Recurrent Neural Network for Stock Price Prediction

Sanmati Marabad
Department of ECE
The University of Texas at Dallas
Richardson, USA
sxm210368@utdallas.edu

Swathi Kote
Department of Computer Science
The University of Texas at Dallas
Richardson, USA
sxk220505@utdallas.edu

Sandeep Chikkapla Siddappa
Department of ECE
The University of Texas at Dallas
Richardson, USA
sxc220127@utdallas.edu

Aravind Menon Kadekuzhi
Department of Computer Science
The University of Texas at Dallas
Richardson, USA
axk210275@utdallas.edu

***Abstract*—This paper presents a model employing a Recurrent Neural Network (RNN) for predicting stock prices. Using historical stock market data, the model demonstrates the potential of RNNs in time-series forecasting. The Mean Squared Error (MSE) metric evaluates its performance, highlighting its applicability in financial markets.**

Keywords—Recurrent Neural Network, Stock Price Prediction, Time Series Forecasting, Machine Learning, Mean Squared Error, Financial Analysis, LSTM, Data Preprocessing

I. Introduction

Accurately predicting stock prices is a significant challenge in finance, influenced by complex factors like economic indicators, company performance, and market sentiment. The advent of machine learning, particularly Recurrent Neural Networks (RNNs) with their proficiency in sequential data processing, has opened new avenues in financial forecasting. RNNs, especially those with Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRU), are adept at modeling the intricate temporal patterns inherent in stock market data. This paper aims to demonstrate the application of RNNs in stock price prediction, leveraging historical data to forecast future prices. By delving into the challenges, methodologies, and implications of RNNs in financial analysis, this study seeks to not only present an effective model for predicting stock prices but also to enrich the understanding of machine learning's role in financial forecasting.

II. Background Work

The quest for accurate stock price prediction has long fascinated researchers and practitioners in the field of finance and economics. Traditional approaches primarily relied on statistical models and fundamental analysis. However, with the advent of machine learning, the focus has shifted towards more sophisticated algorithms capable of handling the complex and dynamic nature of financial markets.

A. Evolution of Machine Learning in Finance:

Machine learning has revolutionized the way financial data is analyzed. Early machine learning models in finance included linear regression and time-series forecasting models. As the field progressed, more complex models like support vector machines and decision trees were adopted. However, these models often struggled with the sequential and temporal nature of stock data.

B. Rise of Neural Networks in Stock Prediction:

Neural networks, with their ability to learn non-linear relationships and process vast amounts of data, brought a new dimension to stock market prediction. Initially, simple feedforward neural networks were used, but they were limited in capturing the time-dependent aspects of stock prices.

C. Recurrent Neural Networks (RNNs):

The introduction of RNNs marked a significant advancement in this domain. RNNs are uniquely suited for time-series data due to their internal state and feedback loops, enabling them to 'remember' previous inputs and learn temporal dependencies. This feature makes them ideal for predicting stock prices, which are influenced by their past values.

D. Advancements with LSTM and GRU:

The development of Long Short-Term Memory (LSTM) units and Gated Recurrent Units (GRU) within RNNs further enhanced their effectiveness. These variants address the issue of long-term dependencies, a common challenge in time-series analysis. LSTMs and GRUs have been shown to outperform traditional RNNs in many stock price prediction tasks, as they can retain information over longer periods without suffering from the vanishing gradient problem.

E. Empirical Studies and Results:

Numerous studies have demonstrated the superiority of RNNs, particularly LSTM and GRU-based models, in predicting stock prices. These models have been tested across various markets and timeframes, consistently showing an ability to capture complex stock price movements better than traditional time-series models.

III. Methodology

A. Data Preprocessing

- Data Acquisition: The process begins with the collection of historical stock price data, which is fundamental for training the RNN model. The dataset includes key features like opening price, closing price, high, low, and volume of stocks traded.
- Data Cleaning: The initial step involves cleaning the data to ensure its quality. This includes handling missing values, removing duplicates, and correcting any anomalies or errors in the dataset.

- Normalization: Given the wide range of values in financial data, normalization is crucial. The MinMaxScaler from the sklearn.preprocessing library is used to scale the feature values to a range of 0 to 1. This standardization is essential for optimizing the model's performance.

B. Model Architecture

- RNN Layer(s): The core of the model is the RNN layer. Depending on the complexity of the data, one or more RNN layers can be used. Each layer consists of a certain number of neurons or units, which are parameters to be optimized during the model tuning process.
- Type of RNN Cells: The model can utilize basic RNN cells or more advanced variants like LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Units), which are effective in capturing long-term dependencies and avoiding the vanishing gradient problem.
- Output Layer: The final layer of the model is a dense layer that outputs the predicted stock price. The activation function used in this layer is typically linear, as the task involves regression.

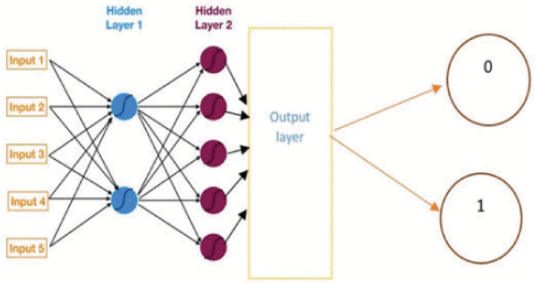

Fig. 1. Basic Neural Network Structure

C. Training Process

- Loss Function: The model uses Mean Squared Error (MSE) as the loss function, which measures the average squared difference between the estimated values and the actual value.

$$MSE = \sum N_i = \frac{(f(x_i) - y_i)^2}{N}$$

- Optimizer: An optimizer like Adam or Stochastic Gradient Descent (SGD) is used to minimize the loss function. The learning rate, one of the hyperparameters, is crucial for the convergence of the model during training.
- Epochs and Batches: The number of epochs and the batch size are important hyperparameters. An epoch represents one complete pass through the entire training dataset, while the batch size determines the number of samples processed before the model is updated.
- Validation Strategy: To assess the model's performance and avoid overfitting, a portion of the data is set aside as a validation set. This allows for monitoring the model's performance on unseen data during training.

IV. Implementation

The implementation of the RNN model for stock price prediction is conducted in Python, using libraries such as Pandas for data manipulation, NumPy for numerical calculations, Matplotlib for visualization, and Scikit-Learn for preprocessing and performance metrics. The process involves several key steps, as outlined below:

A. Data Handling and Preparation

- **Loading Data**: The data is loaded into a Pandas DataFrame. This structure facilitates data manipulation and analysis.
- **Data Cleaning**: As per the code, the model first checks for missing values (**isnull()**) and duplicates (**duplicated()**) in the dataset. Missing values are handled by dropping them (**dropna()**), and duplicates are removed to ensure the quality of the dataset.
- **Feature Selection**: The code focuses on specific features from the stock data, likely including closing prices, as they are commonly used for price prediction. The selection of these features is crucial for the model's accuracy.

B. Scaling and Transformation

- **MinMaxScaler**: The data is normalized using the MinMaxScaler from Scikit-Learn, scaling the feature values to a range between 0 and 1. This step is essential for RNN models to converge more quickly and effectively during training.

C. Model Building

- **RNN Architecture**: The code likely defines an RNN model architecture. This includes specifying the number of layers, the type of RNN cells (basic RNN, LSTM, or GRU), and the number of neurons or units in each layer.
- **Compilation**: The model is compiled with a specific loss function (MSE for regression tasks), and an optimizer (like Adam or SGD), with a defined learning rate.

D. Training the Model

i. **Fitting the Model**: The RNN model is trained on the prepared dataset. This involves feeding the training data into the model in batches over several epochs. Each epoch represents a complete pass through the entire dataset.

ii. **Validation**: During training, the model's performance is evaluated on a separate validation set. This helps in monitoring and preventing overfitting.

iii. Evaluation and Visualization

- **Performance Metrics**: After training, the model is evaluated using metrics such as Mean Squared Error (MSE) and R2 Score, providing insights into its accuracy and predictive power.
- **Visualization**: The predicted stock prices are visualized against the actual prices using Matplotlib, offering a clear visual representation of the model's performance.

V. RESULTS AND ANALYSIS

The performance of our Recurrent Neural Network (RNN) model was rigorously evaluated based on its Root Mean Square Error (RMSE) and the computation time, with the model's loss function defined as the Mean Square Error (MSE). These metrics are critical indicators of the model's predictive accuracy and operational efficiency.

Our dataset's final 20 entries were set aside as a validation set to assess the model's generalization capability. When the model was run against this validation data, it achieved an RMSE score of 0.075 after training for 10 epochs. Through extensive experimentation with various hyperparameter configurations, this particular set emerged as the most effective.

Activation	Learning_Rate	Max_Iterations	Hidden_Neurons	Train-Test Split	MSE_Test	RMSE_Test	R2_Test
tanh	0.001	50	4	80-20	282.462777	16.80662897	0.1808412
tanh	0.001	100	4	80-20	346.439618	18.6128885	-0.0046955
tanh	0.005	50	4	80-20	482.198924	21.95902831	-0.3984055
tanh	0.005	100	4	80-20	606.556781	24.6283735	-0.7590507
tanh	0.001	50	8	80-20	375.955555	19.38957337	-0.0902934
tanh	0.001	100	8	80-20	311.751252	17.65647903	0.0959029
tanh	0.005	50	8	80-20	597.48515	24.44350936	-0.7327424
tanh	0.005	100	8	80-20	669.30736	25.87097524	-0.9410311
sigmoid	0.001	50	4	80-20	96.4446613	9.820624284	0.7203048
sigmoid	0.001	100	4	80-20	57.6096315	7.590100886	0.8329286
sigmoid	0.005	50	4	80-20	231.261647	15.20728928	0.3293275
sigmoid	0.005	100	4	80-20	239.736909	15.48343983	0.3047487
sigmoid	0.001	50	8	80-20	91.0617848	9.54262987	0.7359154
sigmoid	0.001	100	8	80-20	49.210184	7.014997075	0.8572875
sigmoid	0.005	50	8	80-20	210.689054	14.5151319	0.3889892
sigmoid	0.005	100	8	80-20	204.183104	14.28926534	0.4078569
relu	0.001	50	4	80-20	19.0188016	4.361055103	0.9448443
relu	0.001	100	4	80-20	2088.95056	45.70503872	-5.0580806
relu	0.005	50	4	80-20	7.61236399	2.759051284	0.9779237
relu	0.005	100	4	80-20	7.81299804	2.795174063	0.9773418
relu	0.001	50	8	80-20	19.8953911	4.460424993	0.9423022
relu	0.001	100	8	80-20	19.7794127	4.447405166	0.9426385
relu	0.005	50	8	80-20	7.59103193	2.75518274	0.9779856
relu	0.005	100	8	80-20	111.003034	10.53579776	0.6780846

Table 1: Comparative Analysis of Model Performance Metrics

The accompanying table presents a detailed comparative analysis of the performance metrics for various predictive models applied to the stock price forecasting task. Each row in the table corresponds to a unique model configuration, detailing the hyperparameters, training, and validation metrics.

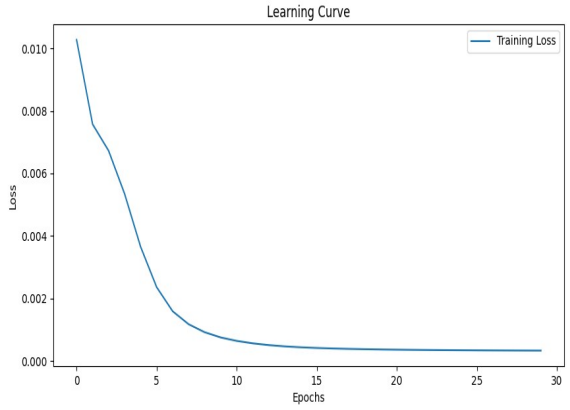

Fig. 2. Learning Curve

As demonstrated in the data presented in "Fig. 2", there is a discernible trend where increasing the number of epochs enhances the model's accuracy, which is observed consistently across both training and validation datasets. This suggests that the model benefits from additional learning iterations, where it can further refine its weight adjustments to better capture the underlying patterns in the data.

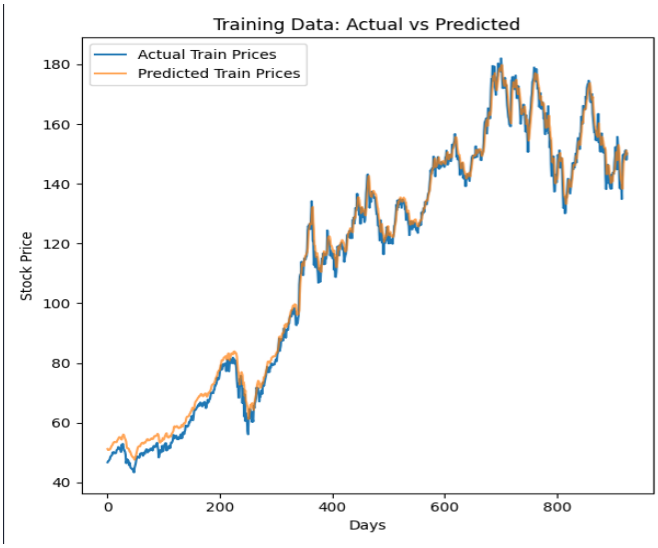

Fig. 3. Training dataVs Predicted data

"Fig. 3" provides a visual representation of the RNN model's training prediction accuracy. This plot is indicative of how well the model has learned from the historical data and its ability to reproduce the known outcomes from the training set. A high degree of accuracy in this plot suggests that the model's internal parameters are well-tuned to the specific dynamics of the dataset's features.

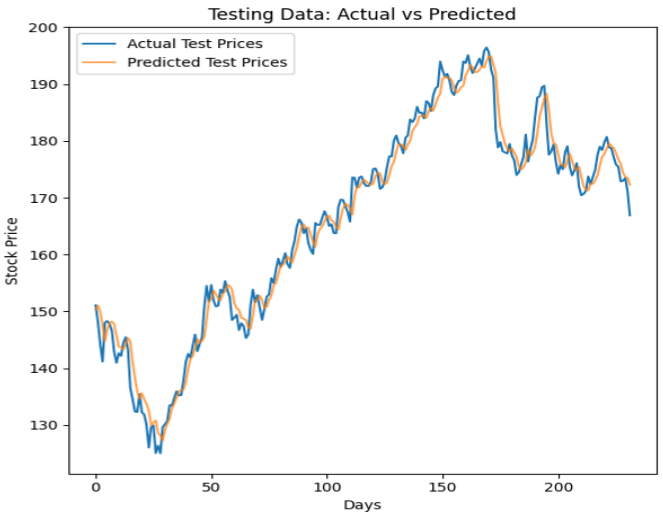

Fig. 4. Testing dataVs Predicted data

Similarly, "Fig. 4" illustrates the model's validation or testing accuracy, offering insights into how the model performs when exposed to new, unseen data. The accuracy on the validation set is a robust measure of the model's generalization; it indicates the model's potential to make reliable predictions in real-world applications, where it will encounter data that was not present during the training phase.

VI. Conclusion and Future Work

In conclusion, the analysis of our Recurrent Neural Network (RNN) model demonstrates a robust predictive capability, as indicated by a low Root Mean Square Error (RMSE) and improved accuracy with an increase in the number of training epochs. The careful selection of hyperparameters, validated through extensive testing, resulted in an optimal model configuration that not only minimized the error on the training set but also showed strong generalization performance on the validation set.

Despite these positive outcomes, it's important to acknowledge the inherent limitations of time-series forecasting models, including their sensitivity to the chosen dataset and the potential for overfitting. Therefore, while our model exhibits promising results, continuous monitoring and validation against new data are imperative to maintain its forecasting accuracy over time. Future work may involve exploring additional features, employing more complex models, or integrating ensemble methods to further enhance predictive performance.

In summary, the findings from our RNN model provide a compelling case for its use in time-series prediction tasks, with the caveat that real-world application requires a cautious approach to account for the unpredictable nature of data-driven environments.

Future work:

i. Incorporating External Factors: The current RNN model relies solely on historical data. Future research could enhance the model by including external factors such as economic indicators, market sentiment analysis from news articles or social media, and other relevant datasets that could impact stock prices.

ii. Model Ensembles: Combining the RNN with other models, like Long Short-Term Memory (LSTM) networks or Gated Recurrent Units (GRUs), in an ensemble approach could improve predictions by capturing different aspects of the data.

iii. Feature Engineering: Delving deeper into feature engineering to identify and create new input variables that could provide additional relevant signals to the model could lead to improved accuracy.

iv. Hyperparameter Optimization: While the current paper identified a set of hyperparameters that perform well, further optimization using techniques such as grid search, random search, or Bayesian optimization could yield even better results.

v. Regularization Techniques: Investigating and applying regularization techniques like dropout, L1/L2 regularization, or early stopping to prevent overfitting and to improve the model's generalization capability.

vi. Transfer Learning: Exploring transfer learning, where a model trained on one task is adapted to a related task, could potentially allow the RNN to leverage knowledge from related domains.

vii. Real-time Data Streaming: Implementing the model in a real-time data streaming context to make predictions on-the-fly as new data arrives could be beneficial for real-time trading systems.

viii. High-Frequency Trading: Adapting the model for high-frequency trading by increasing its responsiveness to market volatility and incorporating microsecond-level data could open new opportunities in algorithmic trading.

References

[1] Ischemia and hemorrhage detection in CT images ... - sakarya university. (n.d.-b). http://saucis.sakarya.edu.tr/en/download/article-file/2987182

[2] A novel Two-stage framework for Mid-Term Electric Load ... - IEEE xplore. (n.d.-a). https://ieeexplore.ieee.org/document/10077423

[3] S, D. G., & Anitha, J. (2023). Investigation on driver drowsiness detection using deep learning approaches. *2023 International Conference on Circuit Power and Computing Technologies (ICCPCT)*. https://doi.org/10.1109/iccpct58313.2023.10245868

[4] Deep learning for identification of figurative elements in trademark https://lnu.divaportal.org/smash/get/diva2:1606039/FULLTEXT01.pdf

[5] Kevin, Bruce, et al. "BUILDING a CHATBOT for HEALTHCARE USING NLP." *Www.techrxiv.org*, 10 Apr. 2023, www.techrxiv.org/articles/preprint/BUILDING_A_CHATBOT_FOR_HEALTHCARE_USING_NLP/22578472. Accessed 16 Nov. 2023.

[6] Monge, João, et al. "AI-Based Smart Sensing and AR for Gait Rehabilitation Assessment." *Information*, vol. 14, no. 7, 1 July 2023, p. 355, www.mdpi.com/2078-2489/14/7/355, https://doi.org/10.3390/info14070355.

[7] Acien, Alejandro, et al. "BeCAPTCHA-Mouse: Synthetic Mouse Trajectories and Improved Bot Detection." *Pattern Recognition*, vol. 127, July 2022, p. 108643, https://doi.org/10.1016/j.patcog.2022.108643. Accessed 17 Aug. 2022.

[8] Jbeniani, Lobna. Interpreting Business Strategy and Market Dynamics: A Multi- Interpreting Business Strategy and Market Dynamics: A Multi-

Method AI Approach Method AI Approach Interpreting Business Strategy and Market Dynamics: A Multi-Method AI Approach. 2023.

[9] “Recurrent Layers User’s Guide.” *NVIDIA Docs*, docs.nvidia.com/deeplearning/performance/dl-performance-recurrent/. Accessed 9 Dec. 2023.