Predictive Analysis of S&P 500 Using Recurrent Neural Network Models

Shavin Kaluthantri
The University of Adelaide
a1904121@adelaide.edu.au

November 23, 2023

Abstract

This paper explores the application of Recurrent Neural Network (RNN) models for predicting the Standard & Poor's 500 (S&P 500) index prices. We detail the process of data preparation, feature selection, and the deployment of various RNN architectures including LSTMs and Gated Recurrent Units (GRUs). The LSTM model augmented with a dropout layer outperformed others in the study, showcasing its ability to capture complex temporal dependencies while avoiding overfitting through regularization. Our findings demonstrate the potential of LSTM models to provide accurate and robust predictions in the volatile domain of stock price forecasting.

1 Introduction

The stock market, pivotal for the global economy, presents both opportunities and challenges for investors and analysts. The S&P 500 index, a leading U.S. equities indicator, is a critical benchmark for assessing financial performance. Predicting stock prices, especially for the S&P 500, is economically significant but highly complex and uncertain due to the market's inherent volatility arising from political events, economic indicators and company performance metrics. Stock prices are sensitive to both past trends and unpredictable future events, making accurate prediction a challenging endeavor. [9]

Accurate predictions can lead to profitable investment strategies, informed decision-making for portfolio management, and a deeper understanding of market dynamics. In this context, the use of Recurrent Neural Networks (RNNs), known for their efficacy in handling sequential data, presents a promising approach to model and predict the complex temporal patterns in stock prices.

A pivotal study has compared various RNN architectures, like Long Short-Term Memory (LSTM) networks and Deep Neural Networks (DNNs), for their effectiveness in stock prediction of the Indian BSE Sensex index, focusing on daily and weekly time scales, demonstrating LSTM's effectiveness in understanding long-term market dynamics, inspiring examination of various RNN architectures for different temporal scales in stock data analysis. [9]

A subsequent study adopted a model-independent approach, analyzing stock data dynamics through various deep learning architectures, including RNNs, LSTMs, and Convolutional Neural Networks (CNNs), instead of confining to a single model. This approach emphasized the importance of identifying underlying patterns in complex and often non-linear financial time series data. [8] This demonstrated the benefits of employing multiple architectures for stock price prediction.

Another study ventured into the realm of stock price correlation by combining the LSTM model with an ARIMA model, the research adeptly captured both linear and nonlinear trends in the stock mar-

ket data.[2] This fusion provides a novel perspective to improve stock market forecast accuracy by leveraging the strengths of different predictive models.

A novel dual-stage attention-based recurrent neural network (DA-RNN), excelled in handling large datasets with multiple influencing factors and predicting stock market indices due to its ability to capture long-term dependencies and select relevant driving series, offering an effective and interpretable framework [7].[7]

This research builds upon previous studies to evaluate various RNN models, ascertaining the most efficient approach for predicting the S&P 500 index. It uses past methodologies and findings to guide model selection and evaluation, addressing the complexities of stock market data.

2 Methodology

2.1 Data Preparation

Historical S&P 500 index data from the past 20 years was gathered using the 'yfinance' library from Yahoo Finance. The data set includes 'Open', 'High', 'Low', 'Close', 'Adj Close', and 'Volume' columns. 'Open' represents the starting trading price of a given day. 'High' and 'Low' are the maximum and minimum prices (respectively) during the trading day. 'Close' is the final trading price at market closure, whilst 'Adj Close' adjusts for dividends or splits prior to the next day's opening. 'Volume' is the day's traded shares. The 'Close' price is crucial for stock performance prediction as it incorporates all daily market information such as economic news and market sentiment, the most stable and analysed price due to its independence from intra-day volatility, and is recognized as the stock's official price for the day. Fig. 1 displays these closing prices of the S&P 500 over the last 20 years.

The plot illustrates the stock market's long-term upward trajectory, interspersed with declines and volatility due to market corrections, economic downturns or financial crises. Increasing volatility can be observed in the last 5 years. Fig. 2 decomposes the closing price into trend, seasonal, and remainder components, highlighting a consistent upward trend with uniform seasonal changes and nonuniform residuals. The analysis employs the Percentage Change Ratio (PCR) for predicting stock movements, rather than utilizing raw closing prices.

Figure 1: S&P closing price 20 years

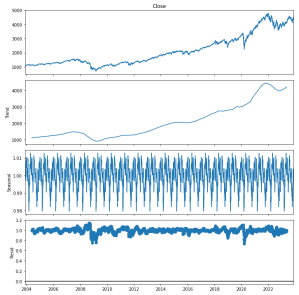


Figure 2: Time series decomposition of S&P closing price

For this analysis, the Percentage Change Ratio (PCR) has been used to predict the stock movement instead of using the raw closing price.

$$\text{PCR} = \frac{\log(\text{Close}_t) - \log(\text{Close}_{t-1})}{\log(\text{Close}_{t-1})} \times 100 \qquad (1)$$

The PCR provides a normalized measure of relative change from one day to the next. The logarithmic transformation stabilizes the variance across time. This approach, using the natural logarithm of closing prices, addresses non-stationarity issues, focusing on relative rather than absolute changes. This scale-invariant measure enhances the comparison of

price movements over time and across various stocks. [6]

2.1.1 Feature Selection

Several technical indicators, listed below, are considered as potential features for the prediction model:
Relative Strength Index (RSI): Measures the speed and change of price movements.

$$RSI = 100 - \frac{100}{1 + RS} \tag{2}$$

where,

$$RS = \frac{\text{Average Gain}}{\text{Average Loss}}. \tag{3}$$

Simple Moving Average (SMA): Calculates the average of the closing prices, by the number of periods in that range.
Exponential Moving Average (EMA): Similar to SMA but gives more weight to recent prices.
Moving Average Convergence Divergence (MACD): Shows the relationship between two moving averages of a security's price.

$$MACD = EMA_{\text{fast}} - EMA_{\text{slow}} \tag{4}$$

where,
EMA_{fast} and EMA_{slow} are the fast and slow EMAs.

Average True Range (ATR): Measures market volatility.

$$ATR = \frac{1}{n} \sum_{i=1}^{n} \text{TR}_i \tag{5}$$

where,

$$TR = \max\left[(\text{H} - \text{L}), |\text{H} - \text{C}_{\text{prev}}|, |\text{L} - \text{C}_{\text{prev}}|\right] \tag{6}$$

H, L, and C are the High, Low, and Close prices respectively. **Stochastic Oscillator (Stoch)**: Compares a closing price to a range of its prices over a certain period.

$$\%K = \frac{(\text{Close} - \text{Low}_n)}{(\text{High}_n - \text{Low}_n)} \times 100 \tag{7}$$

These indicators are known to uncover hidden stock price patterns, supported by past research [9]. A correlation analysis, visualized through a heatmap Fig. 3, helps in understanding the interdependencies among these features.

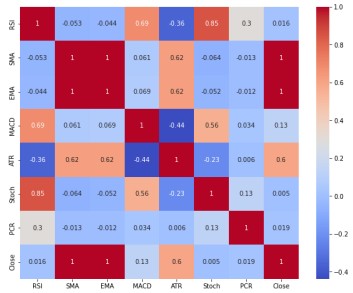


Figure 3: correlation of features

The technical indicators, EMA and SMA are excluded from RNN model training due to their high correlation with closing prices. These averages can introduce redundancy and multicollinearity in the model, skewing results and interpretations. The final features selected were: Close, RSI, MACD, ATR, Stoch, and PCR. The training set constitutes 60% of the data, 20% validation sets, and 20% test sets. This split is a crucial step in preparing the data for the predictive modeling process and it is executed whilst maintaining chronological order for accurate time series forecasting. To prevent data leakage and maintain uniform scaling across datasets, the training data is used to fit a scaler. The training data is used to fit the scaler, and then this scaler is applied to transform the training, validation, and test sets. This scaler then normalizes the features of the training, validation, and test sets, ensuring effective learning by the model.

2.1.2 Sliding Time Window

The dataset sequences are generated through a sliding window technique, essential for preprocessing in sequence prediction models. This method assumes future series values depend on past ones. [12] A fixed-size window slides over the time series, capturing a sequence of data points at each position. These sequences serve as model input, aiming to predict the subsequent point in the series.

For each window position, the function extracts a sequence of a specified length (sequence length) as the input (features) and the next data point as the label (target). These sequences and labels are stored in separate lists. With N data points and a sequence length of L, there are N - L sequences and corresponding labels. The sliding window method preserves the chronological sequence, enabling pattern recognition over time. Longer windows capture long-term trends, and shorter ones focus on short-term patterns.

2.2 Model Architectures

Three primary RNN architectures are employed for this time series prediction task: basic RNN, LSTM, and GRU. Each of these models has its unique way of handling temporal dependencies in data.

2.2.1 RNN Architecture

RNNs are foundational architectures in neural network models dealing with sequential data. A basic RNN processes input sequences one element at a time, maintaining a hidden state vector that implicitly contains information about the history of all the past elements of the sequence. [10]

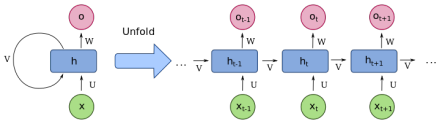


Figure 4: Reccureent Neural Network [3]

The basic RNN architecture shown in Fig. 4 consists of a layer of neurons, each connected to itself and to every other neuron. This architecture is particularly suited for time series data like stock prices, as it can process sequences of data and retain information about previous inputs through its connections.

When predicting the S&P 500 price change ratio, an RNN can be trained on historical price data. It uses this data to learn patterns and trends in the stock's past performance. Each neuron in the RNN processes the input and combines it with its previous state (the information it has retained from previous inputs), generating an output that represents the next prediction.

The ability of RNNs to maintain a sort of 'memory' over the inputs makes them suitable for predicting stock prices, where past trends and patterns are often indicative of future movements. However, basic RNNs have limitations, particularly in capturing long-term dependencies because of the vanishing gradient problem, which can make them less effective for predicting stock prices over longer periods. For such tasks, more advanced variants like LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Unit) are generally preferred due to their ability to retain information over longer sequences without suffering from the vanishing gradient issue.

2.2.2 LSTM Architecture

LSTM networks are a specialized form of RNNs that are designed to overcome the limitations of traditional RNNs, especially the vanishing gradient problem. This makes LSTMs particularly well-suited for learning from long sequences of data.

The key to LSTM's effectiveness lies in its unique structure as , which includes three types of gates: input, output, and forget gates. These gates determine what information should be retained or discarded as the data passes through the sequence of the network [4]:

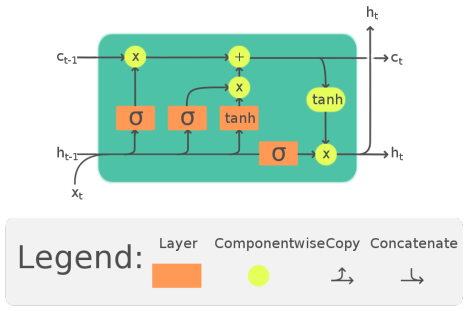


Figure 5: LSTM architecture [11]

The Input Gate Decides which values from the

input to update the memory state and the Forget Gate determines what information should be discarded from the cell state. The Output Gate controls the output based on the cell state and the input.

The forward pass of an LSTM cell (shown in Fig. 5) with a forget gate can be defined as.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{8}$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{9}$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{10}$$

$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{11}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{12}$$

$$h_t = o_t \odot \sigma_h(c_t) \tag{13}$$

where the initial values are $c_0 = 0$ and $h_0 = 0$ and the operator $\odot$ denotes the Hadamard product (element-wise product). The subscript t indexes the time step. Variables:

$x_t \in \mathbb{R}^d$: input vector to the LSTM unit

$f_t \in (0,1)^h$: forget gate's activation vector

$i_t \in (0,1)^h$: input/update gate's activation vector

$o_t \in (0,1)^h$: output gate's activation vector

$h_t \in (-1,1)^h$: hidden state / output vector

$\tilde{c}_t \in (-1,1)^h$: cell input activation vector

$c_t \in \mathbb{R}^h$: cell state vector

$W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h}$: weight matrices

$b \in \mathbb{R}^h$: bias vector parameters

σ_g : sigmoid function

σ_c, σ_h : hyperbolic tangent functions

where the superscripts d and h refer to the number of input features and number of hidden units, respectively.

Each LSTM cell maintains a state over time, effectively allowing the network to remember or forget information dynamically. This ability to capture long-term dependencies makes LSTMs highly effective for sequential data prediction tasks like stock price forecasting.

2.2.3 Gated Recurrent Units

The Gated Recurrent Unit (GRU) is an advanced variant of the standard recurrent neural network, which is designed to better capture dependencies in sequences of data, particularly for longer sequences. The GRU simplifies the LSTM model by combining the forget and input gates into a single "update gate," and it merges the cell state and hidden state, resulting in a more streamlined architecture. Key components of the GRU architecture, as shown in Fig. 6, include:

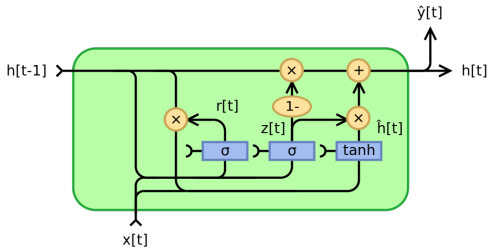


Figure 6: GRU architecture [11]

Update Gate (z): Determines how much of the past information needs to be passed along to the future. It helps the model to decide how much of the previous information to keep versus the new information from the current input. Reset Gate (r): Decides how much of the past information to forget. It allows the model to decide how much of the past information is irrelevant for the current prediction. Current Memory Content: Combines the past information (based on the reset gate) and the current input to create a candidate for the new hidden state. Final Memory at Current Time Step: A combination of the old state and the new memory content, modulated by the update gate.

The forward pass of an GRU cell (shown in Fig. 6) can be defined as:

Initially, for $t = 0$, the output vector is $h_0 = 0$.

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{14}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{15}$$

$$\hat{h}_t = \phi(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{16}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t \quad (17)$$

Variables:

x_t : input vector
h_t : output vector
$\hat{h}_t$: candidate activation vector
z_t : update gate vector
r_t : reset gate vector
W, U, and b : weight matrices and bias
σ : logistic activation function
ϕ : hyperbolic tangent activation function

GRUs can effectively handle the time-series data, capturing the essential temporal dynamics and dependencies in stock price movements. The GRU's ability to remember and forget information adaptively makes it suitable for modeling financial time series data.

2.3 Training and Evaluation

The models undergo training and validation using the prepared dataset. The training process involves feeding sequences of data into the models and adjusting their weights through backpropagation, using Adam optimizer.

The Adam optimizer was chosen for training the models due to its effectiveness and efficiency in handling large datasets and high-dimensional parameter spaces, characteristics common in stock price prediction tasks. Adam stands out for its adaptive learning rate, which adjusts as the training progresses, making it more efficient than traditional gradient descent methods.[5] This optimizer computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. This feature helps in converging faster towards the optimal solution, particularly in scenarios with noisy or sparse gradients, as often encountered in financial time series data.

The performance of each model is assessed using metrics such as Mean Squared Error (MSE) and Root Mean Squared Error (RMSE), vital for evaluating prediction accuracy.

2.3.1 Hyperparameter Optimization

Hyperparameter optimization is conducted using the ree-structured Parzen Estimator (TPE) algorithm hyperopt library, searching for the optimal configuration of parameters like the number of layers, hidden dimensions, learning rate, dropout rate, and batch size.

The TPE algorithm, used within the hyperopt library for hyperparameter optimization, represents a sophisticated approach to tuning model parameters. Unlike grid or random search, TPE constructs a model of the objective function and uses this to select the most promising hyperparameters to evaluate in the true objective function.[1] TPE works in two phases: first, it samples a set of hyperparameters and evaluates them; then, based on these results, it builds a probabilistic model that predicts which hyperparameter values are likely to yield better performance. This method is particularly efficient because it progressively narrows the search to the most promising regions of the hyperparameter space. This optimization is crucial as it fine-tunes the models to achieve the best possible performance on the dataset.

2.3.2 Final Model Selection and Testing

After training and optimizing the models, the best-performing model is selected based on the lowest MSE. The chosen model then undergoes further evaluation on the test dataset, which is crucial for assessing its real-world applicability. The evaluation involves converting predicted PCRs to stock prices and comparing actual stock prices with the model's predictions, providing a concrete measure of the model's predictive power.

3 Results

In the initial experimentation stage, the following parameters shown in Tab. 1 were kept constant:

The validation loss (MSE) for the proposed architectures have been summarised in Tab. 2 where column HL represents the number of hidden layers.

From the three architectures chosen, the LSTM architecture performed the best during initial experi-

Parameter	Value
Learning Rate	0.0001
Epochs	5
Batch Size	32
Loss Function	MSELoss
Optimizer	Adam
Sequence Length	60
Hidden Layer Dimension	8

Table 1: Hyperparameters of the Model

Model	HL	Dropout	MSE	RMSE
base_rnn	2	No	0.0039	0.0626
lstm	2	No	0.0016	0.0412
gru	2	No	0.0053	0.0730
ml_lstm	5	No	0.0132	0.1147
ml_lstm_do	5	0.3	0.0013	0.0360

Table 2: Accuracies for different architectures

mentation stage with least MSE, the LSTM was then tested highers number of hidden layers and a drop out rate of 0.3. The multilayer LSTM performed poorly but on the validation set, However the multilayer LSTM model with dropout showed the best performance with the lowest MSE.

Hyperparameter	Space
Number of Layers	2, 3, 4
Hidden Dimensions	4, 8, 16, 32
Learning Rate	eˆ-5 to eˆ0 (log uniform)
Dropout Rate	0 to 0.5 (uniform)
Batch Size	16, 32, 64
Sequence Length	10, 30, 60, 90

Table 3: Hyperparameter Space for Optimization

From the hyperparameter space shown in Tab. 3, the best parameters obtained were: Number of Layers: 4, Dimensions of hidden layers: 32, learning rate: 0.2930, dropout rate: 0.0630, batch size: 64 and sequence length: 90

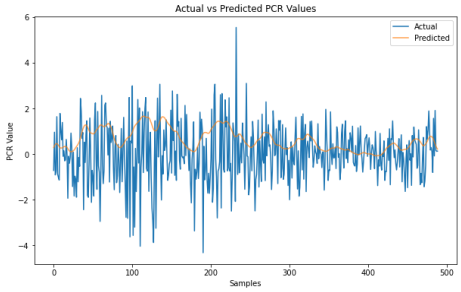
The final model was trained on these parameters 10 Epochs, the learning rate was reduced gradually using a learning rate schedular. This model obtained an MSE of 0.0011 with. This model obtained an MSE of 0.0022 on the test dataset. The accuracy curves and loss curves are shown in Fig. 8. The final model is evaluated on the test dataset and the actual and predicted PCR values obtained from the final model are shown in Fig. 7a, the predicted closing values are calculated by inverse transformation of the PCR values, these predicted and actual closing values are shown in Fig. 7b.

4 Discussion

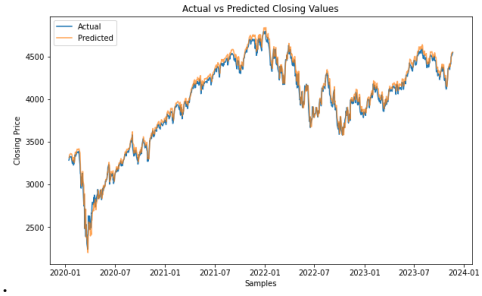
From RNN, LSTM, and GRU architectures the LSTM had the least MSE. Therefore this architecture was chosen for the rest of the analysis. The ability of the LSTM model to handle and remember log-term dependencies resulted in higher performance than the rest of the models. To further improve the performance of the model, several hidden layers were added to create a deeper model and dropout was introduced in the LSTM model to prevent the models from overfitting.

As shown in Tab. 2 The LSTM model with a dropout layer exhibited the best performance, a result that is substantiated by the training and validation loss curves shown in crefFig. 7a. These curves show a swift decrease in training loss, suggesting that the model quickly assimilated the patterns within the data. The stability of the validation loss at higher epochs indicates that the model's learning generalized well to new data, avoiding overfitting.

The Actual vs Predicted PCR and Closing Prices graph shown in Fig. 7 further supports this observation. The predictions closely shadow the actual PCR values and Closing prices despite their volatility, demonstrating the model's capability to capture the complex, time-dependent structures within the stock market data. This adherence to the actual trend, even amidst market fluctuations, underscores the LSTM's proficiency in handling long-term dependencies, a feature enhanced by the inclusion of dropout layers. The dropout layers contribute significantly to the model's performance by introducing regularization, which discourages overfitting and encourages the learning of a more generalized represen-



(a) Actual vs Predicted Closing PCR



(b) Actual vs Predicted Closing price

Figure 7: Actual vs Predicted

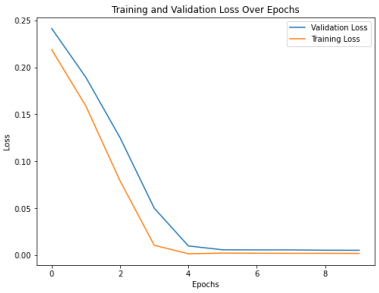


Figure 8: Trainig and validation loss

tation of the data.

The utilization of the TPE algorithm for hyperparameter optimization played a significant role in fine-tuning the model, ensuring that the most effective parameters were chosen to maximize predictive accuracy. The results from the optimized hyperparameters, characterized by a low MSE on the validation set, confirm the effectiveness of the LSTM with dropout in predicting S%P 500 index prices. These outcomes, together with the systematic decrease in learning rate over epochs, reveal an important relationship between model architecture, regularization techniques, and hyperparameter tuning, which is critical for developing high-performing predictive models in the financial domain.

Overall, the LSTM model with dropout layers, trained with an appropriately adaptive learning rate and rigorously optimized hyperparameters, stands out as a robust predictive tool. It showcases an enhanced ability to learn from historical data and predict future stock prices accurately as shown in Fig. 7b, reflecting its superior capability to navigate the complexities of financial time series forecasting.

5 Code

The code can be accessed here: https://github.com/shavinkalu23/7318_assignment_3

6 Conclusion

The project's design choices were steered toward optimizing the prediction accuracy of the S&P 500 index prices. The LSTM model with a dropout layer yielded the best performance, as evidenced by the lowest Mean Squared Error (MSE) on the validation set. This effectiveness can be credited to the LSTM's proficiency in managing long-term dependencies and the dropout layer's role in mitigating overfitting. Future work could explore the integration of external economic indicators into the model and the application of ensemble methods to further refine predictions. Additionally, the use of transformer networks with multi-headed attention could be investigated, offering a new avenue for capturing complex temporal relationships in financial data. Such advancements could significantly enhance the model's interpretability and forecasting accuracy, propelling the field of financial analytics forward.

References

[1] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011. 6

[2] Hyeong Kyu Choi. Stock price correlation coefficient prediction with arima-lstm hybrid model. *arXiv preprint arXiv:1808.01560*, 2018. 2

[3] Wikipedia contributors. Recurrent neural network — wikipedia, the free encyclopedia, 2023. [Online; accessed 20-November-2023]. 4

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 4

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[6] VV Kondratenko and Yu A Kuperin. Using recurrent neural networks to forecasting of forex. *arXiv preprint cond-mat/0304469*, 2003. 3

[7] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017. 2

[8] Sreelekshmy Selvin, R Vinayakumar, EA Gopalakrishnan, Vijay Krishna Menon, and KP Soman. Stock price prediction using lstm, rnn and cnn-sliding window model. In *2017 international conference on advances in computing, communications and informatics (icacci)*, pages 1643–1647. IEEE, 2017. 1

[9] Dev Shah, Wesley Campbell, and Farhana H Zulkernine. A comparative study of lstm and dnn for stock market forecasting. In *2018 IEEE international conference on big data (big data)*, pages 4148–4155. IEEE, 2018. 1, 3

[10] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404, 2020. 4

[11] Long short-term memory. Recurrent neural network — wikipedia, the free encyclopedia, 2023. [Online; accessed 20-November-2023]. 4, 5

[12] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988. 3