Stock Prediction using RNN Model

Saurab Kharel

December 5, 2024

Abstract

With the change in technology, the static analysis of stock has been changed to dynamic with the help of different algorithms. The unpredictable nature of stocks, which is basically based on the speculative market price of an individual stock, helps to earn money i.e. return of investment of an investor. With the help of deep learning using RNN, it is possible to collect the different dotes or features of stock data, process and analyze the behavior of it which helps to correctly predict the price of stock price. With the help of RNN, LSTM, GRU etc. are used to predict the price of stock calculating the RMSE and MSE.

With the introduction of Artificial Intelligence and Machine Learning Language along with deep learning techniques using different models like RNN, LSTM, GRU are the powerful tools to analyze the patterns of stock and can forecast it. Due to its ability to understand a variety of data, including real-time information and unstructured data data, temporal dependencies, model complex, non-linear relationships, the model like RNN's adaptability has been the excellent preferences for predicting the stock and its future patterns.

1. Introduction

A stock is a representation of company ownership in a publicly listed company. Once who owns the company, he/she can be the one of the decisions maker of that company. Similarly, the company also has the responsibility of providing the dividend to its owners. Usually, the stock price is determined by supply and demands whereas there are multiple factors associated with it like financial position, investor's interest, political situation, economic indicators etc. Price forecasting is a crucial area of financial analysis supporting investors and traders to make their decisions.

The traditional approaches are widely adopted apart from using sophisticated languages or models to predict the price of stock. The fundamental areas of analyzing a stock are earning per shares (EPS), Price to Earnings (P/E) Ratio, Growth of an organization with respect to revenue etc. helps to study the financial position of that company, analyzing the macroeconomic of the country and company leadership, mission-vision helps to understand the price of shares which are publicly available for trading. In the same way, technical analysis is also done to predict the future price of stock after collecting the historical limited data and plotting the graph or charts accordingly. With traditional approaches, changes in political events, natural disasters, abrupt changes in leadership etc. it is quite difficult to understand the market dynamics.

Apart from the RNN, there are several models that have been considered for prediction of stock which are broadly classified into traditional statistical models and machine learning models. Models like Autoregressive Integrated Moving Average (ARIMA) using time-series itself relying on the basic data distribution and inter-relationship often falls short. It is primarily used for short term forecasting with linear trends, but it lacks capturing complex non-linear data patterns.

Similarly, Generalized Autoregressive Conditional Heteroskedasticity (GARCH) is another model used to predict the unusual pattern of stock focusing on conditional change like period of highly active of stock and useful to manage portfolio or risk appetite whereas it lacks predicting the price of stock. In the same way, Machine Learning models like Linear Regression, Decision Tree, Random Forest, Support Vector Machines etc. are widely adopted but it is also not able to predict the exact patterns of stock. With the introduction of Deep learning and its high-end capabilities to handle complex data i.e. both structured and unstructured, providing an accuracy of prediction has been widely acclaimed in the different sectors like Financial, Weather forecasting and Energy Consumption Forecasting. The good part about the RNN is that it holds data for future relationship and back propagation so that the outcome can be more accurate whereas the traditional feedforward neural network like CNN can helps to classify the challenges with respect to images but has limitation of forming complex correlations between information. Vanilla Recurrent Neural Networks (RNNs) are one of the simplest forms of RNN architectures and are the introductory for understanding more complex architecture like LSTMs and GRUs. It processes the data sequentially making sure that the previous inputs cannot affect the current output.

After the limitation being identified in the traditional RNN or simple RNN, the Long Short-Term Memory (LSTM) has been introduced by Sepp Hochreiter and Jürgen Schmidhuber in 1997 to overcome the challenges as mentioned above. LSTM helps to overcome challenges like vanishing gradient problem, Handling Long-Term Dependencies etc. With the help of LSTM, the problem related to gradient problem is solved by holding the information for longer time intervals. Similarly, the Long-term dependencies can be resolved after using memory cells and gates to carefully store or remove related information.

The Gated Recurrent Unit has been introduced in 2014 by KyungHyun Cho et al as a simple variant of LSTM which is simpler and easy to get the required output. With GRU, the computational becomes easy and faster to train the dataset because it has fewer gates and captures the dependencies in sequential data with simple architecture.

2. Additional Information Related Works

Before the advent of Deep learning concepts, the varieties of approaches were adopted in different life cycle. With the change of time, different technologies, models, were introduced and followed for the accuracy in the prediction. In the early 20th Century, the people were heavily dependent on the fundamental analysis where the analyst go through the company's profile related to financial situation, market position etc. concentrating the parameters like PE ratios, balance sheets which was time consuming and tedious work.

The technical analysis started to gain momentum focusing on historical price trends and volume of shares traded in the market. Based on it, the projection of price related to stock started with the help of tools like Moving Average, RSI and candlestick patterns to learn patterns of stock so that the investor gains the good capital return of their initial investment. With these approaches, it has a limitation of predicting the stock for long term investment. Techniques like linear regression, ARIMA and other time-series models were tested using historical data and forecast the future trends. The introduction of machine learning and non-linear methods like SVMs, neural networks etc. were far better to identify difficult patterns of datasets and help to foresee the future price of stocks.

Simon Haykin highlighted the use of Artificial Neural Networks (ANNs) to predict the stock price providing substantial evidence for remarkable accuracy compared with respect to traditional analysis model in 1994. With this model, it is very easy to process large volumes of historical data like prices and volumes of stock and can capture the complex patterns and trends of stocks which are not easily identifiable by traditional model. ANN can be called as a breakthrough for further research into machine learning for analyzing the complex pattern of data and predict the future.

In 2002, Leigh and colleagues did some research and explored the possibility of using neural networks to predict the stock market of IBM. Their focus was to demonstrate that the single model prediction cannot correctly predict the market whereas combining the output of multiple neural networks must be adopted to bring accuracy and robustness in the stock price prediction. IBM's stock analysis was done due to availability of historical data and its high liquidity making it a easy for time-series modeling and forecasting analysis.

Huang CJ et al. performed a feasibility study of different data mining classification algorithms, including Artificial Neural Networks (ANNs), with respect to stock market prediction. Their research identified that ANNs model dominates other classification methods providing the evidence to capture non-linear patterns and dependencies in the market but also identified the challenges of inconsistent and unpredictable behavior with ANNs model when applying to noisy and high-dimensional datasets. The very common problem with its overfitting to noise in the data which can be mitigated by combining the different model's approaches.

In 2006, Chen et al demonstrated that PSO-enhanced ANNs can drastically correct in predicting the stock market than the traditional approaches by increasing the efficiency of weights, biases and hyperparameters. It is basically based on the collective behavior of each swarm where each are adjusted to its position based on their own experience.

Vapnik et al. introduced the hybrid approach of ANNs and SVMs in 1997 to predict better results in stock market by combining the strength of both models. Both models conglomerate strong framework for predicting the stock price by their unique strengths like SVMs is purely focused on non-linear data and finding optimal decision areas whereas ANNs are quick at learning complex patterns from raw data. This approaches has multiple benefits like generalization of dataset where standalone ANNs might overfit additionally efficiently handling of high-dimensional financial data with the help of ANN in simplifying the inputs and SVM for predicting the accurate number.

With the introduction of Recurrent Neural Networks, the popularity of ANNs has faded because RNN has a capability of handling the sequential and time-series data more appropriately which is very important in stock price analysis. Stock prices are naturally sequential with latest price of stock has been influenced by previous prices or trends and market behavior i.e. demand and supply, so RNN can hold both short-term and long-term dependencies which are vital point of understanding the patterns of markets, trends etc. Advanced variant like LSTM, GRU overcomes the limitation of traditional RNNs such as vanishing gradient problem. So, the paper is more focused on the three areas of RNN e.g. vanilla RNN, LSTM and GRU using different optimizer to the difference in the prediction along with calculating the RMSE and MSE.

3. Methodologies Adopted:

3.1. Dataset description: The dataset used in the script is a time-series financial dataset of Nepal stock exchange tailored for capturing temporal patterns in stock price movements, making it suitable for predictive modeling with RNNs particularly the closing price close along with Date, Open, High, Low, and Turnover, representing daily stock data.

The necessary preparation has been done like cleaning of unnecessary columns, removing the commas and converting them into float to make sure that the accuracy in the calculation.

The 7-day and 30-day moving averages (MA7, MA30) to capture short-term and long-term trends, as well as lagged features (Lag1, Lag2, Lag3) representing the Close price from 1, 2, and 3 days prior, respectively are also added along with Day, Month, Year, and Day of Week are extracted from the Date column to capture time-based patterns.

The moving average is calculated using the formula:

$$MA_k = \frac{1}{k} \sum_{i=t-k+1}^{t} \text{Close}_i$$

where k is the window size (7 or 30 days), and t is the current time step.

Lagged features are defined as:

$$\text{Lag}_n = \text{Close}_{t-n}$$

where n is the lag period.

For training, the data has been split into 80% and 20% for testing sets. All features are scaled using Min Max Scaler to the range of [0, 1] to make it compatible with RNN model. The datasets have been used by vanilla RNN, LSTM and GRU along with different optimizers like adam, sgd and calculate MSE loss functions. Predictions are opposite to the original scale, and the model's performance is evaluated using the Root Mean Squared Error (RMSE) metric for both training and testing data.

4. Vanilla RNN Model

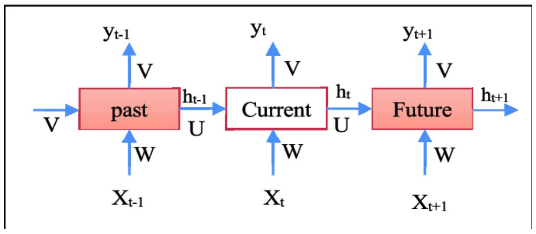


Figure 1: Vanilla RNN

Vanilla RNN is one of the foundations to understand and implement sequential data processing by storing the data in memory of previous time stamps. This stored data in memory is achieved through a hidden state that gets updated as the network processes each time step in a sequence. During the processing time, the Vanilla RNN takes an input vector, combines it with the hidden state from the previous, and computes the present state using the formula mentioned below.

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

where W_{xh} and W_{hh} are weight matrices, b_h is a bias term, and tanh is the activation function. The network can optionally produce an output at each step using:

$$o_t = W_{ho}h_t + b_o$$

The hidden state allows Vanilla RNN to retain information from previous steps, enabling it to model temporal dependencies in sequential data.

The final prediction is based on the information processed by the different time steps of the hidden layer.

The formula has been mentioned below.

$$y_t = \text{softmax}(U_{yh} h_t + b_y) \in R^{|V|}$$

$$P(x_l | x_0, x_1, \cdots, x_{l-1}) = y_t(i)$$

where:

- y_t is the output vector at time t obtained by applying the softmax function.
- U_{yh} is the weight matrix that connects the hidden state h_t to the output.
- b_y is the bias vector for the output layer.
- $P(x_l | x_0, x_1, \ldots, x_{l-1})$ represents the probability of the next token x_l compared to the previous tokens $x_0, x_1, \ldots, x_{l-1}$.
- $y_t(i)$ denotes the i-th element of the vector y_t.

In order to train the Vanilla, to train a Vanilla RNN, the Backpropagation Through Time (BPTT) algorithm is used which access the network across all time steps and calculates gradients for the all sequence to update the model's parameters.

Vanilla has a simple architecture which is easy to understand and implement and increases efficiency by sharing the weight mechanism. It also has limitations like vanishing gradient, short term memory, and limited performance on complex tasks.

5. Long Short Term Memory

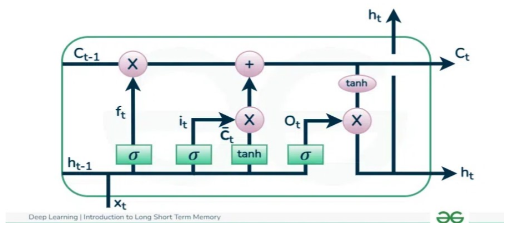


Figure 2: Vanilla RNN

In 1997, Hochreiter and Schmidhuber introduced the concept of Long Short-Term Memory in RNN to address the issue related to vanishing gradient in the traditional RNN network. It has been built in part to collect both short-term and long-term dependencies in sequential data with the help of memory cells and gating mechanisms. It is used for tasks such as language translation, speech recognition, and time series forecasting. The memory cell is controlled by three gates, as mentioned below.

- **Input Gate** - Controls what information is added to the cell state.
- **Forget Gate** - Controls what information is removed from the cell state.
- **Output Gate** - Controls what information is output from the memory cell.

With the help of above flow, the LSTM makes the decision either to store or delete information as it flows through the network and allows them to learn long-term dependencies. It also maintains a hidden state to act as a short-term memory of the network.

Below is the formula of LSTM

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\ C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t \odot \tanh(C_t) \end{aligned}$$

where:

- $[h_{t-1}, x_t]$ represents the concatenation of the previous hidden state h_{t-1} and the current input x_t.
- The sigmoid function σ squashes values to be between 0 and 1.
- The $\odot$ denotes element-wise multiplication.

6. Gated Recurrent Unit (GRU)

In 2014, Cho et al. introduced the concept of Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network (RNN), a simplified form of Long Short-Term Memory (LSTM). It addresses the issue related to vanishing gradient problem with simpler design with less gates and parameters. It helps GRUs to increase the efficiency of computation without changing any performance issue basically in modeling sequential data with temporal dependencies. Both

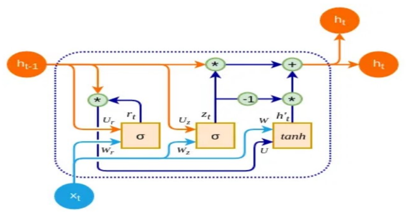


Figure 3: GRU

gate forget and input gates are consolidated into unified update gates which makes the architecture simpler.

The GRU consists of two main gates: the update gate and the reset gate.

1. **Update Gate (z_t):** It helps to control how much information from the past to retain and how much new information to add. The formula is:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

   Where:

   - W_z: Weight matrix for the update gate.
   - b_z: Bias term.
   - σ: Sigmoid activation function.

2. **Reset Gate (r_t):** It determines how much of the past information to remove when computing the new hidden state. The formula is:

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

3. **Candidate Hidden State ($\tilde{h}_t$):** It is computed using the output of the reset gate. The formula is:

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

   Where:

   - $r_t \odot h_{t-1}$: Element-wise multiplication of the reset gate output and the previous hidden state.

4. **Final Hidden State (h_t):** It combines both the previous hidden state and the candidate hidden state based on the update gate. The formula is:

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

7 Optimizer

7.1 Adam Optimizer

Adaptive Moment Estimation is an algorithm used for optimization technique for gradient descent. It is useful with large problems using huge datasets or parameters which require less memory. We can say that it is a conglomerate of gradient descent with momentum and RMSP algorithm. The extensions are defined below.

- **Momentum:** Keeps track of an exponentially decaying average of past gradients to accelerate convergence in a specific direction.
- **RMSProp:** Adjusts the learning rate for each parameter individually based on the magnitude of the gradients.

Adam focuses on two parameters as mentioned below:

- **First Moment Estimate (m_t):** Exponentially weighted average of past gradients.
- **Second Moment Estimate (v_t):** Exponentially weighted average of squared gradients.

Adam uses the following formulas to update parameters:

- **Compute the bias-corrected moment estimates:**

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- **Parameter update:**

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Here:

- β_1: Decay rate for the first moment estimate (default: 0.9).
- β_2: Decay rate for the second moment estimate (default: 0.999).
- ϵ: Small constant for numerical stability (default: 10^{-7}).
- η: Learning rate.

The Adam optimizer is particularly suited for financial data because such data often lack stationary characteristics. Its adaptive learning rate helps optimize over unstable data distributions. Adam also performs well across various architectures without requiring significant changes or tuning. It is primarily used to minimize the loss (e.g., Mean Squared Error) by adjusting learning rates dynamically for all model parameters based on past gradients and squared gradients.

7.2 Stochastic Gradient Descent

SGD makes computation faster and more efficient for large datasets because it updates the parameters for each data sample or mini-batch, minimizing the loss function. However, it has a lower convergence rate as noisy examples may distract the convergence path.

Parameter Update Formula

The parameters are updated using the formula:

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta \mathcal{L}(\theta)$$

Explanation of Terms

- θ_t: The parameter values at step t.
- η: The learning rate.
- $\nabla_\theta \mathcal{L}(\theta)$: The gradient of the loss function $\mathcal{L}$ with respect to θ.

Usage and Characteristics

SGD is particularly effective for:

- Handling large datasets, as it updates parameters incrementally for each sample or mini-batch.
- Simpler architectures, as it requires fewer computational resources.

It is also used to examine the learning rate for convergences, which is crucial to achieving optimal training performance.

7.3 Mean Squared Error (MSE)

It is used to calculate the average squared difference between future values and actual values. It estimates how well a regression model fits the data and penalizes larger errors more heavily because of the squaring operation. Additionally, it provides an aggregate value to evaluate the model's performance.

Expression for Mean Squared Error (MSE):

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

Where:

- n: The number of observations.
- y_i: The actual value of the i-th observation.
- $\hat{y}_i$: The predicted value of the i-th observation.

7.4 Root Mean Squared Error (RMSE)

It is the square root of the Mean Squared Error (MSE). It provides a more understandable metric because it is in the same unit as the target variable, making it easier to interpret the scale of the errors.

Expression for Root Mean Squared Error (RMSE):

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

Where:

- n: The number of observations.
- y_i: The actual value of the i-th observation.
- $\hat{y}_i$: The predicted value of the i-th observation.

4.5 Transfer Learning

Transfer learning involves pre-trained models for new tasks within a similar domain, such as ResNet or VGG.

8 Findings and Results

8.1 Vanilla RNN using SDG Optimizer

The Vanilla RNN implemented to analyze stock prices demonstrates a robust approach to time-series modeling. The data go through cleaning and normalization, including the conversion of numerical columns and the addition of key features such as moving averages (MA7, MA30) and lagged values (Lag1, Lag2, Lag3) to capture temporal patterns. The data was scaled using `MinMaxScaler` and reshaped into a 3D format compatible with the RNN structure to ensure effective handling of sequential dependencies.

The model architecture consisted of two `SimpleRNN` layers with 50 units each, followed by a dense output layer. The first RNN layer stored sequential outputs, while the second layer processed them into the final representation. Stochastic Gradient Descent (SGD) was employed as the optimizer, minimizing the Mean Squared Error (MSE) with 10 epochs.

The model achieved the following results, indicating robust generalization and minimal overfitting:

- **Train RMSE:** 13.26
- **Test RMSE:** 13.38
- **Train MSE:** 175.8914
- **Test MSE:** 179.1469

Although the model performed well, the RMSE suggests slight deviations on predictions. Future improvements could include advanced architectures such as LSTMs or GRUs, optimizing hyperparameters, and enriching the dataset with additional features.

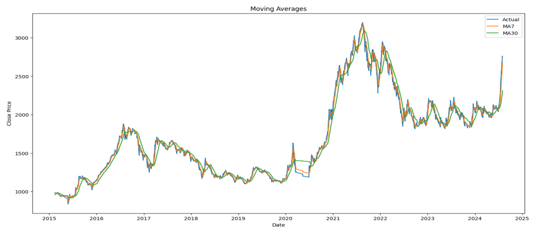


Figure 4: Moving Average

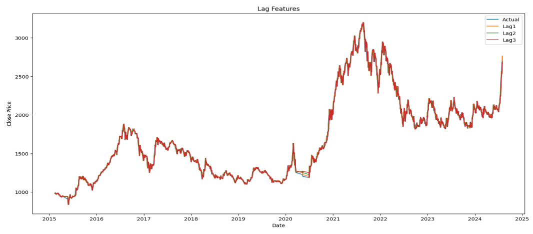


Figure 5: Lag Features

8.2 Vanilla RNN using Adam Optimizer

Using the Adam optimizer with Mean Squared Error (MSE) as the loss function and training for 10 epochs, the model produced the following results:

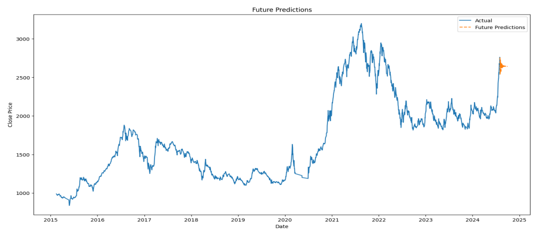


Figure 6: Future Prediction

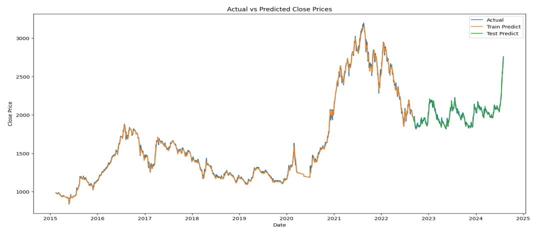


Figure 7: Actual vs Predicted Close Price

- The training process achieved a final loss of 9.8204×10^{-5}, indicating steady convergence.
- The Train RMSE of 59.26 and Test RMSE of 77.78 signifies reasonable prediction errors, but it highlights room for improvement.
- Train and Test MSE values of 3511.95 and 6048.98, respectively, reflect higher variance on test predictions, suggesting potential overfitting or underfitting.

These results demonstrate the model's ability to capture temporal pattern but also highlight the need for refinements. Advanced architectures, such as LSTMs or GRUs, along with hyperparameter optimization, could improve predictive performance and address the observed variance in predictions.

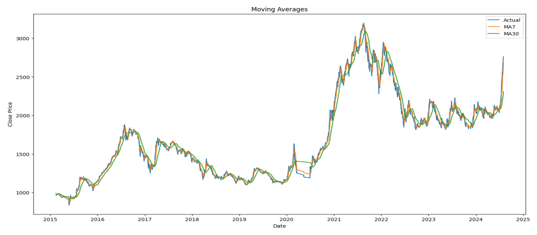


Figure 8: Moving Average

8.3 GRU using Adam Optimizer

The Gated Recurrent Unit (GRU) model has demonstrated a highly efficient approach for handling se-

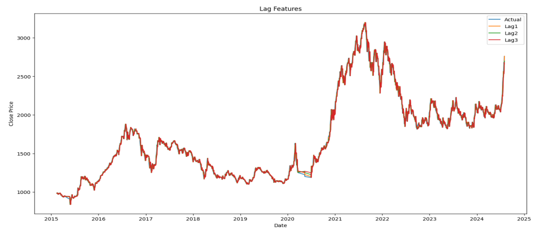


Figure 9: Lag feature

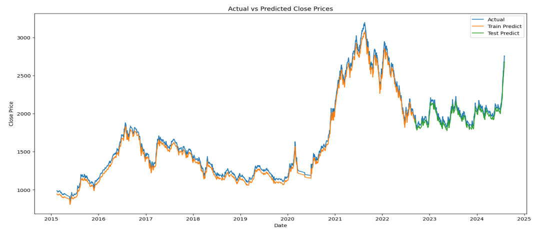


Figure 10: Actual

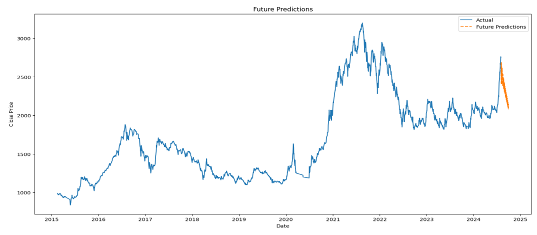


Figure 11: Price Prediction

quential data, showing strong learning and predictive accuracy. The GRU model was trained using the Adam optimizer with Mean Squared Error (MSE) as loss function over 10 epochs and was able to achieve a final training loss of 8.5367×10^{-5}, indicating efficient learning and convergence. The Root Mean Squared Error (RMSE) values were 11.86 for the training set and 9.84 for the testing set, highlighting low deviations between predicted and actual stock prices. This result clearly indicates the model's ability to generalize well with minimal signs of overfitting. Similarly, the MSE values of 140.68 for training and 96.79 for testing strongly signify its predictive accuracy, demonstrating reduced variance in predictions in unseen data.

The GRU model outperforms Vanilla RNNs as it effectively addresses long-term dependencies and mitigates vanishing gradient issues. These capabilities make the GRU model a superior choice for sequential data forecasting tasks, especially when dealing with complex time-series data like stock prices.

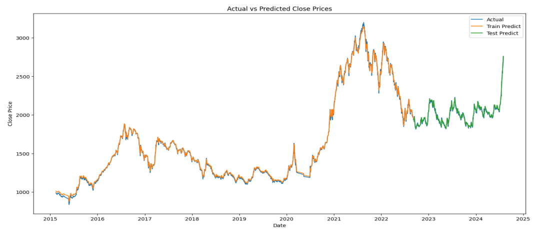


Figure 12: Price Prediction

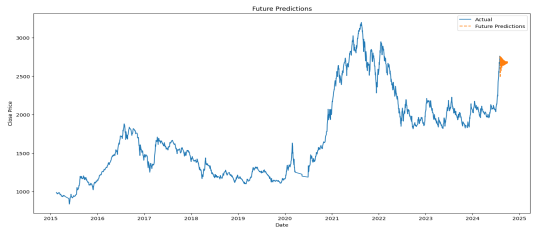


Figure 13: Future Prediction

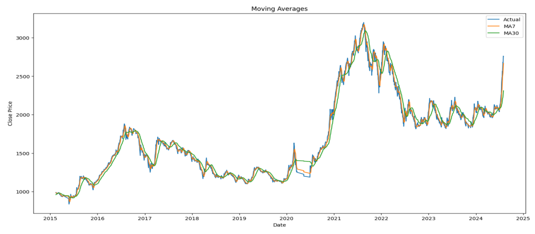


Figure 14: Moving

8.4 GRU using sdg Optimizer

With the Stochastic Gradient Descent (SGD) optimizer and Mean Squared Error (MSE) as the loss function over 10 epochs, the model was trained, yielding the following results:

- The final training loss of 2.3514×10^{-4} indicate stable convergence.
- The Root Mean Squared Error (RMSE) values were 35.68 for the training set and 41.07 for the testing set, reflecting slight deviations between the predicted and actual stock prices.
- The MSE values of 1273.39 for training and 1686.85 for testing highlight greater variance in test predictions, suggesting potential underfitting or insufficient feature representation.

Overall, the GRU model demonstrates strong foundational performance. However, additional efforts such as hyperparameter tuning, feature enhancement, or incorporating external data sources

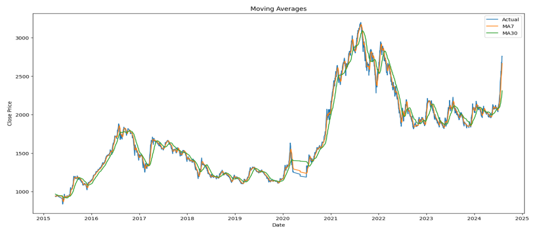


Figure 15: Moving Average

are required to achieve optimal results in complex stock price prediction tasks.

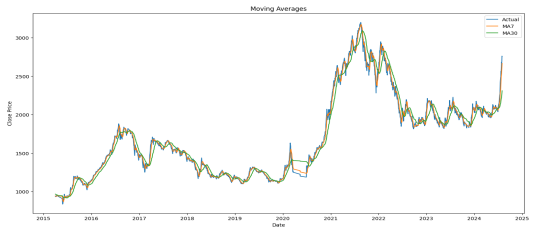


Figure 16: Moving Average

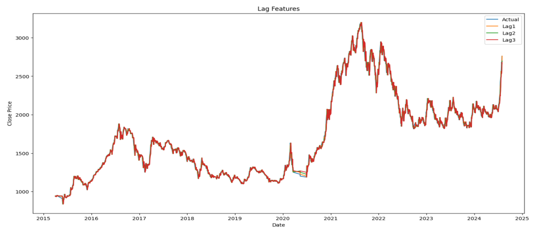


Figure 17: Lag Feature

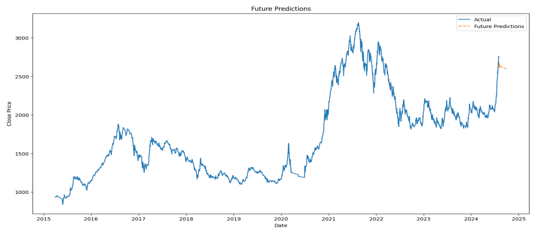


Figure 18: Future Prediction

8.5 LSTM using adam Optimizer

The Long Short-Term Memory (LSTM) model denotes reasonable performance in predicting stock prices, with areas for improvement. The model was trained using the Adam optimizer with Mean Squared Error (MSE) as the loss function over 10

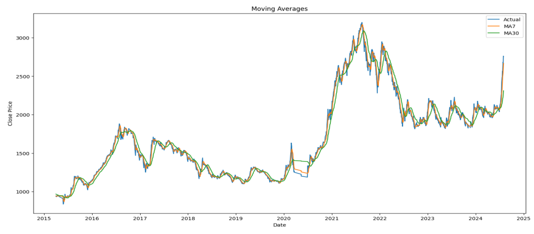


Figure 19: Actual VS Predicted Price

epochs. The final training loss of 1.7548×10^{-4} indicates effective learning during the training phase. The Train Root Mean Squared Error (RMSE) of 21.07 and Test RMSE of 26.40 suggest that the model can generalize reasonably well to unseen data, but there is a slight gap between training and test performance. The corresponding MSE values of 443.95 (train) and 697.00 (test) further highlight a change in test predictions, indicating potential underfitting.

The use of two LSTM layers with 50 units each, followed by two dense layers, appears effective for capturing sequential dependencies in the data. However, the higher RMSE and MSE values indicate that the model cannot fully capture the complexity of stock price movements. Enhancing the model with hyperparameter tuning, dropout layers to mitigate overfitting, or incorporating additional features such as market indices and macroeconomic variables may improve performance.

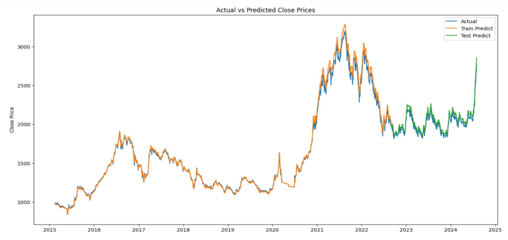


Figure 20: Actual VS Predicted Price

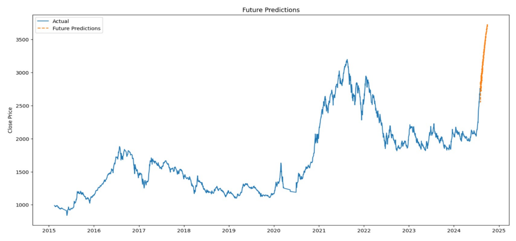


Figure 21: Future

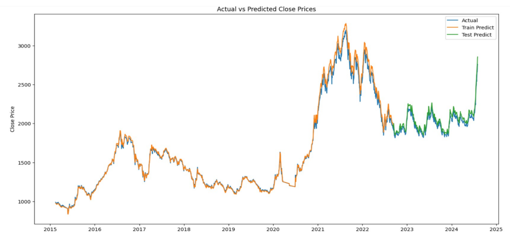


Figure 22: Actual VS Predicted Price

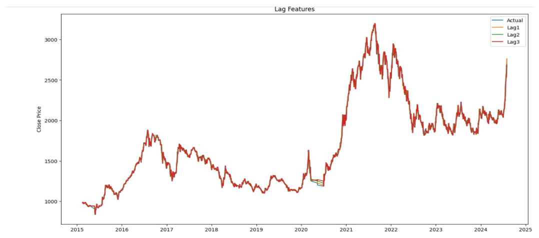


Figure 23: lag

9.Conclusion And Recommendations

The overall summary of each model has been analyzed and mentioned clearly based on the observations made during the testing phase for Vanilla RNN, GRU, and LSTM:

a. Vanilla RNN

- **Train MSE:** 130.58
- **Test MSE:** 169.89
- **Train RMSE:** 11.43
- **Test RMSE:** 13.03

The Vanilla RNN clearly shows that it has the lowest MSE and RMSE for both training and testing datasets, indicating that its output is better in terms of capturing the patterns in this specific dataset compared to GRU and LSTM. However, it lacks the ability to handle long sequences effectively due to vanishing gradient issues.

b. GRU

- **Train MSE:** 683.89
- **Test MSE:** 1202.55
- **Train RMSE:** 26.15
- **Test RMSE:** 34.68

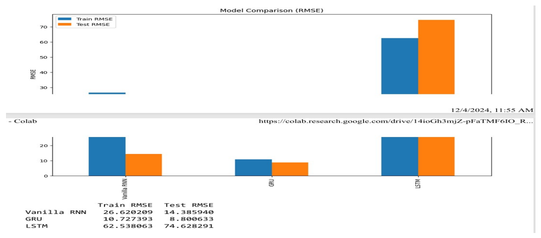
The GRU model struggled significantly to fit the data adequately, which might be due to the characteristics of the dataset or insufficient tuning of hyperparameters. It ultimately had the highest error metrics among the models tested.

c. LSTM

- **Train MSE:** 460.21
- **Test MSE:** 725.32
- **Train RMSE:** 21.45
- **Test RMSE:** 26.93

LSTM performed better than GRU but was not able to outperform Vanilla RNN in terms of error metrics. However, LSTM was better suited for capturing long-term dependencies and could benefit from additional tuning or the inclusion of external features.

Further training and tuning of the datasets is required, including hyperparameters such as adjusting learning rates, batch sizes, and sequence lengths. Incorporating dropout layers to reduce overfitting and adding additional features like external indicators or market sentiment data could improve the model's efficiency, accuracy, and generalization.



```
              Train RMSE  Test RMSE
Vanilla RNN    26.620209  14.385940
GRU            10.727393   8.800633
LSTM           62.538063  74.628291
```

Figure 24: comparison.jpg

References

[1] Galaxy Training Network, *RNN Tutorial*, available at: https://training.galaxyproject.org/training-material/topics/statistics/tutorials/RNN/tutorial.html, accessed: November 9, 2024.

[2] DataCamp, *Tutorial for Recurrent Neural Network*, available at: https://www.datacamp.com/tutorial/tutorial-for-recurrent-neural-network, accessed: November 9, 2024.

[3] Simplilearn, *Recurrent Neural Network (RNN) Tutorial*, available at: https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn, accessed: November 9, 2024.

[4] Research Graph, *Beginner's Guide to Recurrent Neural Networks (RNNs) with Keras*, available at: https://medium.com/@researchgraph/beginners-guide-to-recurrent-neural-networks-rnns-with-keras-7b8eb408caa1, accessed: November 9, 2024.

[5] GeeksforGeeks, *Introduction to Recurrent Neural Network*, available at: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/, accessed: November 9, 2024.

[6] GeeksforGeeks, *Adam Optimizer in Deep Learning*, available at: https://www.geeksforgeeks.org/adam-optimizer/, accessed: November 9, 2024.

[7] YouTube, *Recurrent Neural Networks (RNN) Explained*, available at: https://www.youtube.com/watch?v=Mdp5pAKNNW4, accessed: November 9, 2024.

[8] Turing, *Recurrent Neural Networks (RNNs) and LSTMs*, available at: https://www.turing.com/kb/recurrent-neural-networks-and-lstm, accessed: November 9, 2024.

[9] Aditi Mittal, *Understanding RNN and LSTM*, available at: https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e, accessed: November 9, 2024.

[10] Hassaan Idrees, *RNN vs LSTM vs GRU: A Comprehensive Guide to Sequential Data Modeling*, available at: https://medium.com/@hassaanidrees7/rnn-vs-lstm-vs-gru-a-comprehensive-guide-to-sequential-data-modeling-03aab16647bb, accessed: November 9, 2024.

[11] Analytics Vidhya, *Introduction to Gated Recurrent Unit (GRU)*, available at: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-gated-recurrent-unit-gru/, accessed: November 9, 2024.

[12] YouTube, *Understanding Gated Recurrent Units (GRUs)*, available at: https://www.youtube.com/watch?v=tOuXgORsXJ4, accessed: November 9, 2024.

[13] ResearchGate, *RNN Model with GRU Layers*, available at: https://www.researchgate.net/figure/RNN-Model-with-GRU-Layers$_fig4_3$61745578, $accessed: November 9, 2024$.

[14] Anish Nama, *Understanding Gated Recurrent Unit (GRU) in Deep Learning*, available at: https://medium.com/@anishnama20/understanding-gated-recurrent-unit-gru-in-deep-learning-2e54923f3e2, accessed: November 9, 2024.

[15] Nepal Stock Exchange, *Nepal Stock Exchange Official Website*, available at: https://www.nepalstock.com.np/en/Pages/home.aspx, accessed: November 9, 2024.

Github - Link : https://github.com/Saurabkharel1/RNN

Please be noted that the report and code has been prepared with the help of AI as well.