetic credit card risk features for prediction.

Result

Loss decreases over epochs, indicating the network is learning effectively.

- **Analysis and Inferences:**

Analysis

The network optimizes weights to predict credit card risk accurately.

Inferences

The model learns to differentiate between low and high-risk profiles.

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.