Google Stock Price Prediction Using Recurrent Neural Network(RNN), Long Short-Term Memory(LSTM) and Gated Recurrent Unit(GRU)

Anonymous CVPR submission

Paper ID *****

Abstract

Recurrent Neural Networks (RNNs) are powerful tools for time-series forecasting due to their ability to learn temporal dependencies. The performance of three RNN variants — Vanilla RNN, GRU, and LSTM—for stock price prediction is compared in this work. We conducted hyperparameter tuning throughout a range of settings, including the number of units, epochs, batch sizes, and optimizers (Adam, SGD, RMSprop), using the Google Stock Price dataset. Our findings show that the Vanilla RNN with RMSprop optimizer (150 units, 10 epochs, batch size 32) is the optimal model, with a Mean Absolute Error (MAE) of 806.49 and a Mean Squared Error (MSE) of 650,642.19.

1. Introduction

One of the hardest things to do in financial modeling is to anticipate stock prices [2]. Models that can recognize and take advantage of temporal dependencies in previous data are necessary for accurate stock price forecasting. In this field, recurrent neural networks (RNNs) and their derivatives, GRU and LSTM, have shown remarkable performance. This paper compares the predictive performance of Vanilla RNN, GRU, and LSTM models [2]trained on the Google Stock Price dataset. We investigate the effect of hyperparameter tuning, including optimizers, units, epochs, and batch sizes. The contributions of this work are as follows:

1. Comparison of RNN architectures: Vanilla RNN, GRU, and LSTM.
2. Systematic hyperparameter tuning: Evaluation of the optimizers and key hyper parameters.
3. Identification of the best-performing model: Based on Mean Squared Error(MSE) and MEan Absolute Eror(MAE).

2. Methodology

2.1. Dataset and Preprocessing

Historical prices with attributes like Open, High, Low, Close, and Volume are included in the Google Stock Price dataset. For simplicity, the Open column was selected as the [1] primary feature. The dataset was split into training and testing sets, and the following preprocessing steps were applied:

1. Normalization: Data was normalized using Min-Max scaling:

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

This indicates that all values were scaled between 0 and 1 to facilitate the training.
2. Sequence Generation: The stock price for the 31st day of each day was predicted using the stock price for the 30 days prior. The structure of the training data was Xtrain(input sequences) and y train (output labels).
3. Reshaping for the RNN Input: To make the input data compatible with RNN models, it was transformed into a 3D array of dimensions (samples, timesteps, features).

2.2. Model Architecture

I have implemented three RNN architectures:

1. Vanilla RNN: three SimpleRNN layers stacked together with dropout regularization.
2. GRU : three GRU layers in a layered architecture with dropout.
3. LSTM: Three LSTM layers with dropout are stacked in this architecture.

For the estimated stock price, each model employed a dense output layer of one unit.

2.3. Hyper-parameter Tuning

2.3.1 Grid Search Methodology

I have used a grid search over the following hyper parameters:

1. RNN: Vanilla RNN, GRU, LSTM. These were evaluated to determine the architecture most suitable for the stock price prediction.

2 Optimizers: Three widely used optimizers were tested.

a. Adam: for adaptive learning rates and robust convergence.
b. RMSprop: Time-series forecasting frequently uses this method since it works well in non - stationary environments.
c. SGD: A baseline optimizer with its fixed learning rates.

3 Hidden UNits: Each RNN layer's unit count was adjusted between 50, 100, and 150 in order to balance computational efficiency with model complexity.

4. Epochs: To evaluate the models' capacity to learn from data over varying time periods, they were trained for 10 and 20 epochs.

5 Batch Sizes: We experimented with both small (16) and large (32) batch sizes to assess the trade-off between speed and training stability.

2.3.2 Evaluation Metrics

Two measures are used to assess each set of hyperparameters:

1 Mean Squared Error(MSE): it calculates the average squared difference between the numbers that were expected and those that were found.Better prediction accuracy is shown by a lower MSE.

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

2. Mean Absolute Error (MAE): it provides specific data of model accuracy by capturing the average size of prediction mistakes.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

2.3.3 Impact of Hyperparameters

1. Rnn Type:

a. Despite its simplicity, Vanilla RNN showed competitive performance, obtaining the lowest MAE and MSE when tuned optimally.
b. GRU's effective handling of temporal dependencies allowed it to perform better for smaller datasets or fewer epochs.
c. Although it required more processing, LSTM performed well in situations that called for longer training times [**?**].

2. Optimizers:

a. RMSprop: For both Vanilla RNN and LSTM, RMSprop constantly provided faster and smoother convergence.
b. Adam: it used adaptive learning rates to maximize training and did well with GRU.
c. SGD: it found it difficult to converge efficiently, indicating its shortcomings for this use case.

3. Units:

a. The model's ability to understand intricate temporal patterns was enhanced by increasing the number of units.
b. 150 units were employed by the best-performing models, indicating that adequate complexity is essential for stock price prediction.

4. Epochs:

a. For simpler designs like Vanilla RNN, lower profits were seen after 10 epochs, even though more epochs generally resulted in better performance.
b. Given its capacity to learn long-term [**?**]dependencies, LSTM gained more with longer training (20 epochs).

5. Batch Sizes:

a. GRU's performance was improved by smaller batch sizes (16), which allowed for more frequent weight adjustments.
b. Training for LSTM and Vanilla RNN was stabilized by larger batch sizes (32) especially when RMSprop was used.

3. Results and Analysis

3.1. Performance Summary

The table below summarizes the results of the hyperparameter tuning:

Model Type	Optimizer	Units	Epochs	Batch Size	MSE	MAE
Vanilla RNN	RMSprop	150	10	32	650,642.19	806.49
GRU	Adam	50	10	16	141,400.00	9.28
LSTM	RMSprop	150	20	32	511.24	19.91

Table 1. Performance of RNN Variants with Different Hyperparameters

3.2. Best Model

With an MSE of 650,642.19, the Vanilla RNN with RMSprop optimizer demonstrated the best performance. Its prediction is contrasted with the actual stock values in Figure 1. Despite achieving reduced MSE in other situations, GRU with Adam was unable to generalize as well in this particular setup.

4. Discussion

Despite having a simpler architecture, the Vanilla RNN performed better than the GRU and LSTM. This result emphasizes how crucial hyperparameter adjustment is. The high number of units (150) allowed the model to capture intricate temporal correlations, while the RMSprop optimizer probably helped the Vanilla RNN converge more smoothly. The relatively high MSE, however, raises the possibility that hybrid architectures or other factors (such trade volume) could enhance performance. On bigger datasets or sequences, GRU and LSTM models—which are renowned for their resilience to vanishing gradients—may perform better than Vanilla RNNs.

5. Conclusion

A comparative study of RNN variations for stock price prediction was reported in this work. With an MSE of 650,642.19, the Vanilla RNN with RMSprop optimizer configuration performed the best. The findings highlight how important hyperparameter adjustment is for time-series forecasting. For improved accuracy, future studies will investigate multi-feature datasets and hybrid RNN architectures.

References

[1] N. Mohana Sundaram R. Santhosh E.T. Sivadasan. Stock market forecasting using deep learning with long short-term memory and gated recurrent unit. *springerNatureLink*, 28:3267–3282, 2024. 1

[2] Chandra Prakash Narendra Sharma, Siddhant Behera. A combined model for index price forecasting using lstm, rnn, and gru. *springerLink*, 890:499–514, 2024. 1

5.1. Link to the github

https://github.com/Khurrat123/DLAssignment3.git