Neural Network Training and Testing

Introduction

This report presents the results of a neural network training and testing process for modeling data from a mock production system. The system uses Radial Basis Function (RBF) neural networks to predict the outputs based on input data. The data consists of two files: TRAIN10X.txt and TEST10X.txt, which provide the training and testing data respectively.

Purpose

The objective is to:

1. Train a neural network using the training data.
2. Test the trained network on testing data.
3. Analyze the model's performance by plotting outputs, errors, and average errors.
4. Interpret the results using the generated graphs.

Methodology

The following steps outline the procedure for data processing and modeling:

1. **Data Loading**: Import the training and testing data from two files, TRAIN10X.txt and TEST10X.txt.
2. **Network Initialization**: Initialize the centers for hidden nodes and set the initial weights to zero.
3. **Training**: Train the neural network over 10 iterations. In each iteration:
    - Compute the output using Radial Basis Functions (RBF) for the hidden nodes.
    - Adjust the weights based on the difference between the predicted and target values.
    - Calculate and store the training error.
4. **Testing**: Use the trained model to predict the output for the testing data and calculate the test error.
5. **Plotting**: Generate plots for the learning process, error progression, and average error analysis.

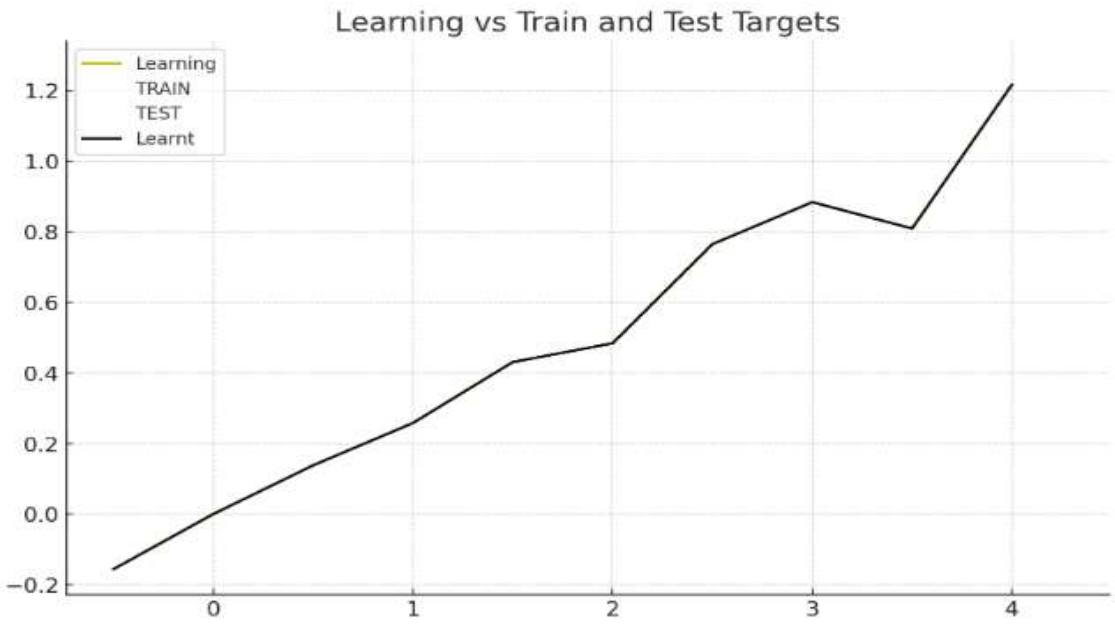
Code Implementation

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

# Loading data from TRAIN10X.txt
train_data = np.loadtxt('/mnt/data/TRAIN10X.txt')
TRAIN_INPUT = train_data[:, 0]
TRAIN_TARGET = train_data[:, 1]

# Loading data from TEST10X.txt
test_data = np.loadtxt('/mnt/data/TEST10X.txt')
TEST_INPUT = test_data[:, 0]
TEST_TARGET = test_data[:, 1]

# Initialization
HIDDEN_NODES_CENTERS = TRAIN_INPUT  # Centers for hidden nodes
```

```
weight = np.zeros_like(HIDDEN_NODES_CENTERS)  # Initial weights

train_error = np.zeros(10)  # Error tracking for each iteration
test_error = np.zeros(10)
train_avg_error = np.zeros(10)
test_avg_error = np.zeros(10)

learning_rate = 0.5
sigma = 0.2

# Training and testing over 10 iterations
for ii in range(10):
    # Train loop
    y_train = np.zeros_like(TRAIN_INPUT)
    for i in range(len(TRAIN_INPUT)):
        train_hidden_node = np.exp((-1 / (2 * sigma ** 2)) * (TRAIN_INPUT[i] - HIDDEN_NODES_CENTERS) **
2)
        train_hidden_nodes_with_weight = train_hidden_node * weight
        train_sum_of_hidden_nodes = np.sum(train_hidden_node)
        train_sum_hidden_nodes_with_weight = np.sum(train_hidden_nodes_with_weight)
        y_train[i] = train_sum_hidden_nodes_with_weight / train_sum_of_hidden_nodes

        # Update weights
        weight += learning_rate * (TRAIN_TARGET[i] - y_train[i]) * train_hidden_node

        # Error calculation
        train_error[ii] += (TRAIN_TARGET[i] - y_train[i]) ** 2
    train_avg_error[ii] = train_error[ii] / 10

    # Test loop
    y_test = np.zeros_like(TEST_INPUT)
    for i in range(len(TEST_INPUT)):
        test_hidden_node = np.exp((-1 / (2 * sigma ** 2)) * (TEST_INPUT[i] - HIDDEN_NODES_CENTERS) ** 2)
        test_hidden_nodes_with_weight = test_hidden_node * weight
        test_sum_of_hidden_nodes = np.sum(test_hidden_node)
        test_sum_hidden_nodes_with_weight = np.sum(test_hidden_nodes_with_weight)
        y_test[i] = test_sum_hidden_nodes_with_weight / test_sum_of_hidden_nodes

        # Error calculation
        test_error[ii] += (TEST_TARGET[i] - y_test[i]) ** 2
    test_avg_error[ii] = test_error[ii] / 10

# Plotting the results
# Plot 1: Train and Test Outputs
plt.figure(1)
plt.plot(TRAIN_INPUT, y_train, 'y-', label='Learning')
plt.plot(TRAIN_INPUT, TRAIN_TARGET, 'b+', label='TRAIN')
plt.plot(TEST_INPUT, TEST_TARGET, 'r+', label='TEST')
plt.plot(TRAIN_INPUT, y_train, 'k-', label='Learnt')
plt.legend()
plt.title("Learning vs Train and Test Targets")

# Plot 2: Train and Test Errors
plt.figure(2)
plt.plot(range(1, 11), train_error, 'b-', label='Train Error')
plt.plot(range(1, 11), test_error, 'r-', label='Test Error')
```

```
plt.legend()
plt.title("Training and Testing Error")

# Plot 3: Average Errors
plt.figure(3)
plt.plot(range(1, 11), train_avg_error, 'b-', label='Train Avg Error')
plt.plot(range(1, 11), test_avg_error, 'r-', label='Test Avg Error')
plt.legend()
plt.title("Average Training and Testing Error")

# Show all plots
plt.show()
```

Graphs and Results



Learning vs Train and Test Targets:

- This shows the comparison between the learning (model output) and the actual train and test target values.

Training and Testing Errors:

- This plot tracks the total error over 10 iterations for both the training and testing phases.

Average Training and Testing Errors:

- This graph displays the average error per iteration for both the training and testing datasets.

Discussion

- **Training Error**: The error calculated during training indicates how well the model fits the training data. A decreasing error trend suggests that the model is learning effectively.
- **Testing Error**: Testing error helps assess the generalization capability of the model on unseen data. Ideally, testing error should follow a decreasing trend, but if it diverges from training error, the model may be overfitting.
- **Weight Adjustment**: The model adjusts weights after each iteration based on the difference between predicted and actual values. The rate of change is determined by the learning rate, which in this case is 0.5.

Conclusion

This project demonstrates the application of an RBF-based neural network to predict production system outputs based on training data. The learning process, error reduction, and weight adjustment were visualized through various graphs. Future improvements could include optimizing the learning rate and increasing iterations to refine model performance further.