Using Recurrent Neural Networks to Predict Stock Prices

Brian Drake
University of Adelaide
THE UNIVERSITY OF ADELAIDE 5005 AUSTRALIA
brian.drake@student.adelaide.edu.au

Abstract

T

1. Introduction

Stock price prediction is a fundamental challenge in the financial industry, attracting attention due to its potential to inform investment strategies and mitigate risks. Stock prices are influenced by various factors, such as market sentiment, economic indicators, and historical trends. Predicting these prices accurately, requires models capable of capturing complex temporal relationships and sequential patterns inherent in time-series data.

To address this challenge, this paper employs Recurrent Neural Networks (RNNs), inclusive of Simple RNN, Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU), to predict Google stock prices. Achieved via a dataset provided by Sah online with Kaggle [3]. The dataset contains 1258 stock market datapoints in a date range of 03Jan2012 to 30Dec2016, each categorised with features Open, High, Low, Close, and Volume. The dataset is preprocessed and split into training, validation, and testing subsets, with sliding windows of historical data used as input to forecast future prices.

RNNs are particularly suited for time-series prediction due to their ability to maintain information across time steps, addressing dependencies that traditional models struggle with. GRUs and LSTMs further enhance this capability by mitigating issues such as vanishing gradients, making them strong candidates for this task. This paper compares these architectures on their predictive performance and evaluates their ability to learn temporal patterns using key metrics like Mean Squared Error (MSE), Mean Absolute Error (MAE), and R^2.

Through rigorous experimental analysis, this study not only aims to predict stock prices but also explore the effectiveness of hyperparameter tuning, architecture design, and regularization techniques in optimising model performance. Shedding light on the practical applications and limitations of RNN-based models in the domain of financial forecasting.

2. Method

RNNs are a class of deep learning models that are designed to accept sequential data. Unlike other feedforward nueral networks, RNNs possess a unique feature, the ability to retain a memory of previous datapoints, or internal state, to process a sequences of inputs. Creating a system that ideally processes natural language, speech recognition and time series forecasting. Modern RNNs have been utilised for many more tasks, such as multimodal learning and real time decision making systems [2].

SimpleRNN is a basic RNN structure offered by Keras. It is a deep learning network that takes in sequential time series data where each time step is processed sequentially and outputs either the final state or the sequence of hidden states. This is a minimalistic system that calculates the hidden state recursively as new datapoints are introduced. This occurs at each time step where each input is computed alongside the previous hidden state. A weighted sum followed by an activation function, typically tanh, is also utilised. Mathematically this is represented as:

$$h_t = activation(W \cdot x_t + U \cdot h_t - 1 + b)$$

Where W is the weight matrix for the input, U is the weight matrix for the recurrent connection, and b is the bias term [1]. The structure of this can be seen in figure 1.

LSTM RNN provides a more complicated architecure. Rather than only having one previous hidden state that is recursively updated as each new input is calculated, it contains a memory cell and three gates to control information flow. The memory cell st

RNN blah blah -creating datasets and what the RNN needs out fo them -ReLu and tanh -Simple -LSTM -GRU -Use of Dense here -preprocess scaling and its place

2.1. Regularisation

-dropout -L2

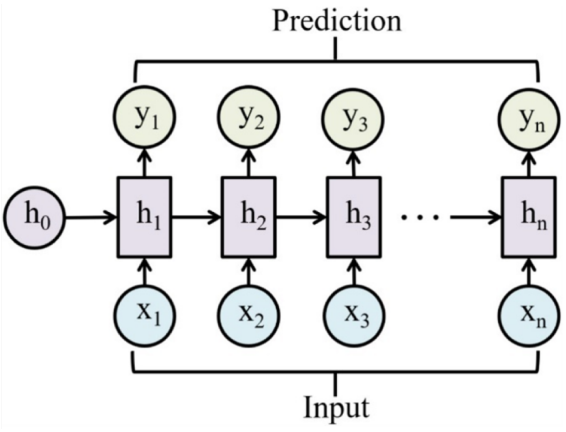

Figure 1. Generic RNN structure reflective of SimpleRNN [2]. Where h_n is the hidden state, x_n is the input tensor, and the y_n is the output tensor.

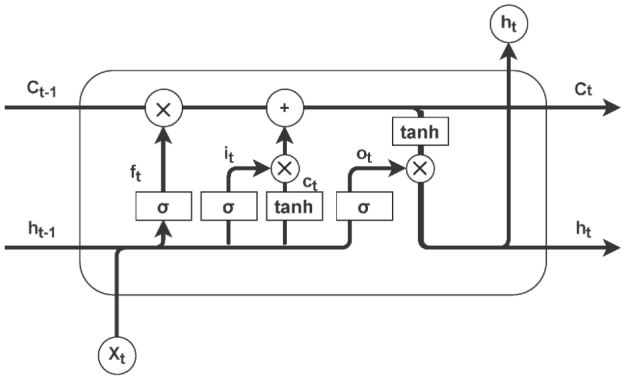

Figure 2. Simplified LSTM diagram [2].

2.2. Early Stopping

-validation loss -MSE

2.3. Performance Metrics

-MSE -RMSE -MAE -r2

2.4. Hyperband Tuner

-optimisers

3. Experimental Analysis

The experimental analysis was conducted using python, alongside the Kaggle Google stock price training dataset. The dataset was read from CSV (Comma Seperated Value file type) into a DataFrame. This data was then visually scanned for inconsistencies. It was noted that timepoints from 03Jan2012 until 26Mar2014 contained Close values that were double the next day's Open value. To remove as much data as possible while maintaining coherency, the latest 2 years of data with a 30 day sliding window were selected. Of this subset, the first 65 values contained the erroneous values. Providing more complicated relationships for the model to learn but remain for robustness and completeness. The intact dataset was then preprocessed. The dates were converted into a datetime format and the numerical values, Open, High, Low, Close, and Volume, were all sanitised from strings into floats. Each of these features were then scaled using sklearn.preprocessing MinMaxScaler. The default parameters for this function were used, causing the dimensions to be independently scaled between 0 and 1.

Four helper functions were then developed to complete the analysis. They were create_dataset, predict_future, plot_predictions and evaluate_model. The function create_dataset transforms the dataset from a 2D array into a 3D format that is compatible with RNNs. It assumes the data is an array or dataframe with dimensions as [samples, features] and sorted in chronological order; it then outputs two tuples, X_train and y_traion, in the format of (day, sliding window, features). The function predict_future predicts future values for a given number of days for the model passed, outputting a list of predicted value arrays for each day. The plot_predictions function plots predicted values against true values using a list of feature labels as labels for the plots, outputting a 2 x 3 plot of features. Finally, the evaluate_model funciton calculates the MSE, RMSE, MAE and R^2 values for a set of predictions against the true values, outputting a tuple in the same order.

The now scaled training set was then split, such that the final 3 days would become a prediction comparison set, while the rest was retained. The retained data was then processed by the helper function, create_dataset, to make X_train and y_train subsets, with 2 years of datapoints each with 30 days of context.

TensorFlow was used to develop a simple model that has architecture compatible and comparable between SimpleRNN, LSTM, and GRU. The specific structure can be seen in figure 3. Tanh activation function was used in all layers to maintain comparability and enable non-linear data relationships to be explored. The first two layers had return sequences enabled, as this option is only required in intermediate layers as they are only required for learning. All layers had 20% dorpout to introduce regularisation. The layer units were made to be equal to the quantity of historical datapoints so what the mapping of temporal patterns is in the smae dimensionality of learning. Scaling the learning and complexity as required. An early stopping function via Keras was implemented that monitors validation loss for improvements over 10 epochs; if none are observed, then the model rolls back to the optimal epoch and discontinues training. This was implemented to reduce overfitting. All models were compiled to optimise loss with a MSE calculation that is optimised by Adam. In addition, MAE and

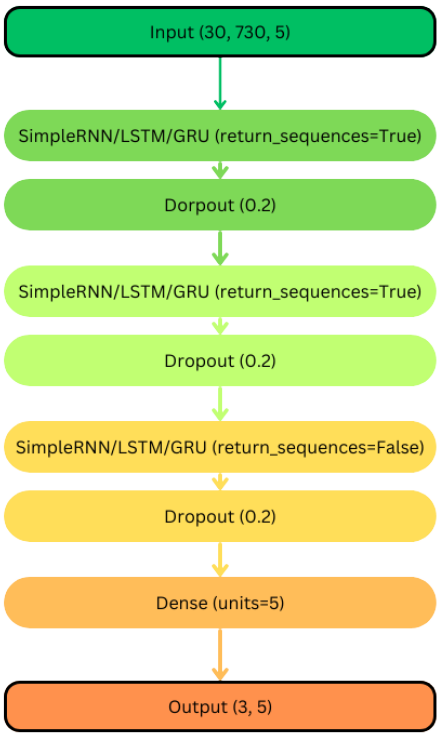

Figure 3. The standard architecture used for simplicity and comparability between SimpleRNN, LSTM, and GRU.

MAPE metrics were used to observe performance, due to the task being framed as a regression task. The model funciton .fit() was used to create an active 20% validation split. The results of the base model comparison are as seen in figure 1.

GRU was the best performing base model. To explore this further, an inverse ablation study was performed. Where complexity was slowly built upon the model and assessed to find an optimal architecture. The results of the 12 trials are as per figure 2. From these trials additional complexity of any variety was found to be detrimental to the model's performance.

To supplement optimisation of the best model, the original GRU model, Keras Hyperband was used. Variance was provided in the quantity of unit size (0.5, 1, 1.5 and 2 times the amount of temporal datatpoints), activation function (tanh and ReLu), drouput rate (0.2, 0.3, 0.4, 0.5), and optimisers (Adam, RMSProp, and SGD). Where each feature was tuned at every instance within the architecture. The Hyperband was allowed up to 25 epochs to choose between features, based on the original model stopping at 23 epochs. The resultant optimised model was then trained, displaying an overall worse performance than the original model, as seen in figure 1. The optimised model performed marginally better in one metric, the MAE.

The original GRU model, performing the best, was then given the original training data to make predictions. The outcome can be seen in figure 4 This task intends to characterise prediction patterns against true data. It can be seen that even though the model has learned to predict quite well with Open, High, Low, and Close, the model is having trouble capturing Volume. The peaks and troughs although align in location, do not reflect the intensity. The model is more so predicting a local average for Volume. The model is also largely able to match all features but is unable to emulate the sharp spikes that comes with market data.

4. Code

Please find the code here.

5. Conclusion

T

References

[1] Keras Community. *Keras Documentation*, 2015. Accessed: December 4, 2024. 1

[2] Ibomoiye Domor Mienye, Theo G. Swart, and George Obaido. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information*, 15(9), 2024. 1, 2

[3] Rahul Sah. Google stock price dataset, 2024. 1

RNN Model	Epoch Count	MSE Score	RMSE Score	MAE Score	R2 Score
Simple	17	1.43	1.20	1.06	-16000
LSTM	51	0.054	0.233	0.210	-620
GRU	23	0.0138	0.117	0.0983	-154
Hyperband GRU	31	0.0198	0.141	0.0891	-295

Table 1. Performance metrics for different RNN models.

Ablation Model	Epoch Count	R2 Score	Feature Changes
01	10	-4010	All layers `tanh` converted to `ReLU`
02	10	-2010	Bidirectionality added to the first layer, building on 01
03	10	-2690	`L2` regularisation at 0.005 added to all layers, building on 02
04	10	-1720	An additional dense layer was added after GRU layers, where dense=max(10, 0.5*past_days), building on 03
05	10	-2710	Unit quantity was multiplied by 1.5 in the first GRU layer, then 1.25 in the second GRU layer, building on 04
06	10	-1350	All layers changed back to `tanh`, building off 04
07	10	-1120	`L2` regularisation reduced to 0.001, building off 06
08	10	-1770	`L2` regularisation removed, building from 07
09	10	-1240	Third GRU layer removed, additional dense layer retained, building from 01
10	10	-872	`L2` regularisation at 0.001 added to all layers, building from 09
11	10	-1190	`L2` regularisation increased to 0.005, building from 10
12	10	-1290	Bidirectionality added to the first layer, building on 11

Table 2. Performance metrics and feature changes for inverse ablation study models.

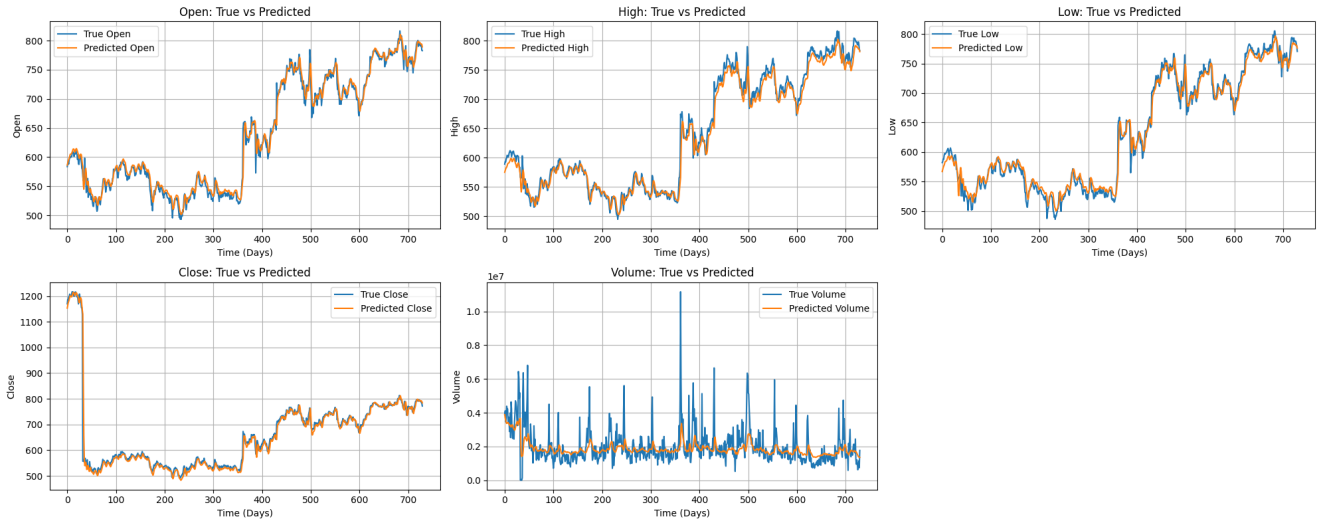

Figure 4. Predicted training values against true values, as made by the original GRU model.