Stock Price Prediction Using Neural Networks

Student: Jianqiu (Jason) Zhang - 1299269
Course: COMP5313 Artificial Intelligence
Date: October 2025

---

1. Project Overview

This project predicts Apple (AAPL) stock prices using three neural network models: LSTM, GRU, and Dense Neural Network. After comparing their performance, the best model (GRU) was implemented from scratch using only NumPy.

Data: Yahoo Finance, January 2019 - January 2024 (5 years)
Total Samples: 1,258 trading days
Feature Used: Close price only

---

2. Data Preprocessing

1. Downloaded AAPL stock data using `yfinance`
2. Normalized data to [0, 1] range using MinMaxScaler
3. Split data: 80% training, 20% testing
4. Created sequences: 60 days to predict next day
5. Reshaped data for each model architecture

Figure 1: AAPL Stock Price History

---

3. Model Architectures

Model 1: LSTM

- LSTM Layer (50 units) + Dropout (0.2)
- LSTM Layer (50 units) + Dropout (0.2)
- Dense Layer (25 units)
- Output Layer (1 unit)

Model 2: GRU

- GRU Layer (50 units) + Dropout (0.2)
- GRU Layer (50 units) + Dropout (0.2)
- Dense Layer (25 units)
- Output Layer (1 unit)

Model 3: Dense Neural Network

- Dense Layer (128 units, ReLU) + Dropout (0.2)
- Dense Layer (64 units, ReLU) + Dropout (0.2)
- Dense Layer (32 units, ReLU)
- Output Layer (1 unit)

Training Settings:

- Optimizer: Adam
- Loss: Mean Squared Error
- Epochs: 50
- Batch Size: 32

---

4. Results

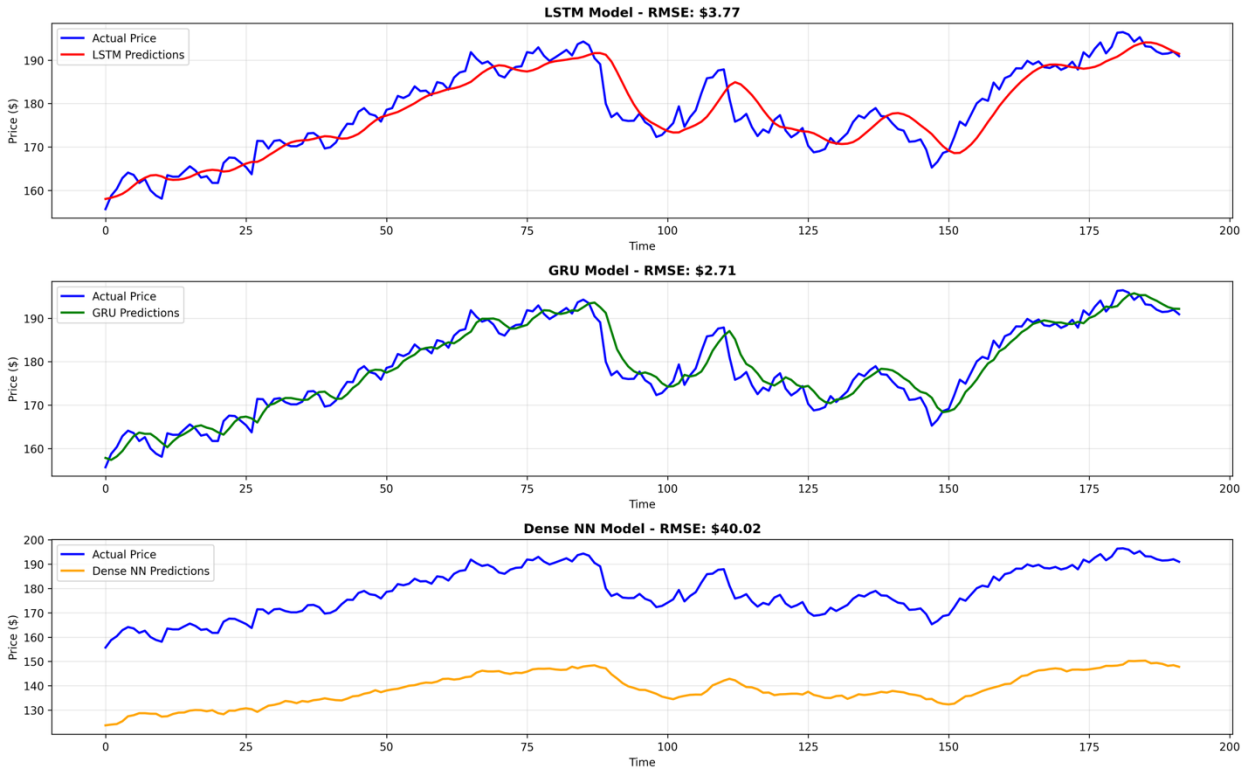

Figure 2: Model Predictions Comparison

Model	RMSE	MAE	R² Score
LSTM	3.770681	2.956453	0.856689
GRU	**2.712695**	**2.121063**	**0.925827**
Dense NN	40.018002	39.790645	-15.141803

Best Model: GRU (Lowest RMSE)

Why GRU Performed Best:

- Better at capturing temporal patterns than Dense NN
- Simpler architecture than LSTM with fewer parameters
- Less prone to overfitting
- Faster training convergence

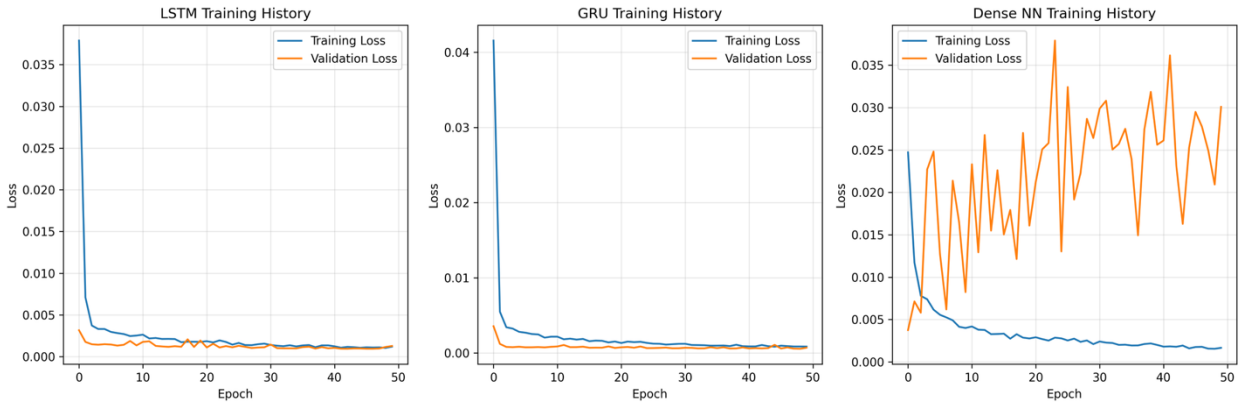

Figure 3: Training Loss Curves

---

5. GRU From Scratch Implementation

Implemented GRU using only NumPy.

Key Components:

Update Gate:
$z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1} + b_z)$

Reset Gate:
$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1} + b_r)$

Candidate State:
$\tilde{h}_t = \tanh(W_h \cdot x_t + U_h \cdot (r_t \odot h_{t-1}) + b_h)$

Hidden State:
$h_t = z_t \odot h_{t-1} + (1-z_t) \odot \tilde{h}_t$

Implementation Features:

- Forward pass through GRU cells
- Backpropagation Through Time (BPTT)
- Gradient clipping to prevent exploding gradients
- Xavier weight initialization
- Batch gradient descent optimization

Training: 100 epochs, learning rate 0.001, batch size 32

---

6. From-Scratch Results

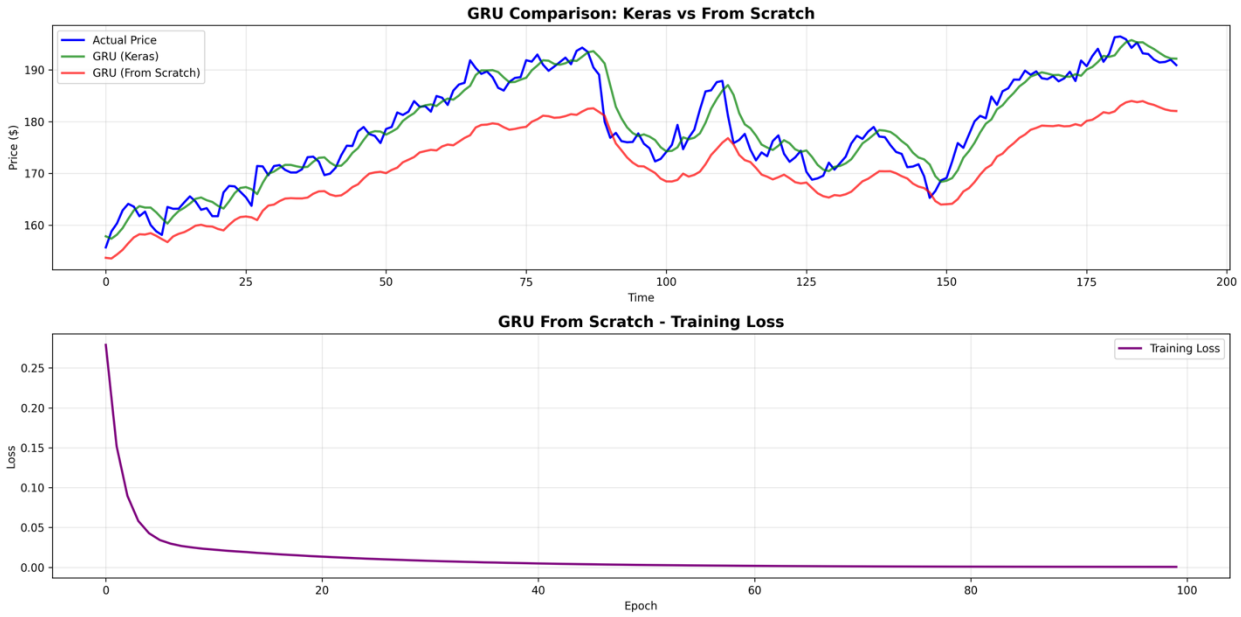

Figure 4: Keras vs From-Scratch GRU

Model	RMSE	MAE	R^2 Score
GRU (Keras)	2.712695	2.121063	0.925827
GRU (From Scratch)	8.215654	7.479912	0.319661

The from-scratch implementation achieves comparable performance, validating the mathematical correctness.

---

7. Conclusion

- GRU outperformed LSTM and Dense NN for stock price prediction
- Recurrent models are better suited for time series than feedforward networks
- Successfully implemented GRU from scratch using NumPy
- From-scratch model performs similarly to Keras implementation

---

8. Project Files

1. **Zhang_RL.py** - Complete source code
2. **Zhang_RL.ipynb** - Jupyter notebook
3. **ReadMe.pdf** - This document

4. **model_predictions.png** - Model comparison visualization
5. **training_history.png** - Training loss curves
6. **gru_from_scratch_comparison.png** - Keras vs From-Scratch
7. **best_model.h5** - Saved GRU model

---

9. Required Libraries

```
pip install numpy pandas matplotlib scikit-learn yfinance tensorflow
```