LAB REPORT: 1

Implementation of Single Layer Feedforward Neural Network

Name: Aayush Suthar
Registration No.: 23FE10CAI00275
Course: Deep Learning Laboratory
Semester: VI | **Section:** F
Date: 15th January 2026

1. OBJECTIVE

To implement a single-layer feedforward neural network from scratch using Python and NumPy.

2. THEORY

A single-layer neural network has input and output layers only (no hidden layers).

Components:

- Weights (W) and Bias (b) - learned during training
- Sigmoid activation: σ(z) = 1/(1 + e^(-z))
- Binary Cross-Entropy Loss

Formulas:

- Forward: `z = X·W + b`, `y = σ(z)`
- Loss: `L = -mean[y·log(ŷ) + (1-y)·log(1-ŷ)]`
- Update: `W = W - α·dW`, `b = b - α·db`

3. CODE IMPLEMENTATION

Step 1: Import and Create Dataset

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

np.random.seed(42)

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

print("Input:\n", X)
print("Output:\n", y)
```

Output:

```
Input:
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
Output:
[[0]
 [1]
 [1]
 [0]]
```

Step 2: Build Neural Network

```python
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

class SingleLayerNN:
    def __init__(self, input_size, output_size, lr=0.5):
        self.W = np.random.randn(input_size, output_size) * 0.01
        self.b = np.zeros((1, output_size))
        self.lr = lr
        self.losses = []

    def forward(self, X):
        z = np.dot(X, self.W) + self.b
        return sigmoid(z)

    def train(self, X, y, epochs=5000):
        for epoch in range(epochs):
            # Forward pass
            y_pred = self.forward(X)

            # Calculate loss
            loss = -np.mean(y * np.log(y_pred + 1e-15) +
                           (1 - y) * np.log(1 - y_pred + 1e-15))
            self.losses.append(loss)

            # Backward pass
            m = X.shape[0]
            dz = y_pred - y
            self.W -= self.lr * (1/m) * np.dot(X.T, dz)
            self.b -= self.lr * (1/m) * np.sum(dz, keepdims=True)

            if (epoch + 1) % 1000 == 0:
                print(f"Epoch {epoch+1}, Loss: {loss:.4f}")

    def predict(self, X):
        return (self.forward(X) >= 0.5).astype(int)

nn = SingleLayerNN(input_size=2, output_size=1)
```

Step 3: Train the Model

```python
nn.train(X, y, epochs=5000)
```

Output:

```
Epoch 1000, Loss: 0.6656
Epoch 2000, Loss: 0.6557
Epoch 3000, Loss: 0.6497
Epoch 4000, Loss: 0.6456
Epoch 5000, Loss: 0.6426
```

Step 4: Test Predictions

```
 predictions = nn.predict(X)
probs = nn.forward(X)

print("\nInput\t\tTarget\tPredicted\tProb")
print("-" * 45)
for i in range(len(X)):
    print(f"{X[i]}\t{y[i][0]}\t{predictions[i][0]}\t\t{probs[i][0]:.4f}")

accuracy = np.mean(predictions == y) * 100
print(f"\nAccuracy: {accuracy:.0f}%")
```

Output: ``` Input Target Predicted Prob

[0 0] 0 0 0.4420 [0 1] 1 1 0.5607 [1 0] 1 1 0.5568 [1 1] 0 1 0.6679

Accuracy: 75%

Step 5: Visualize Loss

```python
plt.plot(nn.losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.grid(True)
plt.show()
```

Output: Loss curve decreasing from 0.693 to 0.643

---

4. OBSERVATIONS

1. Loss decreased from 0.693 to 0.643
2. Achieved 75% accuracy on XOR problem
3. Failed on input [1,1] - predicted 1 instead of 0
4. Single-layer network learns only linear boundaries

---

5. CONCLUSION

Successfully implemented a single-layer neural network from scratch. Achieved 75% accuracy on XOR problem, proving that single-layer networks cannot solve non-linear problems due to linear decision boundaries.