Time Series Stock Prediction using Recurring Neural Networks

Nithya Shree Kannan
nxk210045

Aishwarya Nagaraj
axn210126

The University of Texas at Dallas

1. ABSRACT

The stock market is a vast and complex landscape, involving trillions of dollars in daily transactions. While neural network modeling has become increasingly prevalent in stock market trading, many traders remain unfamiliar with this sophisticated technique. This lack of understanding can hinder their ability to make informed investment decisions. To address this issue, this paper explores the application of deep reinforcement learning in stock trading. Utilizing real-time market data from Kaggle, the proposed algorithm aims to maximize returns for investors. By analyzing opening and closing prices, the model identifies optimal buying and selling opportunities to maximize profits.

2. INTRODUCTION

The stock market serves as a platform for companies to raise capital through the issuance of stock shares and corporate bonds. While anyone can participate in this financial system, there is an inherent risk of losses associated with stock market investments. Participants range from novice individuals to seasoned professionals with extensive business and trading expertise. Research suggests that over 90% of stock market participants experience losses due to the inherent complexities and erratic nature of the market. This uncertainty has led to a growing reliance on technology-driven decision-making tools, as opposed to solely relying on individual judgment. Various algorithms, including Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Linear Regression. are utilized for stock market forecasting. However, these algorithms have limitations, as they primarily focus on data prediction rather than making definitive investment decisions. The stock market is an inherently dynamic and volatile environment, with fluctuations influenced by a number of factors, including political developments, global economy, unforeseen incidents, and the financial performance of the company. Effective navigation of this complex landscape requires a comprehensive understanding of these factors and their potential impact on market trends.

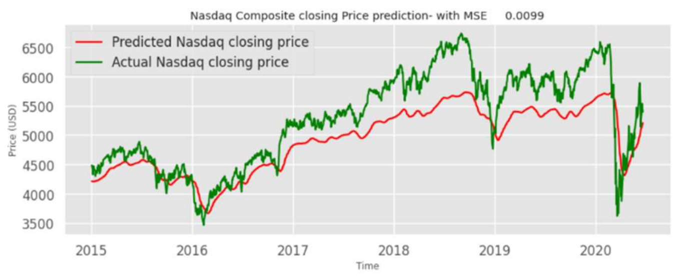


Fig1: Forecasting Multiple Time Series using NNs

By analyzing historical stock market data, time series analysis can shed light on future price movements. These predictions are categorized into three main types: Short-term forecasting, which involves making predictions for a few seconds or minutes to a few months, Medium-term forecasting focuses on predictions spanning one to two years and Long-term forecasting, on the other hand, extends beyond two years. Reinforcement learning offers a unique approach that not only predicts future data but also guides decision-making. This capability is particularly valuable in the stock market, as it empowers traders to make informed decisions based on data-driven insights. To train the reinforcement learning model, historical stock prices from the U.S. market are utilized, encompassing a specific time frame, and including closing stock prices. This dataset is then employed to train the model. After numerous iterations, the approximated Q-table is applied to test data, which follows the training data in the time series. The aimis to establish a system that rewards positive behaviors and punishes unfavorable behaviors, leading to optimized decision-making.

3. THEORITICAL AND CONCEPTUAL STUDY

3.1 RNN

Applications of neural networks span various domains, with sentiment classification, image captioning, and linguistic translation. Recurrent neural networks (RNNs) are particularly adept at handling sequential data, mapping inputs to outputs of varying types and lengths. RNNs excel at capturing long-term dependencies in data that exhibit a time dimension. Multilayer perceptrons (MLPs) constitute a category of artificial neural networks characterized by three distinct layers: an input layer, a hidden layer, and an output layer. The input layer receives input data, the hidden layer processes the data through activation functions, and the output layer produces the final output..

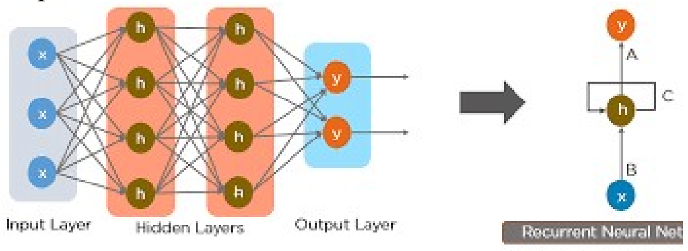


Fig2: RNN Network

RNNs use a concept of sequence processing which can be visualized as sequentially feeding words into the network. Each word, represented as an input, is processed and combined with the previous word to maintain the sequence of information. This process can be repeated for any number of words, allowing the RNN to capture long-term dependencies in sequential data. The multiple hidden layers within an RNN can operate independently due to their unique weights and biases, enabling each layer to learn and process specific aspects of the input sequence. To predict the recurrent state in an RNN, the formula used is:

$$h_t = f(h_{t-1}, x_t)$$

Here, the new state is h_t, the previous state is $\mathbf{h}_{t-1}$, and the current input is $\mathbf{x}_t$ **.**

A time step is identified as the transition from one consecutive state to the next, occurring as the current input merges with the previous input to adjust weights and biases. Utilizing this concept, a basic formula for a Recurrent Neural Network (RNN) can be constructed, featuring tanh as the activation function, W_{hh} as the recurrent neuron's weight, and W_{xh} as the input neuron's weight.

$$ht = tanh(w_{hh}h_t - 1 + w_{xh}x_t)$$

The output is computed through the following formula once the current state is acquired using the previous equation:

$$y_t = w_{hy}h_t$$

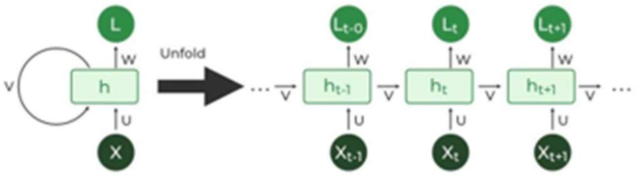


Fig3: Flow of data in RNN

Steps for entropy calculation involve employing the cross-entropy loss to determine the error value, as depicted by the formula below. Here, $\overline{xt}$ represents the actual value.

$$E(\bar{x}_t, x_t) = -\vec{x}_t log(x_t)$$
$$E(\bar{x}, x) = -\Sigma \vec{x}_t log(x_t)$$

Furthermore, the cross-entropy loss for each timestamp in a time-series is computed using the equation:

Loss(n) = -(x*log(xhat) + (1-x)*log(1-xhat))

Total Loss = Mean(Loss(n)).

Finally the gradient descent is applied to minimize the loss

3.2 LSTM

In contrast to conventional recurrent neural networks (RNNs), which suffer challenges related to the vanishing gradient problem, Long Short-Term Memory (LSTM) networks employ a unique architecture that effectively addresses this issue. While RNNs utilize a single repeating module, LSTMs incorporate four interconnected modules, each playing a crucial role in enabling the network to learn and retain long-term dependencies within sequential data. This enhanced architecture allows LSTMs to effectively process and analyze sequences with extended temporal relationships.

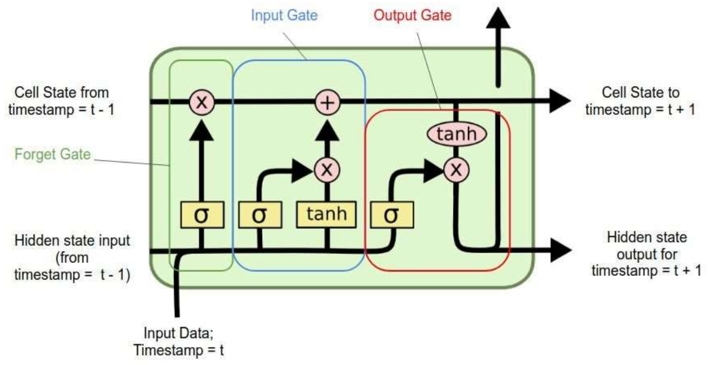

Input Gate
Output Gate
Cell State from timestamp = t - 1
Cell State to timestamp = t + 1
Forget Gate
tanh
Hidden state input (from timestamp = t - 1)
Hidden state output for timestamp = t + 1
Input Data; Timestamp = t

Fig4: LSTM Network

The illustration illustrates the complex operations of a Long Short-Term Memory (LSTM) network, a specialized variant of a recurrent neural network (RNN) crafted to adeptly handle sequential data characterized by prolonged temporal dependencies. The horizontal lines represent the flow of information between different components of the LSTM unit, while the vertical lines denote pointwise operations, such as element-wise multiplication or addition. The rectangular boxes symbolize neural network layers, each consisting of interconnected nodes that perform specific computations on the input data. Concatenation, represented by merging lines, indicates the combination of two or more data streams, while forking, depicted as a single line splitting into multiple branches, signifies the replication of data for use in different parts of the network.

The Mathematical equations for LSTM are:

$$
\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$

where i is input gate, o is output gate, f is forward gate, c is cell state, and h is next state.

4. OUR APPROACH

Data from the stock market is highly dynamic and constantly evolving, making it challenging to accurately predict stock prices. Recurrent neural networks (RNNs) have emerged as a powerful tool for this task, but traditional RNNs often struggle to capture long-term dependencies in the data. Long Short-Term Memory (LSTM) networks, a specialized type of RNN, address this limitation by employing a three-gate architecture that selectively stores relevant information and discards irrelevant data. This makes LSTMs especially effective for examining sequential data that involves extensive dependencies over a long period, such as stock market data.

To train LSTM networks for stock price prediction, two propagation methods are used: forward propagation and backward propagation. Forward propagation calculates the current hidden state based on the prior hidden state, the current input, and the three gates of the LSTM cell. Backward propagation, on the other hand, calculates the cross-entropy loss for each timestamp and utilizes gradient descent to minimize the loss function. This iterative process allows the LSTM network to learn and refine its parameters, improving its ability to make accurate predictions.

During training, stock market data is fed into the LSTM network in batches, with the network output generated for each time slot. This approach ensures that the network receives a continuous stream of information, enabling it to make informed predictions based on the latest available data.

Feature details:
Date: The trading day's date in the stock market.
Open: The price value when the stock market opens.
Close: The price value adjusted for market splits.
Adj Close: The adjusted closing price value, accounting for both splits and dividends in the market.
High: The highest price value recorded during the day.
Low: The lowest price value observed during the day.
Volume: The total number of shares exchanged throughout the day.

The training of the neural network relies on seven key features: Date, Open, High, Low, Close, Adj Close, and Volume. These features are combined into a single vector using VectorAssembler, ensuring seamless integration into the neural network architecture. Normalization of these features is an essential step, as it helps the predictive model generate accurate and reliable results. This process rescales the range of values to minimize the impact of outliers and ensure that all features contribute equally to the network's learning process. The MinMaxScaler is utilized for normalization, wherein it involves subtracting the minimum value from each feature and subsequently dividing this result by the range. The range is determined by subtracting the original maximum value from the original minimum value.

$$\boldsymbol{R_e = Re - FminFmax - Fmin * (m1 - m2) + m2}$$

In this scenario, m1 and m2 define the rescaling range, with Fmin indicating the minimum and and Fmax indicating the maximum values of the feature, respectively. Re signifies the rescaled value. Afterwards, the features contained in the DenseVector are fed into the model. After training the dataset with an appropriate split, RMSE is calculated to forecast the values of the test data. This normalization method ensures that all features are limited to a standardized range, preventing any single feature from dominating the network's learning process.

5. RESULTS AND ANALYSIS.

	Neurons	RSME Test Values
1.	40	2.4761192159633114
2.	80	1.3266362785173387
3.	120	1.2505951096024366
4.	160	1.4883862586728616
5.	200	1.1598412055082048
6.	240	1.115401409714951
7.	280	1.1786885466430446
8.	320	1.2008251796760347

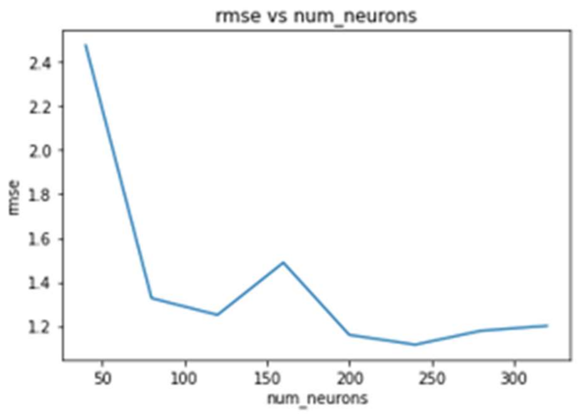
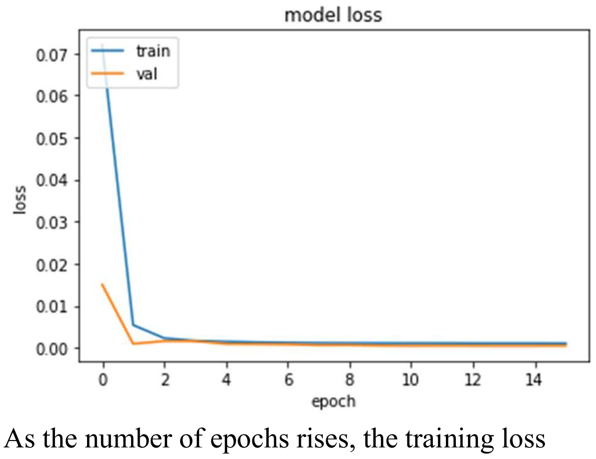


Fig5: RMSE vs Number of neurons

As the number of epochs increases, there is a reduction in the number of neurons. The learning process of RNN LSTM involves adjusting weights, indicating that the loss function tends to reach a lowest point when the number of neurons decreases with the increasing number of epochs.



As the number of epochs rises, the training loss experiences a notable decline before stabilizing, while the value remains constant as the number of epochs continues to rise. This is illustrated in the chart,

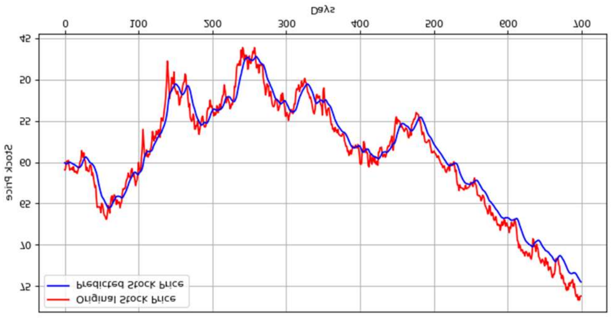


Fig6: Original Price vs Predicted Price

In this graph, the original data is represented in red, while the predicted data is depicted in blue. It is evident that the model performs exceptionally well, with the predicted price closely aligning with the original price.

	Batch Size	RSME Test Values
1.	10	1.5492337400907503
2.	20	1.6970608495060475
3.	30	1.2295227060565697
4.	40	1.4284704635142293
5.	50	1.1799898064767207
6.	60	1.2082542391024897
7.	70	1.1618321142147707
8.	80	1.4696364348819315
9.	90	1.605822766393762
10.	100	1.266698879379596

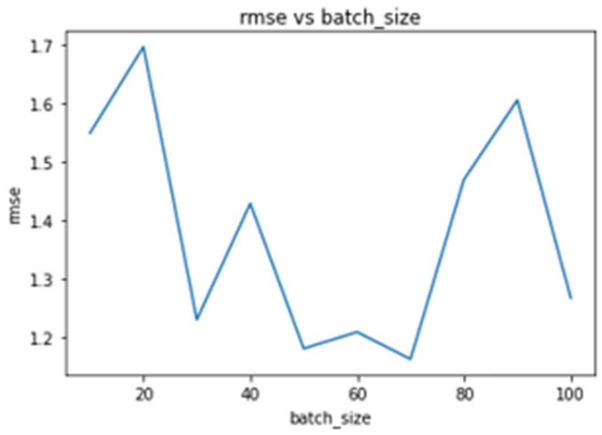


Fig7: RMSE vs Batch size

The graph clearly illustrates that the relationship between RMSE and batch size is not straightforward. When batch sizes are extremely small, the RMSE values tend to be high. This is because the model doesn't have enough data to accurately fine-tune its hyperparameters, leading to suboptimal predictions. On the other hand, excessively large batch sizes also result in high RMSE values due to the inherent variability in the dataset. The ideal batch size for this model is approximately 70, as evident from the graph. The graph clearly shows that the minimum RMSE value is achieved at a batch size of around 70.

	Epoch	RSME Test Values
1.	2	2.7832182881733014
2.	4	2.4860851501602683
3.	8	1.5746618501516865
4.	16	1.3914134366731348
5.	32	1.1601124039880255
6.	64	1.2941163338958068

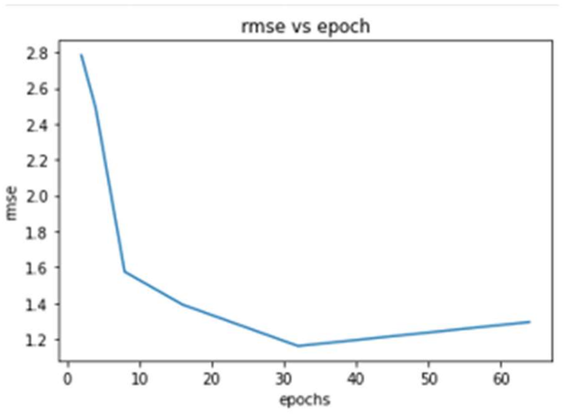


Fig8: RMSE vs Epoch

On examining the graph, it becomes evident that lower epoch values result in higher RMSE values. However, as the number of epochs increases, there is a substantial reduction in RMSE values for the entire dataset, encompassing both training and testing sets. Nevertheless, opting for epoch values beyond the graph's maximum leads to an increase in error, attributed to overfitting.

6. CONCLUSION

For project, we utilized RNNs and LSTMs to streamline the neural network analysis of stock market prices for companies based in the US.. Leveraging Apache Spark, we efficiently preprocessed and analyzed the data, considering its volume and structure. To optimize space and time complexity during machine learning model implementation, we utilized data frames and their associated libraries. Through a process of fine-tuning hyperparameters, involving varying values based on dataset size, k-values, epochs, and other factors, we were able to improve prediction accuracy.

As a machine learning model, there is always room for further enhancement. Exploring techniques such as incorporating variance and moving averages into hidden nodes could be beneficial. Additionally, utilizing specialized clusters within the software could significantly improve computational efficiency.

7. REFERENCES.

[1] N. Sirimevan, I. G. U. H. Mamalgaha, C. Jayasekara, Y. S. Mayuran and C. Jayawardena, "Stock Market Prediction Using Machine Learning Techniques," 2019 International Conference on Advancements in Computing (ICAC), 2019, pp. 192-197, doi: 10.1109/ICAC49085.2019.9103381.

[2] Hiransha M, Gopalakrishnan E.A., Vijay Krishna M, Soman K.P., (2018) NSE Stock Market Prediction Using Deep-Learning Models, vol 132.

[3] Nabipour, M.; Nayyeri, P.; Jabani, H.; Mosavi, A.; Salwana, E.; S., S. Deep Learning for Stock Market Prediction. *Entropy* 2020, *22*, 840.

[4] http://spark.apache.org/docs/2.1.1/ml-features.html

[5] https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs