# Recurrent neural network model for time series analysis[*]

Bilal Masih[1,2,*,†]

[1]*Silesian University of Technology, Akademicka 2A, 44-100, Gliwice, Poland*

[2]*University of L'Aquila, Via Vetoio, 40, 67100 Coppito AQ, Italy*

**Abstract**

Time series analysis is a critical component in various fields such as finance, economics, climate science, and healthcare, where accurate forecasting and pattern recognition are paramount. This research explores the application of recurrent neural networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, for time series prediction, using Google stock prices as a case study. The study begins with a comprehensive literature review, highlighting the evolution and advancements in RNN architectures, their theoretical foundations, and diverse applications in time series forecasting. Methodologically, this study outlines the data preprocessing techniques employed, including scaling and partitioning the dataset into training and testing sets. The RNN model architecture is meticulously designed, featuring multiple LSTM layers and dropout regularization to prevent overfitting and enhance model robustness. The model is trained and evaluated using different metrics (MAE, MSE, RMSE). Empirical results demonstrate the efficacy of the RNN model in capturing the temporal dependencies and producing accurate forecasts of stock prices.

**Keywords**

Recurrent neural networks (RNNs), Time series analysis, Long Short-Term Memory (LSTM), Deep learning, Machine learning, Artificial intelligence (AI), Financial forecasting

## 1. Introduction

In the data-driven decision-making landscape, time series analysis is essential for understanding sequential data across diverse domains. Traditionally, statistical methods like autoregressive models, moving averages, and exponential smoothing have been used to analyze time series data. However, these methods often fall short due to assumptions of linearity and stationarity, which are rarely met in real-world datasets characterized by nonlinearity and volatility.

The advent of deep learning, especially recurrent neural networks (RNNs), has transformed time series analysis. RNNs, with their recurrent connections and ability to process variable-length sequences, excel at capturing temporal dependencies in data. This study explores the application of RNNs for predicting Google's stock prices. **The goals of this study** are:

[*] *IVUS2024: Information Society and University Studies 2024, May 17, Kaunas, Lithuania*

[1,*] Corresponding author

[†] These author contributed equally.

bm313809@student.polsl.pl; bilalkhokhar1002@gmail.com (B. Masih)

0009-0008-8649-2435 (B. Masih)

***1.*** *To develop a bespoke recurrent neural network model tailored specifically for time series analysis, with a focus on predicting Google stock prices; and*

***2.*** *To empirically evaluate the performance of the proposed model using real-world financial data.*

The research addresses the demand for reliable predictive models in financial decision-making, considering challenges such as nonlinearity and irregularities in financial data. The study also examines methodological aspects like data preprocessing, model selection, hyperparameter tuning, and evaluation metrics, aiming to elucidate RNNs' strengths and limitations in financial forecasting.

## 2. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed for process- ing sequences of data. Unlike traditional feedforward neural networks, RNNs have connections that form directed cycles, allowing them to maintain a memory of previous inputs. This makes RNNs particularly well-suited for tasks where the order of data points is important, such as time series [1] forecasting, language modeling, and speech recognition.

### 2.1. Architecture of RNNs

The basic architecture of an RNN consists of a set of hidden states that are updated at each time step based on the current input and the previous hidden state. This can be mathematically described as follows:

Let $x_t$ be the input at time step $t$, $h_t$ be the hidden state and $h_{t-1}$ be the previous hidden state at time step $t$, and $y_t$ be the output at time step $t$. The equations governing the RNN are:

$$h_t = f(W_h x_t + U_h h_{t-1} + b_h) \tag{1}$$

$$y_t = W_y h_t + b_y \tag{2}$$

Here $W_h$ and $W_y$ are the weight matrices for the input and output respectively. $U_h$ is the weight matrix for the hidden state, $b_h$ is the bias vector for the hidden state, $b_y$ is the bias vector for the output, $f$ is the activation function, often used for its nonlinear properties.

The below diagram 2.1 illustrates the architecture of a Recurrent Neural Network (RNN). On the left, the compact representation shows a single recurrent unit with input $x$, hidden state $h$, and output $y$. The connections demonstrate how the hidden state $h$ is influenced by the current input $x$, previous hidden state (loop), and contributes to the output $y$.

On the right, the unfolded representation depicts how the RNN processes a sequence of inputs over time steps $t-1$, $t$, and $t+1$. Each time step $t$ has its own input $x_t$, hidden state $h_t$, and output $y_t$. The hidden state $h_t$ is updated by the previous hidden state $h_{t-1}$ and the current input $x_t$. The weight matrices $U$, $V$, and $W$ are shared across all time steps, illustrating the RNN's ability to handle sequential data by maintaining and updating a memory of previous inputs.

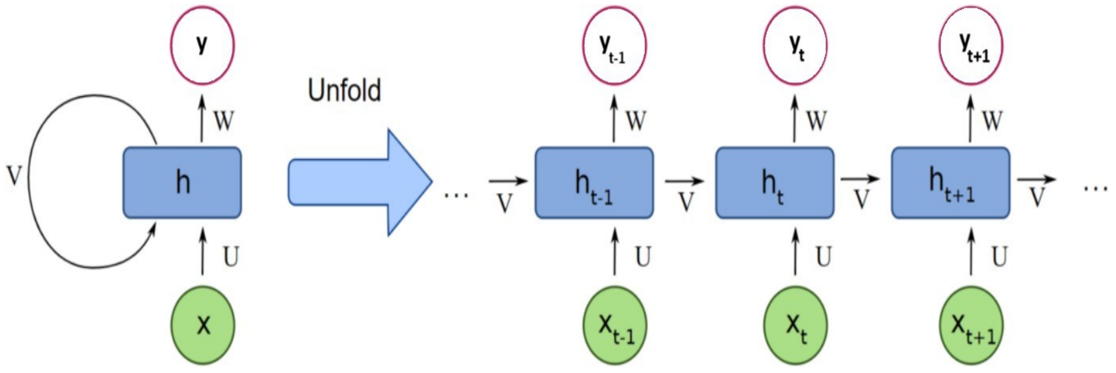

**Figure 2.1:** RNN Architecture. [2]

## 2.2. Training Recurrent Neural Networks

Training an RNN involves adjusting the weights and biases to minimize the error between the predicted output and the actual target. This is typically done using the backpropagation through time (BPTT) algorithm, which is an extension of the backpropagation algorithm used in feedforward neural networks.

**Forward Pass:** During the forward pass, the network processes the input sequence from the first time step to the last, updating the hidden states and producing the outputs. The loss is calculated by comparing the predicted outputs to the actual targets.

Let $\mathscr{L}$ be the loss function. The total loss over a sequence of length $T$ is given by:

$$\mathcal{L} = \sum_{t=1}^{T} \ell(y_t, \hat{y}_t) \tag{3}$$

where $\ell$ is the loss function (i.e. Mean Squared Error) at each time step, $y_t$ is the actual target, and $\hat{y}_t$ is the predicted output. The loss function is given by:

$$\ell(y_t, \hat{y}_t) = \frac{1}{2}(y_t - \hat{y}_t)^2 \tag{4}$$

**Backward Pass (Backpropagation Through Time):** During the backward pass, the gradients of the loss with respect to the weights are calculated by propagating the error backwards through the network. The gradients are then used to update the weights.

The gradients for the weights and biases can be computed as follows:

$$\frac{\partial \mathcal{L}}{\partial W_y} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial y_t} \cdot h_t^T \tag{5}$$

$$\frac{\partial \mathcal{L}}{\partial b_y} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial y_t} \tag{6}$$

For the hidden weights and biases, assuming $f'$ is the derivative of $f$:

$$\frac{\partial \mathcal{L}}{\partial h_t} = \left(\frac{\partial \mathcal{L}}{\partial y_t} \cdot W_y^T\right) + \left(\frac{\partial \mathcal{L}}{\partial h_{t+1}} \cdot U_h^T \cdot f'(W_h x_t + U_h h_{t-1} + b_h)\right) \tag{7}$$

$$\frac{\partial \mathcal{L}}{\partial W_h} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial h_t} \cdot f'(W_h x_t + U_h h_{t-1} + b_h) \cdot x_t^T \tag{8}$$

$$\frac{\partial \mathcal{L}}{\partial U_h} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial h_t} \cdot f'(W_h x_t + U_h h_{t-1} + b_h) \cdot h_{t-1}^T \tag{9}$$

$$\frac{\partial \mathcal{L}}{\partial b_h} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial h_t} \cdot f'(W_h x_t + U_h h_{t-1} + b_h) \tag{10}$$

**Algorithm 1** Training RNN using BPTT

```
 1: Initialize weights and biases: $W_h, U_h, b_h, W_y, b_y$ 2:
 for each training sequence in dataset do
 3:     Initialize loss to 0
 4:     for $t = 1$ to $T$ do
 5:         $h_t \leftarrow f(W_h \cdot x_t + U_h \cdot h_{t-1} + b_h)$
 6:         $y_t \leftarrow W_y \cdot h_t + b_y$
 7:         $loss \leftarrow loss + loss\_function(y_t, \hat{y}_t)$
 8:     end for
 9:     gradients ← $compute\_gradients(loss)$
10:     update_weights_and_biases(gradients)
11: end for
```

Training Recurrent Neural Networks (RNNs) using Backpropagation Through Time (BPTT) involves several steps. Initially, the weights $W_h$, $U_h$, $b_h$, $W_y$, and $b_y$ are randomly initialized. For each training sequence, a forward pass computes hidden states $h_t$ using an activation function (e.g., tanh, ReLU) applied to the current inputs and previous hidden states, weighted by $W_h$ and $U_h$ and shifted by bias $b_h$. Outputs $y_t$ are derived from $h_t$ using $W_y$ and $b_y$. The loss is computed by comparing predicted outputs with target values. A backward pass calculates the gradients of the loss with respect to the weights and biases, propagating the error backward through time. Weights and biases are updated using these gradients via an optimization algorithm like gradient descent. This iterative process refines the network parameters, minimizing loss and enhancing performance.

## 2.3. Long Short-Term Memory (LSTM)

LSTMs are a type of RNN designed to overcome the vanishing gradient problem [3]. They include special units called memory cells to store information over long periods.

An LSTM cell consists of three gates: input gate ($i_t$), forget gate ($f_t$), and output gate ($o_t$). The equations governing the LSTM cell are:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{11}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{12}$$

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c) \tag{13}$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{14}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{15}$$

$$h_t = o_t * \tanh(c_t) \tag{16}$$

where $x_t$ is the input at time $t$, $h_{t-1}$ is the previous hidden state, $c_t$ is the cell state, $\sigma$ is the sigmoid function, and $W$ and $b$ are the weight matrices and bias vectors respectively.

## 3. Time Series Analysis

### 3.1. Introduction to Time Series

A time series is a sequence of data points typically measured at successive points in time, spaced at uniform intervals. Time series analysis involves methods for analyzing time series data to extract meaningful statistics and other characteristics. Time series forecasting is the use of a model to predict future values based on previously observed values. Mathematically, a time series can be represented as:

$$Y(t) = T(t) + S(t) + C(t) + I(t) \tag{17}$$

where $Y(t)$ is the observed value at time $t$, $T(t)$ is the trend component, $S(t)$ is the seasonal component, $C(t)$ is the cyclic component, and $I(t)$ is the irregular component [4].

Components of Time Series: Time series data can be decomposed into several components, each representing an underlying pattern or structure in the data:

*Trend (T):* The long-term progression of the series. It represents the general direction in which the data is moving over a long period.

*Seasonality (S):* The repeating short-term cycle in the data. This is often observed in data with periodic fluctuations, such as monthly sales data.

*Cyclic (C):* Long-term oscillations in the data, typically spanning several years.

*Irregular (I):* The random noise component which cannot be attributed to the other components.

RNNs have found widespread applications in time series analysis, where the objective is to analyze historical data and make predictions about future trends. Some common applications of RNNs in time series analysis include: stock price prediction, weather forecasting, economic modeling, signal processing [5], health monitoring and diagnosis.

## 3.2. Time Series Analysis Methods

Several methods are employed to analyze time series data:

**Moving Average:** The moving average method smooths the data to identify the trend component by averaging adjacent data points. The moving average at time $t$ for a window size $w$ is given by:

$$MA_t = \frac{1}{w} \sum_{i=0}^{w-1} Y_{t-i} \tag{18}$$

**Exponential Smoothing:** Exponential smoothing assigns exponentially decreasing weights to past observations. The simple exponential smoothing forecast for time $t+1$ is:

$$\hat{Y}_{t+1} = \alpha Y_t + (1-\alpha)\hat{Y}_t \tag{19}$$

where $\alpha$ is the smoothing parameter ($0 < \alpha < 1$) [6].

**Autoregressive Integrated Moving Average (ARIMA):** The ARIMA model combines autoregression (AR), differencing (I), and moving average (MA) to model time series data:

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t \tag{20}$$

where $Y_t$ is the differenced series, $c$ is a constant, $\varphi$ represents the autoregressive parameters, $\theta$ represents the moving average parameters, and $\epsilon_t$ is the white noise error term [7].

## 4. Related Work

The theoretical underpinnings of recurrent neural networks trace back to the foundational concepts of artificial neural networks and computational neuroscience. Early research in neural network theory laid the groundwork for understanding the principles of learning, representation, and computation in interconnected networks of artificial neurons [8]. The introduction of recurrent connections endowed neural networks with the ability to process sequential data and capture temporal dependencies, paving the way for the development of RNNs [9].

Over the years, several architectural variants of recurrent neural networks have been proposed to address the challenges of training deep networks and mitigating the issues of vanishing and exploding gradients. Among these variants, Long Short-Term Memory (LSTM) networks and Gated Recurrent Unit (GRU) networks have emerged as prominent choices due to their ability to capture long-range dependencies and facilitate more stable training dynamics [10]. The architectural design principles and computational mechanisms underlying LSTM and GRU networks have been extensively studied, highlighting their strengths and limitations in modeling sequential data.

Recurrent neural networks have found widespread applications in time series analysis, spanning various domains such as finance, economics, climate science, healthcare, and engineering [11]. In the context of financial time series analysis, RNNs have been extensively employed for stock price prediction, market trend forecasting, risk assessment, and algorithmic trading [12]. Similarly, in climate science, RNN-based models have been used for weather forecasting, climate modeling, and environmental monitoring, leveraging the temporal dependencies inherent in meteorological data [13]. Other applications include speech recognition, natural language processing, physiological signal analysis, and anomaly detection [5], where RNNs excel in capturing sequential patterns and extracting meaningful insights from temporal data.

Despite their versatility and effectiveness, recurrent neural networks are not without limitations. Challenges such as the vanishing gradient problem, the curse of dimensionality, overfitting, and computational inefficiency pose significant hurdles in training deep RNN architectures on large-scale datasets [14]. Moreover, the interpretability of RNN-based models remains a concern, as the black-box nature of deep learning algorithms may hinder their adoption in domains where transparency and explainability are paramount.

Looking ahead, several avenues for future research and innovation in the field of RNNs for time series analysis can be identified. These include the development of hybrid architectures integrating RNNs with other deep learning techniques, advancements in optimization algorithms and regularization techniques, and efforts to enhance the interpretability and transparency of RNN-based models [15]. Additionally, exploring applications in emerging domains such as healthcare, cybersecurity, and smart manufacturing holds promise for extending the scope and impact of RNN-based models in real-world scenarios.

## 5. Introduction of the Dataset

The dataset for this study comprises daily Google stock prices from March 21, 2019, to March 20, 2024, sourced from finance.yahoo.com. It includes 1259 observations of five key attributes: Open, High, Low, Close, and Volume, representing different aspects of stock prices and trading volumes. Significant variability and large standard deviations indicate considerable daily fluctuations and market volatility. Quartile values highlight the distribution of stock prices and volumes, showing moderate values with some extreme fluctuations. This dataset is essential for time series analysis and forecasting, aiding in understanding stock behavior and improving prediction accuracy using RNN models.

### 5.1. Correlation Matrix

The correlation matrix provides a numerical summary of the linear relationships between pairs of variables in the dataset: Open, High, Low, Close, and Volume. Each value in the matrix ranges from -1 to 1, where 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation.

**Interpretation:** The prices show extremely high positive correlations, close to 1, indicating they move together. Trading volume has a weak negative correlation with prices, suggesting that higher volumes may slightly correspond to lower prices. This correlation matrix is crucial

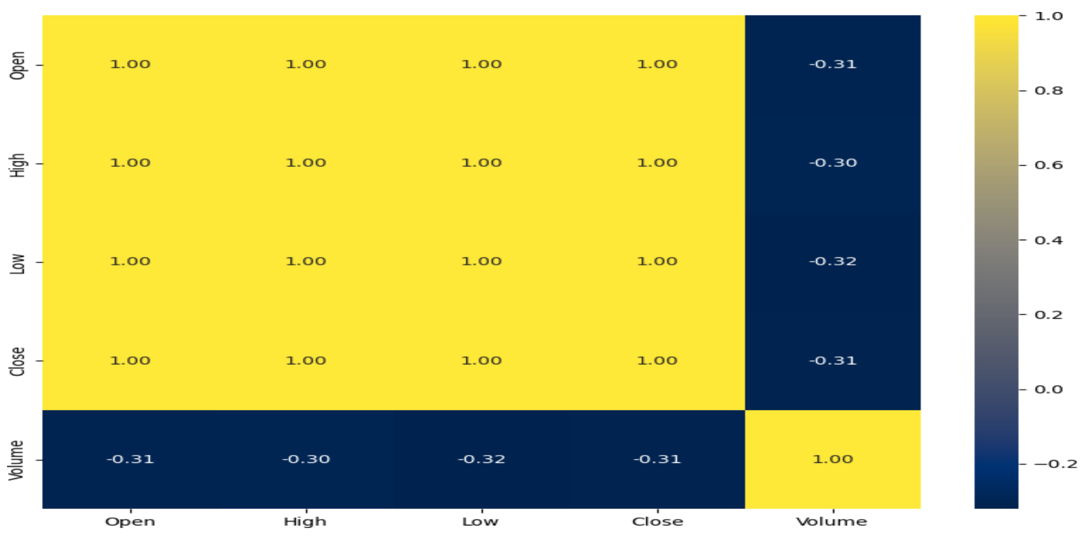

**Figure 5.1:** Correlation Matrix shows strong positive correlations between the price variables and moderately negative correlations between the volume and price variables.

for understanding relationships between stock prices and trading volume, informing predictive modeling and analysis in financial studies, as it highlights typical price movements and potential influences of large trades. [Figure 5.1]

## 5.2. Histogram Plot

The histogram displays the distribution of five variables: Open, High, Low, Close, and Volume. Each histogram provides insight into the frequency and distribution of these values over the observed period. The histogram for the 'Open' prices shows a distribution that ranges from The histogram for 'Open' prices ranges from about 50 to 150, with distinct clusters peaking around 60, 80, 100, and 140, indicating common opening price ranges. 'High' prices follow a similar pattern, suggesting the highest prices during trading often align with opening prices. The 'Low' prices histogram also shows clustering at similar levels, indicating that the lowest prices frequently fell within these groups. 'Close' prices follow the same distribution pattern, with peaks at 60, 80, 100, and 140, suggesting consistency and stability across opening, high, low, and closing prices during trading periods.

The 'Volume' histogram, however, shows a different pattern, ranging from 0 to approximately 1.2e8, with a significant peak around 0.2e8, indicating many trading periods had this volume level. There is a noticeable decrease in frequency as the volume increases, suggesting fewer periods with extremely high trading volumes.[Figure 5.2]

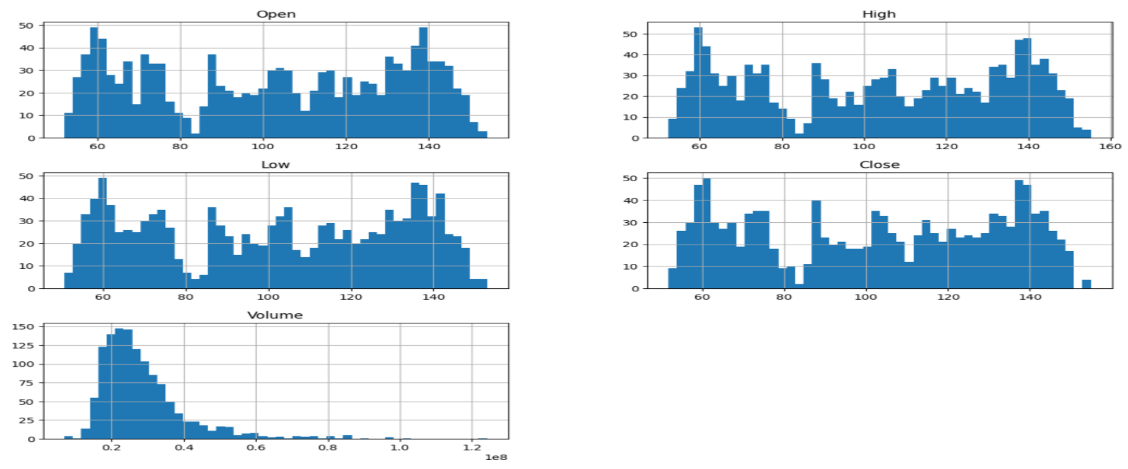

**Figure 5.2:** Histogram Plot of all columns

# 6. Methodology

This chapter outlines the methodology employed to develop and evaluate a Recurrent Neural Network (RNN) for time series analysis, specifically for predicting Google stock prices. The approach involves several key steps: preprocessing, model architecture design, training, and evaluation. Each step is described in detail to provide a comprehensive understanding of the processes involved in this study.

## 6.1. Data Preprocessing

Data preprocessing is crucial for training the RNN model, involving several steps. The dataset is first loaded into a pandas DataFrame for structured manipulation and analysis, followed by inspecting its structure, checking for missing values, and reviewing basic statistics. The data is then split into training (80%) and testing (20%) sets to evaluate the model's performance on unseen data. The 'Open' price is chosen for predicting future stock prices, and Min-Max normalization scales the values between 0 and 1, aiding the RNN model's convergence. Training sequences of 60 time steps are created, with each sequence comprising 'Open' prices for 60 consecutive days and the target being the 'Open' price of the next day, capturing temporal dependencies. Finally, testing sequences are prepared from the combined training and testing 'Open' prices, ensuring continuity and smooth transition in the time series.

## 6.2. Model Architecture and Process

The RNN model is designed using a stacked Long Short-Term Memory (LSTM) network. LSTMs are chosen for their ability to learn long-term dependencies, making them suitable for time series prediction. The model consists of several LSTM layers, each followed by a Dropout layer to prevent overfitting. The final layer is a Dense layer that outputs the predicted stock price.

**LSTM Layers and Dropout:** The Dropout layers, which randomly set a fraction of the input units to 0 during training, prevent overfitting while each LSTM layer records the temporal dependencies in the stock price data.

**Compiling the Model:** The Adam optimizer is used to compile the model, and the MSE (Mean Squared Error) is used as the loss function. The Adam optimizer is chosen for its adaptive learning rate capabilities, which help in faster and more efficient convergence.

**Model Training:** The model is trained with a batch size of 32 across 100 epochs. This choice balances training time with the model's ability to learn from the data effectively.

**Model Evaluation:** The model's performance is assessed using the testing set. Predicted stock prices are compared with actual prices, and the Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) are used to evaluate the accuracy of the model.

**Making Predictions:** Utilizing the testing data, the trained model is employed to make predictions. The predicted prices are then transformed back to their original scale using the inverse of the Min-Max scaler.

## 7. Empirical Analysis

In this study, the performance of recurrent neural networks (RNNs) was analyzed with varying numbers of layers and activation functions to predict time series data for a specific number of input values. The performance metrics used were Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). The activation functions evaluated included no activation key, tanh, ReLU, sigmoid, and combinations thereof. This comprehensive analysis aims to identify the optimal configuration for time series prediction and understand the impact of different network architectures on model performance.

### 7.1. General Observations:

**1.** The best performing models balance simplicity and sufficient depth to capture temporal patterns without overfitting.
**2.** Deep networks with complex activations tend to perform poorly, indicating the need for careful tuning and possibly alternative architectures or regularization techniques to manage deeper models. The issues like exploding gradients can affect deeper networks, particularly those using ReLU and sigmoid activation, which can result in poor model performance.

### 7.2. Impact of Network Depth

**Shallow Networks (4-6 layers):** These generally performed well, especially with no activation key or a combination of no activation key and tanh. This suggests that for this specific time series data, a less complex model is sufficient to capture the necessary patterns.

**Medium Networks (8-15 layers):** Performance begins to degrade as the number of layers increases. For instance, at 10 layers, the errors increase notably, particularly with the ReLU and sigmoid activations.

**Deep Networks (20-100 layers):** The performance significantly worsens with deeper networks. Activation functions like ReLU and sigmoid show particularly high errors, likely due to gradient issues and overfitting.

## 7.3. Impact of Activation Functions

**No Activation Key:** Consistently shows good performance across different network depths, indicating stability and robustness.

**tanh:** Performs well in shallow and medium networks but shows increased errors in very deep networks.

**ReLU and sigmoid:** These activation functions result in poor performance, particularly as the network depth increases. This is likely due to their susceptibility to gradient issues.

**Combinations of Activation Functions:** The combination of no activation key and tanh shows promise in shallow networks but degrades in performance as the network depth increases.

## 7.4. Model Performance Visualization

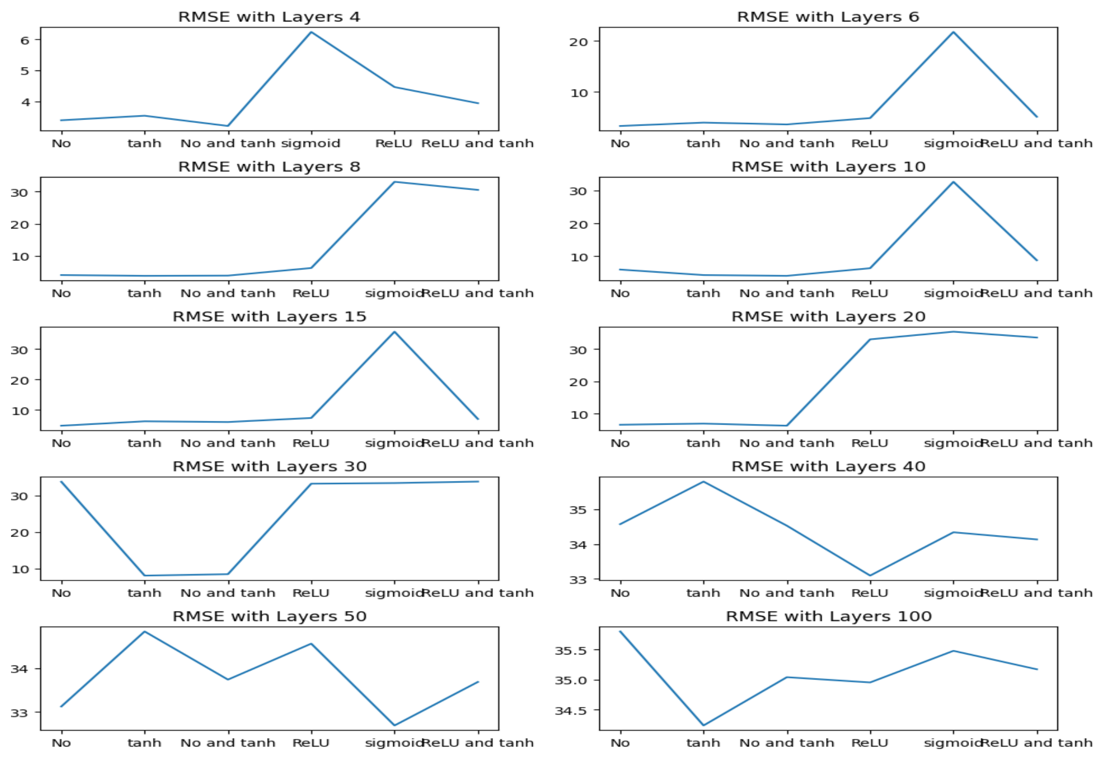

**Figure 7.1:** Impact of Layer Variation

The figure 7.1 shows the Root Mean Squared Error (RMSE) of a Recurrent Neural Network (RNN) model with different numbers of layers and various activation functions. Each subplot represents the RMSE for a specific number of layers, ranging from 4 to 100 layers, as indicated by the titles of the subplots. The x-axis of each subplot lists different activation functions used in the RNN layers, and the y-axis indicates the RMSE value corresponding to each activation function.

Similarly, we can see the activation key wise visualization of the results in figure 7.2. The figure shows the Root Mean Squared Error (RMSE) of a Recurrent Neural Network (RNN) model with different number of layers and various activation functions. Each subplot represents the

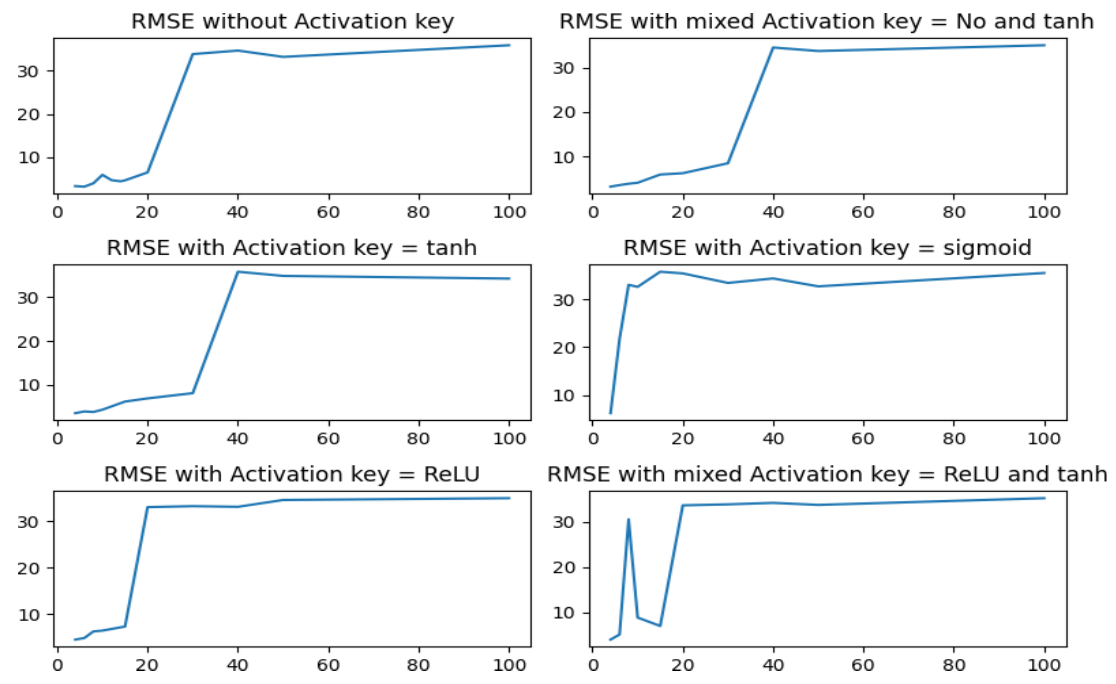

**Figure 7.2:** Impact of Activation Key Variation

RMSE for specific activation function keys, ranging from no activation key to ReLU and tanh activation keys, as indicated by the titles of the subplots. The x-axis of each subplot lists different layers used in the RNN model, and the y-axis indicates the RMSE value corresponding to each activation function.

**Best Performance:** The best performance in terms of the lowest MAE, MSE, and RMSE was observed with a 4-layer RNN using a combination of no activation key and tanh activation key. This configuration provides MAE: 2.532684078, MSE: 10.31147237, RMSE: 3.21114814. This indicates that a relatively shallow network with mixed activation functions can effectively capture the temporal dependencies in the data without overfitting. [Figure 7.3]

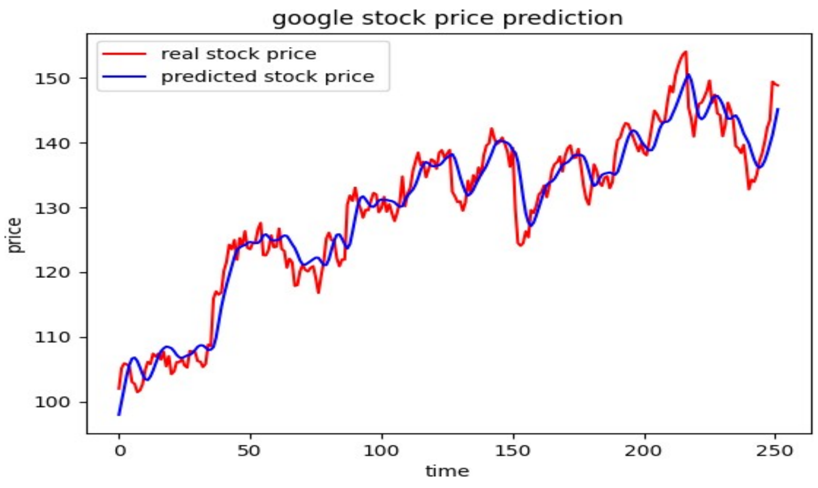

**Figure 7.3:** Model Prediction with 4 LSTM Layers with Mixed Activation “No and tanh". Here mixed activation means that half of the hidden layers are using activation function “tanh" and the rest are not using any activation key.

## 8. Conclusion

This study explored the application of recurrent neural networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, for time series prediction using Google stock prices as a case study. The study comprehensively evaluated different RNN configurations, varying the number of network layers and activation functions, to determine the optimal setup for accurate forecasting.

The empirical analysis revealed that the best performance was achieved with a 4-layer RNN using a combination of no activation key and tanh activation function. This configuration produced the lowest Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE), indicating its effectiveness in capturing temporal dependencies in the data. Conversely, the worst performance was observed with a 100-layer RNN without using any activation function, which highlighted issues such as gradient explosion and overfitting in deeper networks.

The findings underscore the importance of network depth and activation function selection in RNN-based time series analysis. Shallow networks with appropriate activation functions can effectively model temporal data, while deeper networks require careful tuning to avoid performance degradation.

## 9. Acknowledgments

I would like to express my deepest gratitude to my supervisor, Professor Marcin Woźniak, for his invaluable guidance, encouragement, and support throughout the course of this study. His profound knowledge and expertise have been instrumental in shaping my research and providing me with the direction needed to navigate through various challenges. Professor

Woźniak's insightful feedback and constructive criticism have significantly enhanced the quality of this work. His patience and willingness to share his time and knowledge have been greatly appreciated.

## References

[1] J. Siłka, M. Wieczorek, M. Woźniak, Recurrent neural network model for high-speed train vibration prediction from time series, Neural Computing and Applications 34 (2022) 13305–13318.

[2] A brief overview of recurrent neural networks (rnn), 2024. URL: https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/, accessed on June 14, 2024.

[3] M. Woźniak, M. Wieczorek, J. Siłka, Bilstm deep neural network model for imbalanced medical data of iot systems, Future Generation Computer Systems 141 (2023) 489–499.

[4] D. C. Montgomery, C. L. Jennings, M. Kulahci, Introduction to Time Series Analysis and Forecasting, John Wiley & Sons, 2015.

[5] M. Woźniak, J. Siłka, M. Wieczorek, M. Alrashoud, Recurrent neural network model for iot and networking malware threat detection, IEEE Transactions on Industrial Informatics 17 (2020) 5583–5594.

[6] R. G. Brown, Smoothing, Forecasting and Prediction of Discrete Time Series, Prentice-Hall, 1963.

[7] G. E. Box, G. M. Jenkins, G. C. Reinsel, G. M. Ljung, Time series analysis: forecasting and control, John Wiley & Sons, 2015.

[8] A. N. Ranganathan, K. Reddy, Time series forecasting using lstm neural networks, in: Recent Trends in Image Processing and Pattern Recognition, Springer, Singapore, 2021, pp. 129–135.

[9] Z. C. Lipton, J. Berkowitz, C. Elkan, A critical review of recurrent neural networks for sequence learning, arXiv preprint arXiv:1506.00019 (2015).

[10] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735–1780.

[11] Q. Ke, X. Jing, M. Woźniak, S. Xu, Y. Liang, J. Zheng, Apgvae: Adaptive disentangled representation learning with the graph-based structure information, Information Sciences 657 (2024) 119903.

[12] X. Zhang, X. Wu, Time series prediction using a deep learning model based on lstm network, in: Journal of Physics: Conference Series, volume 1529, 2020, p. 032035.

[13] N. Boulanger-Lewandowski, Y. Bengio, P. Vincent, Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription, arXiv preprint arXiv:1206.6392 (2012).

[14] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: International Conference on Machine Learning, 2013, pp. 1310–1318.

[15] Q. Yao, Q. Hu, X. Zhao, Recent advances in deep learning for time series forecasting, IEEE Transactions on Neural Networks and Learning Systems 32 (2020) 3730–3751.