Neural Network

VISHAL GHASOLIYA- 24UADS1056

Propmpt:

Objective:

Visualize the Perceptron Learning Algorithm with step-by-step training details.

Requirements:

1. Implementation:

- Use NumPy and Matplotlib

- Initialize all weights to zero (including bias)

- Train on NAND and XOR gate datasets

2. Training Output (Each Epoch):

Print a table showing for every sample:

- Input values

- Target output

- Predicted output

- Error

- Updated weights

3. Visualization:

- Plot decision boundary after “each epoch”

- Show correctly classified points (one color/style)

- Show misclassified points (different color/style)

- Add “2-second delay” between epoch updates

- Continue until convergence or max epochs

4. Final Plot:

- Error vs. Epoch graph after training completes

Deliverables:

- Python code with zero-weight initialization
- Animated training visualization with 2s delays
- Epoch-by-epoch console output tables
- Error convergence plot

Code: # NAND

```
import numpy as np
import matplotlib.pyplot as plt
import time
# Step function (activation)
def step(z):
    return 1 if z >= 0
# Perceptron training function
def train_perceptron(X, T, lr=1.0, max_epochs=20):
  n_samples, n_features = X.shape
    # Initialize weights + bias = 0
    W = np.zeros(n_features)
    b = 0
     errors_per_epoch = []
     plt.ion()
   for epoch in range(max_epochs):
        total_error = 0
        print(f"\nEpoch {epoch+1}")
        print("x1  x2 | Target | Pred | Error | Weights (w1,w2,b)")
        for i in range(n_samples):
```

```
        x = X[i]
        t = T[i]

        z = np.dot(W, x) + b
        y = step(z)

        error = t - y
        total_error += abs(error)

        # Update rule
        W = W + lr * error * x
        b = b + lr * error

        print(f"{x[0]}  {x[1]}  |  {t}  |  {y}  |  {error}  | {W}, {b}")

    errors_per_epoch.append(total_error)

    plot_decision_boundary(X, T, W, b, epoch)

    time.sleep(2)

    if total_error == 0:
        print("\n✓ Converged!")
        break

plt.ioff()
plot_error_curve(errors_per_epoch)
```

```
# Plot decision boundary
def plot_decision_boundary(X, T, W, b, epoch):
    plt.clf()
    # Separate correct/misclassified
    for i in range(len(X)):
        pred = step(np.dot(W, X[i]) + b)
        if pred == T[i]:
            plt.scatter(X[i][0], X[i][1], c='green', s=100)
        else:
            plt.scatter(X[i][0], X[i][1], c='red', s=100, marker='x')
    # Decision boundary: w1*x + w2*y + b = 0
    if W[1] != 0:
        x_vals = np.array([-0.5, 1.5])
        y_vals = -(W[0]*x_vals + b) / W[1]
        plt.plot(x_vals, y_vals)
    plt.xlim(-0.5, 1.5)
    plt.ylim(-0.5, 1.5)
    plt.title(f"Epoch {epoch+1}")
    plt.grid(True)
    plt.pause(0.01)
# Error vs Epoch graph
def plot_error_curve(errors):

    plt.figure()
    plt.plot(range(1, len(errors)+1), errors)
    plt.xlabel("Epoch")
    plt.ylabel("Total Error")
```

```
    plt.title("Error Convergence")
    plt.grid(True)
    plt.show()

# NAND Dataset
X_nand = np.array([
    [0,0],
    [0,1],
    [1,0],
    [1,1]
])
T_nand = np.array([1,1,1,0])
print("\n===== Training NAND Gate =====")
train_perceptron(X_nand, T_nand)
# XOR Dataset
X_xor = np.array([
    [0,0],
    [0,1],
    [1,0],
    [1,1]
])

T_xor = np.array([0,1,1,0])

print("\n===== Training XOR Gate =====")
train_perceptron(X_xor, T_xor)
```

#XOR:

import numpy as np

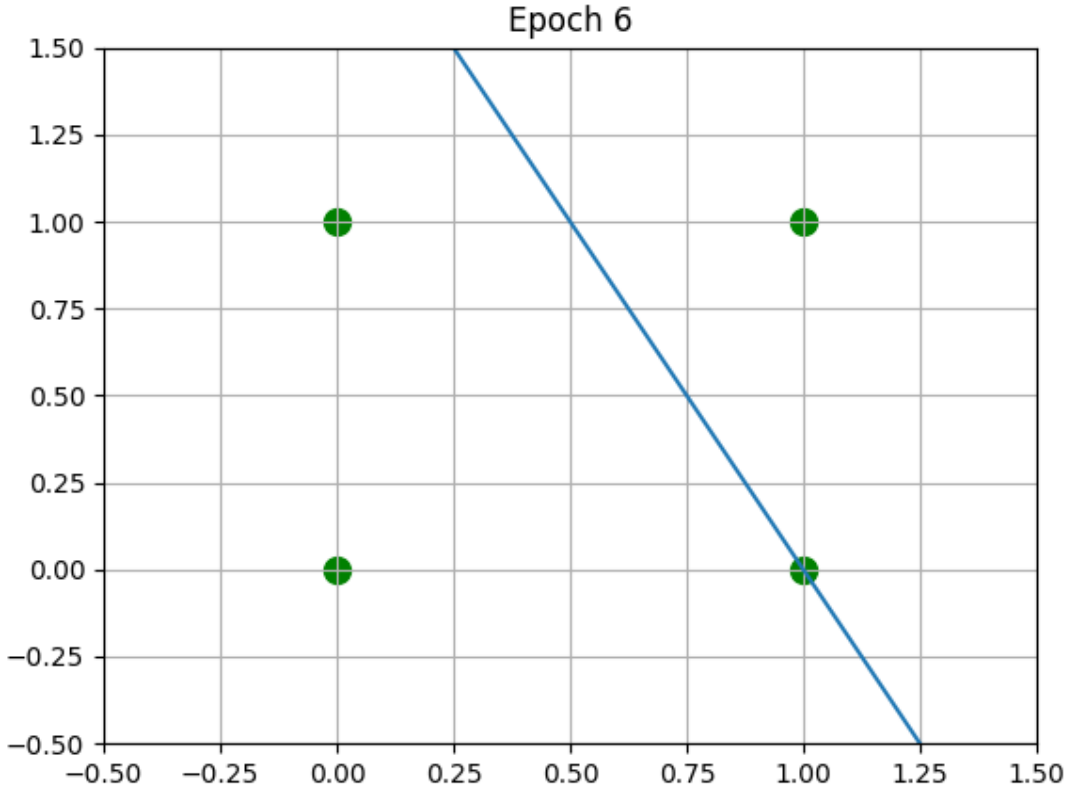
```
import matplotlib.pyplot as plt

# Sigmoid + derivative

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

def dsigmoid(a):
    return a * (1 - a)

# XOR dataset

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

T = np.array([[0],[1],[1],[0]])

# Initialize (small random)
np.random.seed(0)

W1 = np.random.randn(2,2)
b1 = np.zeros((1,2))

W2 = np.random.randn(2,1)
```

```
b2 = np.zeros((1,1))

lr = 0.5
epochs = 5000

losses = []

# Training loop

for epoch in range(epochs):

    # Forward
    z1 = X @ W1 + b1
    h = sigmoid(z1)

    z2 = h @ W2 + b2
    y = sigmoid(z2)

    # Loss
    loss = np.mean((T - y)**2)
    losses.append(loss)

    # Backprop
    delta_out = y - T
    delta_h = delta_out @ W2.T * dsigmoid(h)

    # Updates
    W2 -= lr * h.T @ delta_out
```

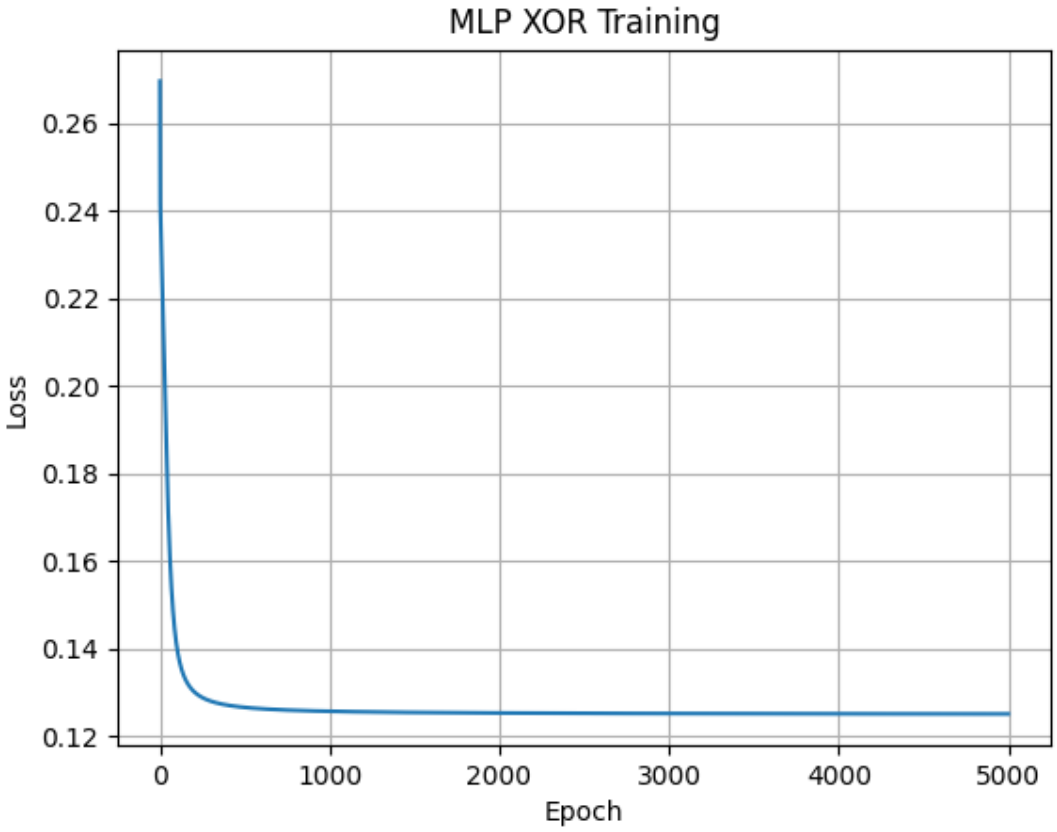
```
    b2 -= lr * np.sum(delta_out, axis=0)

    W1 -= lr * X.T @ delta_h
    b1 -= lr * np.sum(delta_h, axis=0)

    if epoch % 500 == 0:
        print(f"Epoch {epoch} Loss: {loss:.4f}")

# Final predictions

print("\nFinal XOR Predictions:")
for x in X:
    h = sigmoid(x @ W1 + b1)
    y = sigmoid(h @ W2 + b2)
    print(x, "→", round(float(y),3))
plt.plot(losses)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("MLP XOR Training")
plt.grid()
plt.show()
```

output:

```
PS C:\Users\kumaw\AppData\Local\Programs\Microsoft VS Code> & c:\python314\python.exe "c:/vishal practical/rishab.py"
Epoch 1
x1  x2 | Target | Pred | Error | Weights (w1,w2,b)
0   0  |   1    |  1   |  0    | [0. 0.], 0.0
0   1  |   1    |  1   |  0    | [0. 0.], 0.0
1   0  |   1    |  1   |  0    | [0. 0.], 0.0
1   1  |   0    |  1   |  -1   | [-0.1 -0.1], -0.1

Epoch 2
x1  x2 | Target | Pred | Error | Weights (w1,w2,b)
0   0  |   1    |  0   |  1    | [-0.1 -0.1], 0.0
0   1  |   1    |  0   |  1    | [-0.1  0. ], 0.1
1   0  |   1    |  1   |  0    | [-0.1  0. ], 0.1
1   1  |   0    |  1   |  -1   | [-0.2 -0.1], 0.0

Epoch 3
x1  x2 | Target | Pred | Error | Weights (w1,w2,b)
0   0  |   1    |  1   |  0    | [-0.2 -0.1], 0.0
0   1  |   1    |  0   |  1    | [-0.2  0. ], 0.1
1   0  |   1    |  0   |  1    | [-0.1  0. ], 0.2
1   1  |   0    |  1   |  -1   | [-0.2 -0.1], 0.1

Epoch 4
x1  x2 | Target | Pred | Error | Weights (w1,w2,b)
0   0  |   1    |  1   |  0    | [-0.2 -0.1], 0.1
0   1  |   1    |  1   |  0    | [-0.2 -0.1], 0.1
1   0  |   1    |  0   |  1    | [-0.1 -0.1], 0.2
1   1  |   0    |  1   |  -1   | [-0.2 -0.2], 0.1

Epoch 5
x1  x2 | Target | Pred | Error | Weights (w1,w2,b)
0   0  |   1    |  1   |  0    | [-0.2 -0.2], 0.1
0   1  |   1    |  0   |  1    | [-0.2 -0.1], 0.2
1   0  |   1    |  1   |  0    | [-0.2 -0.1], 0.2
1   1  |   0    |  0   |  0    | [-0.2 -0.1], 0.2

Epoch 6
x1  x2 | Target | Pred | Error | Weights (w1,w2,b)
0   0  |   1    |  1   |  0    | [-0.2 -0.1], 0.2
0   1  |   1    |  1   |  0    | [-0.2 -0.1], 0.2
1   0  |   1    |  1   |  0    | [-0.2 -0.1], 0.2
1   1  |   0    |  0   |  0    | [-0.2 -0.1], 0.2

✓ Converged!
```


Epoch 6
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50


Error Convergence
Total Error
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
Epoch


MLP XOR Training
Loss
0.26
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0
1000
2000
3000
4000
5000
Epoch