A Comparative Analysis of Long Short-Term Memory and Gated Recurrent Unit for Stock Price Prediction

Baibhav Datta

baibhav.datta@city.ac.uk

Abstract

Long Short-Term Memory networks (LSTM) and Gated Recurrent Unit networks (GRU) represent two prevalent variations of recurrent neural networks (RNN) renowned for their ability to handle sequential information and retain long-term memory. This investigation delves into the comparative performance of these two deep learning architectures, assessing factors such as running speed, accuracy, and Root Mean Square Error (RMSE). The study employs the Netflix Stock Price dataset, accessible on Kaggle, covering a span of five years.

1. Introduction

In the realm of financial markets, accurate prediction of stock prices remains a formidable challenge, crucial for informed decision-making and mitigating investment risks. In recent years, deep learning techniques, particularly Recurrent Neural Networks (RNNs), have emerged as powerful tools for tackling this task. Among the variants of RNNs, Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks stand out for their ability to capture intricate temporal dependencies inherent in stock price data. In this study, I embark on a comparative analysis of these two prominent architectures, evaluating their performance in stock price prediction and aim to discern the strengths and weaknesses of LSTM and GRU models in this critical financial forecasting domain.

1.1. Long Short-Term Memory

Long Short-Term Memory (LSTM) networks represent a specific architecture within the realm of recurrent neural networks (RNNs). Their design aims to tackle the vanishing gradient problem and effectively capture extended dependencies in sequential data. They incorporate memory cells and gating mechanisms to selectively retain and update information over time, allowing them to effectively model temporal relationships in various tasks, including time series prediction.

In Long Short-Term Memory (LSTM) networks, the process involves several key components. Firstly, LSTM units consist of a cell state that runs along the entire sequence, allowing information to flow unchanged. They also feature three gates: the input gate, which controls the flow of new information into the cell state; the forget gate, which regulates the retention or removal of information from the cell state; and the output gate, which determines the information to be output based on the current cell state. These gates are controlled by sigmoid activation functions, allowing them to selectively update the cell state and output information.

1.2. Gated Recurrent Unit

Gated Recurrent Unit (GRU) networks are another variant of RNNs, introduced as a simpler alternative to LSTMs. GRUs also feature gating mechanisms, but with a reduced number of gates compared to LSTMs, resulting in a more streamlined architecture. Despite their simplicity, GRUs have demonstrated competitive performance in tasks requiring the modelling of long-range dependencies, offering computational advantages in terms of training speed and parameter efficiency.

Gated Recurrent Unit (GRU) networks streamline the process by combining the forget and input gates into a single "update gate." Similar to LSTMs, GRUs also incorporate a cell state, but they utilize an "update gate" to regulate the flow of information into the cell state and the output gate to control the information to be output based on the current state.

Additionally, GRUs have a "reset gate" that determines how much of the past information to forget when computing the current state. By simplifying the architecture and reducing the number of parameters, GRUs offer computational advantages while maintaining competitive performance in capturing long-range dependencies in sequential data.

2. Dataset

The dataset utilized for analysis and experimentation comprises Netflix stock price data obtained from Kaggle, spanning five years from February 5, 2018, to February 5, 2022, and containing 1009 records. It encompasses seven features: Date, Open, High, Low, Close, Adj Close, and Volume. For this study, focus will be placed solely on the closing price (Close) feature. Detailed statistics summarising the closing prices are provided in Table 1.

	Close
count	1009.000000
mean	419.000733
std	108.289999
min	233.880005
25%	331.619995
50%	378.670013
75%	509.079987
max	691.690002

Table 1: Statistic Summary of Closing Price

2.1. Initial Data Analysis

During the preliminary data analysis phase, the data values for each feature were plotted over the timeline. It was observed that the distribution of values across the timeline was fairly uniform for all features, with a gradual increase over time. However, no consistent trend was evident in the graph, as depicted in
Figure 1.

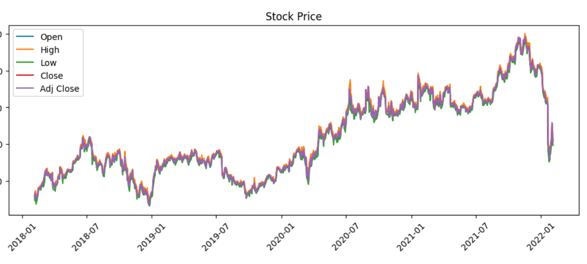


Figure 1: Features Values Distribution

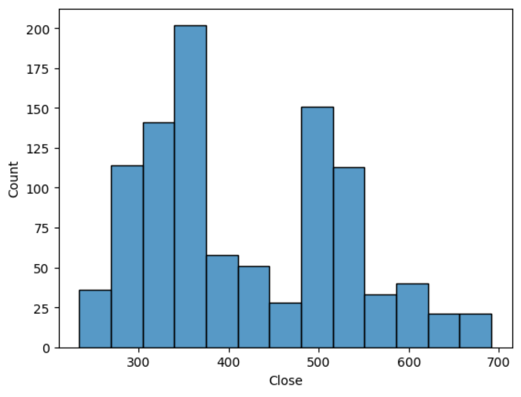


Figure 2: Closing Price Histogram

Since we'll train our model using a sliding window approach, we only require the closing price of the stocks. Figure 2 illustrates the distribution of this closing price feature.

3. Methods

This section outlines the methodology employed for data preprocessing, training, and testing, along with a description of the architectural design and hyper-parameters utilized in constructing both the LSTM and GRU models.

3.1. Methodology

The methodology involved preprocessing the dataset, beginning with the conversion of the 'Date' feature to the standard date-time format using the pandas to_datetime function. Subsequently, the relevant feature for our modelling, namely the closing price, is copied to a separate variable and the values are scaled to a range of -1 to 1 using Scikit-Learn's MinMaxScaler function. This scaling process is crucial to ensure stability in the learning process, as variables with a wide range of values can lead to unstable weight updates due to large error gradient values.

Following preprocessing, the dataset is partitioned, with 20% reserved as testing data for algorithm comparison between the best-performing LSTM and GRU models. The remaining 80% of the data is utilized for training, facilitating model selection and comparison between both algorithms. Lastly, all of the training and testing data are converted into PyTorch tensors using the torch.from_numpy().

The sliding window approach has been adopted to prepare the dataset creating training and testing sets for time series forecasting tasks. This methodology involves creating overlapping sequences of data points from the original dataset, where each sequence represents a historical window of observations. By sliding the window along the dataset, we generate multiple sequences for both training and testing. We set the lookback window, which refers to the number of previous data points or time steps considered when making predictions in a time series forecasting model. It represents the historical context used by the model to generate forecasts for future time points. In financial forecasting, for instance, a lookback period of 20 days means that the model considers the past 20 days' worth of data to predict the next day's stock price. Adjusting the lookback period can influence the model's ability to capture patterns and trends in the data, with longer lookback periods providing more historical context but potentially increasing computational complexity.

The training set comprises sequences of historical data along with their corresponding target values, enabling the model to learn temporal patterns. Meanwhile, the testing set consists of sequences of historical data used for evaluation, allowing us to assess the model's performance on unseen data. This approach ensures that the model learns from past observations while being evaluated on future data, facilitating robust performance assessment and generalization.

For model evaluation, a function has been defined to calculate the RMSE by first computing the squared differences between each corresponding pair of true and predicted values, then taking the mean of these squared differences, and finally taking the square root of the resulting mean.

3.2. Implementation

- In this project, we utilise LSTM and GRU models for time series forecasting. For both models, the architecture is configured with 2 hidden layers, each containing 32 neurons. To prevent overfitting, the 'Adam' activation function is chosen, which dynamically adjusts learning rates during training.
- Training occurs over a specified number of epochs using a loop structure. During each epoch iteration, the model generates predictions for the training input data. Subsequently, the Mean Squared Error (MSE) loss between the predicted and true outputs is calculated. The MSE serves as the optimization criterion, guiding the model's parameter updates to minimize prediction errors.
- To optimize the model parameters, the gradients of the loss function with respect to the parameters are computed through backpropagation. Before each gradient computation,

the optimizer's gradients are reset to zero to prevent accumulation. The optimizer then updates the model parameters using the computed gradients, facilitating the model's convergence to an optimal state.

- Upon completion of training, the model's performance is evaluated using various performance metrics tailored to time series forecasting tasks. These metrics provide insights into the model's accuracy, precision, and ability to generalize to unseen data.

3.3. Algorithm

Both the models' codes are implemented in a similar way, therefore the following code implementation applies to both the models.

Model Definition:

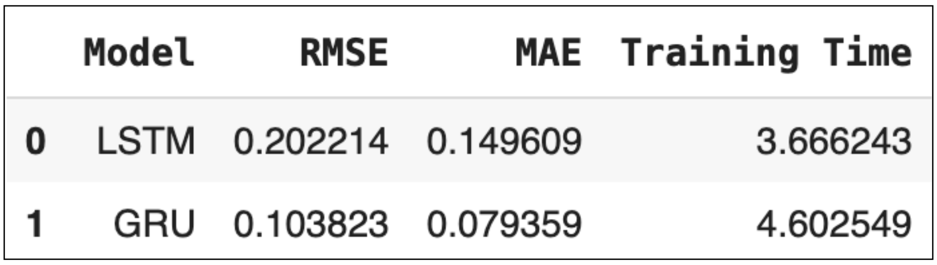
- The **LSTM/GRU** class inherits from **nn.Module** and defines the LSTM/GRU model architecture.
- In the constructor, it initializes the LSTM/GRU layers and linear layer. It takes input arguments such as input dimension, hidden dimension, number of layers, and output dimension.
- Inside the constructor, an LSTM/GRU layer and a linear layer are defined using the specified dimensions.
- The **forward** method implements the forward pass of the LSTM/GRU model. It takes an input tensor, initializes the hidden and cell states, passes the input through the LSTM/GRU layer, processes the LSTM/GRU output using the linear layer, and returns the final output.

Model Training:

- An instance of the **LSTM/GRU** class is created with the specified input, hidden, and output dimensions, as well as the number of LSTM/GRU layers.
- The Mean Squared Error loss function and the Adam optimizer are defined.
- Training is performed for 30 epochs using a loop.
- Within each epoch loop, the model predicts the output for the training input data.
- The MSE loss between the predicted output and the true outputis computed.
- The loss is printed and stored in an array for visualization.
- The optimizer's gradients are zeroed, backpropagation is performed to compute gradients, and the optimizer updates the model parameters using the gradients.
- Finally, the training time is calculated and printed after training completes.

4. Results, Findings & Evaluation

Upon completing the training of both the LSTM and GRU models, we proceeded to evaluate their performance by comparing the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) when predicting on the testing set. Evaluation metrics for both models are illustrated in Figure 3.



	Model	RMSE	MAE	Training Time
0	LSTM	0.202214	0.149609	3.666243
1	GRU	0.103823	0.079359	4.602549

Figure 3: Evaluation Metrics

As we can observe, the GRU model has lesser RMSE and MAE, which indicates its better performance in predicting on unseen data. But the training time for GRU is longer than LSTM.

Figure 4 & 5 shows the training loss curves for both the LSTM and GRU models respectively, it was observed that the loss steadily decreased for approximately 20 epochs. During this initial phase, the models effectively learned from the training data, resulting in a reduction of the loss function. However, beyond the 20th epoch, the decline in loss became less pronounced, and the curves exhibited a tendency to plateau. Despite further iterations till 100, there was minimal improvement in the loss values, indicating that the models had reached a saturation point in their learning process. So to avoid overfitting, the number of epochs was reduced to 30.

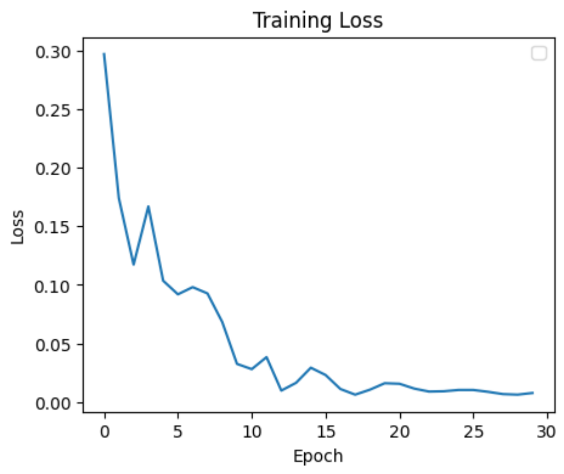


Figure 4: LSTM Loss Curve

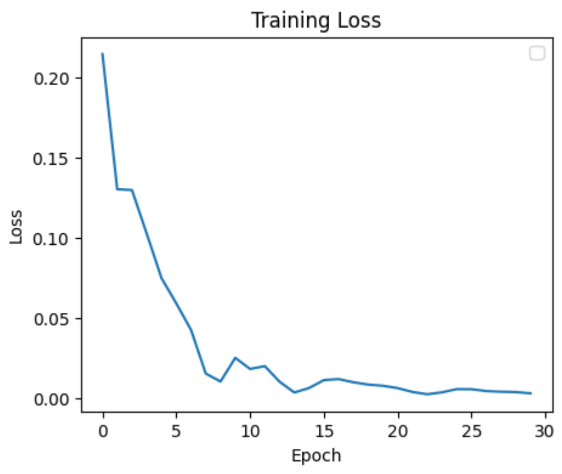


Figure 5: GRU Loss Curve

Figure 6 & 7 shows the graphical representation comparing the actual stock prices with the predicted stock prices reveals distinct characteristics between the LSTM and GRU models respectively.

It is evident that the predicted stock prices generated by the GRU model closely align with the actual stock prices, exhibiting a high degree of accuracy. Conversely, the predicted stock prices produced by the LSTM model display noticeable deviations from the actual prices, indicating comparatively lower accuracy. This disparity underscores the superior predictive performance of the GRU model over the LSTM model in capturing the underlying patterns and trends in the stock price data. The clear distinction in accuracy between the two models emphasizes the effectiveness of the GRU architecture in generating more precise forecasts for stock price movements.

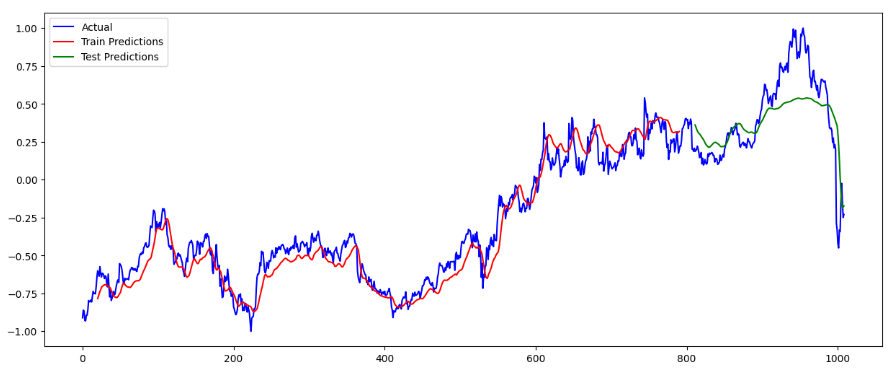


Figure 6: LSTM Predicted vs Actual Curve

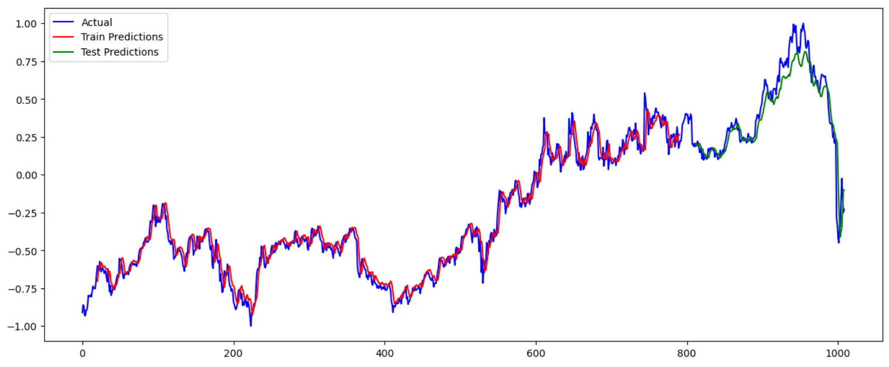


Figure 7: GRU Predicted vs Actual Curve

5. Conclusion

This research compared the performance of LSTM and GRU models in predicting stock prices based on the past 20 days' data. In conclusion, both models exhibited a similar pattern of saturation after approximately 20 epochs of training, where the loss ceased to decrease further. However, upon evaluation, the GRU model consistently outperformed the LSTM model in terms of accuracy metrics such as Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). But the training time was longer for GRU than LSTM. Most notably, the comparison of actual versus predicted stock price graphs revealed that the GRU model's predictions closely matched the actual data, demonstrating its superior predictive capabilities. These findings highlight the effectiveness of the GRU architecture in capturing the intricate patterns and dynamics of stock price movements. Moving forward, leveraging GRU models may offer enhanced accuracy and reliability in stock price forecasting applications, thus warranting further exploration and adoption in financial markets.

During the course of this project, I gained valuable insights into the significance of the number of epochs in model training. Initially, training the models for 100 epochs resulted in excessively accurate predictions, indicating overfitting. Recognising this, I subsequently reduced the number of epochs to 30. This adjustment proved beneficial, as it led to more satisfactory results.

6. References

[1] Yongqiong Zhu. 2020 Journal of Physics: Conference Series. Stock price prediction using the RNN model.

[2] Ruobing Zhang. Beijing University of Technology. LSTM-based Stock Prediction Modeling and Analysis

[3] B. N. Varaprasad, C. Kundan Kanth, G. Jeevan and Y. K. Chakravarti, "Stock Price Prediction using Machine Learning," *2022 International Conference on Electronics and Renewable Systems (ICEARS)*, Tuticorin, India, 2022

[4] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon and K. P. Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model," *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*

[5] Mohammadreza Ghadimpour and Seyed Babak Ebrahimi. Forecasting Financial Time Series Using Deep Learning Networks: Evidence from Long-Short Term Memory and Gated Recurrent Unit

Appendix 1- Glossary

Root Mean Squared Error (RMSE): A metric used to measure the average magnitude of the differences between predicted and actual values in forecasting tasks.

Mean Absolute Error (MAE): A metric used to measure the average magnitude of errors between predicted and actual values in forecasting tasks.

Sliding Window Approach: A technique used to create overlapping sequences of data points from the original dataset for time series forecasting tasks.

Backpropagation: The process of computing gradients of the loss function with respect to model parameters to update them efficiently.

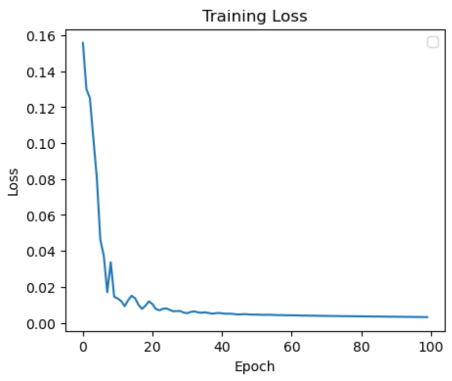
Optimizer: An algorithm used to adjust the learning rate and update model parameters during training to minimize the loss function.

Overfitting: A condition where the model learns to memorize training data excessively, leading to poor generalization on unseen data.

Temporal Dependencies: Patterns or relationships between data points that occur over time, crucial for accurate forecasting in time series data.

Appendix 2- Implementation Details

Following are some intermediate results- Initially I had trained the models with 100 epochs, which was leading to overfitting. As we can see in this loss curve below for lstm model, the loss value had reached the saturation point around 20 epochs, and the model training for the remaining 80 epochs was in vain. The case for gru model was similar. Therefore, I trained the models with various lower values of epochs like 10,15,20,25,30,.. and finally selected 30 epochs for training as it yielded the best results in lesser time.


Training Loss
Loss
Epoch