Exp-6 IMPLEMENT GRADIENT DESCENT AND BACKPROPAGATION IN DEEP NEURAL NETWORK

AIM

To implement gradient descent and back propagation algorithms in a deep neural newrak and understand how weight updates occur during training.

OBJECTIVES.

① To understand the working principels of gradient descent and backpropagation.

② To implement forword progration, loss calculation. and backword pragation from scratch

③ To observe how the network learns by minimizing. the loss function through iterative updates.

④ To analyze how learning rate affets the convegence of the network

⑤ To compare performance for different initialization and activation functions.

PSEUDOCODE:

-> Initalize network parameters:
  - Number of layers, neurons/layer
  - Weights: randomly, biases to 0 or ~ values.

-> Define activation functions.

-> for each training iteration (epoch)

  a. Forword Propagation:
    - Compute input to each layer $z = W * x^T + b$
    - Apply activation $A = g(z)$
    - Reapt for all layers.

NOTE:

$$\nabla_\theta J(\theta) = \left[\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \cdots \frac{\partial J}{\partial \theta_n}\right]$$

$$\boxed{\theta_{t+1} = \theta_t - \alpha \nabla_\theta J(\theta_t)}$$

Taylor series approximation around θ_t

$$J(\theta_t + \Delta\theta) \approx J(\theta_t) + \nabla_\theta J(\theta_t)^T \Delta\theta$$

To decrease J choose $\Delta\theta$

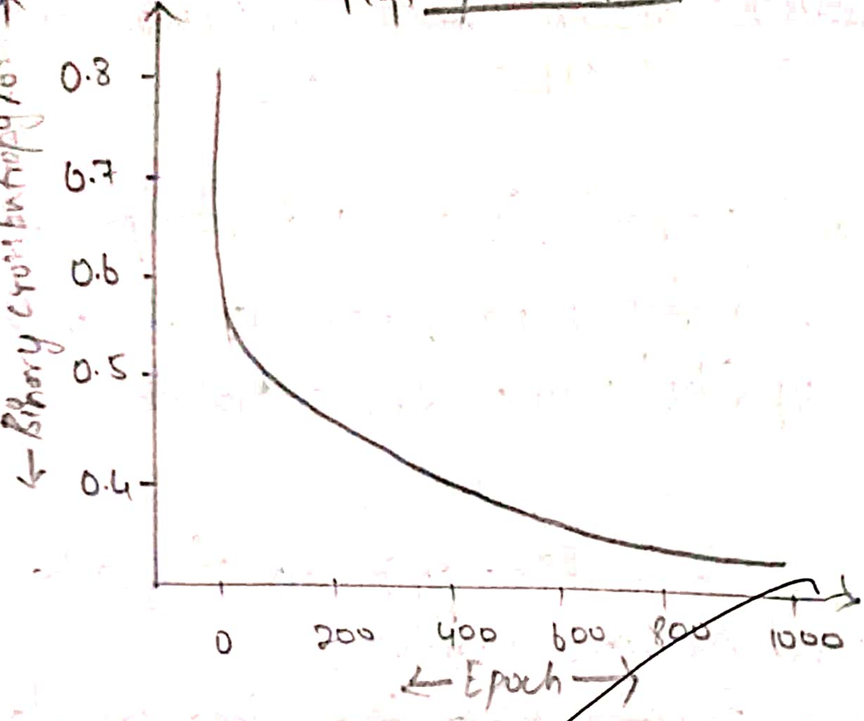
$$\nabla_\theta J(\theta_t)_T \Delta\theta < 0$$

The steepest descent direction.

$$\Delta\theta = -\alpha \nabla_\theta J(\theta_t)$$

which justifies the update rule.

Fig1 Epoch vs Loss



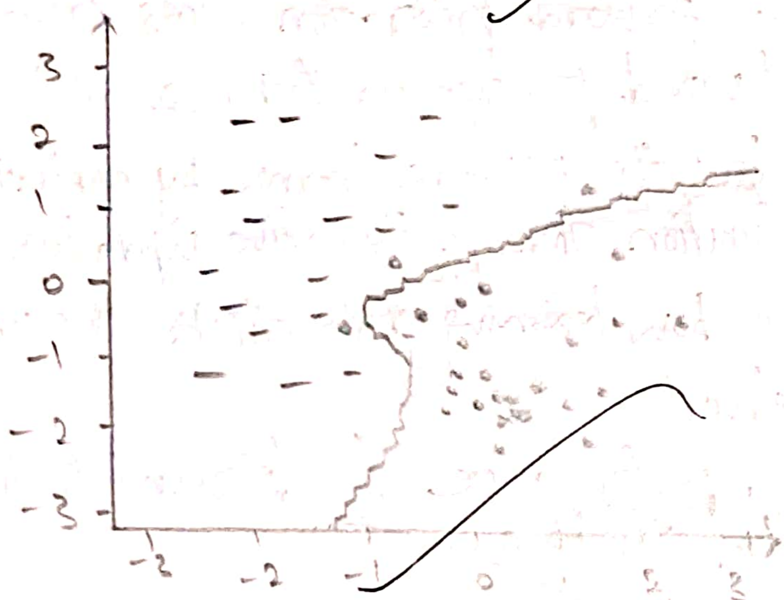
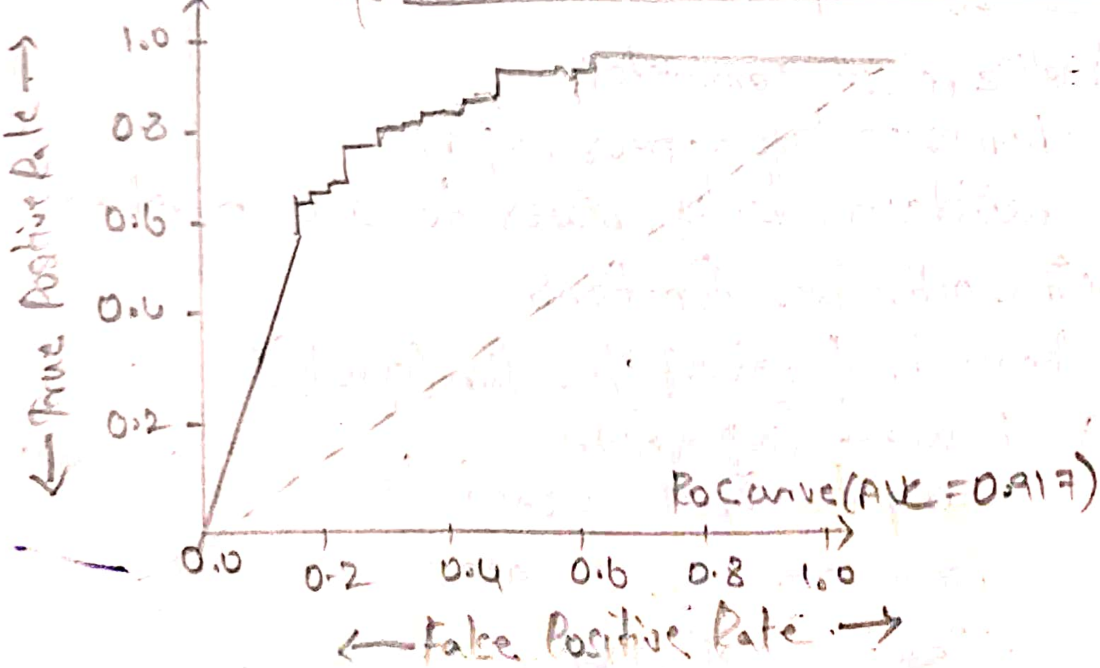


Fig: 2

Fig3: Receiver Operating Characteristic (ROC) Curve



b) Compute loss:

$loss = \left(\frac{1}{2}\right) * (A\_final - Y\_actual)^2$ // Mean sq error

c) Backword Propagation:
- Calculate gradient of loss w.r.t. o/p layer
- For each layer o/p to I/p
  1. dA = gradient of activation
  2. $dZ = dA * g'(z)$
  3. $dW = dZ * A\_prev^T$
  4. $db = sum(dZ)$
  5. $dA\_prev = W^T * dZ$

d) Update parameters:

$W_{t+1} = W_t - \alpha * dW$

$b_{t+1} = b - \alpha * db$

↳ Repeat until convergence or max epochs.

↳ After training.

OBSERVATION:

* The ROC curve (Fig 3) and AUC provide insights into models ability to distinguish b/w 2 classes. An AUC of 0.917 suggest that the model has good discriminatory power.
* 86.00% which means the model correctly clasifed 86% of the test samples.
* The training loss (fig 1) shows that the binary cross-entropy loss decreases over epochs indication that the model is learning and converging.

Fig: 2 Dicision boundary plot on the test shows.
How the trained model sepetats 2 cabuses
by a non-linear boundry.

The experiment was implemented and obtained
result sucssfully

https://colab.research.google.com/drive/1acqtNpF_J9XfrAnsvDXakZyDB6qMdF9S?authuser=2

lab_6

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

Share Gemini

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc
```

```
# Activation functions and derivatives

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

def sigmoid_derivative(x):
    s = sigmoid(x)
    return s * (1 - s)

def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return (x > 0).astype(float)

# Loss function: binary cross-entropy

def binary_cross_entropy(y_true, y_pred):
    eps = 1e-8
    y_pred = np.clip(y_pred, eps, 1 - eps)
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
```

Variables Terminal

India | ISRO | GATE | GATE | GATE | GATE | Inbox | Inbox | URO | 21AI | Non | drive | Colab | lab_5 | Lab 5 | KAVI | jpg t | Conv | DLT

https://colab.research.google.com/drive/1acqtNpF_J9XfrAnsvDXakZyDB6qMdF9S?authuser=2

lab_6

File Edit View Insert Runtime Tools Help

Share Gemini K

Commands + Code + Text Run all Connect

```python
    s = sigmoid(x)
    return s * (1 - s)

def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return (x > 0).astype(float)

# Loss function: binary cross-entropy

def binary_cross_entropy(y_true, y_pred):
    eps = 1e-8
    y_pred = np.clip(y_pred, eps, 1 - eps)
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

def binary_cross_entropy_derivative(y_true, y_pred):
    eps = 1e-8
    y_pred = np.clip(y_pred, eps, 1 - eps)
    return -(y_true / y_pred) + (1 - y_true) / (1 - y_pred)
```

```python
# Deep Neural Network with 2 hidden layers

class DeepNeuralNetwork:
    def __init__(self, input_dim, hidden_dims, output_dim, learning_rate=0.01):
        self.learning_rate = learning_rate

        # Initialize weights with He initialization for ReLU
        layer_sizes = [input_dim] + hidden_dims + [output_dim]
        self.weights = []
        self.biases = []
```

Variables Terminal

https://colab.research.google.com/drive/1acqtNpF_J9XfrAnsvDXakZyDB6qMdF9S?authuser=2

lab_6

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

```python
# Deep Neural Network with 2 hidden layers

class DeepNeuralNetwork:
    def __init__(self, input_dim, hidden_dims, output_dim, learning_rate=0.01):
        self.learning_rate = learning_rate

        # Initialize weights with He initialization for ReLU
        layer_sizes = [input_dim] + hidden_dims + [output_dim]
        self.weights = []
        self.biases = []
        for i in range(len(layer_sizes) - 1):
            W = np.random.randn(layer_sizes[i], layer_sizes[i+1]) * np.sqrt(2 / layer_sizes[i])
            b = np.zeros((1, layer_sizes[i+1]))
            self.weights.append(W)
            self.biases.append(b)

    def forward(self, X):
        self.zs = []  # weighted sums
        self.activations = [X]  # store activations for each layer

        for i in range(len(self.weights) - 1):
            z = self.activations[-1].dot(self.weights[i]) + self.biases[i]
            a = relu(z)
            self.zs.append(z)
            self.activations.append(a)

        # Output layer with sigmoid
        z = self.activations[-1].dot(self.weights[-1]) + self.biases[-1]
        a = sigmoid(z)
        self.zs.append(z)
        self.activations.append(a)
```

Variables Terminal

https://colab.research.google.com/drive/1acqtNpF_J9XfrAnsvDXakZyDB6qMdF9S?authuser=2

lab_6

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

```python
            z = self.activations[-1].dot(self.weights[i]) + self.biases[i]
            a = relu(z)
            self.zs.append(z)
            self.activations.append(a)

        # Output layer with sigmoid
        z = self.activations[-1].dot(self.weights[-1]) + self.biases[-1]
        a = sigmoid(z)
        self.zs.append(z)
        self.activations.append(a)
        return a

    def backward(self, y_true):
        m = y_true.shape[0]
        y_pred = self.activations[-1]

        # Gradient of loss w.r.t output activation
        dA = binary_cross_entropy_derivative(y_true, y_pred) / m

        self.dW = [None] * len(self.weights)
        self.db = [None] * len(self.biases)

        # Backprop for output layer (sigmoid)
        dZ = dA * sigmoid_derivative(self.zs[-1])
        self.dW[-1] = self.activations[-2].T.dot(dZ)
        self.db[-1] = np.sum(dZ, axis=0, keepdims=True)

        dA_prev = dZ.dot(self.weights[-1].T)

        # Backprop for hidden layers (ReLU)
        for i in reversed(range(len(self.weights) - 1)):
```

Variables Terminal

```python
        # Backprop for output layer (sigmoid)
        dZ = dA * sigmoid_derivative(self.zs[-1])
        self.dW[-1] = self.activations[-2].T.dot(dZ)
        self.db[-1] = np.sum(dZ, axis=0, keepdims=True)

        dA_prev = dZ.dot(self.weights[-1].T)

        # Backprop for hidden layers (ReLU)
        for i in reversed(range(len(self.weights) - 1)):
            dZ = dA_prev * relu_derivative(self.zs[i])
            self.dW[i] = self.activations[i].T.dot(dZ)
            self.db[i] = np.sum(dZ, axis=0, keepdims=True)
            dA_prev = dZ.dot(self.weights[i].T)

        # Update weights and biases
        for i in range(len(self.weights)):
            self.weights[i] -= self.learning_rate * self.dW[i]
            self.biases[i] -= self.learning_rate * self.db[i]

    def train(self, X, y, epochs=1000, print_every=100):
        self.losses = []
        for epoch in range(epochs):
            y_pred = self.forward(X)
            loss = binary_cross_entropy(y, y_pred)
            self.losses.append(loss)
            self.backward(y)
            if epoch % print_every == 0 or epoch == epochs - 1:
                print(f"Epoch {epoch}, Loss: {loss:.4f}")

    def predict(self, X):
```

```python
        self.losses = []
        for epoch in range(epochs):
            y_pred = self.forward(X)
            loss = binary_cross_entropy(y, y_pred)
            self.losses.append(loss)
            self.backward(y)
            if epoch % print_every == 0 or epoch == epochs - 1:
                print(f"Epoch {epoch}, Loss: {loss:.4f}")

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

```python
# Prepare data

X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)
y = y.reshape(-1, 1)

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Train model
model = DeepNeuralNetwork(input_dim=2, hidden_dims=[10, 5], output_dim=1, learning_rate=0.01)
model.train(X_train, y_train, epochs=1000)
```

```
Epoch 0, Loss: 0.8247
Epoch 100, Loss: 0.5036
Epoch 200, Loss: 0.4592
```

https://colab.research.google.com/drive/1acqtNpF_J9XfrAnsvDXakZyDB6qMdF9S?authuser=2#scrollTo=XwCnRax4XnRX

lab_6

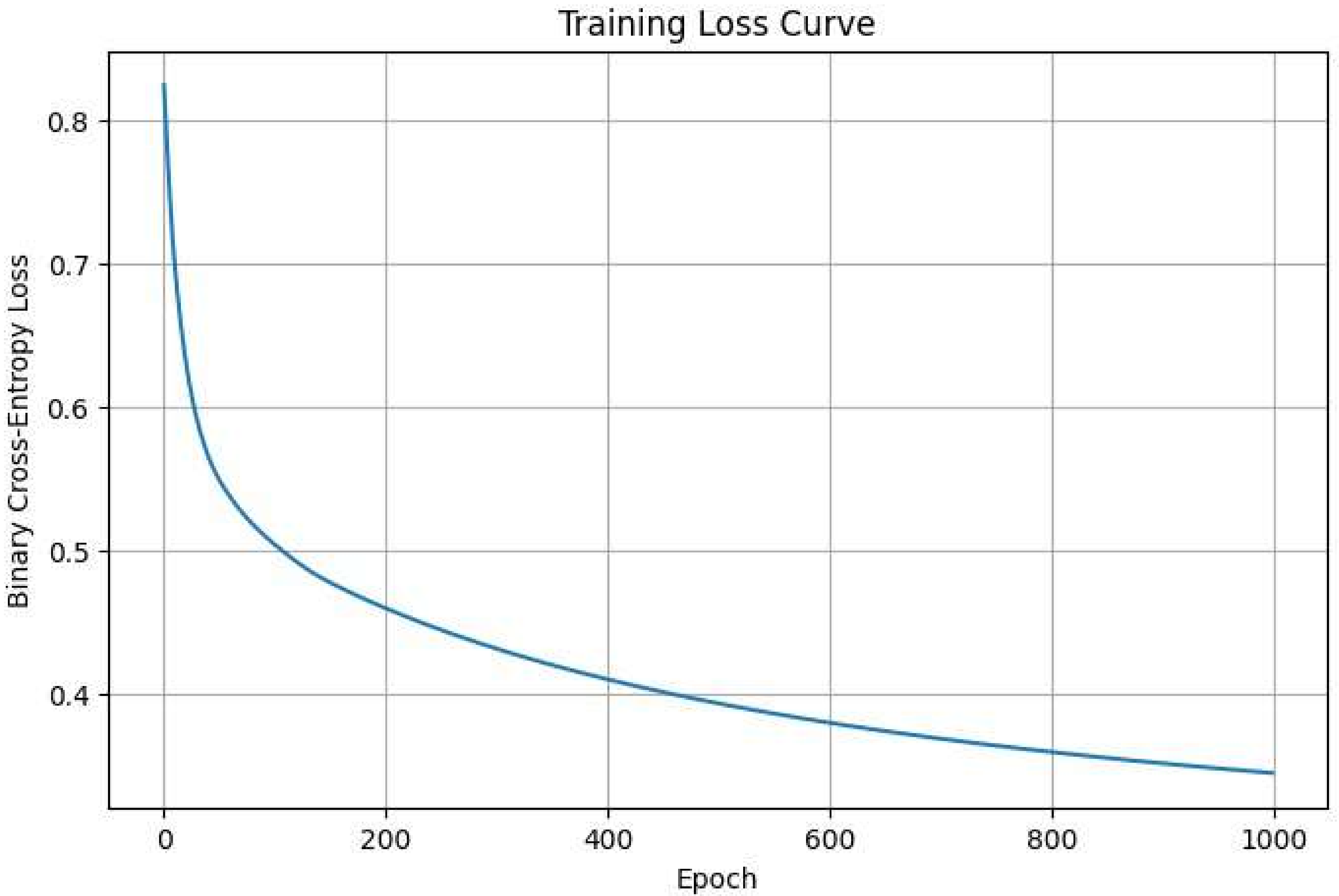
File Edit View Insert Runtime Tools Help

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Train model
model = DeepNeuralNetwork(input_dim=2, hidden_dims=[10, 5], output_dim=1, learning_rate=0.01)
model.train(X_train, y_train, epochs=1000)
```

```
Epoch 0, Loss: 0.8247
Epoch 100, Loss: 0.5036
Epoch 200, Loss: 0.4592
Epoch 300, Loss: 0.4309
Epoch 400, Loss: 0.4095
Epoch 500, Loss: 0.3927
Epoch 600, Loss: 0.3792
Epoch 700, Loss: 0.3680
Epoch 800, Loss: 0.3587
Epoch 900, Loss: 0.3508
Epoch 999, Loss: 0.3441
```

```python
# Plot loss curve
plt.figure(figsize=(8, 5))
plt.plot(model.losses)
plt.title("Training Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Binary Cross-Entropy Loss")
plt.grid(True)
plt.show()
```



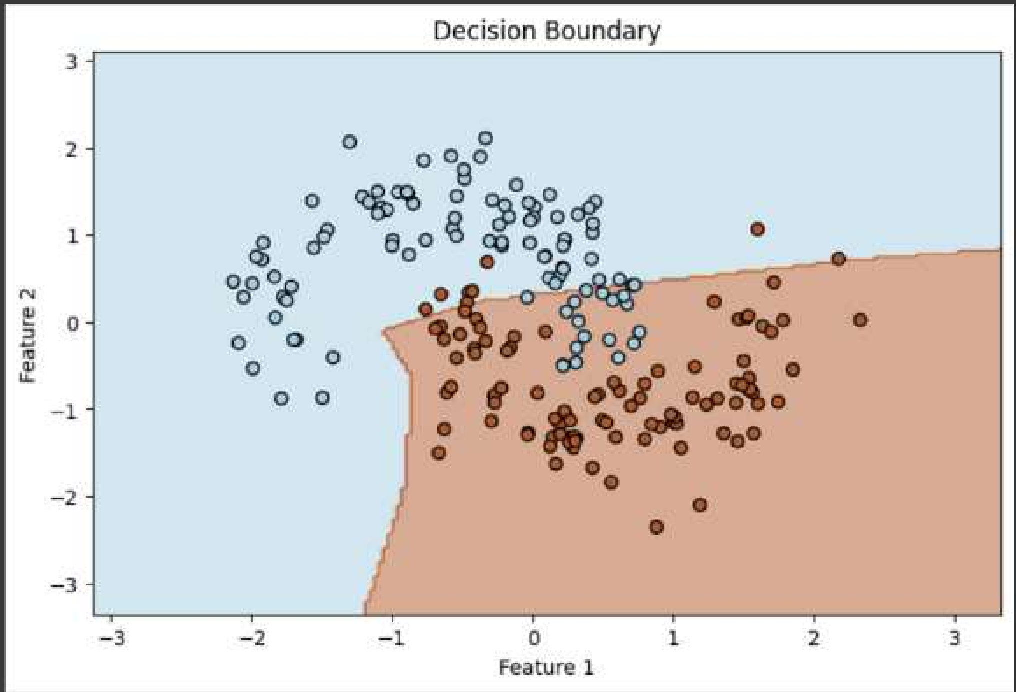
Training Loss Curve
Binary Cross-Entropy Loss
0.8
0.7
0.6
0.5
0.4
0
200
400
600
800
1000
Epoch


```python
# Plot decision boundary function
def plot_decision_boundary(model, X, y):
    # Create mesh grid
    x_min, x_max = X[:,0].min() - 1, X[:,0].max() + 1
    y_min, y_max = X[:,1].min() - 1, X[:,1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
                         np.linspace(y_min, y_max, 200))
    grid = np.c_[xx.ravel(), yy.ravel()]
    preds = model.predict(grid)
    preds = preds.reshape(xx.shape)

    plt.figure(figsize=(8, 5))
    plt.contourf(xx, yy, preds, alpha=0.5, cmap=plt.cm.Paired)
    plt.scatter(X[:,0], X[:,1], c=y.ravel(), edgecolors='k', cmap=plt.cm.Paired)
    plt.title("Decision Boundary")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()
```

```python
# Plot decision boundary on test set
plot_decision_boundary(model, X_test, y_test)
```

```
# Plot decision boundary on test set
plot_decision_boundary(model, X_test, y_test)
```




```
# Test accuracy
y_pred_test = model.predict(X_test)
```

https://colab.research.google.com/drive/1acqtNpF_J9XfrAnsvDXakZyDB6qMdF9S?authuser=2#scrollTo=BQB8i3rXZulw

lab_6

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Share Gemini Connect

```python
# Test accuracy
y_pred_test = model.predict(X_test)
accuracy = np.mean(y_pred_test == y_test)
print(f"Test accuracy: {accuracy * 100:.2f}%")
```

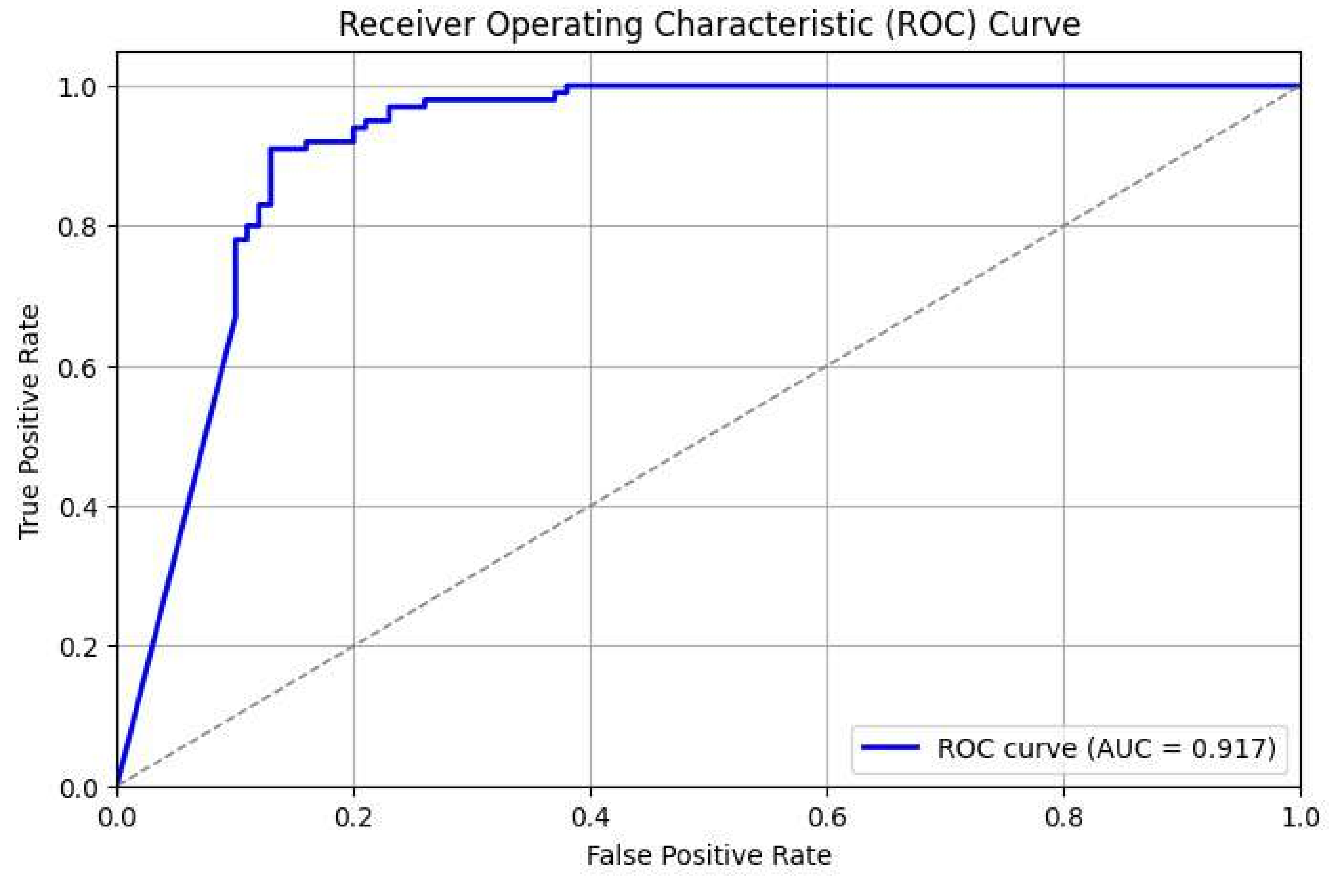
```
Test accuracy: 86.00%
```

```python
# Predict probabilities for test set (not thresholded)
y_scores = model.forward(X_test).ravel()
```

```python
# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)
```

```python
# Plot ROC curve
plt.figure(figsize=(8,5))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.3f})')
plt.plot([0,1], [0,1], color='gray', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

Variables Terminal


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.917)