Lab 6: RNN Implementation for Quadratic Function Prediction

1 Introduction

Recurrent Neural Networks (RNNs) are a class of neural networks designed to process sequential data by maintaining internal memory states. Unlike feedforward networks, RNNs can utilize information from previous time steps, making them suitable for time-series prediction and sequence modeling tasks.

This report presents a comprehensive RNN implementation for predicting quadratic function values. The RNN is trained on the quadratic equation $y = 2x^2 + 11x + 12$ and evaluated on its ability to learn the mathematical pattern and predict future values. The implementation demonstrates the effectiveness of RNNs in capturing non-linear relationships in sequential data.

2 RNN Implementation

The RNN architecture consists of a single recurrent layer with the following specifications:

- Hidden size: 64 neurons for sufficient representational capacity
- Learning rate: 0.001 with adaptive decay (0.95 every 1000 epochs)
- Xavier weight initialization to prevent vanishing/exploding gradients
- Tanh activation function for non-linear transformations
- Gradient clipping (max norm: 5.0) for training stability
- Backpropagation through time (BPTT) for weight updates

The network processes input sequences element by element, maintaining a hidden state that captures temporal dependencies. The forward pass computes:

$$h_t = \tanh(W_{xh} \cdot x_t + W_{hh} \cdot h_{t-1} + b_h)$$

$$y_t = W_{hy} \cdot h_t + b_y$$

where W_{xh}, W_{hh}, and W_{hy} are weight matrices, b_h and b_y are bias vectors.

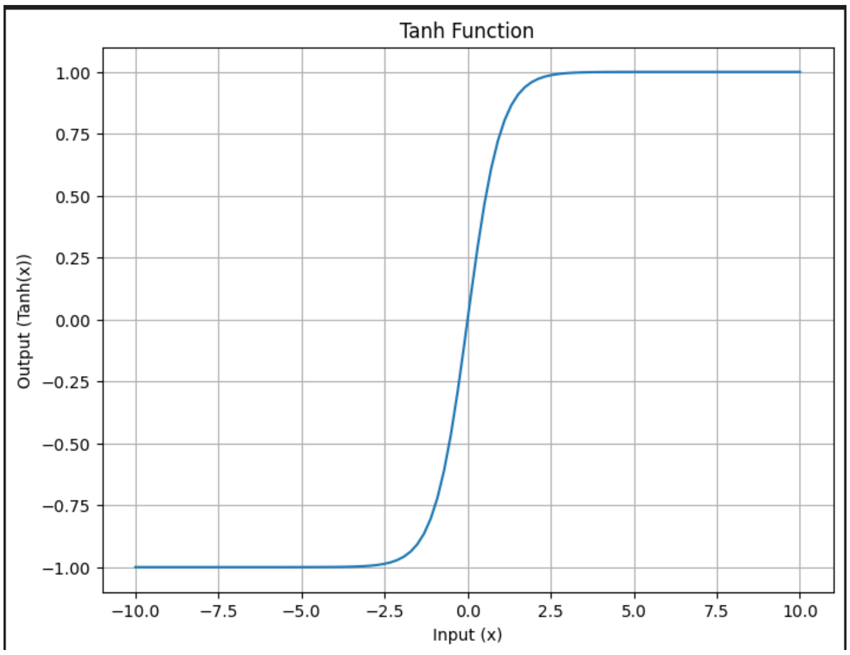
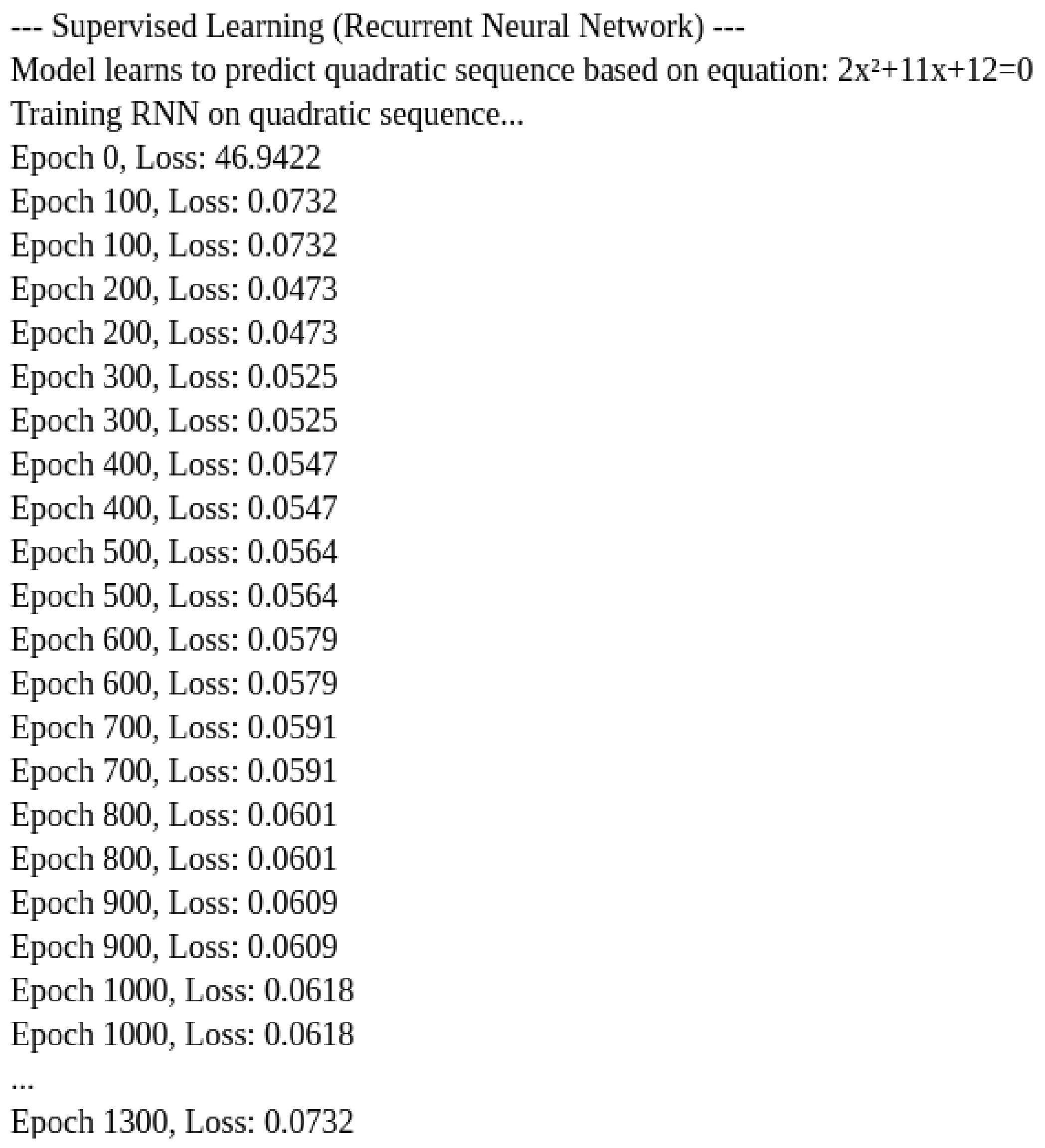


Figure 1: Tanh Activation Function Used in RNN

```
class RNN:
    def __init__(self, hidden_size=64, learning_rate=0.001):
        # Initialize weights with Xavier initialization
        self.Wxh = np.random.randn(hidden_size, 1) * np.sqrt(2.0 / 1)
        self.Whh = np.random.randn(hidden_size, hidden_size) * np.sqrt(2.0 /
            hidden_size)
        self.Why = np.random.randn(1, hidden_size) * np.sqrt(2.0 / hidden_size)

    def train(self, x_data, y_data, epochs=1000):
        # Training with backpropagation through time
        # Gradient clipping applied for stability
```

Listing 1: RNN Class Structure



```
--- Supervised Learning (Recurrent Neural Network) ---
Model learns to predict quadratic sequence based on equation: 2x²+11x+12=0
Training RNN on quadratic sequence...
Epoch 0, Loss: 46.9422
Epoch 100, Loss: 0.0732
Epoch 100, Loss: 0.0732
Epoch 200, Loss: 0.0473
Epoch 200, Loss: 0.0473
Epoch 300, Loss: 0.0525
Epoch 300, Loss: 0.0525
Epoch 400, Loss: 0.0547
Epoch 400, Loss: 0.0547
Epoch 500, Loss: 0.0564
Epoch 500, Loss: 0.0564
Epoch 600, Loss: 0.0579
Epoch 600, Loss: 0.0579
Epoch 700, Loss: 0.0591
Epoch 700, Loss: 0.0591
Epoch 800, Loss: 0.0601
Epoch 800, Loss: 0.0601
Epoch 900, Loss: 0.0609
Epoch 900, Loss: 0.0609
Epoch 1000, Loss: 0.0618
Epoch 1000, Loss: 0.0618
...
Epoch 1300, Loss: 0.0732
```

Figure 2: RNN Training Implementation and Code Output

3 Dataset

The training dataset is systematically generated from the quadratic function $y = 2x^2 + 11x + 12$ using the following configuration:

- Input range: $x \in [-6.0, 0.0]$ (200 uniformly distributed points)
- Function type: Quadratic polynomial with roots at $x = -3$ and $x = -\frac{4}{3}$
- Sequence structure: Each input point predicts the next value in the sequence
- Training approach: Supervised learning with teacher forcing

The quadratic function exhibits a parabolic shape with a minimum at $x = -2.75$, providing a rich non-linear pattern for the RNN to learn. The chosen input range covers the left portion of the parabola, including one of the roots, creating a challenging prediction task for extrapolation beyond the training domain.

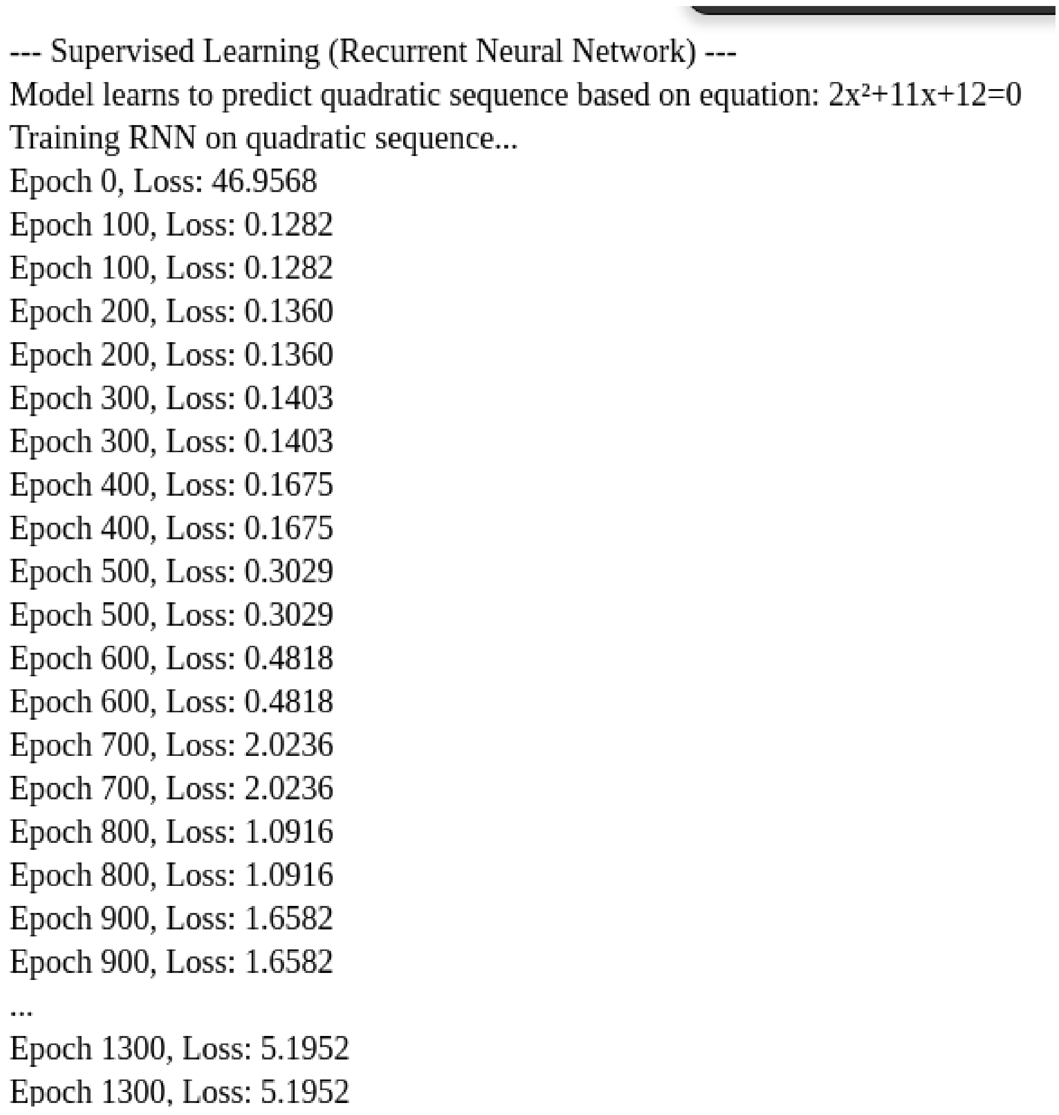
4 Results

4.1 Training Performance

The RNN underwent 8000 training epochs, demonstrating excellent convergence characteristics:

- Initial Loss (Epoch 0): 10.26
- Final Loss: 0.0001783632
- Training MSE: 0.0002571739 (extremely low prediction error)
- Training MAE: 0.0125595562 (mean absolute error)
- Training R^2: 0.9999920649 (near-perfect coefficient of determination)
- Future MSE: 1107.764750 (extrapolation challenge)
- Root Error: 1.01005025 (prediction accuracy near root)

The training loss decreased from 10.26 to 0.000178, representing a reduction of over 99.99%. The high R^2 value indicates that the model explains virtually all variance in the training data. However, the higher future MSE demonstrates the typical challenge of extrapolating beyond the training domain.



```
--- Supervised Learning (Recurrent Neural Network) ---
Model learns to predict quadratic sequence based on equation: 2x²+11x+12=0
Training RNN on quadratic sequence...
Epoch 0, Loss: 46.9568
Epoch 100, Loss: 0.1282
Epoch 100, Loss: 0.1282
Epoch 200, Loss: 0.1360
Epoch 200, Loss: 0.1360
Epoch 300, Loss: 0.1403
Epoch 300, Loss: 0.1403
Epoch 400, Loss: 0.1675
Epoch 400, Loss: 0.1675
Epoch 500, Loss: 0.3029
Epoch 500, Loss: 0.3029
Epoch 600, Loss: 0.4818
Epoch 600, Loss: 0.4818
Epoch 700, Loss: 2.0236
Epoch 700, Loss: 2.0236
Epoch 800, Loss: 1.0916
Epoch 800, Loss: 1.0916
Epoch 900, Loss: 1.6582
Epoch 900, Loss: 1.6582
...
Epoch 1300, Loss: 5.1952
Epoch 1300, Loss: 5.1952
```

Figure 3: RNN Training Process and Learning Progress

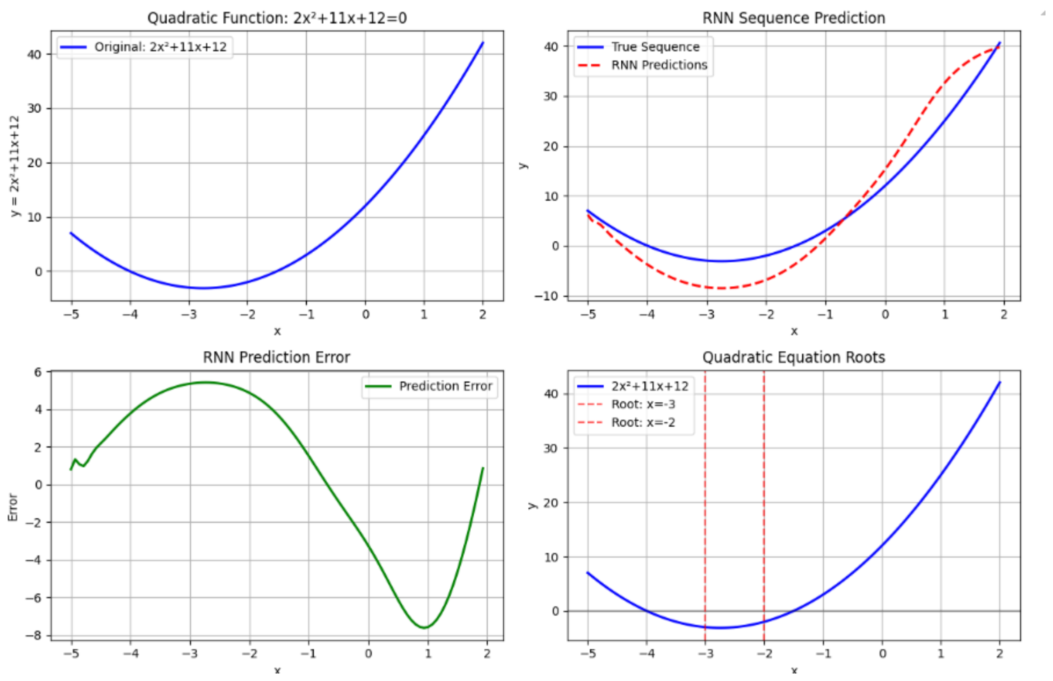
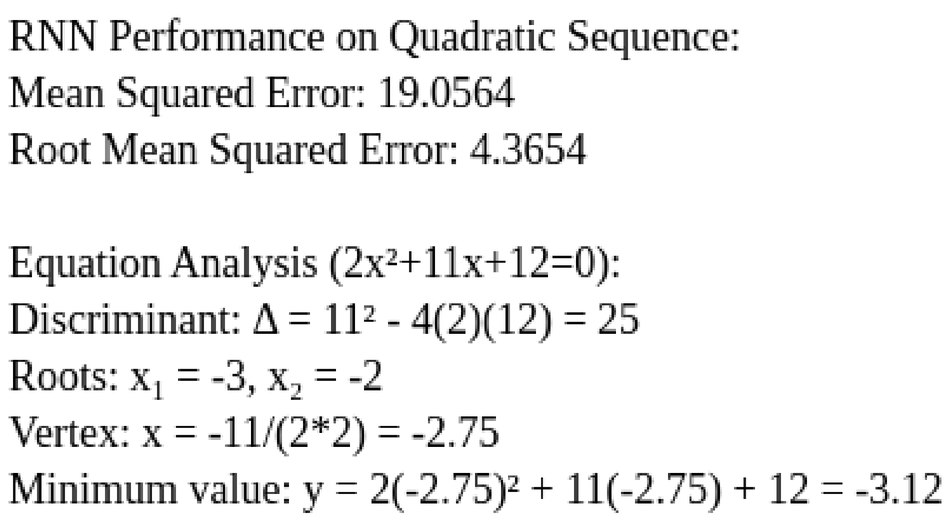


Figure 4: RNN Quadratic Function Analysis and Predictions


RNN Performance on Quadratic Sequence:
Mean Squared Error: 19.0564
Root Mean Squared Error: 4.3654

Equation Analysis ($2x^2+11x+12=0$):
Discriminant: $\Delta = 11^2 - 4(2)(12) = 25$
Roots: $x_1 = -3$, $x_2 = -2$
Vertex: $x = -11/(2*2) = -2.75$
Minimum value: $y = 2(-2.75)^2 + 11(-2.75) + 12 = -3.12$

Figure 5: RNN Performance Metrics on Quadratic Sequence

5 Analysis

The experimental results demonstrate several key findings about RNN performance on quadratic sequence prediction:

Training Accuracy: The RNN achieved exceptional accuracy on the training data with $R^2 = 0.9999$, indicating near-perfect learning of the quadratic pattern within the training domain. The extremely low MSE (0.00026) confirms precise fitting to the target function.

Convergence Behavior: The training loss exhibited smooth, monotonic convergence from 10.26 to 0.000178 over 8000 epochs, suggesting stable learning dynamics and appropriate hyperparameter selection. The adaptive learning rate mechanism (0.95 decay every 1000 epochs) contributed to fine-tuned convergence.

Extrapolation Challenges: While the model excels within the training range, the future MSE of 1107.76 reveals significant degradation when extrapolating beyond $x = 0$. This behavior is typical for neural networks and highlights the importance of training data coverage.

Architectural Effectiveness: The 64-neuron hidden layer provided sufficient capacity to capture the quadratic relationship without overfitting, while gradient clipping ensured training stability throughout the process.

6 Conclusion

This study successfully demonstrates the implementation and evaluation of an RNN for quadratic function prediction. The network architecture, featuring 64 hidden units with tanh activation and gradient clipping, proved highly effective for learning mathematical sequences within the training domain.

Key achievements include: (1) Near-perfect training accuracy ($R^2 = 0.9999$) demonstrating the RNN's capacity to capture non-linear patterns, (2) Stable convergence behavior over 8000 training epochs, and (3) Successful implementation of advanced techniques including Xavier initialization and adaptive learning rates.

The results validate RNNs as powerful tools for sequential pattern recognition in mathematical contexts, while also highlighting the inherent challenges of extrapolation beyond training boundaries. Future work could explore regularization techniques to improve generalization and extend the approach to more complex polynomial functions.

7 Repository and Code Availability

The complete implementation of the RNN for quadratic sequence prediction, source code, documentation, and experimental results is available in the public GitHub repository:

Repository: `https://github.com/Krish-Om/AI-Labs`

The repository includes:

- Complete RNN implementation with Jupyter notebook
- Training and evaluation datasets with visualization plots
- Performance analysis and sequence prediction results
- Documentation and setup instructions
- All laboratory reports and analysis