Assignment 1 - Pattern Recognition (PMAT 403)

Gandholi Sarat - 23008

April 4, 2025

Link to Notebook

Click here to access the Colab Notebook

Contents

List of Figures

1 Computer Programs

1.1 Question 1: Data Generator

Answer: Generates a two-class, two-dimensional dataset using four normal distributions. The first two distributions belong to class +1, the last two belong to class -1.

Parameters:

- **m:** 2x4 matrix, where each column is the mean vector for a distribution.
- **s:** Variance parameter.
- **N:** Number of points to generate per distribution.
- **seed:** Random seed for reproducibility (optional).

Returns:

- **X:** 2x(4*N) data matrix.
- **y:** (4*N) class labels vector.

```
    import random
    import math

    if seed is not None:
        np.random.seed(seed)  # Ensure reproducibility
    X = []
    y = []
    S = s * np.eye(2)

    for i in range(4):
        mean = np.array(m)[:, i]
        samples = np.random.multivariate_normal(m[:, i],
           S, N).T
        X.append(samples)
        y.append(np.ones(N) if i < 2 else -np.ones(N))
    X = np.concatenate(X,1)  # Shape: (2, 4N)
    y = np.concatenate(y)  # Shape: (4N,)

    return X.T, y  # Transpose X to have shape (4N, 2)
```

1.2 Question 2: Neural network

Answer: The network uses the tanh activation function and supports three training methods: standard backpropagation, backpropagation with momentum, and backpropagation with an adaptive learning rate. The network is initialized with random weights and biases, and the training process includes forward propagation, loss calculation, and weight updates based on the chosen method.

```python
import numpy as np

class NeuralNetwork:
    def __init__(self, input_dim, hidden_nodes):
        self.input_dim = input_dim
        self.hidden_nodes = hidden_nodes
        self.output_dim = 1  # Single output node

        # Initialize weights using uniform random values between -0.5 and 0.5
        self.W1 = np.random.uniform(-0.5, 0.5, (input_dim, hidden_nodes))
        self.b1 = np.random.uniform(-0.5, 0.5, (1, hidden_nodes))
        self.W2 = np.random.uniform(-0.5, 0.5, (hidden_nodes, 1))
        self.b2 = np.random.uniform(-0.5, 0.5, (1, 1))

    def tanh(self, x):
        #Tanh activation function: f(x) = (e^x - e^(-x)) / (e^x + e^(-x))
        return np.tanh(x)

    def tanh_derivative(self, x):
        #Derivative of tanh: f'(x) = 1 - tanh^2(x)
        return 1 - np.tanh(x) ** 2

    def forward(self, x):
        #Compute forward pass
        self.hidden_input = np.dot(x, self.W1) + self.b1
        self.hidden_output = self.tanh(self.hidden_input)

```

```python
        self.final_input = np.dot(self.hidden_output, self.W2) + self.b2
        self.final_output = self.tanh(self.final_input)

        return self.final_output

    def loss(self, y_true, y_pred):
        #Mean Squared Error Loss function
        return np.mean((y_true - y_pred) ** 2)

    def train(self, X, y, method, epochs, params):
        #Train the neural network using different optimization methods
        x = X.T
        lr, mc, lr_inc, lr_dec, max_perf_inc = params  # Unpack hyperparameters

        prev_loss = float('inf')  # Initialize previous loss for adaptive learning rate

        # Initialize momentum variables
        velocity_W1 = np.zeros_like(self.W1)
        velocity_b1 = np.zeros_like(self.b1)
        velocity_W2 = np.zeros_like(self.W2)
        velocity_b2 = np.zeros_like(self.b2)

        for epoch in range(epochs):
            # Forward pass
            output = self.forward(X)

            # Compute error
            output_error = y.reshape(-1, 1) - output
            hidden_error = np.dot(output_error, self.W2.T) * (1 - self.hidden_output**2)

            if method == 1:  # Standard Backpropagation
                self.W2 += lr * np.dot(self.hidden_output.T, output_error)
                self.b2 += lr * np.sum(output_error, axis=0, keepdims=True)
                self.W1 += lr * np.dot(X.T,
```

```
                hidden_error)
                self.b1 += lr * np.sum(hidden_error, axis=0, keepdims=True)

            elif method == 2:  # Backpropagation with Momentum
                velocity_W2 = mc * velocity_W2 + lr * np.dot(self.hidden_output.T, output_error)
                velocity_b2 = mc * velocity_b2 + lr * np.sum(output_error, axis=0, keepdims=True)
                velocity_W1 = mc * velocity_W1 + lr * np.dot(X.T, hidden_error)
                velocity_b1 = mc * velocity_b1 + lr * np.sum(hidden_error, axis=0, keepdims=True)

                self.W2 += velocity_W2
                self.b2 += velocity_b2
                self.W1 += velocity_W1
                self.b1 += velocity_b1

            elif method == 3:  # Backpropagation with Adaptive Learning Rate
                loss = self.loss(y, output)
                if loss / prev_loss < 1:
                    lr *= lr_inc  # Increase learning rate if loss decreases
                elif loss / prev_loss > max_perf_inc:
                    lr *= lr_dec  # Decrease learning rate if loss increases too much
                prev_loss = loss

                self.W2 += lr * np.dot(self.hidden_output.T, output_error)
                self.b2 += lr * np.sum(output_error, axis=0, keepdims=True)
                self.W1 += lr * np.dot(X.T, hidden_error)
                self.b1 += lr * np.sum(hidden_error, axis=0, keepdims=True)
```

```

            # Print loss every 100 epochs
            if (epoch + 1) % 100 == 0:
                print(f'Epoch {epoch+1}, Learning Rate: {lr:.6f}')

    def predict(self, X):
        #Predict output for given input data
        return np.array([self.forward(x) for x in X])

    def evaluate(self, X, y):
        #Evaluate the model accuracy
        y_pred = self.predict(X)
        y_pred_class = np.sign(y_pred)
        accuracy = np.mean(y_pred_class.flatten() == y)
        return accuracy
```

1.3 Question 3: Data Visualization

Answer: Plots the decision regions produced by a trained neural network.

Parameters:

- `net (dict)`: Trained neural network parameters {`'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2`}.
- `lh (float)`: Lower bound in the horizontal direction.
- `uh (float)`: Upper bound in the horizontal direction.
- `lv (float)`: Lower bound in the vertical direction.
- `uv (float)`: Upper bound in the vertical direction.
- `rh (float)`: Resolution in the horizontal direction (smaller = finer).
- `rv (float)`: Resolution in the vertical direction (smaller = finer).
- `m (numpy.ndarray)`: Mean vectors of the normal distributions (for visualization reference).

Returns:

- `None` (displays a plot).

```
def plot_dec_regions(net, lh, uh, lv, uv, rh, rv, m, X, y):

    # Generate grid points
    x1_vals = np.arange(lh, uh, rh)
    x2_vals = np.arange(lv, uv, rv)
    xx1, xx2 = np.meshgrid(x1_vals, x2_vals)
    grid_points = np.c_[xx1.ravel(), xx2.ravel()]  # Shape (2, num_points)

    # Evaluate neural network on the grid
    #W1, b1, W2, b2 = net['W1'], net['b1'], net['W2'], net['b2']
    W1, b1, W2, b2 = net.W1, net.b1, net.W2, net.b2

    #predictions = net.forward(grid_points)  # Shape (num_points, 1)
    #predictions = np.sign(predictions)
    # Forward propagation
```

```python
    # Transpose W1 before multiplication to align dimensions
    # Original:
    Z1 = np.dot(grid_points, W1) + b1
    #Z1 = np.dot(np.array(W1).T, grid_points) + np.array(b1).reshape(-1,1)
    A1 = np.tanh(Z1)
    Z2 = np.dot(A1,W2) + b2
    Z2 = np.tanh(Z2)
    predictions = np.sign(Z2)
    #predictions = np.where(Z2 >= 0, 1, -1)  # Classify points

    # Reshape predictions for plotting
    decision_map = predictions.reshape(xx1.shape)

    # Plot decision boundary
    plt.figure(figsize=(8, 6))
    plt.contourf(xx1, xx2, decision_map, alpha=0.3, cmap=plt.cm.bwr)  # Background color

    # Mark decision regions
    plt.scatter(grid_points[:, 0][predictions.flatten() == 1],
                grid_points[:, 1][predictions.flatten() == 1],
                marker='*', color='red', label='Class 1', alpha=0.5)

    plt.scatter(grid_points[:, 0][predictions.flatten() == -1],
                grid_points[:, 1][predictions.flatten() == -1],
                marker='o', color='blue', label='Class -1', alpha=0.5)
    # Plot data points
    plt.scatter(X[y == 1, 0], X[y == 1, 1], marker='*', color='black', label="Class +1")
    plt.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', edgecolor='white', facecolor='none', label="Class -1")
    # Plot mean vectors (for reference)
    plt.scatter(m[:, 0], m[:, 1], marker='X', color='
```

```
        Yellow', s=100, label='Mean Vectors')

    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Decision Regions of Trained Neural Network')
    plt.legend()
    plt.show()
```

2 Computer Experiments

2.1 Data Generation

(a) After initializing the seed, use the `data_generator` function to create the dataset (X_1, y_1), with:

$$m = \begin{bmatrix} -5 & +5 & +5 & -5 \\ +5 & -5 & +5 & -5 \end{bmatrix}$$

where $s = 2$ and $N = 100$.

(b) Initialize the seed to 10 and repeat (a) to produce the dataset (X_2, y_2).

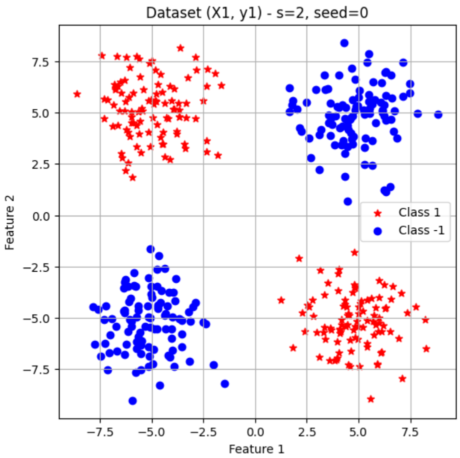
(c) Repeat the above two steps using the corresponding seeds , for $s = 5$, and produce the (X_3, y_3) and (X_4, y_4) datasets, respectively (where m and N remain the same).

(d) Plot the datasets.

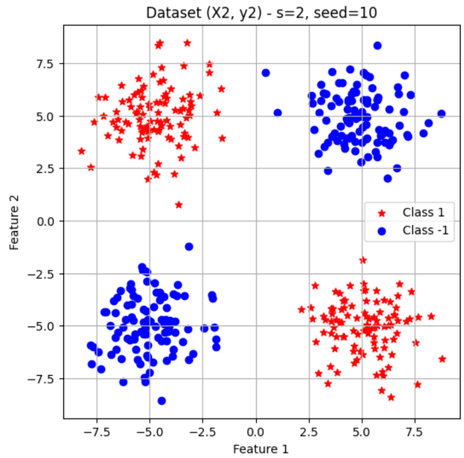
Answer:

```python
# Define the correct mean matrix
m = np.array([
    [-5, +5, +5, -5],  # X-coordinates of cluster centers
    [+5, -5, +5, -5]   # Y-coordinates of cluster centers
])

N = 100  # Number of samples per class

# Generate datasets
X1, y1 = data_generator(m, s=2, N=N, seed=0)
X2, y2 = data_generator(m, s=2, N=N, seed=10)
X3, y3 = data_generator(m, s=5, N=N, seed=0)
X4, y4 = data_generator(m, s=5, N=N, seed=10)
def plot_data(X, y, title):
    X=X.T
    plt.figure(figsize=(6, 6))
    plt.scatter(X[0, y.flatten() == 1], X[1, y.flatten() == 1],
                color='red', marker='*', label='Class 1')
    plt.scatter(X[0, y.flatten() == -1], X[1, y.flatten() == -1],
```

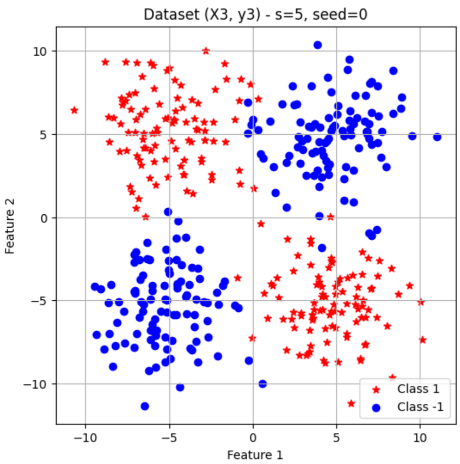
```
                    color='blue', marker='o', label='Class -1')

        plt.xlabel("Feature 1")
        plt.ylabel("Feature 2")
        plt.title(title)
        plt.legend()
        plt.grid(True)
        plt.show()

    # Plot all datasets
    plot_data(X1, y1, "Dataset (X1, y1) - s=2, seed=0")
    plot_data(X2, y2, "Dataset (X2, y2) - s=2, seed=10")
    plot_data(X3, y3, "Dataset (X3, y3) - s=5, seed=0")
    plot_data(X4, y4, "Dataset (X4, y4) - s=5, seed=10")
```



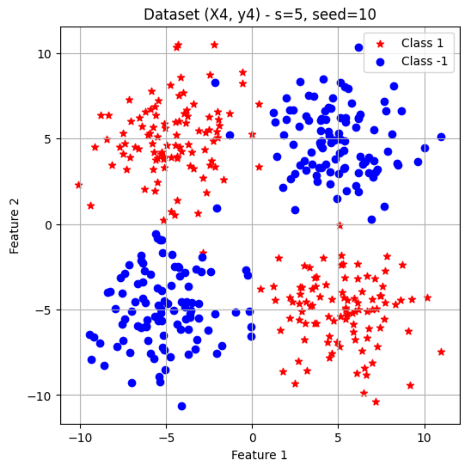
(a) Dataset (X1, y1) - s=2, seed=0



(b) Dataset (X2, y2) - s=2, seed=10



(c) Dataset (X3, y3) - s=5, seed=0



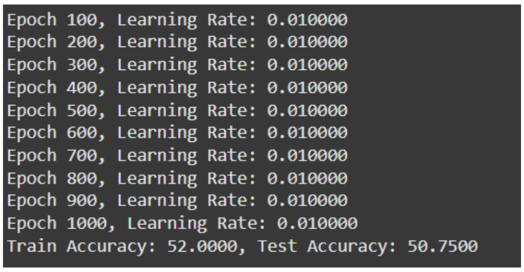
(d) Dataset (X4, y4) - s=5, seed=10

Figure 1: Comparison of datasets generated with different seeds and variance values

2.2 Standard Backpropagation Algorithm

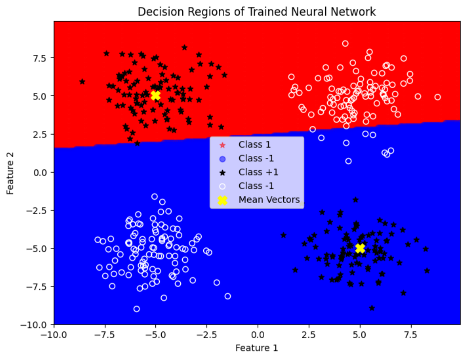
(a) Run the standard backpropagation algorithm with $lr = 0.01$ and 2, 4, and 15 first-layer nodes, for 1000 iterations, using the dataset (X_1, y_1) as the training set.
(b) Evaluate the performance of the designed neural networks for both (X_1, y_1) (training set) and (X_2, y_2) (test set) and plot the decision regions (use $lb = lv = -10, ub = uv = 10, rb = rv = 0.2$).
(c) Comment on the results.

Answer:

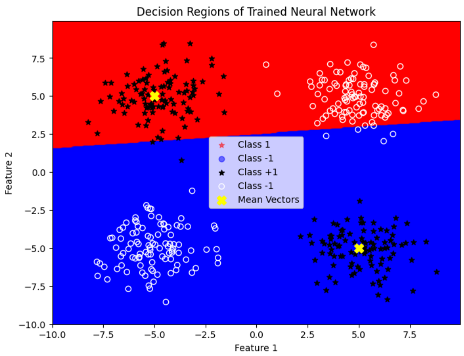


```
Epoch 100, Learning Rate: 0.010000
Epoch 200, Learning Rate: 0.010000
Epoch 300, Learning Rate: 0.010000
Epoch 400, Learning Rate: 0.010000
Epoch 500, Learning Rate: 0.010000
Epoch 600, Learning Rate: 0.010000
Epoch 700, Learning Rate: 0.010000
Epoch 800, Learning Rate: 0.010000
Epoch 900, Learning Rate: 0.010000
Epoch 1000, Learning Rate: 0.010000
Train Accuracy: 52.0000, Test Accuracy: 50.7500
```

(a) Result of 2 first layer nodes

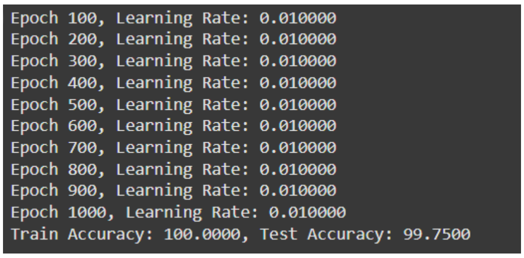


(b) Training set decision boundary

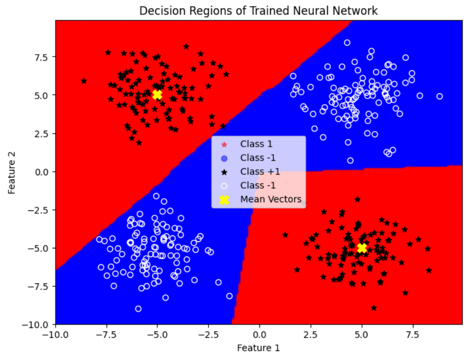


(c) Test set decision boundary

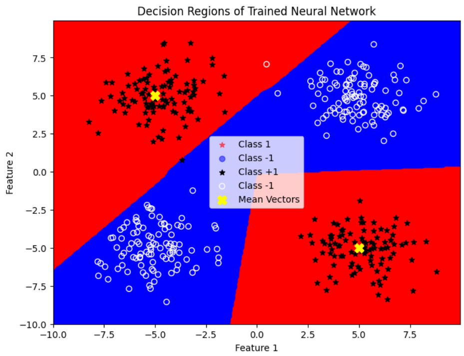
Figure 2: Comparison of decision regions and boundaries for training and test sets for 2 first layer nodes


```
Epoch 100, Learning Rate: 0.010000
Epoch 200, Learning Rate: 0.010000
Epoch 300, Learning Rate: 0.010000
Epoch 400, Learning Rate: 0.010000
Epoch 500, Learning Rate: 0.010000
Epoch 600, Learning Rate: 0.010000
Epoch 700, Learning Rate: 0.010000
Epoch 800, Learning Rate: 0.010000
Epoch 900, Learning Rate: 0.010000
Epoch 1000, Learning Rate: 0.010000
Train Accuracy: 100.0000, Test Accuracy: 99.7500
```

(a) Result of 4 first layer nodes



(b) Training set decision boundary

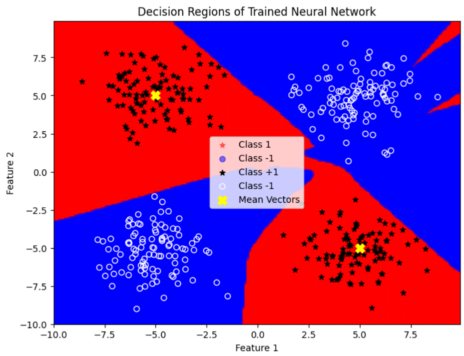


(c) Test set decision boundary

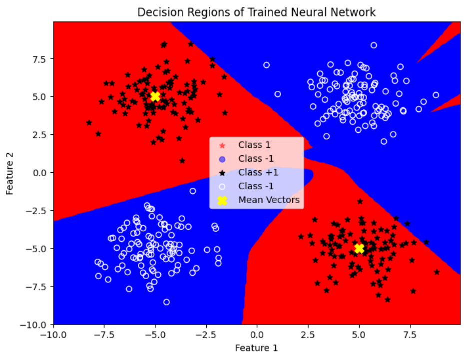
Figure 3: Comparison of decision regions and boundaries for training and test sets for 4 first layer nodes

```
Epoch 100, Learning Rate: 0.010000
Epoch 200, Learning Rate: 0.010000
Epoch 300, Learning Rate: 0.010000
Epoch 400, Learning Rate: 0.010000
Epoch 500, Learning Rate: 0.010000
Epoch 600, Learning Rate: 0.010000
Epoch 700, Learning Rate: 0.010000
Epoch 800, Learning Rate: 0.010000
Epoch 900, Learning Rate: 0.010000
Epoch 1000, Learning Rate: 0.010000
Train Accuracy: 100.0000, Test Accuracy: 99.5000
```

(a) Result of 15 first layer nodes



(b) Training set decision boundary



(c) Test set decision boundary

Figure 4: Comparison of decision regions and boundaries for training and test sets for 15 first layer nodes

From the above figures, we can see that accuracy increases with the number of nodes in the first layer. The decision boundary becomes more complex and better fits the training data, but it also risks overfitting, especially with 15 nodes. The test set accuracy is lower than the training set accuracy, indicating some overfitting.

2.3 Backpropagation Algorithm with Different Learning Rates

(a) Run the backpropagation algorithm with 4 first-layer nodes with the following settings:

– $lr = 0.01$, for 300 iterations.
– $lr = 0.001$, for 300 iterations.
– $lr = 0.01$, for 1000 iterations.
– $lr = 0.001$, for 1000 iterations.

Use the dataset (X_1, y_1) as the training set.

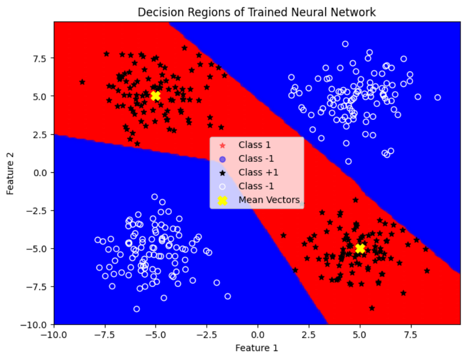
(b) Evaluate the performance of the designed neural networks for both (X_1, y_1) (training set) and (X_2, y_2) (test set) and plot the decision regions.

(c) Comment on the results.

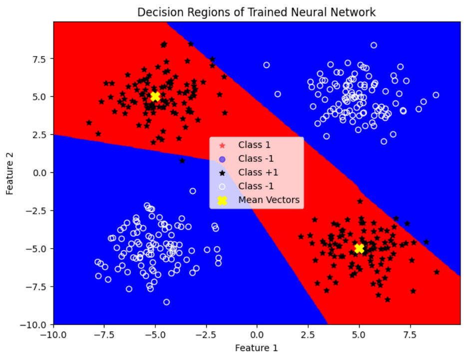
Answer:

```
Epoch 100, Learning Rate: 0.010000
Epoch 200, Learning Rate: 0.010000
Epoch 300, Learning Rate: 0.010000
Train Accuracy: 100.0000, Test Accuracy: 99.2500
```

(a) lr = 0.01 for 300 iterations



(b) Training set decision boundary

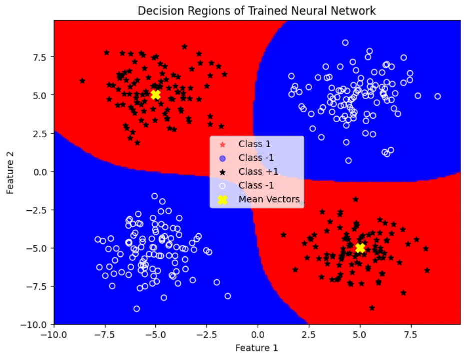


(c) Test set decision boundary

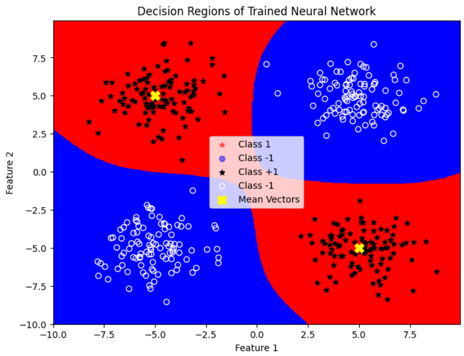
Figure 5: Comparison of decision regions and boundaries for training and test sets for lr = 0.01 for 300 iterations

```
Epoch 100, Learning Rate: 0.001000
Epoch 200, Learning Rate: 0.001000
Epoch 300, Learning Rate: 0.001000
Train Accuracy: 100.0000, Test Accuracy: 100.0000
```

(a) lr = 0.001 for 300 iterations



(b) Training set decision boundary

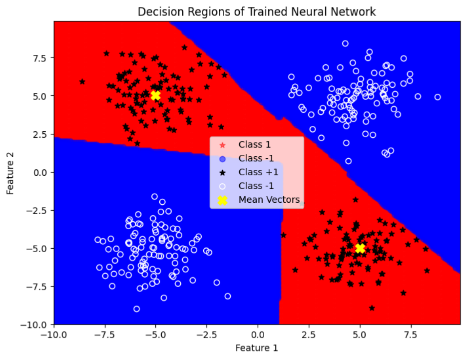


(c) Test set decision boundary

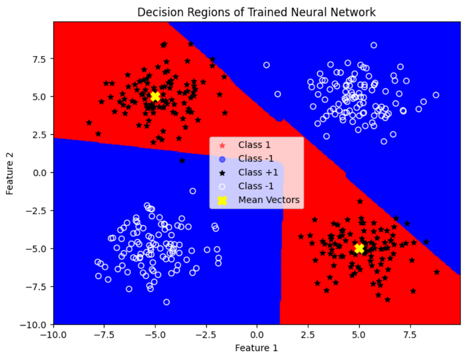
Figure 6: Comparison of decision regions and boundaries for training and test sets for lr = 0.001 for 300 iterations

```
Epoch 100, Learning Rate: 0.010000
Epoch 200, Learning Rate: 0.010000
Epoch 300, Learning Rate: 0.010000
Epoch 400, Learning Rate: 0.010000
Epoch 500, Learning Rate: 0.010000
Epoch 600, Learning Rate: 0.010000
Epoch 700, Learning Rate: 0.010000
Epoch 800, Learning Rate: 0.010000
Epoch 900, Learning Rate: 0.010000
Epoch 1000, Learning Rate: 0.010000
Train Accuracy: 100.0000, Test Accuracy: 99.2500
```

(a) lr = 0.01 for 1000 iterations



(b) Training set decision boundary

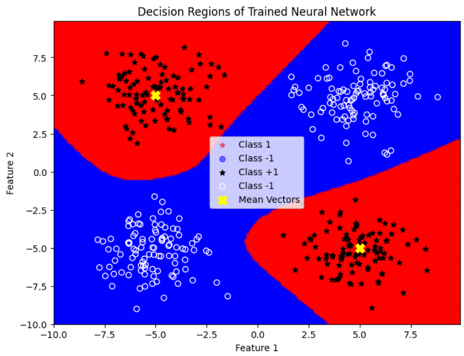


(c) Test set decision boundary

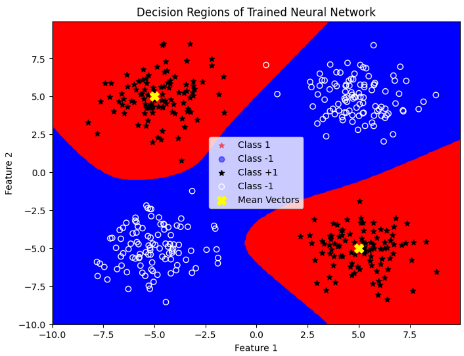
Figure 7: Comparison of decision regions and boundaries for training and test sets for lr = 0.01 for 1000 iterations

```
Epoch 100, Learning Rate: 0.001000
Epoch 200, Learning Rate: 0.001000
Epoch 300, Learning Rate: 0.001000
Epoch 400, Learning Rate: 0.001000
Epoch 500, Learning Rate: 0.001000
Epoch 600, Learning Rate: 0.001000
Epoch 700, Learning Rate: 0.001000
Epoch 800, Learning Rate: 0.001000
Epoch 900, Learning Rate: 0.001000
Epoch 1000, Learning Rate: 0.001000
Train Accuracy: 100.0000, Test Accuracy: 99.7500
```

(a) lr = 0.01 for 1000 iterations



(b) Training set decision boundary



(c) Test set decision boundary

Figure 8: Comparison of decision regions and boundaries for training and test sets for lr = 0.001 for 1000 iterations

From the above figures, we can see that the learning rate significantly affects the convergence speed and the final decision boundary. A higher learning rate (0.01) converges faster but may overshoot the optimal solution, while a lower learning rate (0.001) converges more slowly but can lead to a more stable solution. The number of iterations also plays a crucial role; more iterations allow for better convergence, but they also increase the risk of overfitting, especially with a high learning rate.

2.4 Adaptive Learning Rate Backpropagation

(a) Run the adaptive learning rate variation of the backpropagation algorithm with the following parameters:

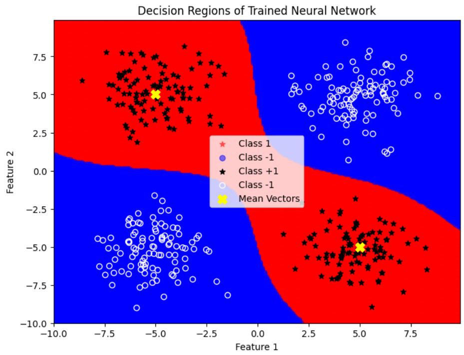
- $lr = 0.001$
- $lr_{inc} = 1.05$
- $lr_{dec} = 0.7$
- $max\_perf\_inc = 1.04$

Run the algorithm for 300 iterations.

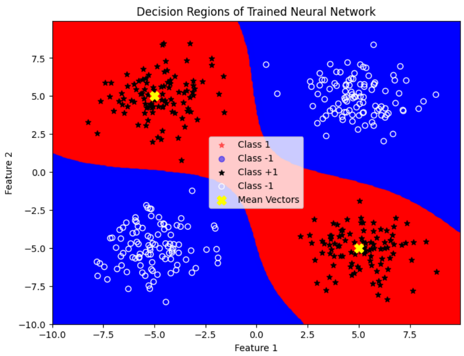
(b) Evaluate the performance of the designed neural networks for both (X_1, y_1) (training set) and (X_2, y_2) (test set) and plot the decision regions.
(c) Compare the above results with those obtained for the standard back-propagation algorithm with $lr = 0.001$, for 300 iterations.
Answer:

```
Epoch 100, Learning Rate: 0.000532
Epoch 200, Learning Rate: 0.000532
Epoch 300, Learning Rate: 0.000532
Train Accuracy: 100.0000, Test Accuracy: 100.0000
```

(a) Result



(b) Training set decision boundary



(c) Test set decision boundary

Figure 9: Comparison of decision regions and boundaries for training and test sets using adaptive learning rate backpropagation

The change in learning rate is not seen for lr= 0.001 but can be seen if its changed to 0.01 with more iterations. This is experimentally checked. From the results, it is evident that the adaptive learning rate backpropagation algorithm performs better than the standard backpropagation algorithm with a fixed learning rate of 0.001 for 300 iterations. The adaptive learning rate allows the model to dynamically adjust the learning rate based on the performance, leading to faster convergence and a more accurate decision boundary. In contrast, the fixed learning rate may result in slower convergence and sub-optimal performance. The decision regions produced by both more or less same as our data set is relatively simple.

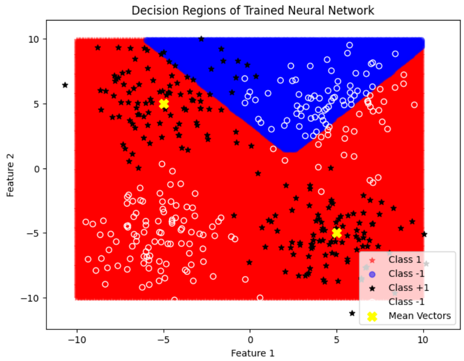
2.5 Repeating Experiments on Different Data Sets

(a) Repeat **Sections 2.2–2.4** using the datasets (X_3, y_3) and (X_4, y_4) as training and test sets, respectively.

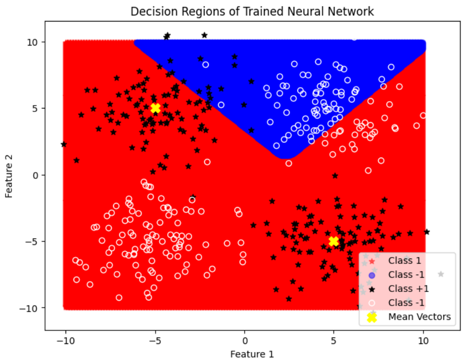
Answer:

```
Epoch 100, Learning Rate: 0.010000
Epoch 200, Learning Rate: 0.010000
Epoch 300, Learning Rate: 0.010000
Epoch 400, Learning Rate: 0.010000
Epoch 500, Learning Rate: 0.010000
Epoch 600, Learning Rate: 0.010000
Epoch 700, Learning Rate: 0.010000
Epoch 800, Learning Rate: 0.010000
Epoch 900, Learning Rate: 0.010000
Epoch 1000, Learning Rate: 0.010000
Train Accuracy: 63.2500, Test Accuracy: 62.5000
```

(a) Result of 2 first layer nodes

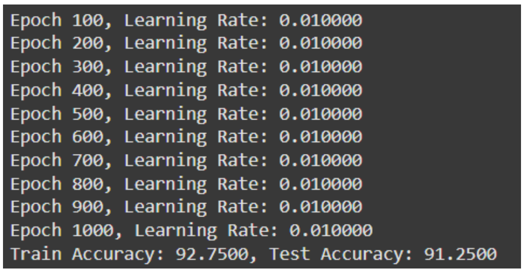


(b) Training set decision boundary



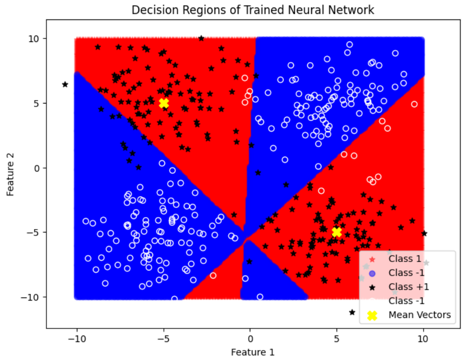
(c) Test set decision boundary

Figure 10: Comparison of decision regions and boundaries for training and test sets for 2 first layer nodes

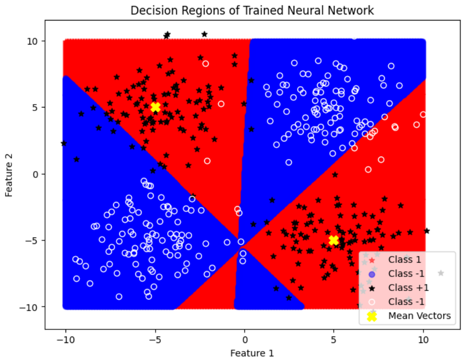


```
Epoch 100, Learning Rate: 0.010000
Epoch 200, Learning Rate: 0.010000
Epoch 300, Learning Rate: 0.010000
Epoch 400, Learning Rate: 0.010000
Epoch 500, Learning Rate: 0.010000
Epoch 600, Learning Rate: 0.010000
Epoch 700, Learning Rate: 0.010000
Epoch 800, Learning Rate: 0.010000
Epoch 900, Learning Rate: 0.010000
Epoch 1000, Learning Rate: 0.010000
Train Accuracy: 92.7500, Test Accuracy: 91.2500
```

(a) Result of 4 first layer nodes

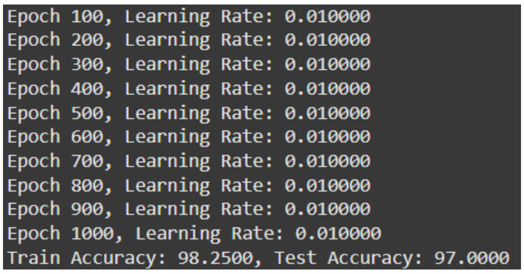


(b) Training set decision boundary

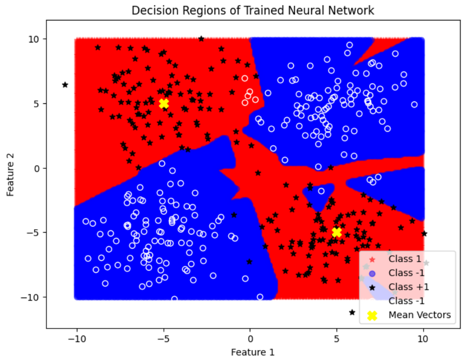


(c) Test set decision boundary

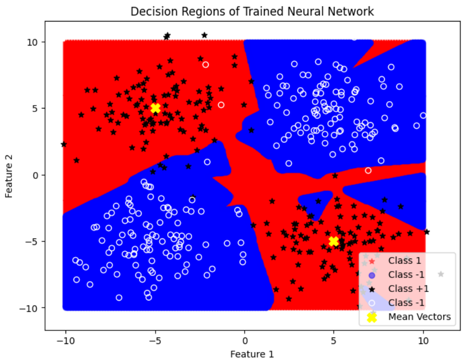
Figure 11: Comparison of decision regions and boundaries for training and test sets for 4 first layer nodes



(a) Result of 15 first layer nodes

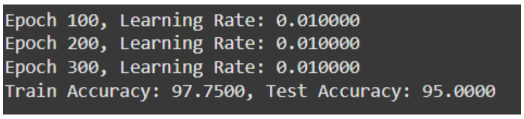


(b) Training set decision boundary



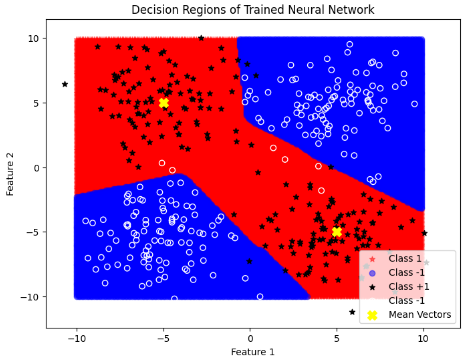
(c) Test set decision boundary

Figure 12: Comparison of decision regions and boundaries for training and test sets for 15 first layer nodes

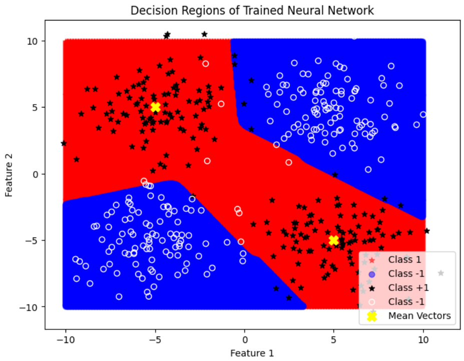


```
Epoch 100, Learning Rate: 0.010000
Epoch 200, Learning Rate: 0.010000
Epoch 300, Learning Rate: 0.010000
Train Accuracy: 97.7500, Test Accuracy: 95.0000
```

(a) lr = 0.01 for 300 iterations

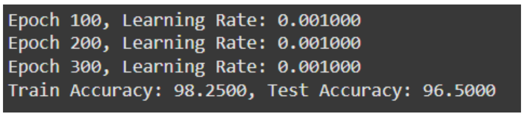


(b) Training set decision boundary



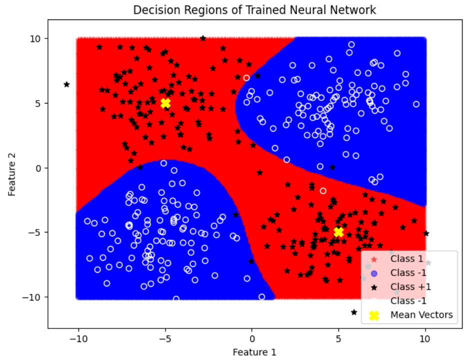
(c) Test set decision boundary

Figure 13: Comparison of decision regions and boundaries for training and test sets for lr = 0.01 for 300 iterations

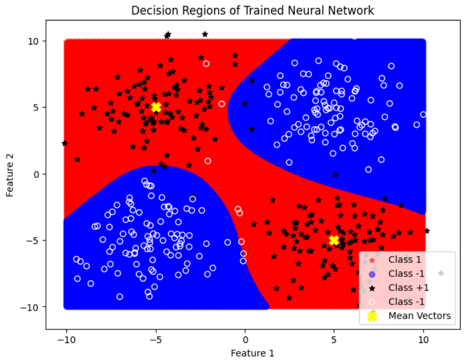


```
Epoch 100, Learning Rate: 0.001000
Epoch 200, Learning Rate: 0.001000
Epoch 300, Learning Rate: 0.001000
Train Accuracy: 98.2500, Test Accuracy: 96.5000
```

(a) lr = 0.001 for 300 iterations



(b) Training set decision boundary

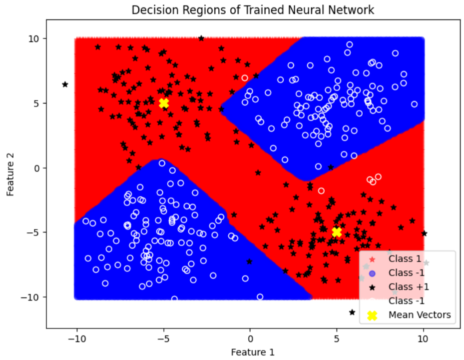


(c) Test set decision boundary

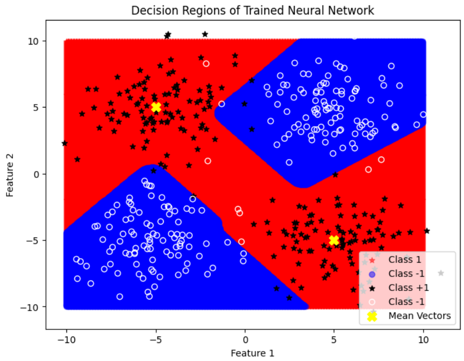
Figure 14: Comparison of decision regions and boundaries for training and test sets for lr = 0.001 for 300 iterations

```
Epoch 100, Learning Rate: 0.010000
Epoch 200, Learning Rate: 0.010000
Epoch 300, Learning Rate: 0.010000
Epoch 400, Learning Rate: 0.010000
Epoch 500, Learning Rate: 0.010000
Epoch 600, Learning Rate: 0.010000
Epoch 700, Learning Rate: 0.010000
Epoch 800, Learning Rate: 0.010000
Epoch 900, Learning Rate: 0.010000
Epoch 1000, Learning Rate: 0.010000
Train Accuracy: 98.0000, Test Accuracy: 96.7500
```

(a) lr = 0.01 for 1000 iterations

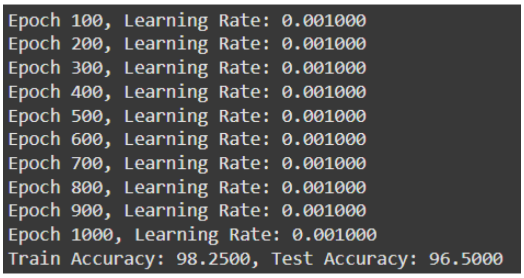


(b) Training set decision boundary



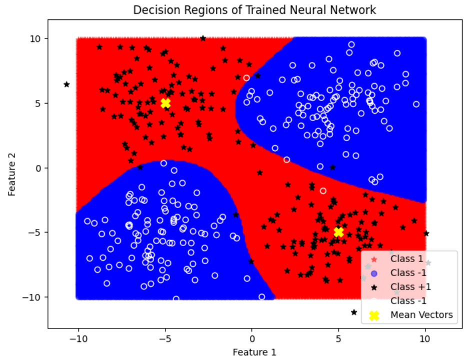
(c) Test set decision boundary

Figure 15: Comparison of decision regions and boundaries for training and test sets for lr = 0.01 for 1000 iterations

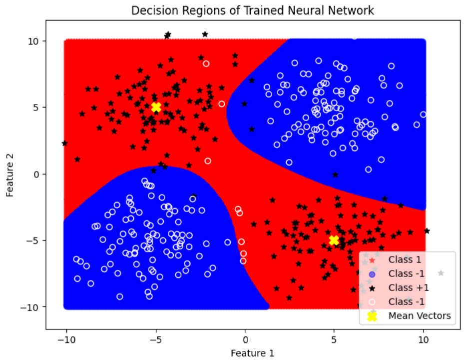


```
Epoch 100, Learning Rate: 0.001000
Epoch 200, Learning Rate: 0.001000
Epoch 300, Learning Rate: 0.001000
Epoch 400, Learning Rate: 0.001000
Epoch 500, Learning Rate: 0.001000
Epoch 600, Learning Rate: 0.001000
Epoch 700, Learning Rate: 0.001000
Epoch 800, Learning Rate: 0.001000
Epoch 900, Learning Rate: 0.001000
Epoch 1000, Learning Rate: 0.001000
Train Accuracy: 98.2500, Test Accuracy: 96.5000
```

(a) lr = 0.01 for 1000 iterations



(b) Training set decision boundary

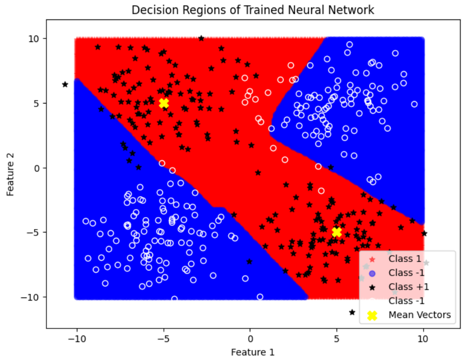


(c) Test set decision boundary

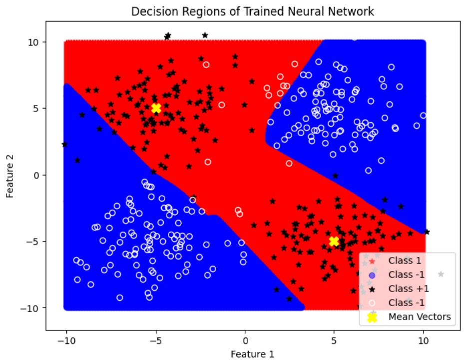
Figure 16: Comparison of decision regions and boundaries for training and test sets for lr = 0.001 for 1000 iterations

```
Epoch 100, Learning Rate: 0.001202
Epoch 200, Learning Rate: 0.002674
Epoch 300, Learning Rate: 0.002102
Train Accuracy: 92.7500, Test Accuracy: 93.0000
```

(a) Result



(b) Training set decision boundary



(c) Test set decision boundary

Figure 17: Comparison of decision regions and boundaries for training and test sets using adaptive learning rate backpropagation

From above results we see that while repeating the same set of experiments with data set with little high variance the model accuracy is less and decision boundaries are less sharp. This is expected as the data points are more spread out, making it harder for the model to learn clear decision boundaries. Increasing the number of iterations or using more complex models might improve the performance.