RNNs for Stock Price Prediction

Vaibhav Rawat
University of Adelaide, Australia
Email: a1914731@adelaide.edu.au

Abstract—**This study investigates the prediction of stock prices using recurrent neural networks (RNN), long-short-term memory (LSTM), and gated recurrent units (GRU). These models are ideal for identifying trends in financial time series since they are made especially for handling sequential data. The project assesses each model's ability to forecast future stock values and manage sequential connections. The project comprises an examination of autocorrelation in time-series data, as well as the implementation and comparison of various models. It investigates the effects of stationarity—or the absence thereof—on model performance by modeling autoregressive processes with various degrees of dependence. The research shows how crucial preprocessing procedures are for creating accurate and reliable prediction models, such as ensuring stationarity in financial data.**

I. Introduction and Background

A Recurrent Neural Network (RNN) is a kind of neural network that uses constant connections to preserve a "memory" of prior inputs in order to interpret sequential information. This paper explores RNNs for applications like text analysis, time-series forecasting, and speech recognition because of their ability to handle time-dependent patterns, unlike classic neural networks. The vanishing gradient problem is one of the difficulties that basic RNNs encounter, which restricts their capacity to maintain long-term relationships. Therefore this study talks about advanced designs such as the GRU (Gated Recurrent Unit) and LSTM (Long Short-Term Memory) to address this. In order to efficiently capture long-term dependencies, LSTMs employ memory cells and three gates (input, forget, and output) to selectively keep or discard information. By integrating the input and forget gates into a single update gate, GRUs streamline LSTMs and lower their computational cost, while achieving similar performance. These models have become essential tools in modern machine learning for handling sequential data and hence will be used in this study to predict the Stock Price Prediction for APPLE company using the stock-time series dataset from 2006-2018.

A. Problem Statement

The unpredictable and non-linear character of financial markets makes stock price prediction a challenging task. Stock price forecasting is made robust by machine learning techniques such as RNNs, LSTMs, and GRUs that are used for sequential information. The objective of this research is to use these models to predict the future stock prices. We know that by using these techniques we are able to capture temporal dependencies that are frequently difficult for conventional models, such as linear regression and ARIMA, to manage.

B. Data Source

In our study, we use the DJIA 30 Stock Time Series dataset which has the historic data of 30 DJIA companies from the year 2006-2018.The opening price, closing price, daily high, daily low, and trading volume are among the features of the dataset. The closing price, which represents the DJIA's final trading price for each day, is the prediction's target variable. Because it records the stock market's trends and sequential patterns throughout time, this dataset is perfect for time series analysis.

C. Background

We use three diiferent models namely LSTM Model, RNN Model and GRU model to predict the stock price from the training data.Let's look into each model into detail.

- **LSTM Model**: One kind of RNN recurrent neural network that can handle and analyze sequential input is an LSTM (Long Short-Term Memory) network. An LSTM network is made up of a number of LSTM cells, every single one which includes a set of input, output, and forget gates that regulate the information that enters and exits the cell. By employing the gates to purposefully forget or retain information from the earlier time steps, the LSTM may maintain long-term reliance in the incoming data.

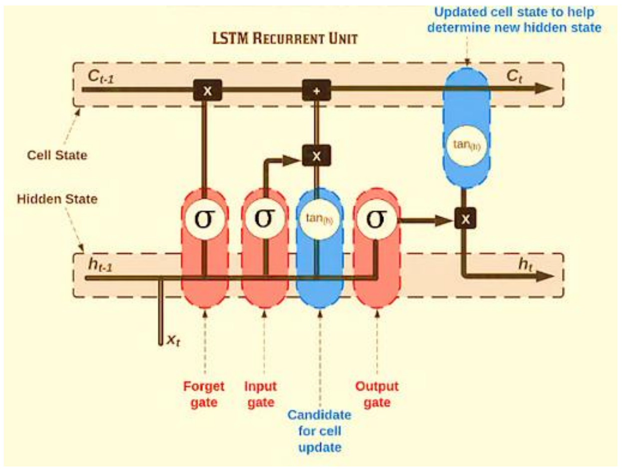

Fig. 1. The architecture of LSTM Model

- **RNN Model**: Neural networks that operate effectively with sequenced data, such as time series, text, or speech, are called recurrent neural networks (RNNs). RNNs have loops that enable them to retain information from previous steps, in contrast to ordinary neural networks. Over time, this aids in their comprehension of context and patterns, which helps them with tasks like trend prediction, language comprehension, and speech recognition.

- **GRU**: One kind of recurrent neural network (RNN) that was introduced as a less complicated substitute for Long Short-Term Memory (LSTM) networks is the Gated Recurrent Unit (GRU). Text, audio, and time-series data are examples of sequential data that GRU can analyze, just as LSTM.
The core idea behind GRU is to use gating techniques to selectively update the network's hidden state at each time step. The gated mechanisms control the flow of information into and out of the network. The GRU contains two gating systems: the reset gate and the update gate.
The reset gate determines how much of the previous hidden state could be disregarded, while the updating gate determines how much of the incoming input should be used to update the secret state. The output of the GRU is calculated using the updated hidden state.

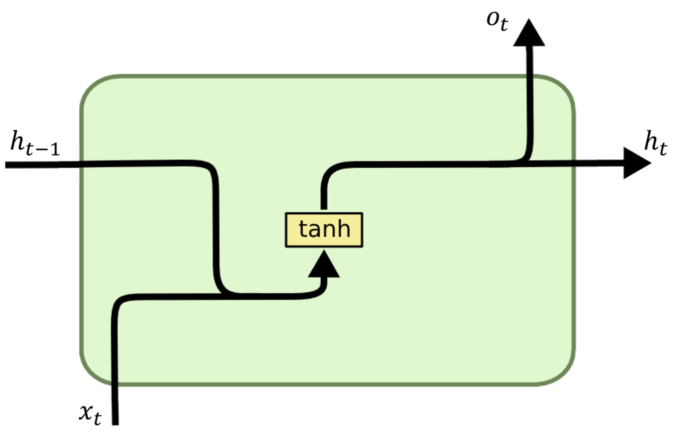

Fig. 2. The architecture of RNN Model

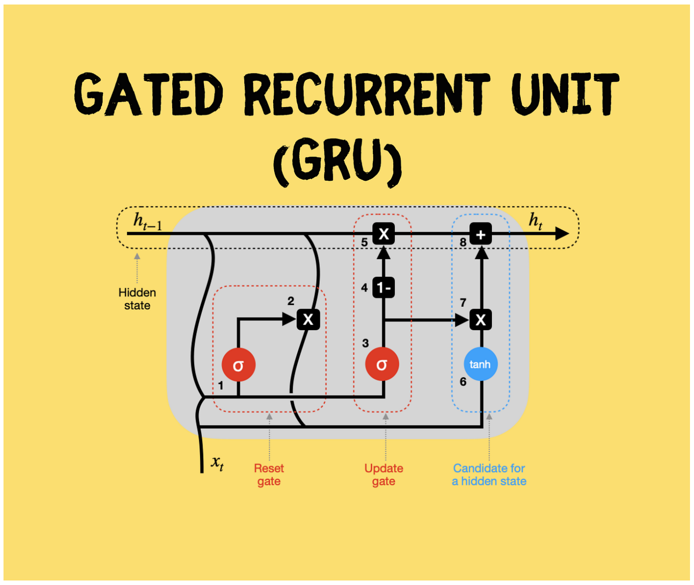

Fig. 3. The architecture of GRU Model

II. METHOD DESCRIPTION

The dataset offers a thorough understanding of Apple Inc.'s (AAPL) market activities throughout time and is composed of daily stock data. Important characteristics include:

- **Open**: The price of the stock at the start of the trading day.
- **Close**: The value of the stock at the conclusion of the trading day.
- **High**: The price attained at the peak of the trading day.
- **Low**: The price that was at its lowest point during the trading day.
- **Volume**: The total quantity of shares exchanged throughout the trading day.

The high price was selected as the study's objective variable. This is a helpful indicator for predicting peak prices since it shows the highest value the stock could reach on a given day.

A. *Data Preparation and Cleaning*

As a part of data preparation and cleaning we separated the Apple stock data from the other stocks in the dataset, rows with Name = "AAPL" were filtered. Secondly, checked for null values and made sure they are not present in the dataset. In order to enable time-series operations, the Date column was designated as the index and transformed into a datetime format for correct chronological ordering. Additionally, in order to concentrate only on numerical factors pertinent to stock price prediction, unnecessary fields like Name were eliminated from the dataframe to focus primarily on the predictions.

B. *Normalization*

We used Min-Max Scaling in our research to scale all numerical features to a range between 0 and 1 in order to prevent features with wider numeric ranges from controlling the model training process. By normalizing the input scale, this change hepled enhance model convergence in addition to stabilizing the training procedure. The formula for Min-Max Scaling is as follows:

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

Where:

- X: Original value.
- X_{min}: Minimum value of the feature.
- X_{max}: Maximum value of the feature.
- X_{scaled}: Scaled value.

By verifying that every feature falls within the same range, this transformation improves training stability and enabled us with a more effective model convergence.

C. *Train-Test Split*

In order to assess how well the models performed in terms of generalization, the dataset was split into training and testing sets. Data from 2017 onward was put aside for testing, while data from the dataset's beginning from 2006 up to 2016 was used for training. In order to simulate real-world situations when predictions are made for future time periods, this chronological divide made sure that the model is evaluated on the unknown data and does not succumb to overfitting.

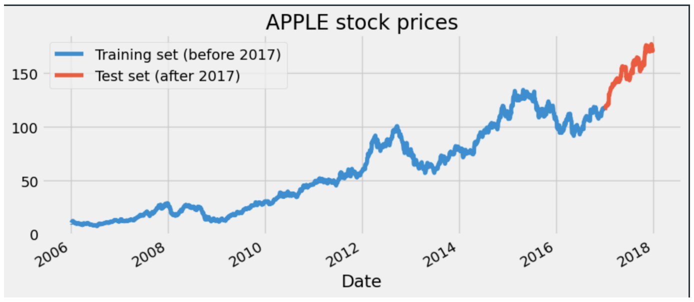

Fig. 4. Training and test data visualization

D. *Sequence Generation*

A sliding window technique was used to generate input-output pairs so that the neural networks could recognize patterns in sequential data. The dataset's temporal dependencies are captured by this method of generation of sequence.

1. *Sliding Window Technique*

- A fixed-length window of 60 days was used as the input features (X) to predict the *High* price of the next day (y).
- The sliding window shifts by one day after each iteration, creating a new sequence until the end of the dataset.
- Mathematically, the process can be represented as:

$$X_i = [H_{i-60}, H_{i-59}, \ldots, H_{i-1}]$$

$$y_i = H_i$$

Where:

  - H_i: The *High* price on day i.

  - X_i: Input sequence consisting of the *High* prices for the 60 days preceding day i.
  - y_i: The target variable, which is the *High* price on day i.

2. Input-Output Pair Generation

- For each sample, the model is trained to map the sequence of past 60 days (X) to the next day's price (y).
- For example:

$$X_1 = [H_1, H_2, \ldots, H_{60}], \quad y_1 = H_{61}$$

$$X_2 = [H_2, H_3, \ldots, H_{61}], \quad y_2 = H_{62}$$

- This process continues until the dataset is exhausted.

3. Reshaping for Compatibility

- The input sequences were reshaped into a three-dimensional format:

$$(\text{samples}, \text{timesteps}, \text{features})$$

  Where:
  - **samples**: The number of sliding windows created.
  - **timesteps**: The length of each sequence (60 days in this case).
  - **features**: The number of features (1 for *High* price).

4. Purpose

- Neural networks (RNN, LSTM, and GRU) are capable of analyzing sequential data because of to this transformation, which trains them the connections between previous prices and forecasted potential values.

III. METHOD IMPLEMENTATION

Training Configuration

The models' training setup was carefully planned to guarantee efficient learning and accurate forecasts. Important elements of the training procedure consist of:

1. Loss Function: The loss function that we used in the study is the Mean Squared Error (MSE). It penalizes greater mistakes more severely by calculating the average squared difference between the actual and anticipated values. The following formula gives the value of MSE:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Where:

- y_i: Actual value.
- $\hat{y}_i$: Predicted value.
- n: Number of data points.

In order to make sure the model learns to reduce prediction errors during training, MSE is frequently utilized in regression tasks.

2. Optimizers: Two optimizers were used based on the models that are being trained:

- **Adam Optimizer**: Used for Simple RNN and LSTM. During training, Adam, an adaptive learning rate optimization method, modifies the learning rate for every parameter. For quicker convergence, it combines the advantages of the Momentum and RMSProp approaches.
- **SGD Optimizer (Stochastic Gradient Descent)**: Used for GRU with specific hyperparameters:
  - Learning Rate: 0.01, which controls the step size in the parameter update.
  - Momentum: 0.9, it smoothes the learning process and speeds up convergence by taking into account the direction of previous gradients.

3. Batch Size: The batch size defines the number of samples processed before updating the model parameters.

- For Simple RNN and LSTM, a batch size of 32 was used to maintain computational efficiency while ensuring sufficient gradient updates per iteration.
- For GRU, a larger batch size of 150 was used in this case, using the ease of use and effectiveness of the model to analyze more data sets with every update.

4. Epochs: The number of training iterations (epochs) was set based on the complexity of the model:

- Simple RNN was trained for 25 epochs.
- LSTM was trained for 25 epochs.
- GRU was trained for 25 epochs.

These values were selected to get a comparision of training time and model performance, avoiding overfitting or underfitting.

Implementation Steps

1. Scaled the Training Data: The training data was normalized using Min-Max Scaling to ensure that all features had values within the range from 0 to 1. It's formula is given by:

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

Where:

- X: Original value.
- X_{min}: Minimum value of the feature.
- X_{max}: Maximum value of the feature.
- X_{scaled}: Scaled value.

This transformation ensured that features with larger numeric ranges did not dominate the training process, promoting stable convergence during optimization.

2. Generated Sequences: Using a sliding window approach, sequences of 60 consecutive days of data were generated as input features (X), with the *High* price of the next day as the target (y). The `create_dataset` function automated this process, ensuring consistent and efficient sequence generation. For example:

$$X_1 = [H_1, H_2, \ldots, H_{60}], \quad y_1 = H_{61}$$
$$X_2 = [H_2, H_3, \ldots, H_{61}], \quad y_2 = H_{62}$$

This approach allowed the models to capture temporal dependencies in the data, essential for accurate stock price predictions.

3. Reshaped Data: The generated sequences were reshaped into a three-dimensional format to be compatible with RNN-based models:

$$(\text{samples}, \text{timesteps}, \text{features})$$

Where:

- **samples**: Total number of sequences (sliding windows).
- **timesteps**: Number of days in each sequence (60).
- **features**: Number of features (1, representing the *High* price).

This reshaping ensured that the models could process the sequential nature of the data effectively.

4. Model Training: Each model was built, compiled, and trained using the preprocessed data. The training was conducted using the specified batch sizes and epochs for each architecture. The loss function (MSE) guided the training process, while the optimizers adjusted the model parameters to minimize errors iteratively.

5. Predicted Stock Prices: After training, the models were tested on unseen data (test set). The test data was normalized using the same Min-Max Scaler applied to the training data to maintain consistency. Predicted prices were then converted back to their original scale using the inverse transform of the scaler, allowing for direct comparison with actual prices.

6. Performance Evaluation: The models' performance was evaluated using the Root Mean Squared Error (RMSE), calculated as:

$$\text{RMSE} = \sqrt{\text{MSE}}$$

RMSE provides an intuitive measure of the average prediction error in the same units as the target variable. The predicted prices were plotted against the actual prices for the test set to visually assess the models' accuracy and highlight areas of deviation.

IV. EXPERIMENT AND ANALYSIS

The **Root Mean Squared Error (RMSE)** was used to evaluate the performance of the models. An straightforward indicator of average error in forecasting, RMSE shows higher accuracy with lower levels of error. The RMSE formula is:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Where:

- y_i: Actual value.
- $\hat{y}_i$: Predicted value.
- n: Number of data points.

Performance Results

The RMSE values for the three models are as follows:

Model	RMSE
Simple RNN	13.936
LSTM	3.320
GRU	6.238

Observations

1) **Simple RNN**: Struggled with long-term dependencies, resulting in higher prediction errors (RMSE: 13.936).
2) **LSTM**: Outperformed others (RMSE: 3.320) due to its ability to retain long-term information but required more computational time.
3) **GRU**: Performed very well (RMSE: 6.238), achieving a balance between accuracy and computational efficiency.

Visual Analysis

- Plots of predicted vs. actual prices showed **GRU** consistently producing predictions closer to actual values.

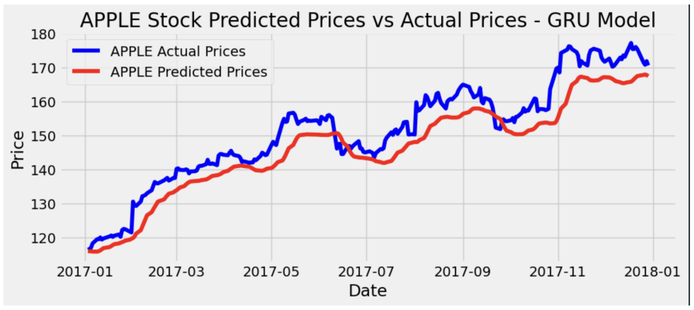

Fig. 5. APPLE Stock Predicted Prices vs Actual Prices - GRU Model

- **LSTM** provided comparable results but required longer training time.
- **Simple RNN** was less effective, particularly during volatile market periods.

V. Conclusion

Key Insights

1. Model Effectiveness:

- **GRU**: With its capacity to manage sequential dependencies with great accuracy and little computing cost, GRU emerged as a successful model.

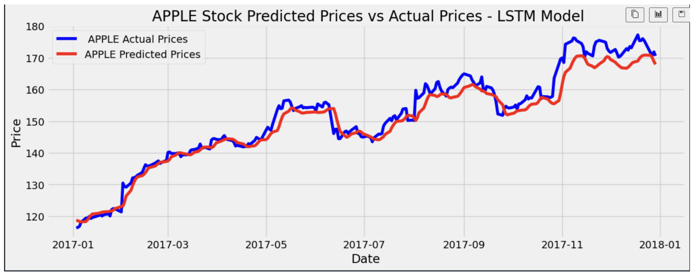

Fig. 6. APPLE Stock Predicted Prices vs Actual Prices - LSTM Model

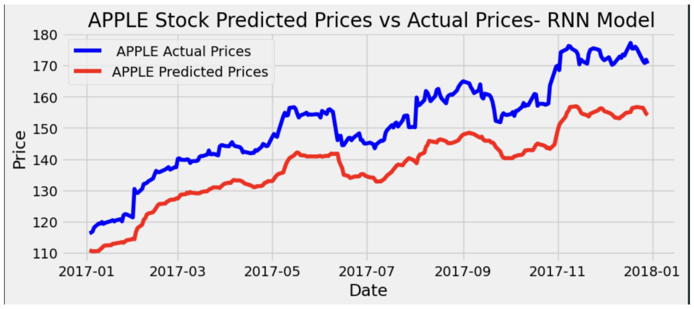

Fig. 7. APPLE Stock Predicted Prices vs Actual Prices- RNN Model

- **LSTM**: Its complex architecture made it computationally demanding even though it was accurate and most successful.
- **Simple RNN**: The least successful was the simple RNN, which had larger prediction errors because it had trouble modeling long-term relationships.

2. Preprocessing Importance:

- **Min-Max Scaling**: To prevent influence by features with wider numerical ranges, parameter values were normalized to a constant range, ensuring steady training.
- **Sequence Generation**: By efficiently capturing temporal relationships using the sliding window approach, the models were able to identify patterns in sequential data.
- The **stationarity analysis** highlighted the importance of transforming raw time-series data. The non-stationarity of the stock prices was confirmed by the Dickey-Fuller test, highlighting the necessity of preprocessing to enhance model performance.

Additional Analysis: Stationarity

- Stationarity's effect on time-series modeling was examined using simulated autoregressive

processes with different autocorrelation levels (ρ).
- The results reaffirmed how crucial it is to deal with non-stationary data so that models may more effectively identify underlying trends.

Reflection

The project provided valuable insights into the strengths and weaknesses of different RNN-based models for stock price prediction:

- **GRU**'s balance of accuracy and efficiency makes it a strong candidate for real-world applications.
- **LSTM**, while accurate, may be better suited for tasks where computational resources are less constrained.
- **Simple RNN**, though simple, is not effective for tasks requiring modeling of long-term dependencies.

All things considered, the research demonstrated how important preprocessing, model selection, and assessment are to creating reliable time-series forecasting solutions. It also underlined how crucial it is to examine data properties like stationarity in order to maximize model performance.

VI. Code

The Python code for implementing RNN for Stock Price Prediction is on GitHub: `https://github.com/VaibhavRawat39/RNNPRED`

References

1) Jeremyhi, J., 2023. *Understanding Recurrent Networks Part 1: Simple RNN & LSTM*. [online] Medium. Available at: https://medium.com/@imjeremyhi/understanding-recurrent-networks-part-1-simple-rnn-lstm-cc53e7475980 [Accessed 7 Dec. 2024].
2) Analytics Vidhya, 2021. *Introduction to Long Short-Term Memory (LSTM)*. [online] Available at: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/ [Accessed 7 Dec. 2024].
3) TensorFlow, 2024. *GRU Layer - TensorFlow Core API*. [online] Available at: https://www.tensorflow.org/api$_docs/python/tf/keras/layers/GRU[Accessed7Dec.2024]$.
4) Towards Data Science, 2023. *Understanding RNNs, LSTMs, and GRUs*. [online] Available at: https://towardsdatascience.com/understanding-rnns-lstms-and-grus-ed62eb584d90 [Accessed 7 Dec. 2024].
5) Data Science Stack Exchange, 2019. *MinMaxScaler when LSTM predictions fall outside of training range*. [online] Available at: https://datascience.stackexchange.com/questions/52727/ when-lstm-predictions-fall-outside-of-training-range [Accessed 7 Dec. 2024].