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

```
# Step 1: Generate Data
def generate_data(num_samples=100):
    np.random.seed(42)
    hours_studied = np.random.uniform(0, 10, num_samples)
    hours_slept = np.random.uniform(0, 10, num_samples)
    # Rule: pass if hours_studied * 0.5 + hours_slept * 0.2 > 4.5
    labels = (hours_studied * 0.5 + hours_slept * 0.2 > 4.5).astype(int)
    data = np.column_stack((hours_studied, hours_slept))
    return data, labels
```

```
# Step 2: Sigmoid Activation Function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))
```

+ Code + Text

```
# Step 3: Train the Network
def train_fnn(data, labels, learning_rate=0.01, epochs=1000):
    np.random.seed(42)
    weights = np.random.randn(2)  # Initialize weights randomly
    bias = np.random.randn()     # Initialize bias randomly

    losses = []  # Store losses for plotting

    for epoch in range(epochs):
        # Forward pass
        linear_output = np.dot(data, weights) + bias
        predictions = sigmoid(linear_output)

        # Compute the loss (Binary Cross-Entropy)
        loss = -(labels * np.log(predictions) + (1 - labels) * np.log(1 - predictions)).mean()
        losses.append(loss)

        # Backward pass (Gradient Descent)
        d_loss_pred = predictions - labels
        d_pred_linear = sigmoid_derivative(linear_output)

        gradient_weights = np.dot(data.T, d_loss_pred * d_pred_linear) / len(data)
        gradient_bias = np.sum(d_loss_pred * d_pred_linear) / len(data)

        # Update weights and bias
        weights -= learning_rate * gradient_weights
        bias -= learning_rate * gradient_bias

        if epoch % 100 == 0:
            print(f"Epoch {epoch}, Loss: {loss:.4f}")

    return weights, bias, losses
```

```
# Step 4: Plot Decision Boundary
def plot_decision_boundary(data, labels, weights, bias):
    x_min, x_max = data[:, 0].min() - 1, data[:, 0].max() + 1
    y_min, y_max = data[:, 1].min() - 1, data[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                         np.linspace(y_min, y_max, 100))
    Z = sigmoid(weights[0] * xx + weights[1] * yy + bias)
    Z = (Z > 0.5).astype(int)

    plt.contourf(xx, yy, Z, alpha=0.6, cmap=plt.cm.Paired)
    plt.scatter(data[:, 0], data[:, 1], c=labels, edgecolor='k', cmap=plt.cm.Paired)
    plt.title("Decision Boundary")
    plt.xlabel("Hours Studied")
    plt.ylabel("Hours Slept")
    plt.show()
```

```
# Step 5: Plot Training Loss
def plot_training_loss(losses):
    plt.plot(losses)
    plt.title("Training Loss Over Epochs")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.show()
```

```
1 # Generate synthetic data
2 data, labels = generate_data()
```

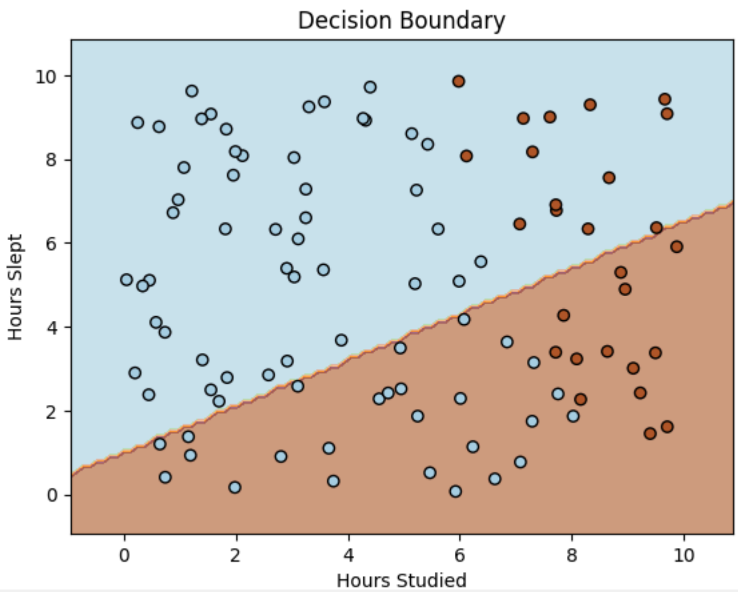
```
1 # Train the FNN
2 trained_weights, trained_bias, losses = train_fnn(data, labels)
```

```
Epoch 0, Loss: 1.4194
Epoch 100, Loss: 0.7582
Epoch 200, Loss: 0.7025
Epoch 300, Loss: 0.6863
Epoch 400, Loss: 0.6758
Epoch 500, Loss: 0.6672
Epoch 600, Loss: 0.6597
Epoch 700, Loss: 0.6531
Epoch 800, Loss: 0.6471
Epoch 900, Loss: 0.6416
```

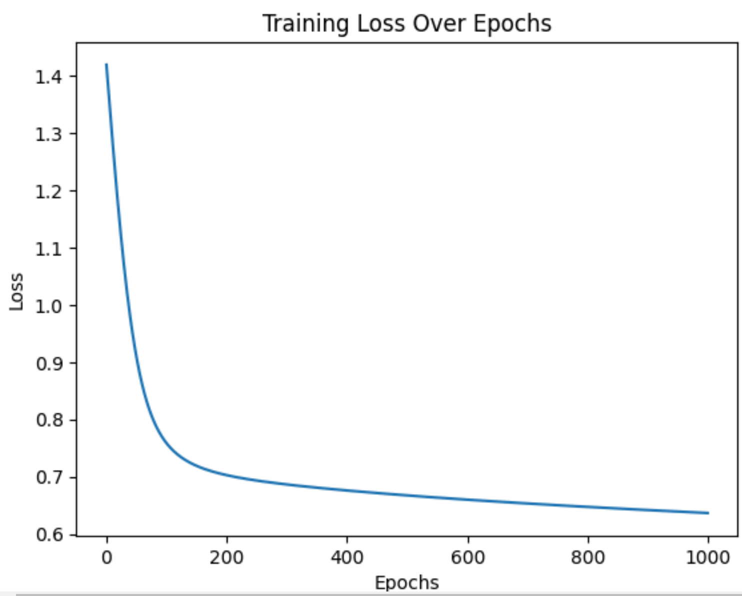
```
1 # Print learned weights
2 print("Learned Weights:", trained_weights)
3 print("Learned Bias:", trained_bias)
```

```
Learned Weights: [ 0.15889573 -0.28865285]
Learned Bias: 0.29157136032525205
```

```
1 # Plot decision boundary
2 plot_decision_boundary(data, labels, trained_weights, trained_bias)
```




```
1 # Plot training loss
2 plot_training_loss(losses)
```




```
1 Start coding or generate with AI.
```

```
1 Start coding or generate with AI.
```