Multi-Stock Neural Network for Price Prediction

Group F

January 8, 2024

Abstract

This research investigates the effectiveness of Long Short-Term Memory (LSTM) neural networks in forecasting intraday stock prices based on historical price data. It draws on past findings in model design, and utilises a grid search technique for optimising model hyperparameters. The proposed model predicts the direction of price movement with considerable accuracy (61%), yet its accuracy of price prediction is comparatively limited. This study serves as a proof of concept into the capability of LSTM models for stock price prediction. Furthermore, the findings imply that more efficient hyperparameter optimisation strategies may increase model accuracy.

1 Introduction

Stock price prediction is a time series modelling problem – the goal is to fit a model that describes the underlying pattern or structure of the time series by studying historical observations. As such, an appropriate model should be able to generate predictions of future values. Forecasting is important in any field that involves time-dependent data, and is particularly indispensable to financial markets. The existing methods for price forecasting in stock markets can be categorised into linear and nonlinear algorithms.

Linear models are the orthodox approach to forecasting problems and owe its popularity to its simplicity and flexible application to different time series. ARIMA model and its subclass models like AR and MA take the assumption that the time series is linear and has a known statistical distribution (e.g., normal distribution) [?]. The main limitation of this approach is its presupposition of linearity, which can become inadequate if the underlying time series is non-linear. Nonlinear models such as NMA [?] or ARCH and its associated algorithms [?] address this issue by, accordingly, either introducing non-linearity in mean , or non-linearity in variance, but make trade-offs in terms of ease of implementation.

A deep learning model for stock price prediction is an approach that could overcome these challenges. Artificial neural networks have the capability to fit models without a priori assumptions about the underlying statistical distribution. Since the approximation is adaptively formed based on the given data, it can deal with more flexible functional forms than traditional statistical models [?]. This makes neural networks uniquely suited for problems that are hard to specify but have abundant data [?, ?]. Most importantly, neural networks have the capability to generalise. After training with a dataset (i.e., historical prices), it should be able to make correct inferences about the unseen part of the data (i.e, future prices). Therefore, in theory, neural networks should be a suitable approach for forecasting stock prices.

Papers on the application of deep learning for stock forecasting propose a number of different neural network architectures. Recurrent neural networks (RNNs) are largely the preferred type of neural network for handling time series data [?] because the recurrent cells have feedback loops that address the temporal dependencies of the sequence [?]. An overview of recent RNN-based stock price forecasting attempts [?] identify a corpus of models that use LSTM (Long Short-Term memory) units or in combination with other kinds of RNN units. Following from this, this paper replicates an existing approach to LSTM modelling, comparing to available models. It contributes a novel grid-based search method for hyperparameter validation which draws on existing studies.

2 Problem Formulation

RNNs are a class of neural networks suited to processing time series data. They can keep information along time, storing it in hidden variable h which is updated based on new inputs. This allows the output to be computed with the hidden variable [?]:

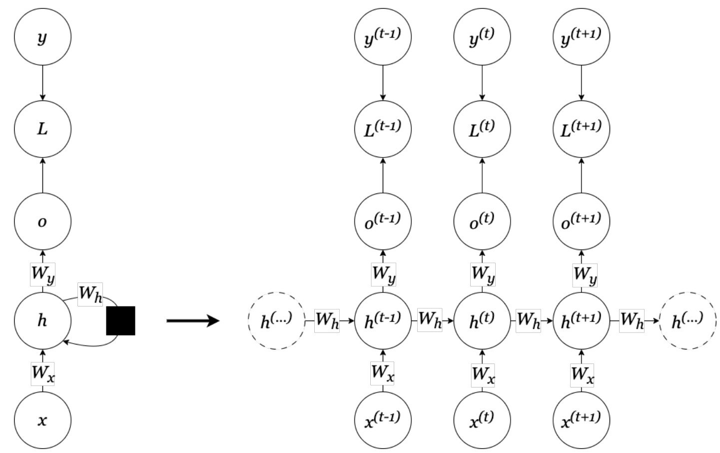

Figure 1: A simple recurrent neural network

For the RNN depicted in Figure 1, we outline here the corresponding equations from [?]11). Supposing we have the sigmoid function as our activation function (this is variable), our output o will take the value of the predicted price. Since our output is not discrete, it does not necessitate a post-processing step. Our $\hat{y}$ is equivalent to our output. Hence, for all time t:

$$a^{(t)} = b + W_h h^{(t-1)} + W_x x^{(t)} \tag{1}$$
$$h^{(t)} = \sigma(a^{(t)}) \tag{2}$$
$$o^{(t)} = c + W_y h^{(t)} \tag{3}$$
$$\hat{y}^{(t)} = o^{(t)} \tag{4}$$

where for the weight matrices, W_x is input-to-hidden, W_h is hidden-to-hidden, W_y is hidden-to-output connections; b and c are bias vectors given by the hyperparameters. We will explore its implications in section 3. Since we are concerned with the difference between $\hat{y}$ and y, the loss function should take a form such as mean squared error:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{5}$$

As price forecasting concerns the learning of long-term dependence series, the issue with 'vanilla' RNN is that gradients propagated over many stages often vanish (i.e., the gradient of the loss function approaches zero), making training the model difficult. Therefore, for time-series modelling, the most effective units are gated RNNs: LSTM [?] and GRU (Gated Recurrent Unit) [?]. These units address the vanishing gradient problems by having connection weights that are manually chosen constants or values of parameters, which allows the network to retain information over a duration. However, once the information is used, it might be practical to forget the old state if the information is no longer necessary. Essentially, we want the neural network to learn when it is appropriate to forget [?]. Hochreiter and Schmidhuber (1997) introduced the idea of create paths where the gradient may flow for longer durations by making the weight of the self-loop gated (i.e., determined by a separate hidden unit), allowing the time scale of integration dynamic instead of static as in vanilla RNN. LSTM structure is represented in Figure 2 below.

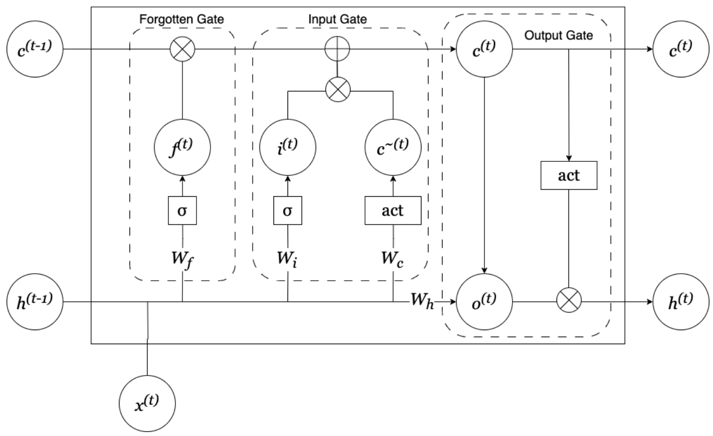

Figure 2: A Long Short-Term Memory recurrent network 'cell'

The work of each LSTM unit can be divided into three parts: forgotten gate, input gate, and output gate. The equations are as follows:

1. Forget gate: Historical information that is judged as useless or irrelevant will be discarded by the forgotten gate

$$f^{(t)} = \sigma(W_f \times (h^{(t-1)}, x^{(t)}) + b_f) \tag{6}$$

2. Input gate: The information retained by the forget gate is combined with the new input then transmitted to the output gate

$$i^{(t)} = \sigma(W_i \times (h^{(t-1)}, x^{(t)}) + b_i) \tag{7}$$
$$\widetilde{c}^{(t)} = act(W_c \times (h^{(t-1)}, x^{(t)}) + b_c) \tag{8}$$
$$c^{(t)} = i^{(t)} \times \widetilde{c}^{(t)} + f^{(t)} \times c^{(t-1)} \tag{9}$$

3. Output gate: The output value $h^{(t)}$ of the current neural unit is output from the output gate

$$h^{(t)} = \sigma(W_h \times (h^{(t-1)}, x^{(t)}) + b_0) \times act(c^{(t)}) \tag{10}$$

GRU differs from LSTM by having fewer parameters to train. There are also many more variants of gated RNNs, but LSTM is chosen for our proposition for two reasons: on the basis of its popularity in recent relevant works, and investigations into architectural variations show that LSTM is as strong as the best explored variants [?].

Apart from deciding the architecture of the neurons themselves, the construction of a neural network requires defining various hyperparameters (which is detailed in Numerical Experiments below). In reviewing the literature we find that in most papers, the selection of hyperparameters is based on scholars' experience. However, the suitability of these chosen parameters can not be guaranteed as a general rule. Therefore, our approach in the next section combines hyperparameter tuning with Grid-Search for RNN and LSTM respectively. This chapter will be divided into three parts: the Grid Search, comparison of the Grid-Search-RNN model and the Grid-Search-LSTM model based on single input, and the Grid-Search-LSTM model based on multiple input.

3 Proposed Solution

This section will be divided into three parts: the Grid Search, comparison of the Grid-Search-RNN model and the Grid-Search-LSTM model based on single input, and the Grid-Search-LSTM model based on multiple input.

3.1 Grid Search

In machine learning models, parameters that need to be manually selected are called hyperparameters. For example, the activation function, number of neurons, and optimizer in the RNN and LSTM models. All hyperparameters must be predetermined, and improper selection may lead to issues of either underfitting or overfitting. Hyperparameter selection involves two main approaches: fine-tune by experience and employing search algorithms to facilitate the identification of superior parameters. Evidently, the initial approach is time-consuming, and it is very complex to manually explore multiple hyperparameter combinations. Hence, search algorithms, including grid search, Bayesian search, and genetic search can be employed to facilitate the estimation of hyperparameters [?]. In this paper, we use the grid search method. Grid search can be seen as an exhaustive search method: it systematically traverses all potential parameter combinations, evaluating each possibility, and ultimately chooses the optimal-performing parameters as the hyperparameters selection [?]. However, given the time complexity as exponential, the use of grid search is computationally intensive: the time complexity is $O(M^N)$ where N is the number of hyperparameters and M is the number of values per hyperparameter.

The two key parts in this algorithm is the setting of the parameter candidates table and the score function. For our model, the negative mean squared error is used as the score function, because our goal is to make the predicted stock prices as close as possible as the real stock prices. As for the parameter candidates table, it is assembled by collecting parameters that are frequently used in past research.

3.2 Grid Search LSTM Model based on Multiple Input

The basic structure of the grid search RNN Model and the grid search LSTM model is shown in Figure 3. According to this figure, after normalising the input data (training set), our model's hyperparameters can be obtained by using a grid search. After training the model using a training data set, the stock price is predicted by inputting the most recent period of stock price data into the trained model.

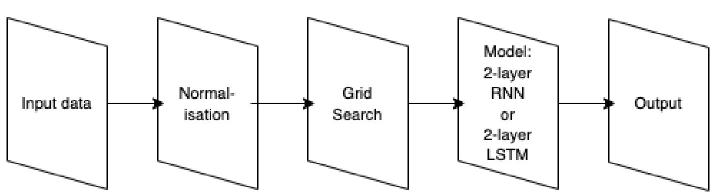

Figure 3: Hyperparameter tuning and training process

We first briefly use a single stock (CSCO) as input data to broadly compare the fit of an LSTM and RNN model for the dataset. The time interval and data processing for this single stock are identical to the processes described in Section 4. The results of the comparison are shown in Figure 4, which clearly displays that the LSTM model is a much stronger fit compared to the RNN model, which informs our selection of using an LSTM layer in our main numerical experiments.

Much of the research on stock prediction stays within single stock input models. However, in financial markets, the future prices of a stock are not only dependent on its own historical data, but also affected by wider market fluctuations. Therefore, a grid search model with multiple inputs is constructed. The model construction is basically the same as in Figure 3, except the input is multiple. This model aims to capture such market information and patterns of similar stocks in the industry, by inputting the historical prices of various stocks within the sector.

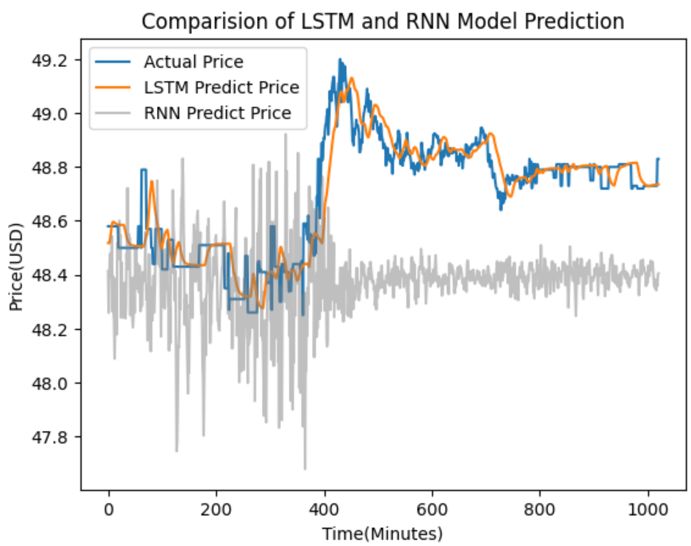

Figure 4: Comparison of LSTM and RNN Model Prediction

4 Numerical Experiments

4.1 Data Collection

4.1.1 Data length

The data sample is set at 5 months of trading, from January to May, 2023. A majority of previous intraday prediction papers utilise a data length less than one year [?]. A shorter time period has a high risk of both overfitting to training data and lacking sufficient training exposure for accurate prediction [?]; conversely, a longer time period increases risk of extraneous time-based variation, including change in market conditions that invalidate previously observed trading patterns [?]. The price data type chosen is minute closing price due it's practicability and simplicity.

4.1.2 Stock selection

The 20 highest volume stocks in the S&P 500 information technology (IT) sector were selected (see list here [?]). Historical market data was extracted from WRDS. High volume stocks were selected to minimise the number of missing values and reduce data noise (e.g. low-volume stochastic fluctuations in price). The IT sector is chosen given its inclusion of volatile and stable stocks, exposing the model to diverse trading patterns. Data completeness is therefore increased, aiding model accuracy [?].

4.2 Data Pre-processing

4.2.1 Missing data

Missing data is relatively rare in the market data sample, however during extended hours trading, some minutes have no recorded price. Forward filling is used to fill in these values, where missing values are imputed propagating the last valid observation forward to the next valid observation [?]. Stock prices were normalised between 0 and 1 to enable aggregating and comparing within the same model, a standard practice [?].

4.2.2 Sliding window

Time-series forecasting is defined by two metrics: Lag: the time length of the input data to be used by the model for a single price prediction Horizon: the time length into the future to be predicted by the model [?]

Chosen values for sliding prediction window:

$$\text{Lag} = 120 \text{ minutes} \tag{11}$$

$$\text{Horizon} = 10 \text{ minutes} \tag{12}$$

This lag is long enough for sufficiently complex intraday price patterns to be trained in the model [?]. These include support and resistance price levels, and day-trading price signals, patterns derived from technical analysis that have been shown to underpin deep learning forecast models [?]. The window remains sufficiently small to focus the model on the most recent price data that has highest relevance for the prediction [?].

The horizon provides a useful gap into the future that can be used for profit generation. The variability between time t and t+10 in terms of price is sufficiently large to create a challenging predictive task that necessitates a non-linear deep-learning approach.

4.2.3 Grid search construction

Grid search is run on a subset of the data length to reduce its computation requirement: one week of price data instead of 5 months. Using a truncated version of time series data for high-frequency price prediction has precedent in Zhang et al. (2020) [?], which demonstrates that optimal hyperparameter variation is relatively insensitive to changes in the time series length. In addition, domain knowledge suggests a week is sufficient to capture latent trends in price fluctuation, particularly in intra-day trading [?].

The following hyperparameters are deemed to be the most crucial to include in the grid search, due to their significance:

Hyperparameter	Candidates	Reason
Number of Hidden Layers	1, 2	An additional hidden Dense layer increases model complexity and possibly predictive power, yet may also lead to overfitting. Given the data size, more than two layers would likely overfit.
Hidden Layer Neurons	40, 60, 80	Candidate values produce a balanced and relatively wide range.
Activation function	Sigmoid, ReLU, Tanh	Selection represents widely used and suitable activation functions [**?**].
Optimizer	SGD, RMSprop, Adam	Selection represents the most prevalent optimizers in related literature [**?**].

Table 1: *Grid Search Hyperparameter Candidates*

The following hyperparameters have been manually set, and are not included in the grid:

Hyperparameter	Defined Value	Reason
Dropout	Single dropout layer with 0.2 probability	Random inactivation of a neuron, which can prevent overfitting; less significant on overall model fit; precedent in [**?**].
Batch Size	32	Lower end of common batch values, influenced by computational complexity.
Epochs	10	Relative flattening out after 10 epochs in initial model observations.

Table 2: *Predefined Hyperparameter Values*

4.3 Determining Hyperparameters Fit for Purpose

The best-performing set of hyperparameters uses 2 hidden layers with 60 neurons in each hidden layer, as well as using a Tanh activation function, and an Adam learning rate optimiser, which dynamically adjusts the learning rate. The additional manually defined hyperparameters are as previously discussed 10 epochs, a batch size of 32, and a dropout rate of 0.2. The activation function between the final hidden layer and the output layer is set as the default linear, since the stock price calculation must produce a valid price figure. This model on the grid search training data produces an MSE of 0.0131. Find the full results of the grid search in the Appendix. The resulting model specification is visualised with the diagram in Figure 5.

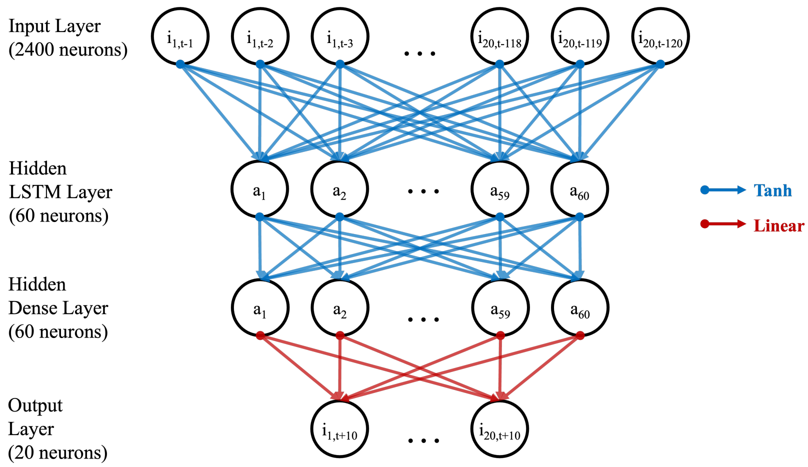

Figure 5: Model Structure and Hyperparameters Illustration

In figure 5, i represents the index of the stock out of the 20 possible stocks, t is the time point (minute) at which the model is run, and a represents a hidden layer neuron.

4.4 Results of Fully Trained Model

Having obtained the optimal model hyperparameters, the model is applied to the full 5 month time frame, resulting in a model with 2432 trained parameters. The experimental results also further support the epoch size selection, as we can observe a clear flattening out of the error throughout the epochs (Figure 6), indicating that additional epochs would not improve error scores significantly.

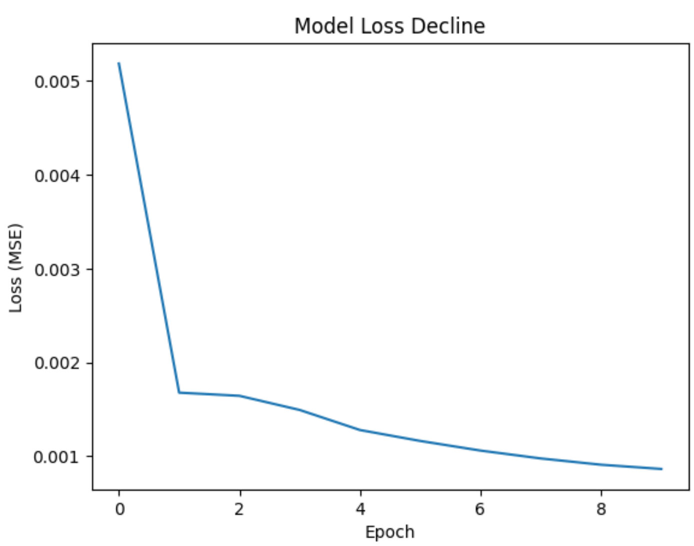

Figure 6: Graph of Model Loss Decline per Epoch

Following this, the prediction is carried out on the test data section. The data that is obtained is then inverse-transformed from the normalisation back into the original price scale for interpretability. The overall MSE across all 20 stocks is 72.8220. A single stock, CSCO is selected to visualise and illustrate the model performance (Figure 7):

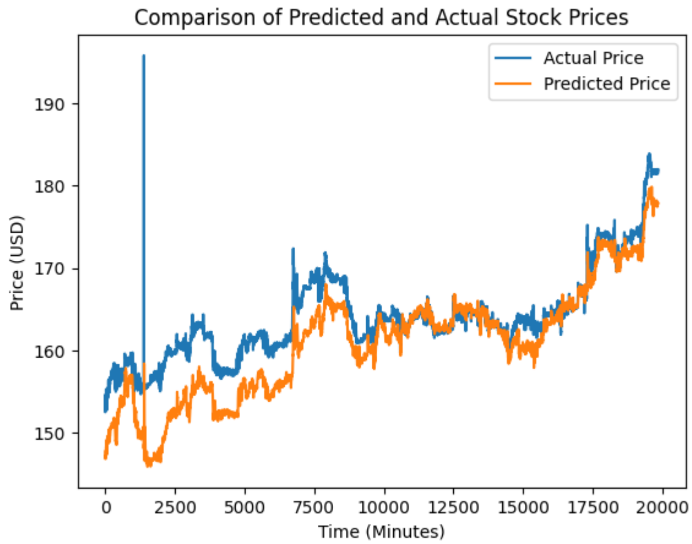

Figure 7: Graph Comparison of Predicted and Actual Stock Prices

5 Conclusion

The predicted prices and the actual prices display a partial alignment between predicted and actual prices. While the overlap is not particularly close, there is a noticeable similarity between the general trend shape of the two graphs. This mirroring suggests that while general market direction and trends are captured, accurate estimations are however not present.

It is notable that the runtime of the prediction step, once the model has already been trained and data obtained, is only about 13 seconds. This suggests, particularly given the 10 minute prediction horizon, that practical applications using such a model as a trading strategy is at least theoretically feasible, if predictive power is assessed to be reliable for such purposes.

5.1 Validation

To verify model profitability, we can assess whether the price information about time t+10 provided to a trader at time t can be utilised to increase/decrease actual trading returns. A bi-directional pseudo-accuracy test is constructed to measure whether the model accurately predicts a price for time t+10 that matches the direction of the actual price movement. Indeed, price direction is correctly predicted in 61% of the minutes within the test data. This is not a particularly impressive or reliable rate of prediction, yet it nonetheless outperforms pure chance (50/50), increasing model usefulness. Worth noting, this bi-directional test is not interpreted as a viable accuracy score as it would for a classification model, since the model was trained as a regression model and is not optimised for that task. However, it does strengthen the argument that the model is not particularly strong as a practical model for a real world intraday trading. This corroborates finance academic theories which posit that high-frequency stock price data follows a random walk [**?**].

5.2 Limitation

Several limitations exist in the proposed method. Firstly, the model is not generalizable to the broader stock market given a limited sample of 20 high-volume IT stocks, and sample size was limited by computation power available. Conclusions about the IT sector overall are therefore not possible, limiting the scope of findings. Secondly, the grid search method of hyperparameter optimization is a less exhaustive method compared to alternatives bayesian optimization and genetic algorithms, which extensively explore the combinations of hyperparameters using an efficient search process that iteratively explores promising hyperparameters without requiring pre-defined parameter combinations [**?**]. Grid search in contrast, is inefficient, given exponential time complexity, and its exploration strategy is un-informed, simply trying all combinations of pre-defined parameters. A more efficient hyperparameter optimization strategy may have resulted in a more accurate final model.

5.3 Further research

The proposed method can be extended in multiple avenues. Firstly, access to large computing power would enable a more powerful model creation and hyperparameter optimization. Secondly, increasing the dimensionality of the input data, adding technical variables including RSI, moving average and volume, would have likely increased prediction accuracy [**?**], but would've necessitated a new set of hyperparameters. Thirdly, a more robust profitability model test could've been developed that tracks the total investment return/loss based on periodic selling/buying using the model forecasts to act as a better sanity check for the model usefulness. Next, convolutional neural networks could have been tested on image data of candle-stick price charts as an alternative approach utilising price data. Finally, the sample used in the model creation could be expanded to increase generalizability of findings, and serve as a minimal test of the persisting perception of stocks as following a random walk.