Predicting NVIDIA Stock Prices Trends with Recurrent Neural Networks

1st Kuei-Yu Tsai
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas, USA
kxt230002@utdallas.edu

2nd Yu-Chen Lin
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas, USA
yxl220044@utdallas.edu

3rd Shiyou Yan
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas, USA
sxy180032@utdallas.edu

4th Yijie Tang
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas, USA
yxt140930@utdallas.edu

5th Nguyen Hoang
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas, USA
hdn220000@utdallas.edu

***Abstract*—This paper presents a comprehensive exploration of recurrent neural networks (RNNs) applied to the task of predicting NVIDIA stock price trends. A unique RNN model is constructed from scratch, delving into the underlying theories, particularly focusing on recurrent connections and hidden states. The back-propagation through time algorithm for training and the architecture of the customized RNN are elucidated. Through a rigorous analysis of benchmark tasks, the effectiveness of the model in capturing sequential patterns, particularly in the context of NVIDIA stock prices, is demonstrated. The insights gained contribute to a deeper understanding of RNNs, furthering the field of deep learning and sequential data analysis.**

Keywords—Deep Learning, Neural Networks, Prediction, Back Propagation, Sequential Pattern

I. Introduction

Because of their cyclic connections that can retain information over many time steps, Recurrent Neural Networks (RNNs) have become the most effective way to process sequential data. In this research, we will build a brand new model step by step so we could know what really happens underneath these concepts and how they work. We want to emphasize that RNN is recurrent and hidden states preserve history information. What we are trying to do is dig deep into mathematical background knowledge as well as algorithmic design principles behind RNNs' behavior. It was starting from scratch at building this RNN model therefore there are insights about its architecture and training procedures which cannot be gained anywhere else except here — Long Short Term Memory (LSTM)[1] or Gated Recurrent Unit(GRU) networks [2] represent more complicated variants of RNNs that could be understood only after knowing everything about them given by our own creation according to [1]…[2]. This study does not use any existing libraries deliberately so that people can better understand what makes up an RNN and how it can be used in real-world situations.

Building on this investigation, the present study explores financial forecasting, with a specific emphasis on utilizing RNNs to predict trends in NVIDIA stock prices. Our method involves painstakingly building a custom RNN model that is adapted to the subtleties of stock price prediction. We want to clarify the complex inner workings of our RNN-based forecasting system by exploring the underlying theories, with a focus on recurrent connections and the function of hidden states. In addition, we give a thorough explanation of the back-propagation through time algorithm that we used to train our customized RNN and outline the architectural details that make it capable of recognizing the sequential patterns present in stock price data.

We extensively experiment and thoroughly analyze benchmark jobs to show that our model is effective in accurately predicting trends in NVIDIA stock prices. We demonstrate the usefulness of RNNs in the field of stock price prediction by carefully examining how well it captures sequential patterns in financial data. The knowledge gained from this project advances sequential data analysis and deep learning, which in turn helps improve financial markets predictive analytics. It also advances our understanding of RNNs.

II. Related Work

Research in the field of Recurrent Neural Networks (RNN) for time series prediction extends beyond stock market forecasting to include various applications in weather forecasting and energy consumption forecasting.

A. Weather Forecasting

In the domain of weather forecasting, Recurrent Neural Networks (RNN) for time series rely on features like air temperature, atmospheric pressure, humidity, and others as input to the predictive model. These features are structured into sequences, where each data point represents the weather conditions at a specific time interval, such as hourly or daily. In this setup, historical weather data sequences are fed into the RNN model as input, while the output consists of predicted weather conditions for future time steps. This approach enables RNNs to learn patterns and dependencies in the data over time, allowing for accurate predictions of future weather conditions.

For instance, Edward Appau Nketiah and his team [3] explored multivariate time series models based on RNN for forecasting atmospheric temperature. To address the challenges of overfitting and underfitting, they used Ridge Regularizer to prevent overfitting and underfitting, and utilized Bayesian Optimization along with Long Short-Term Model (LSTM) modeling of data sequences to improve its performance, ultimately achieving superior accuracy in weather prediction.

B. *Energy Consumption Forecasting*

Recurrent Neural Networks (RNNs) are also highly effective in energy consumption prediction and management, especially for time series data. They can take a sequence of past energy consumption values and predict the next one, thereby helping in optimizing energy distribution. Architectural approaches such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Unit (GRU) networks have been particularly successful in this domain. To support this viewpoint, Ibtissam Amalou presents various architectural approaches, with an emphasis on RNNs, especially LSTM and GRU networks, for the usage of different home appliances. Among these evaluation indicators such as RMSE, MAE and R2_score, the GRU network has shown to be particularly effective, offering the best predicted results for smart grids.

III. Methodlogy

A. *Data Preprocessing*

The first crucial step in our methodology involves preparing the NVIDIA stock data for effective modeling with RNNs. We start by collecting daily stock prices, including the open, close values, and trading volume. Given the importance of capturing trends over time, we preprocess this data to reflect percentage changes rather than absolute values, which helps in normalizing the scale of fluctuations across different periods.

Normalization using the StandardScaler that transforms the data to have a mean (average) of zero and a standard deviation of one. Each feature's value is subtracted by the mean of the feature and then divided by the standard deviation of the feature. The formula is:

$$z = \frac{x-\mu}{\sigma}$$

This is a crucial step in many machine learning algorithms, especially in neural network. Because it can improve the convergence speed during training and the effectiveness of the model by ensuring that all features contribute equally.

Then we divide the data into training, validation, and testing sets. We apply seed to ensure that the randomness in the data splitting process is reproducible. Different runs with the same seed will produce identical splits, which is important for replicating study results. The dataset is split into three parts: The training set is 70% of the data. The validation set is 15% of the data (85% - 70%). The test set is The remaining 15%.

We initializes the weights and biases for a neural network based on a predefined layer configuration. It employs a method based on the He initialization strategy, adjusting weights with a scaling factor derived from the inverse square root of the number of hidden units in each layer. This helps stabilize training by maintaining consistent variance in the neurons' outputs. The function processes a list of dictionaries, each defining the unit configuration for layers, including the number of previous units, hidden units, and output units. Weights and biases are initialized using a uniform distribution around zero, scaled to maintain balanced activation across the network. The function returns a structured list of these parameters for each layer, facilitating further steps in building and training the neural network.

B. *Training Process*

The training process for neural networks is a critical phase where the model learns to map input data to the correct output using the provided training dataset. There are three weights in this model: U, W and V Matrices represent the weights for input-to-hidden, hidden-to-hidden, and hidden-to-output transformations, respectively. The b, c are bias for hidden and output layers. And, x is input. Perform the forward pass as following:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o^{(t)} = c + Vh^{(t)}$$

Where $a^{(t)}$ is the pre-activation vector at timestep t, $h^{(t)}$ is the hidden state vector after applying the hyperbolic tangent function. where $o^{(t)}$ is the output vector before activation at timestep t.

Backpropagation through time is essential component of training recurrent neural networks. It implements the algorithm to compute the gradients of the network's parameters based on the output error and update these parameters accordingly. The function efficiently handles the complexities of temporal dependencies inherent in sequential data. Update the model's weights using gradient descent to minimize the loss function.

$$\theta \leftarrow \theta - \eta \nabla L$$

η is the learning rate.

θ is any parameter (such as weights or biases).

∇L is the gradient of the loss function L with respect to θ.

L is the Mean Squared Error.

$$L = \frac{1}{n}\sum_{n}(t-o)^2$$

Applying general form to each type of parameter in RNN:

$$U \leftarrow U - \eta \sum_t \left(\frac{\partial L}{\partial h^{(t)}} \otimes x^{(t)}\right)$$
$$W \leftarrow W - \eta \sum_t \left(\frac{\partial L}{\partial h^{(t)}} \otimes h^{(t-1)}\right)$$
$$V \leftarrow V - \eta \sum_t \left(\frac{\partial L}{\partial o^{(t)}} \otimes h^{(t)}\right)$$
$$b \leftarrow b - \eta \sum_t \frac{\partial L}{\partial h^{(t)}}$$
$$c \leftarrow c - \eta \sum_t \frac{\partial L}{\partial o^{(t)}}$$

When $a \otimes b = ab^T$ represent the outer product of two vectors.

Each equation follows the principle that parameters are updated by moving in the direction that minimally decreases the loss function. This is achieved by subtracting a portion of the gradient, scaled by the learning rate, from the current parameter values. The sum over tt indicates the gradients are accumulated over all timesteps, which is crucial for capturing temporal dependencies in sequence processing tasks typical of RNNs. The training process is set to run for 100 epochs, it means that the network will process the entire dataset a total of 100 times.

IV. Results

The performance of each model configuration was evaluated based on two primary metrics: the training loss and the validation loss. Additionally, we computed the Root Mean Square Error (RMSE) on the validation set to provide a measure of the predictive accuracy of the models.

A. *Influence of Hidden Layers*

The number of hidden layers in the RNN architecture significantly influenced model performance. We observed that increasing the number of hidden layers beyond a certain threshold led to diminishing returns, as evidenced by fluctuations in both training and validation losses.

B. *Variations in Model Performance*

Our experiments revealed notable variations in model performance across different hyperparameter configurations. The following table summarizes the key findings:

TABLE I. Summary of Model Performance

Learning Rate	Hidden Layers	Train Loss	Validation Loss	Validation RMSE
0.0001	**7**	**5.62**	**32.89**	**2931.85**
0.0001	**9**	**4.46**	**30.01**	**2861.13**
0.0001	**11**	**9.12**	**24.67**	**1995.80**
0.0001	**13**	**7.86**	**23.78**	**1857.61**

This table presents the model performance metrics for various combinations of learning rates and hidden layer configurations.

C. *Visualization of Results*

The results of our experiments are visualized in Figure I, illustrating the variations in training and validation losses, as well as the validation RMSE, across different hyperparameter configurations. Additionally, we provide an epoch-wise plot of the training and validation losses for the configuration with a learning rate of 0.0001 and 13 hidden layers in Figure II.

Figure I.

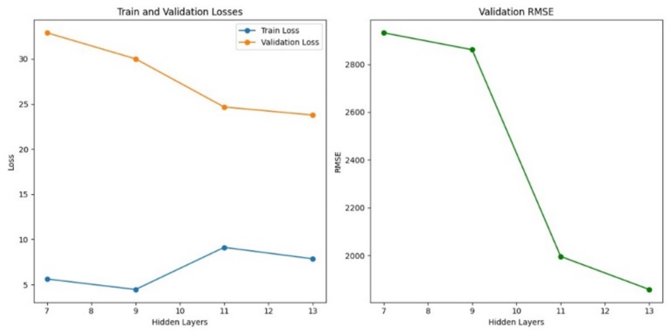
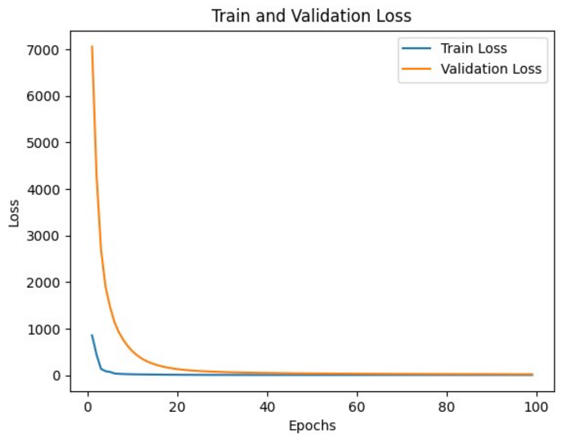


Figure II.



V. Discussion

A. *Key Oberservations from the Table*

As the number of hidden layers increases from 7 to 13, there is a general trend of decreasing training and validation loss. This suggests that deeper models are able to better capture the underlying patterns in the data, potentially leading to more accurate predictions. Notably, the configuration with 13 hidden layers achieves the lowest validation loss (23.78) and training loss (7.86), indicating a robust model performance relative to other tested configurations.

The RMSE follows a declining trend as the number of hidden layers increases. The highest RMSE is observed with 7 hidden layers (2931.85) and the lowest with 13 hidden layers (1857.61). The decrease in RMSE with more layers supports the conclusion that adding layers up to a point significantly enhances the model's predictive accuracy. The reduction in RMSE implies better generalization on the validation set, suggesting that the model with 13 hidden layers not only fits better but also generalizes better than lower configurations.

The data shows diminishing improvements in RMSE with each additional layer beyond 11 layers. While still improving, the rate of improvement in model accuracy slows down, suggesting that simply adding more layers may lead to diminishing returns in terms of loss reduction and accuracy improvement.

The fluctuations in both training and validation losses as the number of layers increases could indicate the onset of overfitting in configurations with fewer layers and underfitting with too many layers, though the latter is less evident in this specific dataset. It's crucial to monitor both losses to ensure that the model does not memorize the training data at the expense of its ability to generalize.

B. *Visual Analysis*

1) *Train and Validation Loss*

Left plot of Figure I displays the training and validation losses as functions of the number of hidden layers in the network. As the number of hidden layers increases from 7 to 13, there is a clear trend of decreasing losses, both training and validation. This suggests that deeper networks are better at capturing the complexity of the dataset and reducing error. The decrease in loss with more hidden layers could indicate that the network is able to learn more abstract representations of the data. However, both lines tend to flatten as the number of layers increases, hinting at a diminishing return on adding more layers beyond 11 or 13.

2) *Validation RMSE*

Right Plot of Figure I shows the Root Mean Square Error on the validation set as a function of the number of hidden layers. There is a sharp decrease in RMSE as the number of layers increases from 7 to 11, after which the decrease in RMSE becomes less pronounced. This further supports the notion of diminishing returns with additional layers. The sharp improvement up to 11 layers indicates significant gains in predictive accuracy, but additional layers beyond this point offer smaller improvements.

3) *Epoch-Wise Train and Validation*

Figure II shows how the training and validation decrease over the number of epochs for the configuration with a learning rate of 0.0001 and 13 hidden layers. Both the training and validation losses drop sharply within the first few epochs and then stabilize. The rapid decrease in losses at the beginning of the training process indicates that the model quickly adapts to the patterns in the data. The stabilization of the losses at a low level suggests that the model has reached a point of convergence, where further training does not significantly change the loss, indicating effective learning and generalization. The close gap between the training and validation losses throughout suggests that the model is not overfitting, which is a positive sign of its ability to generalize well to new data.

C. *Recommendation andOptimization*

1) *Model Complexity and Depth*

The results from Figure I suggest that increasing the number of hidden layers improves model performance up to a point. Careful consideration should be given to the number of layers; beyond a certain point, the benefits in reduced loss and improved accuracy drop off.

2) *Monitoring and Early Stopping*

The insights from Figure II emphasize the importance of monitoring loss during training. Implementing early stopping could prevent overfitting and unnecessary computations once the losses stabilize.

3) *Experimentation with Hyperparameters*

Further experimentation with learning rates, batch sizes, and other architectural features (like different types of RNN cells) could help in fine-tuning the model for even better performance.

4) *Cross-Validation*

To ensure that the model's performance is robust, cross-validation could be employed to validate these findings across different subsets of the data.

5) *Optimal Layer Configuration*

As the analysis suggests diminishing returns beyond 11 to 13 layers, experiment with models within this range to fine-tune the depth for the best performance without unnecessary complexity.

6) *Dynamic Learning Rate*

Implement learning rate schedules such as decay or adaptive learning rate techniques (e.g., Adam, RMSprop) that adjust the learning rate based on the training epochs or loss plateauing. This could enhance the convergence speed and overall training efficiency.

VI. Conclusion

RNNs for time series prediction are not like traditional feedforward and backward artificial neural networks. They can capture temporal dynamics and context, making them ideal for time-series forecasting. In this project, we applied RNNs to predict NVIDIA stock prices from scratch. After normalizing the features and dividing them into a training set, we constructed a model with an input layer of 3 units and 13 hidden RNN layers, with 1 output unit. Then we trained the RNN model on the training data using backpropagation through time to update the network parameters, calculated the mean squared error gradient and evaluated the trained model on the test data by making predictions.

In conclusion, while RNNs still require improvements in handling long-term dependencies and vanishing gradients, advanced architectures like LSTM and GRU are making significant advancements in addressing these challenges. RNNs remain a promising method for time series prediction, offering the potential to capture complex temporal patterns effectively.

Acknowledgment

We would like to express our sincere gratitude to all those who contributed to the completion of this paper. First and foremost, we extend our heartfelt appreciation to our professor Dr. Anurag Nagar and TAs for their invaluable guidance, encouragement, and support throughout the research process.

Additionally, we are grateful to our group members for their work, which greatly enriched the quality of our project.

Lastly, we acknowledge the contributions of all individuals, whether acknowledged here or not, who played a role in the completion of this paper. Their collective efforts are sincerely appreciated.

References

[1] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," in Neural Computation, vol. 9, no. 8, pp. 1735-1780, 15 Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[2] R. Dey and F. M. Salem, "Gate-variants of Gated Recurrent Unit (GRU) neural networks," 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA, USA, 2017, pp. 1597-1600, doi: 10.1109/MWSCAS.2017.8053243.

[3] Nketiah EA, Chenlong L, Yingchuan J, Aram SA. Recurrent neural network modeling of multivariate time series and its application in temperature forecasting. PLoS One. 2023 May 19;18(5):e0285713. doi: 10.1371/journal.pone.0285713.

[4] Amalou, I., Mouhni, N., & Abdali, A. (2022). Multivariate time series prediction by RNN architectures for energy consumption forecasting. Energy Reports, 8(Supplement 9), 1084-1091. ISSN 2352-4847. doi: 10.1016/j.egyr.2022.07.139.