Assignment 3 - Recurrent Neural Networks for Stock Price Prediction

Sanchit Bakshi
The University of Adelaide
Adelaide, Australia
sanchit.bakshi@adelaide.edu.au

Abstract

This paper presents the application of RNN architectures- Vanilla RNNs, LSTMs, and GRUs-to stock price forecasting on historical data from the SP500 dataset [1]. The critical features of this dataset involve daily open, high, and low, close, and volume prices that are preprocessed by normalising, choosing features, and using the sliding window technique to perform the sequential modelling.

First, a baseline model for each architecture was trained and tested (included validation) to analyse their relative strengths and weaknesses. On training, the simple RNN converged fast but, owing to the problem of vanishing gradients, was unable to pick up long-term dependencies of the data. GRUs did an excellent job in terms of learning temporal dependencies. For reasons of better memory management via its update gates and reset gates [11],[3]. Heavy optimisations with hyperparameter tuning of hidden units, batch sizes, and learning rates significantly improved the results for all models.

1. Introduction

The stock market has grown to become part of today's society in terms of the global economy. Among various financial critical problems, one very critical problem is stock price forecasting. A proper forecast will probably provide insight that is valuable to making investment decisions [5]. Hence, the challenge is how advanced computational techniques can provide efficient results in the prediction of stock prices with respect to historical data. Most recently, with increasing access to high-frequency data, advanced developments in machine learning have given rise to very successful performance of RNNs-especially the LSTM and GRU models-for time-series predictions [3], [5], [11], [22].

The aim of this project is to build and estimate the stock prices of SP500 for Google company using the RNN architecture: vanilla RNNs, LSTMs, and GRUs. These models are going to predict the values for the future by looking at the trend of stock prices, helping stakeholders make their decisions wiser and comparing all three models. This will bridge the gap to evaluate the model's usability in real-world scenarios.

2. Background

In this section, we will explain the overview of competing techniques and other models which have been developed in the past.

2.1. Statistical Methods

ARIMA may be among one of the very first time-series forecasting methods used on a wide scale [2]. In an attempt to render times-series data stationary so that it will fit a predictive model, it combines autoregressive models, moving averages, and differencing. Its simplicity and interpretability have gained it a place in financial forecasting. However, nonlinear or nonstationary times series, as found in many stock prices whose trends could result from a myriad of unpredictable causes, are not well addressed by ARIMA.

While ARIMA did a decent job at the short-term level in a stationary data series, it is not designed to model nonlinear complex dependencies. Later models like RNNs and their advanced forms were introduced to tackle this issue.

2.2. Machine Learning Models

Some people use machine learning methods for prediction, such as Support Vector Regression (SVR). SVR changes Support Vector Machines for the purpose of regression to locate a hyperplane that minimises the prediction error within the tolerance margin. While SVR itself comes with some beneficial characteristics, such as high performance on small data and high-dimensional spaces, it naturally does not model sequential dependencies inherent in time-series data; rather, to obtain this, a great amount of feature engineering is done to represent time-lagged variables, which makes this modelling process more complex [19].

2.3. Deep Learning Models

The SVR modifies Support Vector Machines for the purpose of regression to find a hyperplane that minimises the prediction error within the tolerance margin. Whereas the SVR itself has many useful characteristics, such as high performance on small and high-dimensional data, naturally, it does not model a particular representation of sequential dependencies inherent to the time series data [3],[11]. Obtaining this is done after some enormous amount of feature engineering which represents a time-lagged variable against being modelled, making this more complicated. We have RNN-based methods including its advanced forms like LSTM and GRU those that fit the data directly, as they can handle raw sequential data well.

3. Dataset Overview

In this section, we will explore the dataset, preprocessing, and the splitting.

3.1. Dataset

The dataset used in this project was stock market data that is sourced from Kaggle, this comes under the SP500, **"GOOG"** dataset. It includes historical data with features such as Date, day's low, opening price, volume traded, day's high, closing price, and adjusted close. This dataset consists of 4612 daily records in total.

	Date	Low	Open	Volume	High	Close	Adjusted Close
0	19-08-2004	2.390042	2.490664	897427216	2.591785	2.499133	2.499133
1	20-08-2004	2.503118	2.515820	458857488	2.716817	2.697639	2.697639
2	23-08-2004	2.716070	2.758411	366857939	2.826406	2.724787	2.724787
3	24-08-2004	2.579581	2.770615	306396159	2.779581	2.611960	2.611960
4	25-08-2004	2.587302	2.614201	184645512	2.689918	2.640104	2.640104

Figure 1. Stock Market Data

3.2. Preprocessing

The steps involved in this data preprocessing are:

- **Normalisation:** All the stock prices were normalised by using min-max scaling, which scales the data from 0 to 1 [17]. It is essential for gradient-based optimisation techniques.

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

where,

X represents the original value of the feature

X_{min} represents the minimum value of the feature

X_{max} represents maximum value of the feature

- **Feature Selectionz:** The key features were (Open, high, low , volume close) and the "date" column which was preprocessed to suit the requirement of sequential modelling. The target variable in this model was "close".

- **Sliding Window:** The data from the past (N = 30) days is considered as the input for all the features, and (M = 1) day is the target price.

3.3. Train-Test Split

The GOOG dataset was divided into three parts: training, validation, and testing data, 70 %, 15 %, and 15 % respectively. After the split, the split data is obtained as 3207 for training, 687 for validation, and 688 for testing.

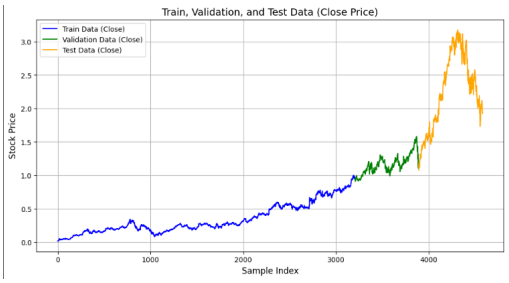

Figure 2. Train, Validation, and Test Data (Close Price).

4. Model Architecture

In this section, we will provide a detailed explanation of the algorithms used, including vanilla RNNs, LSTMs, and GRUs along with the hyperparameter optimisation.

4.1. Simple Recurrent Neural Network (Vanilla RNN)

This model is one of the simplest forms of a recurrent architecture. This is the backbone for probably most time-series prediction task adaptations, including stock price, since knowing the past tells considerably about the future.

It has the data in the form of a sequence, which updates the hidden state at every timestep of the time by taking as well as the hidden state from the previous timestep. The architecture mainly comprises of these components:

- **Input Layer**: It accepts the sequential data (N = 30), which is the number of days of stock price for those past days.

- **Recurrent Layer**: This layer uses one with a fixed number of neurons, for instance, 64 (baseline). The number of neurons was selected to make the model as complex while still keeping in mind the efficiency of computations. Mathematically, it can be represented

as:

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b_h)[12], [6]$$

where,

x_t is the input vector

h_{t-1} is the hidden state from the previous state

W_x, W_y is the weight matrices

b_h is the bias terms

σ is the activation function, typically a *tanh*.

- **Output Layer**: This layer maps the hidden state in order to predict the output, which in turn gets normalised further at the next timestep.

Mathematically, it can be represented as:

$$y_t = W_y h_t + b_y$$

where,

y_t is the predicted output
h_t is the hidden state
W_y are the weight matrices
b_y are bias terms

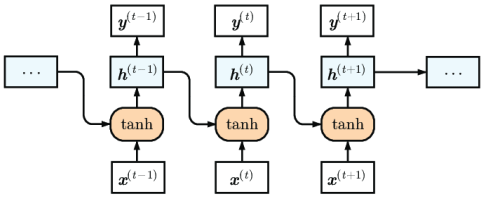

Figure 3. RNN Architecture. Source: ResearchGate, Understanding Hidden Memories of Recurrent Neural Networks.

4.1.1 Baseline RNN

This baseline model consists of one RNN layer—one with 10 units in this case-followed by a dense output layer. To build this model it uses adam optimiser, with 0.001 as its learning rate, 20 epochs for training with a batch size of 64. In first implementation, the model used is vanilla, with a structure and aims at catching immediate sequential dependencies that happen in the data. However, this model suffered from a vanishing gradient problem: the main problem with all neural network models dependent on previous long-distance frames is that the gradients decay while backpropagation goes forward in time. This seriously affected the ability of the model to learn multiple-day patterns, such as seasonal trends in stock prices, or a classic example, the covid era.

4.1.2 Optimisation

During optimisation, the number of hidden units was increased to 50 to capture more complex patterns while keeping in mind not to overfit and keeping a batch size of 32. Next, a critical tuning of the learning rate was done to reach an optimum between convergence speed and stability, being 0.001 (same as baseline). Again, dropout layers are followed to avoid overfitting by randomly disabling certain neurons when training. With those two changes, predictive capability saw huge enhancement, providing much longer-term dependencies the model is now able to manage.

4.2. Long Short-Term Memory (LSTM)

4.2.1 Baseline LSTM

LSTM was introduced by Sepp Hochreiter and Jürgen Schmidhuber in 1997. This model addresses the problem of vanishing gradient, which is done with the help of a gate mechanism. Every LSTM cell has its own cell state C(t), which saves all the values (buffer or memory). The updates are controlled by three gates which control the flow of the information: forgot, input, and output gates [3],[10].

- **Forget Gate:** This uses the sigmoid function to keep the relevant information, and the rest of the information is discarded. It gives values of 0 and 1 for discarded and retained respectively.

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t\right] + b_f\right)$$

- **Input Gate:** New values are generated with the help of tanh and update the information to add to the cell state.

$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, x_t\right] + b_i\right)$$

$$\tilde{C}_t = \tanh\left(W_C \cdot \left[h_{t-1}, x_t\right] + b_C\right)$$

- **Output Gate:** It obtains the values by deciding the output of the cell.

$$o_t = \sigma\left(W_o \cdot \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b_o\right)$$

- **Cell State Update:** It updates the old state with the new information (from the input gate).

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

- **Hidden State Update:** It produces the output h(t) on the basis of the updated cell state c(t).

$$h_t = o_t \cdot \tanh(C_t)$$

where,

W_f, W_i, W_C, W_o are the weight matrices for the forget, input, cell, and output gates, respectively and b_f, b_i, b_C, b_o are the corresponding bias vectors for these gates.

f_t represents the forget gate

i_t is the input gate activation at time t

$\tilde{C}_t$ is the candidate cell state.

o_t is the output gate activation at time t.

C_t represents the cell state at time t.

x_t is the input at time t, and h_{t-1} is the hidden state from the previous time step $t-1$.

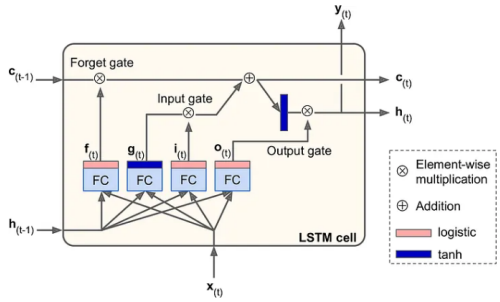

Figure 4. LSTM Architecture. Source: *O'Reilly Neural Networks and Deep Learning*.

4.2.2 Baseline LSTM

This model consists of the above mechanism, which enables this model to retain relevant information. This architecture of the baseline LSTM model consisted of a single LSTM layer with 10 units and a dense output layer. Training was done by using the Adam optimiser with a learning rate of 0.001 (constant for all the models) and keeping 64 as the batch size. Although the LSTM could learn long-term dependencies, it did not perform that well because of overfitting from the small size of the dataset, and we know that LSTMs are computationally complex, which increases training time.

4.2.3 Optimisation

In increasing the units for optimisation of the LSTM to 50, the target was to increase the representational capacity for catching complex patterns. The learning rate has been kept constant at 0.001, for the model to strike a good balance between the speed of convergence and stability. The batch size has been kept the same, 32 (similar to other optimised models), for the sake of effective gradient updates. Validation data was utilised while training the model to check the generalisation performance. To put more elaborate training, predictions on validation and test datasets were to be made after training through the 20 epochs. First of all, these predictions must be inverse-transformed onto the appropriate scale so that the preprocessed stock price can be compared with the actual closing price.

4.3. Gated Recurrent Unit (GRU)

Compared to the LSTM, the GRU model is a simplier architecture by fewer-parameter gating and maintains the ability of modeling long-term dependencies. In GRUs, the combination of the forget and input gates of LSTMs into one update gate, along with a reset gate that controls how much of the previous hidden state is going to contribute to the new candidate state [11],[12]. The important equations governing the GRU are given by the following:

- **Update Gate:** It determines how much of the information has to be retained from the previous state.

$$z_t = \sigma\left(W_z \cdot \left[h_{t-1}, x_t\right] + b_z\right)$$

- **Reset Gate:** It determines how much of the information has to be forgotten from the previous state.

$$r_t = \sigma\left(W_r \cdot \left[h_{t-1}, x_t\right] + b_r\right)$$

- **Hidden State Update:** It determines how much of the information has to be combined from the reset gate along with the hidden states from the previous state.

$$\tilde{h}_t = \tanh\left(W_h \cdot \left[r_t \cdot h_{t-1}, x_t\right] + b_h\right)$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t$$

where,

z_t: Update gate activation at time t.

r_t: Reset gate activation at time t.

$\tilde{h}_t$: Hidden state at time t.

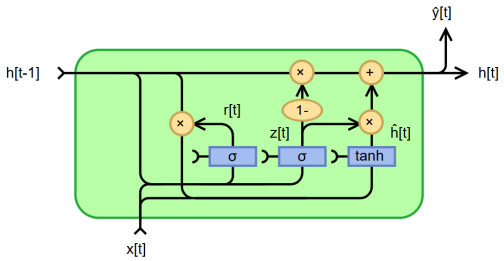

Figure 5. GRU Architecture. Source: *Wikipedia - Gated Recurrent Unit*.

4.3.1 Baseline GRU

This model consists of a layer with single GRU layer along with a 10-unit (similar to LSTM) and a dense output layer. Training was done by using the Adam optimiser with a learning rate of 0.001 (for all models). Having a batch size of 64, this baseline model was trained for 20 epochs.

4.3.2 Optimisation

For the optimisation, an increase in the number of units was done to 50 (keeping in mind about the problem of overfitting the model) for the GRU model. The learning rate was chosen as 0.001 (fixed), with the batch size being fixed to 32 for efficient updates. This model monitors generalisation with validation data and uses that model to generate predictions over both validation and test sets. These predictions were inverse transformed to their original scale for comparison with actual stock prices.

4.4. Device Configuration

During this research, the experiments were done using Jupyter Notebook, an interactive development web application that offers iterative experimentation, modular execution, and real-time visualization. All models were trained using GPU acceleration, as a result of the computation was faster which allowed us to take larger batch sizes.

5. Experimental Analysis

In this section, we will provide a detailed experimental analysis of the performance of all the models based on the different datasets along with their evaluation metrics such as Mean Squared error (MSE), MAE and other losses.

5.1. RNN

The baseline RNN model was trained for 20 epochs and a batch size of 64 with 10 hidden units and a dense layer, keeping the learning rate at 0.001. The model achieved 0.0012 as the validation loss and 0.0253 validation MAE, while a test loss of 0.00760 and test MAE was 0.0665.

On the other hand, the optimised model, which was the improved version of the baseline model was trained for 20 epochs and a batch size of 32 with 50 hidden units and a dense layer, keeping the learning rate at 0.001. The Adam optimiser was used [13] to achieve 4.3712e-04 as the validation loss and 0.0145 validation MAE, while a test loss of 0.003099 and test MAE was 0.03899.

The Vanilla RNN model, though capturing the short-run dependencies, suffered from a problem of vanishing gradient, which restricted it to learning long-run patterns in the baseline.

5.2. LSTM

The baseline LSTM model was trained for 20 epochs and a batch size of 64 with 10 hidden units and a dense layer, keeping the learning rate at 0.001. The model achieved 0.0060 as the validation loss and 0.0542 validation MAE, while a test loss of 0.586 and test MAE was 0.658.

On the other hand, the optimised model, which was the improved version of the baseline model, was trained for 20 epochs and a batch size of 32 with 50 hidden units and a

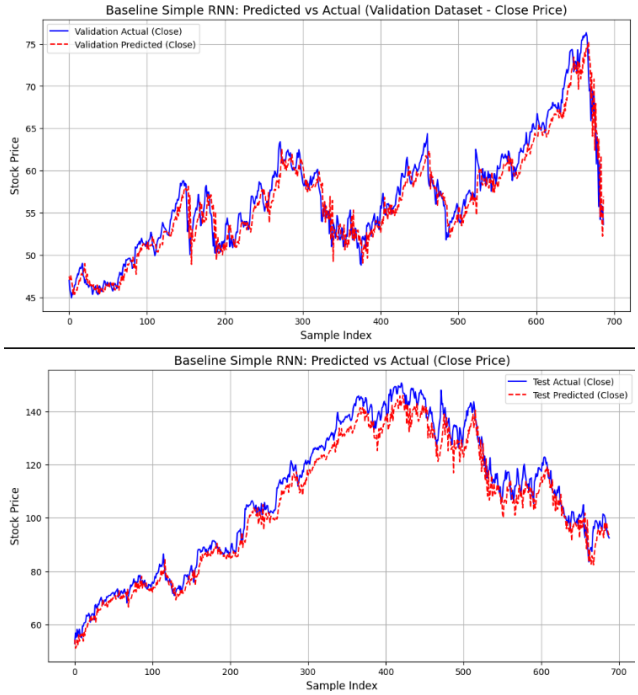

Figure 6. Baseline Simple RNN: Predicted vs Actual for Validation and Test Datasets (Close Price).

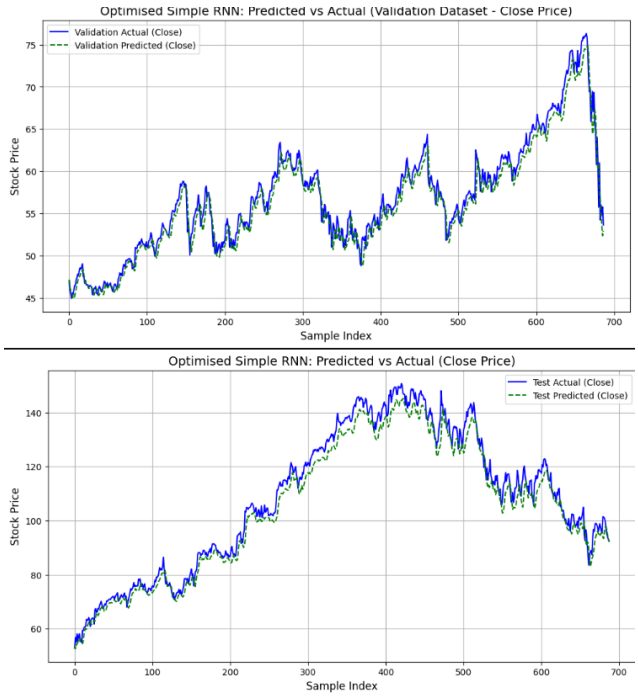

Figure 7. Optimised RNN: Predicted vs Actual for Validation and Test Datasets (Close Price).

dense layer, keeping the learning rate at 0.001. The Adam optimiser [13] was used to achieve 9.4295e-04 as the validation loss and 0.0145 validation MAE, while a test loss of

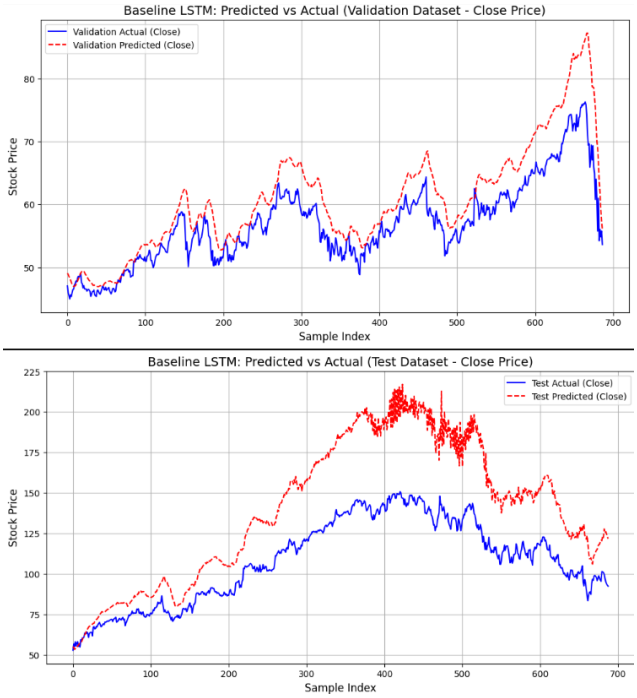

Figure 8. Baseline LSTM: Predicted vs Actual for Validation and Test Datasets (Close Price).

0.124292 and test MAE was 0.258601.

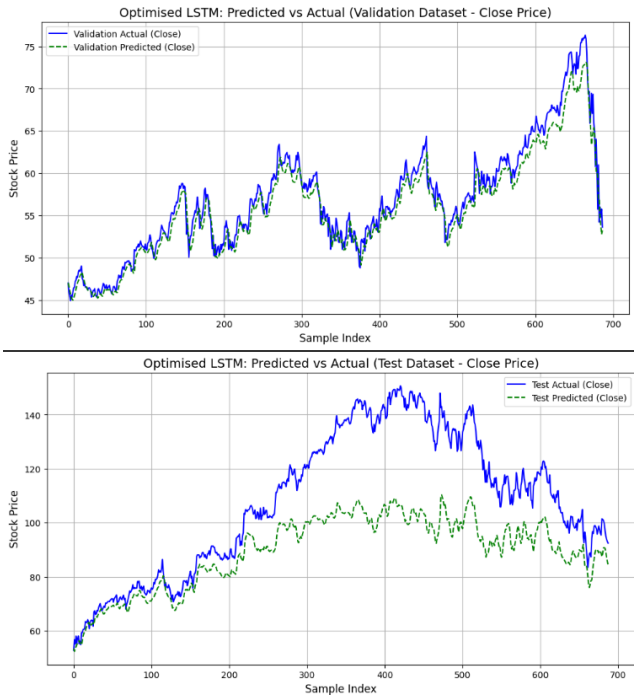

Figure 9. Optimised LSTM: Predicted vs Actual for Validation and Test Datasets (Close Price).

With a more advanced architecture, LSTMs failed to perform better than RNNs and GRUs due to some possible reasons:

- **Small Size of Dataset:** The dataset used for training is small compared to the other large datasets during the LSTM model.
- **Overfitting:** LSTMs have multiple gates and a lot of parameters, which might be prone to overfitting. This was enhanced due to the small dataset and not enough regularisation; that is why generalisation went poorly on the test set.
- **Hyperparameter Sensitivity:** Poor LSTMs are sensitive to the choices of their hyperparameters, such as the number of hidden units, learning rate, and dropout. Poor tuning of those hyperparameters can have huge impacts on the overall performance and has also been observed within this experiment.

5.3. GRU

The baseline GRU model was trained for 20 epochs and a batch size of 64 with 10 hidden units and a dense layer, keeping the learning rate at 0.001. The model achieved 0.0013 as the validation loss and 0.0252 validation MAE, while a test loss of 0.0273 and test MAE was 0.1105.

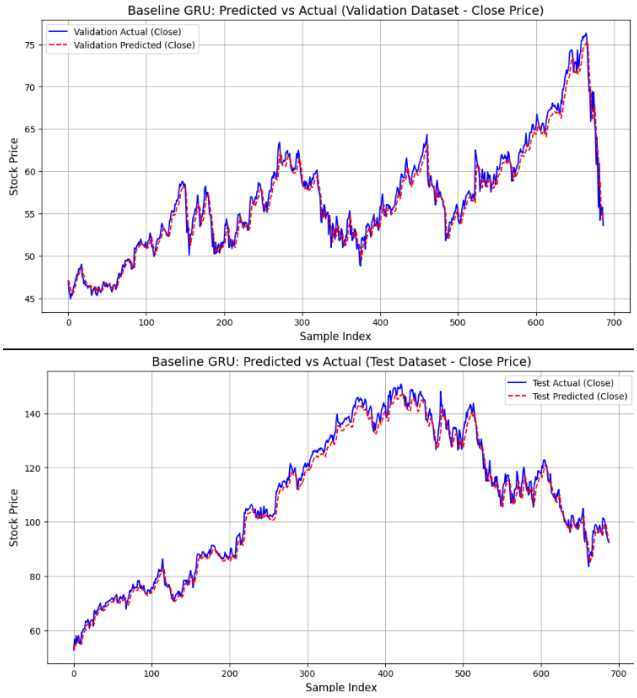

Figure 10. Baseline GRU: Predicted vs Actual for Validation and Test Datasets (Close Price).

On the other hand, the optimised model, which was the improved version of the baseline model was trained for 20

epochs and a batch size of 32 with 50 hidden units and a dense layer, keeping the learning rate at 0.001. The Adam optimiser [13] was used to achieve 4.9825e-04 as the validation loss and 0.0167 validation MAE, while a test loss of 0.022206 and test MAE was 0.11088.

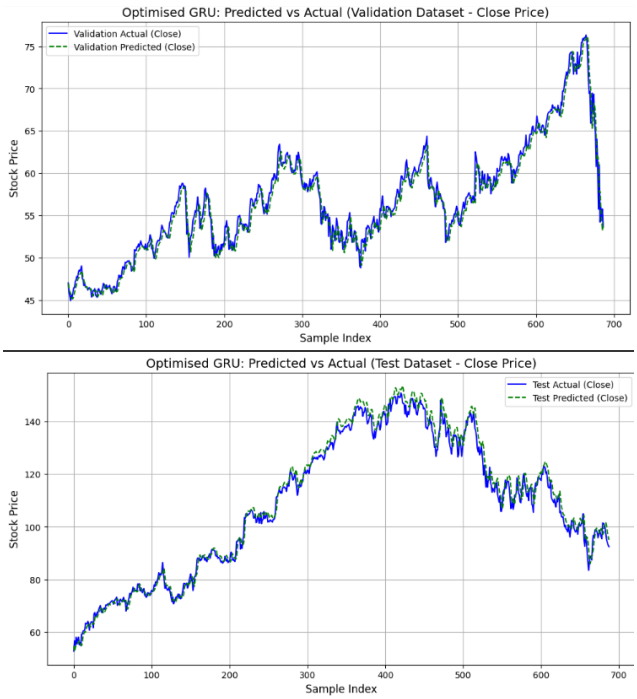

Figure 11. Optimised GRU: Predicted vs Actual for Validation and Test Datasets (Close Price).

6. Conclusion and Future Scope

This research looked at the application of an RNN architecture-Vanilla RNN, LSTM, and GRU-based methods—for stock price forecasting with the SP500 dataset. Comparative performance analysis provided the evidence that, with a more sophisticated gating system, the LSTM model clearly learned long-term dependencies between sequential elements, while such small datasets led to very bad overfitting; that is, it underfitted the training examples and did not generalise well at all. As can be seen, the higher validation and test losses appeared in the prediction graphs.

While both GRU and Vanilla RNN have better generalisation properties, their performance over test data was much improved. The GRU model yielded a lower test loss and MAE, clearly explainable through its strong modelling of sequential dependencies with fewer parameters. Though burdened with the problem of a vanishing gradient [20], the Vanilla RNN ran much better once optimised and hence can be thought of as a good baseline for simple time-series tasks [3].

The future scope of this study puts into perspective the potentials in GRU and Vanilla RNN models in financial forecasting and lays out challenges presented by LSTM. Further, this, therefore, might include regularisation techniques related to LSTMs when there is a need not to suffer much over-fitting, making the integral of exogenous features directly, such as market sentiments or macros, and further research into new transformer-based architecture for this sequential modelling. Moreover, an extension of the scope of either real-time predictions, multi-asset forecasting, and using more extensive datasets would indeed mean a lot for improving insights from these models and their robustness in various situations arising in finance.

Code Section

The link to my GitHub Repository for Deep Learning Assignment 3

References

[1] P. T. Mooney, "Stock Market Data," *Kaggle*. Available: https://www.kaggle.com/datasets/paultimothymooney/stock-market-data.

[2] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, Dec. 1974. doi: 10.1109/TAC.1974.1100705.

[3] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. doi: 10.1162/neco.1997.9.8.1735.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems (NIPS)*, Montréal, QC, 2014, pp. 3104–3112.

[5] A. Graves, "Generating Sequences with Recurrent Neural Networks," *arXiv preprint arXiv:1308.0850*, 2013. Available: https://arxiv.org/abs/1308.0850.

[6] F. Chollet, *Deep Learning with Python*, 2nd ed., Shelter Island, NY: Manning, 2021.

[7] "Recurrent Neural Networks (RNNs): Overview and Applications," *Wikipedia*. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network.

[8] A. Nama, "Understanding LSTM Architecture: Pros and Cons and Implementation," *Medium*. Available: https://medium.com/@anishnama20/understanding-lstm-architecture-pros-and-cons-ar

[9] S. Poudel, "Recurrent Neural Network (RNN) Architecture Explained," *Medium*. Available: https://medium.com/@poudelsushmita878/recurrent-neural-network-rnn-architecture-explained-1d69560541ef.

[10] O'Reilly, *Neural Networks and Deep Learning*. O'Reilly Media, 2017. Available: https://www.oreilly.com/library/view/neural-networks-and/9781492037354/ch04.html.

[11] "Gated recurrent unit," *Wikipedia*. Available: https://en.wikipedia.org/wiki/Gated_recurrent_unit.

[12] ResearchGate, "Understanding Hidden Memories of Recurrent Neural Networks," *ResearchGate*, Oct. 2017. Available: https://www.researchgate.net/publication/320726805_Understanding_Hidden_Memories_of_Recurrent_Neural_Networks.

[13] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014. Available: https://arxiv.org/abs/1412.6980.

[14] J. Brownlee, "A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size," *Machine Learning Mastery*, 2019. Available: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/.

[15] S. Goodfellow, I. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[16] G. Hinton, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, June 2014.

[17] "Stock Price Normalization Techniques," *Kaggle Discussions*. Available: https://www.kaggle.com/discussions/stock-prices-normalization.

[18] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015, pp. 448–456.

[19] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Oct. 2011.

[20] Y. Bengio, P. Simard, and P. Frasconi, "Learning Long-Term Dependencies with Gradient Descent is Difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[21] H. S. Oliveira and H. P. Oliveira, "Transformers for Energy Forecast," *Sensors*, vol. 23, no. 7, 2023.

[22] A. G. AbdElkader, H. ZainEldin, and M. M. Saafan, "Optimizing Wind Power Forecasting with RNN-LSTM Models through Grid Search Cross-validation," *Sustainable Computing: Informatics and Systems*, vol. 36, 2024.

[23] T. Gu, G. Xu, and J. Luo, "Sentiment Analysis via Deep Multichannel Neural Networks With Variational Information Bottleneck," *IEEE Access*, vol. 8, pp. 23940–23950, 2020. doi: 10.1109/ACCESS.2020.2969050.

[24] Y. Liu, J. Tian, Z. Song, F. Li, W. Zhou, and Q. Lin, "Spray Characteristics of Diesel, Biodiesel, Polyoxymethylene Dimethyl Ethers Blends and Prediction of Spray Tip Penetration Using Artificial Neural Network," *Physics of Fluids*, vol. 34, no. 6, pp. 067105, 2022. doi: 10.1063/5.0093675.

[25] T. Mariprasath, K. R. Cheepati, and M. Rivera, *Practical Guide to Machine Learning, NLP, and Generative AI: Libraries, Algorithms, and Applications*, River Publishers, 2024.