computation

MDPI

Article

Sliding Window-Based Randomized K-Fold Dynamic ANN for Next-Day Stock Trend Forecasting

Jaykumar Ishvarbhai Prajapati and Raja Das *

Department of Mathematics, School of Advanced Sciences, Vellore Institute of Technology, Vellore 632014, India; jaykumar.ishvarbhai2022@vitstudent.ac.in
* Correspondence: rajadas@vit.ac.in

Abstract: The integration of machine learning and stock forecasting is attracting increased curiosity owing to its growing significance. This paper presents two main areas of study: predicting pattern trends for the next day and forecasting opening and closing prices using a new method that adds a dynamic hidden layer to artificial neural networks and employs a unique random *k*-fold cross-validation to enhance prediction accuracy and improve training. To validate the model, we are considering APPLE, GOOGLE, and AMAZON stock data. As a result, low root mean squared error (1.7208) and mean absolute error (0.9892) in both training and validation phases demonstrate the robust predictive performance of the dynamic ANN model. Furthermore, high R-values indicated a strong correlation between the experimental data and proposed model estimates.

Keywords: deep learning; forecasting; artificial neural network; stock price prediction; machine learning

check for updates

Academic Editors: Antonio Martínez Raya and Manuel Uche-Soria

Received: 17 April 2025
Revised: 22 May 2025
Accepted: 3 June 2025
Published: 8 June 2025

Citation: Prajapati, J.I.; Das, R. Sliding Window-Based Randomized K-Fold Dynamic ANN for Next-Day Stock Trend Forecasting. *Computation* **2025**, *13*, 141. https://doi.org/10.3390/computation13060141

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Trend theory in stock price forecasting is a crucial technical analysis method primarily associated with the price and time domains. A stock trend refers to a specific configuration observed in a stock's time series chart that signifies a potential price trend. In [1], over 50 extensively recognized stock patterns have been analyzed, and the findings suggest that each stock trend possesses a significant possibility of achieving a forecast price.

In this study, we propose a dynamic artificial neural network (ANN) model integrated with a randomized *k*-fold cross-validation and adaptive sliding *K*-day window technique for predicting future stock price trends. The dynamic structure allows the model to vary the number of hidden-layer nodes for each window shift during training, enabling it to autonomously capture underlying patterns in the data. We focus on forecasting the opening and closing prices for the next day and extend the model to predict a sequence of 10 future days using input parameters from historical stock data.

The experimental results show that the proposed dynamic ANN model achieves high predictive accuracy for three major companies, Apple, Google, and Amazon, with rmse and mae metrics reported across training, validation, and overall windows. The model performs competitively compared to conventional architectures such as LSTM and GRU and demonstrates strong adaptability across different window sizes (K = 30 to 80). This supports its generalizability across market conditions and stocks from different sectors.

The motivation behind this approach lies in the limitations of static ANN architectures, which often require manual tuning of parameters. ANNs, unlike traditional linear or nonlinear forecasting models, can automatically learn complex patterns without prior

assumptions [2–5]. However, optimal network design (for example, the number of nodes in the hidden layer) remains challenging. To address this, our proposed architecture incorporates both a sliding window approach and randomized *k*-fold validation for enhanced generalization and robustness.

The primary contributions of this work are as follows:

- Providing comprehensive mathematical details regarding the Levenberg–Marquardt (L-M) algorithm for its implementation in the proposed model.
- Introducing a novel randomized *k*-fold cross-validation methodology.
- Developing a new ANN model that incorporates a variable number of nodes in the hidden layer with each *K*-days window shift.
- Forecasting the opening and closing prices for the under-researched stocks, along with the correct price trend.
- Assessing and analyzing the efficacy of the proposed dynamic model in comparison to established advanced models, using performance metrics such as rmse, mae, and R-value.

The paper is segmented into multiple sections: Section 2 provides the research findings based on the literature survey. Section 3 provides a comprehensive explanation of the L-M algorithm: both mathematical and practical implementation approaches and explanation of the revolutionary random *k*-fold validation combined with the *K*-day sliding window technique to develop the adaptive neural network construction with the selection of the best system parameters. Section 4 establishes the multiple outcomes and analysis of research work. Section 5 contains a discussion of the study with the practical feasibility of deploying the model. Section 6 reflects the conclusion of the study. Lastly, Appendix A features multiple appendices, containing derivative formulas essential for a comprehensive understanding of the L-M approach.

2. Literature Survey

Buying in equities is the most enticing means of participating in the stock market due to the tremendous rewards; nevertheless, it also carries the extreme trouble of experiencing significant losses. For example, the past performance for the well-known NASDAQ-100 technology index from its inception in 1985-86 shows that successive losing years are extremely rare; in 2022, it fell 33%, followed by a 46% recovery in 2023. Several financial institutions predicted that the index could increase and would go up by 23–24% in 2024 [6,7]. Predicting stock prices is a crucial and sought-after endeavor for investors in the stock market; however, it is also the most difficult challenge due to the volatile nature of stock prices [8]. Consequently, accurate forecasting of stock prices allows investors to enhance returns and mitigate losses [9].

Generally, the stock market is analyzed and predicted using four different methods: technical analysis, fundamental analysis, conventional time series and forecasting techniques, and soft computing approaches. Technical analysis serves as a method for predicting future stock market trends by examining historical time series data on stock prices. Fundamental analysis entails examining a range of metrics related to companies, focusing on aspects that reflect their overall growth, including gain and loss statements, balance sheets, annual reports, and more [10]. Conventional statistical forecasting methodologies encompass time-based models that predict future stock prices by analyzing past stock values using different autoregressive models like ES [11], ARIMA [12], ARCH, and GARCH [13]. Stock price time series data exhibit intricate nonlinear dynamics, significant noise, and dynamic chaos [14]. Machine learning predictions can enhance the efficacy of major regression techniques twofold [15]. In [16], the authors examine the efficacy of deep neural network (DNN) models in forecasting the stock premium. Stock price time

series data exhibit intricate nonlinear dynamics, significant noise, and dynamic chaos [17]. Consequently, ordinary methods for forecasting time series cannot effectively capture the intricate nonlinear and non-stationary dynamics of stock markets [18]. The implementation of machine learning-based methodologies in addressing challenges across various research and industrial domains, including optical networks [19], image segmentation [20], internet of things [21], healthcare [22], and logistics [23], has demonstrated superior performance compared to conventional techniques.

Neural network applications possess a fundamental characteristic: a network must undergo training to address a particular problem. The process of iteratively updating the bias and weight in ANNs is referred to as the learning or training of ANNs. Training ANNs involves reducing the total error function, such as the average of squared error, which quantifies the discrepancy between actual and forecasted values averaged across all training samples. Various training algorithms are directly developed from the original back-propagation approach. The back-propagation technique computes and iterates propagated gradients from the output layer of the network to the input layer until the difference between the actual and forecasted values is minimized or other stopping criteria, such as the number of iterations or total number of epochs, are executed [24]. In addition, there are more intricate algorithms that use Newton's approach, namely the Levenberg–Marquardt (L-M) algorithm, which originally appeared in [25]. It is possible to employ the L-M algorithm with any feedforward neural network since it is a supervised training approach that combines the benefits of the Gauss–Newton method and the gradient descent method [26].

Various neural network training methods such as, the classic L-M algorithm can also become stuck in a local minimum. Secondly, if there are in excess of three layers in the network, the vanishing gradient problem occurs [20]. Classical first-order approaches can address this issue by utilizing the momentum component. This improvement facilitates surpassing local minima and identifying the correct trajectory toward the ideal solution. The momentum factor may be chosen freely and maintained constant throughout the training or be dynamically modified according to the convergence process. This method was described in [27], and the main idea was to combine the good things about the L-M and Conjugate gradient (C-G) approaches to make the training more reliable. In, refs. [28–32], authors endeavored to create two forms of the momentum L-M algorithm, using both fixed and variable momentum sizes. The proposed algorithms were better at cutting down on training time than the traditional L-M algorithm.

As the utilization of ANNs for stock series prediction gains traction, researchers have commenced investigations into its application in stock trend recognition. The study [33] came up with a method to recognize triangular patterns using a recurrent neural network, while [34] developed a way to extract important points and used a multi-output artificial neural network to find stock trends. In [35,36], researchers have used the 10-fold cross-validation approach to validate ANNs and their corresponding models for forecasting diverse trends, including short-term and long-term predictions. Another avenue of investigation involves the segmentation of time series data. In [37], a fixed-length window was employed to segment the time series into subsequences. Then, the time series was shown using the beginning shape patterns that developed [38].

In [39], researchers use a backpropagation ANN with a sliding window method to boost learning over time and make the models more reliable for index forecasting. In [40], the study examines multiple studies that use sliding window approaches inside soft computing models, especially artificial neural networks, to capture short-term fluctuations in the market. In [41], researchers input time-dependent data into several ML models, including ANNs, for directional stock forecasts using a sliding window methodology. The

recently created and more popular deep learning networks, LSTM and GRU, outperform ANNs in terms of accuracy [32,42–45]. This enhanced performance is especially noticeable in applications that use sequential data, such as deep learning and time series prediction. The primary distinction between the shifting window approach and the conventional method is that the former partitions the data into smaller window segments for training and validation, rather than utilizing the entire dataset simultaneously. Subsequently, the previously trained model will be employed to again train and validate the next window segment. Each window segment will be evaluated to forecast stock prices for a certain period. To forecast unseen stock data, the optimum window segment has been finalized after multiple iterations. The issue of significant short-term price fluctuations is effectively managed through the shifting window technique, which facilitates the retraining and updating of the model with each subsequent data prediction.

Problem Statement

Regrettably, all the aforementioned studies employed static methodologies, and the input data consisted of long-term sequences. Static approaches are unsuitable for stock investment due to the time-consuming nature of stock time series, where long-term sequences incur significant computational costs in dynamic methods. Furthermore, the model will consistently incorporate the initial parameters, although their impact on prediction decreases over time. So, from the extensive literature survey, we propose a dynamic methodology aimed at identifying recently formed stock pattern trends, enabling executable investment decisions based on stock series patterns and forecasting the opening and closing prices with limited data.

3. Methodology

3.1. Mathematical Derivation of Levenberg–Marquardt Algorithm

The derivation of the L-M algorithm [25] is contingent upon the prior derivation of the following algorithms: the steepest descent technique (see Appendix A.1), Newton's technique (see Appendix A.2), and the Gauss–Newton technique (see Appendix A.3).

Before deriving the L-M algorithm, we define multiple frequently utilized indices. Let p denote the sequence index, which varies from 1 to $\mathscr{P}$, with $\mathscr{P}$ indicating the total number of samples. m signifies the output index, ranging from 1 to $\mathscr{M}$, with $\mathscr{M}$ indicating the total number of outputs. For weight, consider i and j indices, spanning from 1 to $\mathscr{N}$, where $\mathscr{N}$ denotes the total number of weights, and lastly, k indicates the iteration index.

The error (loss) function has been determined to assess the training procedure. We calculate the total error function for all training sequences and network outputs as:

$$L(x, w) = \frac{1}{2} \sum_{p=1}^{\mathscr{P}} \sum_{m=1}^{\mathscr{M}} l_{p,m}^2 \tag{1}$$

where x represents the input vector, w denotes the weight vector, and $l_{p,m}$ signifies the training error at output m while using the sequence p, defined as

$$l_{p,m} = y_{p,m} - \hat{y}_{p,m} \tag{2}$$

where $y_{p,m}$ is the desired output vector and $\hat{y}_{p,m}$ is the actual output vector.

Levenberg–Marquardt Algorithm

From the Appendix A, to ensure the invertibility of the estimated the Hessian matrix $\mathbb{J}^T\mathbb{J}$, the L-M method offers an alternative estimated of the Hessian matrix:

$$\mathscr{H} \approx \mathbb{J}^T\mathbb{J} + \mu I \tag{3}$$

where μ is a value greater than zero, referred to as the combination coefficient, and I is the identity matrix.

The elements of the approximated matrix $\mathscr{H}$ that are on the main diagonal will be greater than zero, as shown in Equation (3). We can be sure that matrix $\mathscr{H}$ is always invertible with this approximation (3). The revised L–M method can be expressed through incorporating Equations (A21) and (3):

$$w_{k+1} = w_k - (\mathbb{J}_k^T\mathbb{J}_k + \mu I)^{-1}\mathbb{J}_k^T l_k \tag{4}$$

The L–M method is an adaptation of the S-D technique and the Gauss–Newton technique. During training, it changes between the two algorithms. Equation (4) is getting close to Equation (A21) and the Gauss–Newton technique is used when the combination coefficient is very small or almost zero. Equation (4) gets close to Equation (A2) when the combination coefficient is significantly big, and so the S-D technique is employed.

A large μ in Equation (4) can be considered analogous to the learning coefficient in the S-D technique (A2):

$$\beta = \frac{1}{\mu} \tag{5}$$

The method is iteratively run until the loss factor decreases to a minimal threshold number, at which time the desired result and actual output are nearly equal. However, it is crucial to emphasize that the network must be trained with a large enough number of samples and iterations to obtain the desired outcomes.

Choosing the μ Value in Levenberg–Marquardt Algorithm: The μ parameter in the L-M method regulates the equilibrium between the S-D and Gauss–Newton techniques. A larger μ value produces a step approaching the S-D technique, while a minimal μ value yields a step closer to the Gauss–Newton technique.

Here, we are defining the strategies for choosing the μ value:

1. Initial value: Start with a moderate value: A common starting point is $\mu = 1$. Adjust based on problem characteristics: If the problem is known to be highly nonlinear, a larger initial value might be appropriate.
2. Adaptive Updates:
   (i) Increase μ if the step size is not improving: If the sum of squared residuals increases after a step, it indicates that the step size could be too large. Increase μ to take a smaller and more cautious step.
   (ii) Decrease μ if the step size is consistently improving: If the sum of squared residuals decreases consistently after multiple steps, it suggests that the step size is appropriate. Decrease μ to allow for larger steps.

3.2. Data Collection

This research utilizes datasets from three distinct companies across various industries, specifically AMAZON, APPLE, and GOOGLE, which are listed on the NASDAQ Stock Market, United States of America (U.S.A) and included in the S&P 500 Index, covering the period from 1 January 2022 to 31 December 2022. To ensure the reliability and validity of the proposed dynamic model, we have considered complementary data from the Yahoo Finance website (https://finance.yahoo.com/) accessed on 2 March 2025 for the

stocks AMAZON, APPLE, and GOOGLE over a period of 251 days (the market remained closed on holidays and festivals) on a daily basis, consisting of five individual parameters: opening price, closing price (adjusted closing price), day's high price, day's low price, and volume. To perform this experiment, we simulate data and produce graphs in the MATLAB R2024a environment.

3.3. Normalization

Numerous machine learning algorithms endeavor to identify patterns within datasets through the comparison of data point attributes. Nevertheless, a problem arises when the characteristics exhibit significant disparities in their scales. Normalization encompasses a range of strategies used for data processing, especially when the data comes from different scales. If normalization is not applied, it may lead to errors in evaluation or inappropriate conclusions. So, for that we are using Min-Max normalization. This technique involves the process of normalizing the input dataset by fitting it within a predetermined boundary, specifically within a predefined period. The values of numeric variables are within the range of 0 to 1.

To normalize variable $\tilde{x_i}$ into x_i, the formula of Min-Max normalization is given by

$$x_i = \frac{(1-0)(\tilde{x_i} - \tilde{x_{min}})}{(\tilde{x_{max}} - \tilde{x_{min}})} + 0$$

and so,

$$x_i = \frac{(\tilde{x_i} - \tilde{x_{min}})}{(\tilde{x_{max}} - \tilde{x_{min}})}. \tag{6}$$

where $\tilde{x_{max}}$ = Maximum of $\tilde{x_i}$ among dataset, $\tilde{x_{min}}$ = Minimum of $\tilde{x_i}$ among dataset. All elements of each row are considered to be non-equal, and $\tilde{x_i}$ has only finite real values. (If $\tilde{x_{max}} = \tilde{x_{min}}$ or if $\tilde{x_{max}}$ of $\tilde{x_{min}}$ are not finite, then $x_i = \tilde{x_i}$ and there is no change.)

In this study, before feeding input variables to the neural network, we normalize all the input variables into the range [0, 1].

3.4. Artificial Neural Networks

To enhance the understanding of forward and backward calculations, let us consider the two-layer ANN as shown in Figure 1, having an input layer with x_i ($i = 1, 2, \ldots, n$ represents the i-th input neuron). The hidden layer with h_j ($j = 1, 2, \ldots, m$ represents the j-th hidden neuron), the sigmoid activation function $f_h(\cdot)$, and output layer with y_o ($o = 1, 2, \ldots, l$ represents the o-th output neuron) and linear activation function $f_o(\cdot)$. Where weight-and-bias notations can be understood as follows: w_{ij} is the weight connecting input neuron i to hidden neuron j, b_j is the bias of hidden neuron j, v_{jo} is the weight connecting hidden neuron j to output neuron o, and c_o is the bias of output neuron o. For a given sequence, $n = 25$, $l = 2$, and m are variable between 1 and 30.

Our problem can be described as follows: The input layer of the ANN consists twenty-five variables of the stocks' opening price, closing price, volume, daily high, and daily low price for five consecutive days. The variables x_1, x_2, x_3, x_4, and x_5 represent inputs for the day t; x_6, x_7, x_8, x_9, and x_{10} denote inputs for the day $t-1$; x_{11}, x_{12}, x_{13}, x_{14}, and x_{15} correspond to inputs for the day $t-2$; x_{16}, x_{17}, x_{18}, x_{19}, and x_{20} signify inputs for the day $t-3$; and x_{21}, x_{22}, x_{23}, x_{24}, and x_{25} indicate inputs for the day $t-4$.

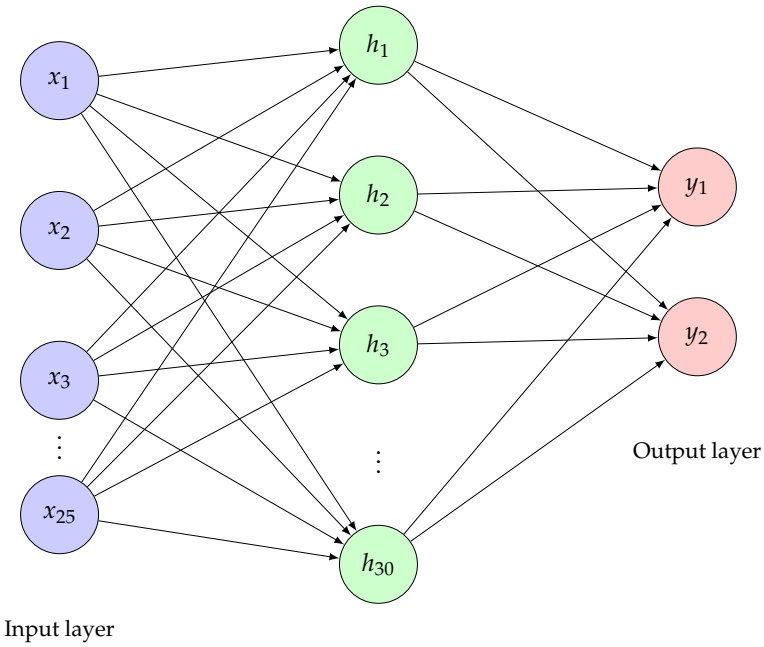


Figure 1. ANN architecture with 25 input nodes in input layer, variable number of nodes from h_1 to h_{30} in the hidden layer, and two output prediction nodes in the output layer.

The forward calculation may be structured as follows:

(a) Compute the net value at the *jth* hidden node and outputs for all nodes in the hidden layer:

$$z_j = \sum_{i=1}^{n} w_{ij} x_i + b_j \tag{7}$$

$$h_j = f_h(z_j) = \frac{1}{1 + e^{-z_j}}. \tag{8}$$

So,

$$\begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{pmatrix} = f_h \left(\begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,25} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,25} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,25} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{25} \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \right) \tag{9}$$

(b) Utilize the first-layer neurons' outputs as the inputs for every neuron in the output layer, calculate the net values for the *kth* output node in a similar manner, and output from them can be expressed as:

$$o_k = \sum_{j=1}^{m} v_{jo} h_j + c_o \tag{10}$$

$$y_k = f_o(o_o) \tag{11}$$

Therefore,

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = f_o \left(\begin{pmatrix} v_{1,1} & v_{1,2} & \ldots & v_{m,1} \\ v_{2,1} & v_{2,2} & \ldots & v_{m,2} \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{pmatrix} + \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \right) \tag{12}$$

(c) Calculate the loss function which is defined in Equations (1) and (2). After the forward calculation, The Jacobian, along with the loss vector and the damping parameter μ, is used in the L-M update rule to calculate how much to adjust each weight in the network.
The backward calculation may be structured as follows:

(d) The matrix $\mathbb{J}$ has dimensions $(\mathscr{P} \cdot \mathscr{M}) \times \mathscr{N}$, where $\mathscr{P}$ is the number of training sequences, $\mathscr{M}$ is the number of output neurons, and $\mathscr{N}$ is the total number of biases and weights in the network. Therefore, for the $\mathscr{p}th$ training sequence with the $\mathscr{m}th$ output neuron and the $\mathscr{n}th$ weight/bias, the entry in the matrix $\mathbb{J}$ will be:

$$\frac{\partial l_{\mathscr{p},\mathscr{m}}}{\partial w_{\mathscr{N}}} = \frac{\partial l_{\mathscr{p},\mathscr{m}}}{\partial y_m} \frac{\partial y_m}{\partial s_m} \frac{\partial s_m}{\partial w_{\mathscr{N}}} \tag{13}$$

where $l_{\mathscr{p},\mathscr{m}}$ is the loss vector for the $\mathscr{p}th$ sequence and the $\mathscr{m}th$ output neuron, y_m is the output of the $\mathscr{m}th$ output neuron, s_m is the weighted sum of inputs to the $\mathscr{m}th$ output neuron.

(e) Back-propagation from the output layer to the hidden layer: If w_n is a weight connecting the hidden layer to the output layer (v_{jk}), then:

$$\frac{\partial s_m}{\partial v_{jo}} = h_j \text{ (output for the } jth \text{ hidden node)} \tag{14}$$

Back-propagation from the hidden layer to the input layer: If w_n is a weight connecting the input layer to the hidden layer (w_{ij}), then:

$$\frac{\partial s_m}{\partial w_{ij}} = \frac{\partial s_m}{\partial h_j} \frac{\partial h_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} \tag{15}$$

$$= v_{jk} f_h'(z_j) x_i \tag{16}$$

(f) After the calculating the back-propagation at each layer, the L-M update:

$$\Delta w = -(\mathbb{J}^T \mathbb{J} + \mu I)^{-1} \mathbb{J}^T l \tag{17}$$

where Δw is the vector of weight-and-bias updates, $\mathbb{J}$ is the Jacobian matrix (with dimensions $P \cdot M \times N$), μ is the damping parameter, I is the identity matrix, and l is the vector of errors $(t_{\mathscr{p},\mathscr{m}} - y_{\mathscr{p},\mathscr{m}})$ for all training patterns p and output neurons $\mathscr{m}$.

(g) Update the rule for new weights and biases as:

$$w = w + \Delta w, b = b + \Delta b.$$

The stages (e)–(g) are repeated for backward propagation. Then, the related parts of the matrix $\mathbb{J}$ can be found using Equation (13). By performing the forward and backward calculations as mentioned above, we can compute the entire matrix $\mathbb{J}$ for other sequences.

3.5. Training Process Design

If the loss value decreases, which means it is smaller than the last loss value, it implies that the quadratic approximation on the total loss function is working and the combination coefficient μ could be reduced to reduce the influence of the gradient descent part (ready to speed up). If the loss value grows, indicating it surpasses the previous loss value, it is essential to adhere more closely to the gradient to identify an appropriate curvature for quadratic estimation, necessitating an increase in the value of μ.

Therefore, the training process using the L–M algorithm is as per the following steps:

1. Assess the total loss value using the first obtained random weights.
2. Execute an update as specified by Equation (4) to modify weights.
3. Utilize the updated weights to compute the overall loss value.
4. If the updated total loss value increases, reverse the step (restore the weight vector to its prior value) and boost μ by a factor of 10. Proceed to step 2 and attempt the update once again.
5. If the updated total loss value is lower, accept the step by retaining the new weight vector as the current one and reduce μ by a factor of 10.
6. Proceed to step 2 with the updated weights until the current overall loss value is less than the specified threshold.
7. For regularization: We keep an eye on validation performance, and if the loss function stagnates or gets worse over the sequence of six iterations, we halt the training to prevent becoming stuck at local minima.

The flow chart for the forward and backward calculations, as well as the training process, is illustrated in Figure 2.

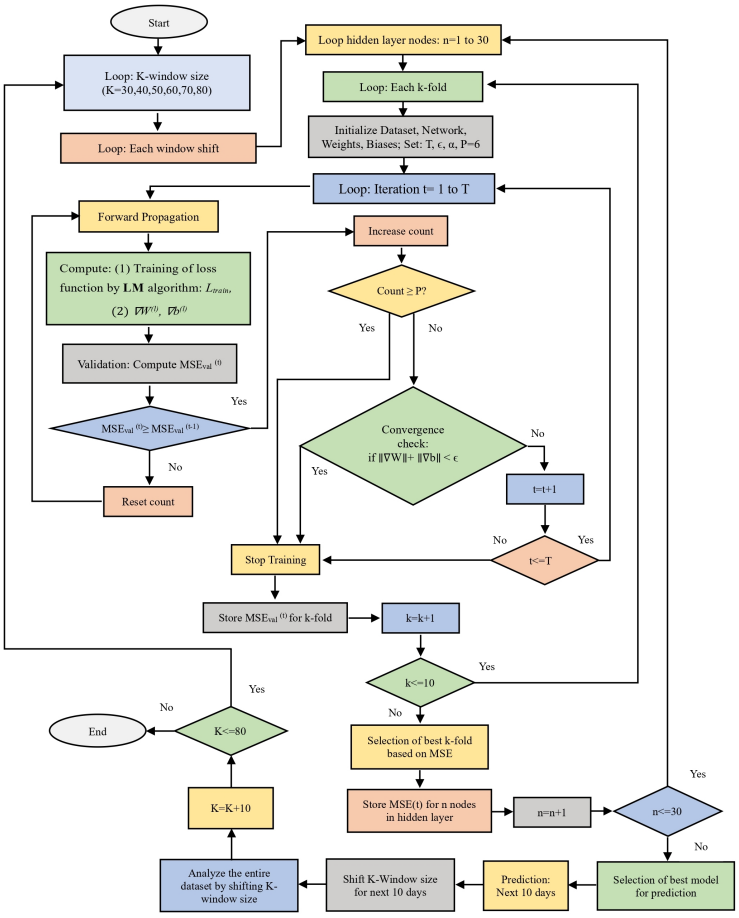


Figure 2. Flowchart of the dynamic ANN.

3.6. Random k-Fold Cross-Validation

The technique of multi *k*-fold validation is used to evaluate predictive models. It is a resampling technique that evaluates and trains a model over several iterations using distinct data folds. Figure 3a illustrates the division of the dataset into *k*-folds, or subsets of equal size. The model is trained and analyzed *k* times, each time with a different fold as the validation set. This method facilitates model evaluation, selection, and hyperparameter tuning by providing a more solid measure of a model's effectiveness. Previous research has employed *k*-fold cross-validation to avoid overfitting. The dataset is divided into two

parts: 80% for training the model and 20% for validating it. This improves the algorithm's generalization capabilities in various aspects. But, for the stock market data, traditional *k*-fold cross-validation is not useful, as standard *k*-fold randomly splits a big portion of the data, which violates the time order and can cause data leakage.

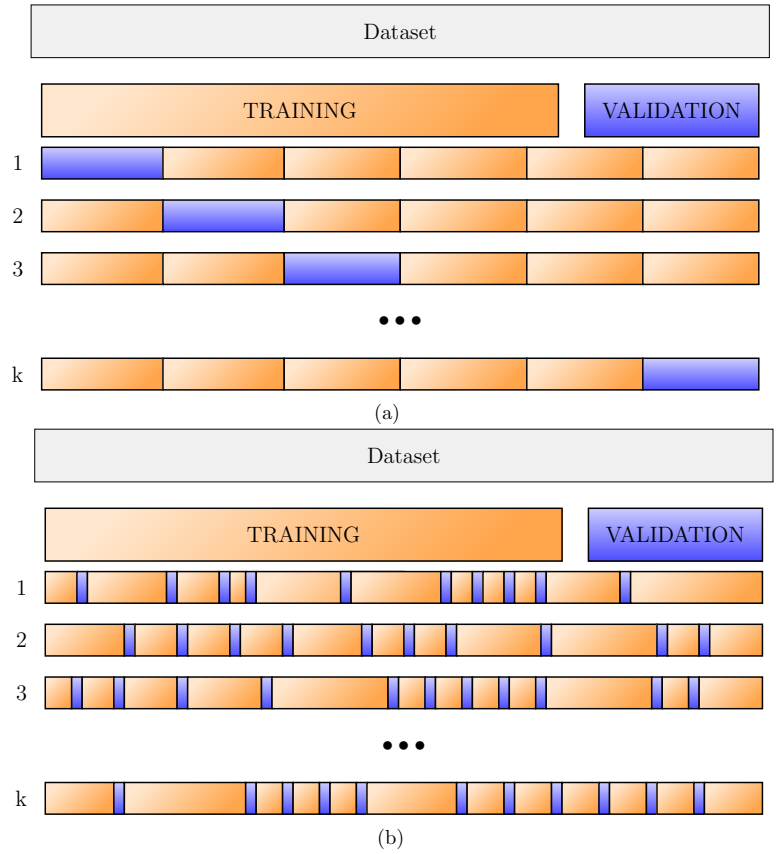


Figure 3. (**a**) Traditional *k*-fold cross validation. (**b**) Random *k*-fold cross validation.

To overcome the drawbacks of traditional *k*-fold cross-validation, this study develops a novel random multi *k*-fold validation to improve the algorithm. It enhances how well the model predicts unknown data based on the known dataset, which helps avoid a problem called "overfitting". The presence of overfitting indicates that the model did not learn the important laws in the data. A random multi *k*-fold validation takes a more dynamic strategy, randomly splitting the data into smaller subsets for training and validation in each cycle. This strategy improves the robustness of model evaluation by removing the potential bias associated with fixed splits and allows for a more comprehensive assessment of the algorithm's effectiveness across varied data distributions. By setting a random number seed to ensure the same number of samples for each data split, the consistency of cross-validation is ensured. Figure 3b shows the procedure for partitioning and training the sample set. This method employs a random approach to create a sample set, allocating 15% for validation and 85% for training data. A random *k*-fold cross-validation facilitates the model's learning from diverse samples, thereby mitigating the risk of converging to local extrema.

3.7. Sliding Window Technique

The sliding window technique, employing a window size of '*K*–days' and a 10–day interval, is utilized to transform raw time series data into sequential input features for the ANN. This method enables the ANN to identify temporal dependencies and trends in the data, thereby improving its predictive efficacy. By methodically adjusting the window,

the model can acquire insights from diverse segments of the data, enhancing its precision in predicting future values. Each window comprises '*K*' consecutive data values, representing '*K*' days of observation. The observations are subsequently normalized and input into the ANN, allowing it to identify patterns and correlations that may not be apparent in the raw data alone. As the model processes these inputs, it adjusts its weights and biases, resulting in more accurate predictions grounded in historical trends. The model subsequently forecasts for the ensuing 10 days. The window subsequently shifts by 10 days, resulting in the revised '*K*' day dataset. For instance, utilizing daily data and $K = 30$, the initial window encompasses days 1–30, followed by window shifts of days 2–31, 3–32, and so forth. This 10-day period facilitates a comprehensive analysis of the data, revealing trends that persist beyond multiple days. For a better understanding of the sliding window mechanism, refer to Figure 4. As for each window shift, we train the data, and based on the MSE value, we try to identify the best neural network architecture to predict next day opening and closing price of the stock. This iterative process enables us to enhance our model by modifying parameters and assessing performance with each new window. Our objective is to improve predictive accuracy and ensure that the architecture aligns with the underlying patterns in the dataset.

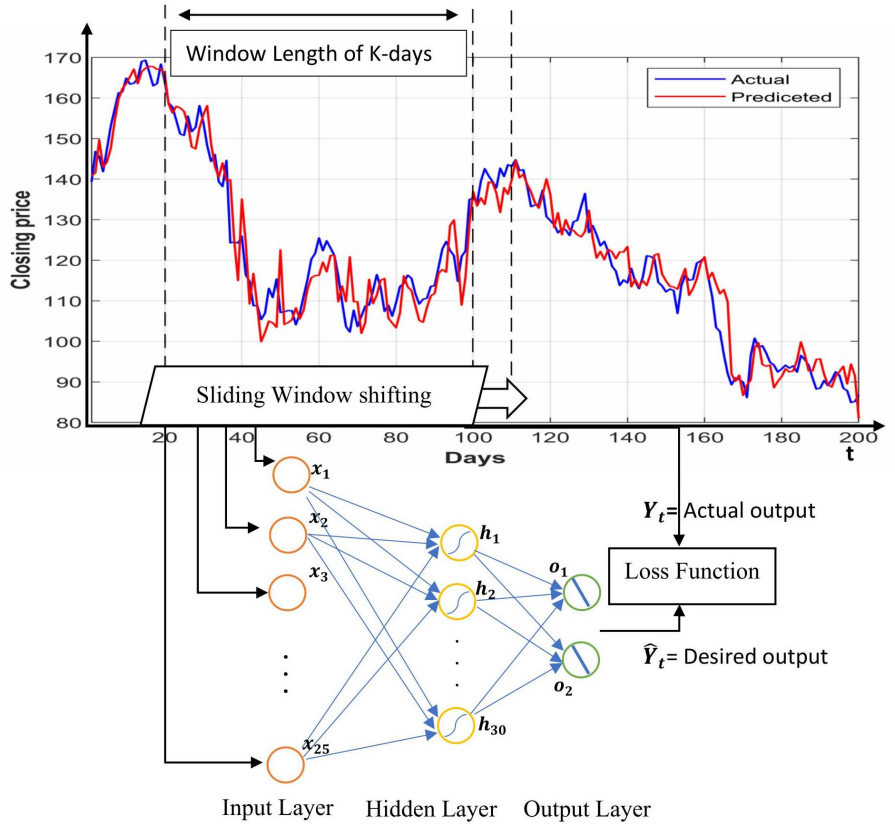


Figure 4. Shifting *K*-day window size mechanism.

3.8. Selection Criteria for Nodes in Hidden Layer for the Dynamic Neural Network Architecture

Initially, we fixed the *K*-day window size, and for each window shift, we determined the *k*-th fold for which the ANN architecture would be developed, and then we varied the number of nodes in the hidden layer to select the optimum ANN architecture. The architecture comprises 25 input neurons in the input layer and 2 output neurons in the output layer, with the hidden layer containing 1, 2, 3, … up to 30 neurons. We construct all plausible ANN structures. The neural network architecture for all combinations will be represented as 25-1-2, 25-2-2, …, 25-29-2, and 25-30-2. So, the total number of possible ANN architectures would be $10 \times 30 = 300$ for each shift. Therefore, we have up to $300 \times 21 = 6300$

different ANN model combinations for each *K*-window size, depending on the fold and number of nodes in the hidden layer. Subsequently, we evaluate the mean square error (mse), which is the loss function for each ANN having a different architecture. From there, we select the optimal architecture with the least overall root mean squared error (rmse), mean absolute error (mae), and R-value. Subsequently, we repeat this process for each fold, selecting the optimal combination of fold and number of nodes in the hidden layer. Through comparison of overall performance metric values across various configurations, we determine the optimal architecture for each window shift that mitigates the error and improves model reliability. Figure 5 demonstrates the flowchart of the selection criteria for the optimal number of nodes in the hidden layer and the number of *k*-folds in determination of the best dynamic architecture for the ANN.

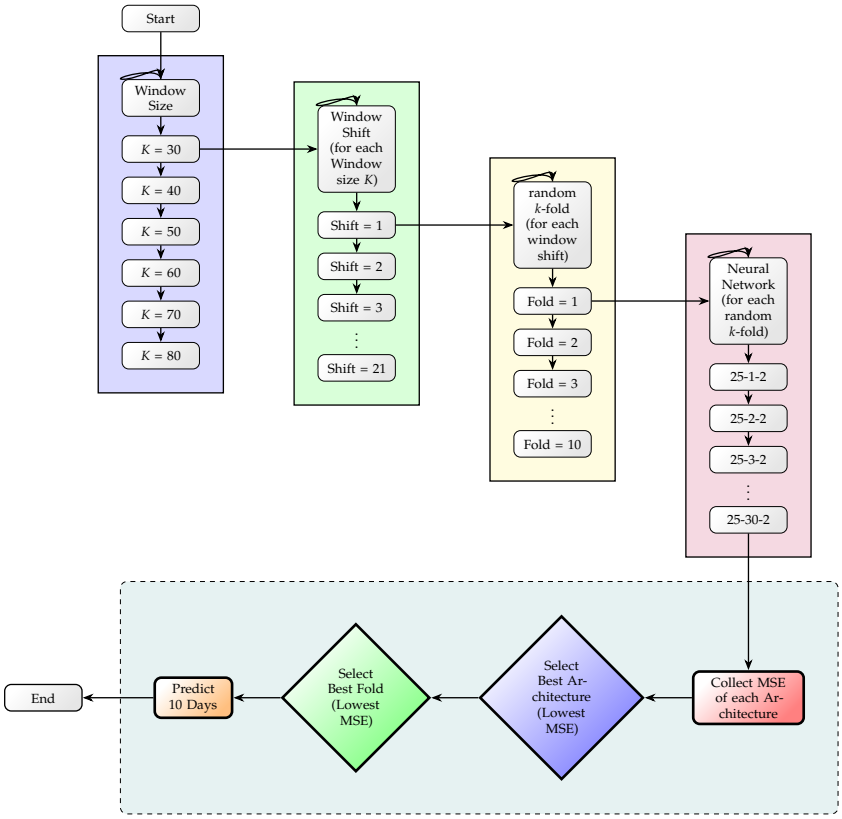
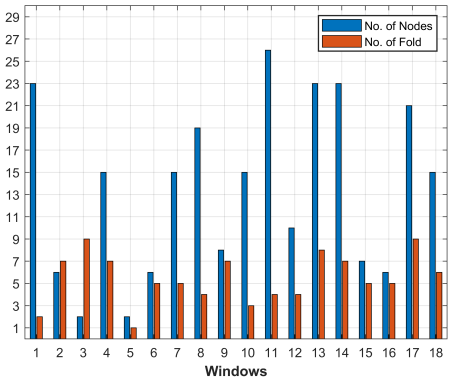


Figure 5. Flowchart of the proposed methodology.

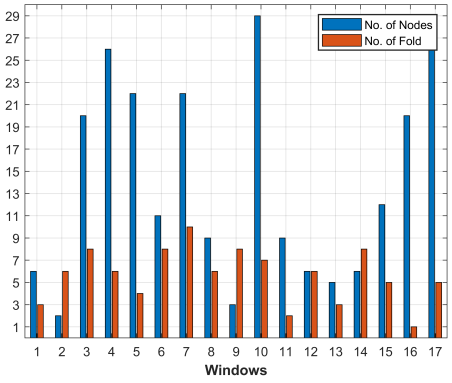
Performance metrics, including rmse, mae, and R-value, are evaluated for each *K*-window size for APPLE, GOOGLE, and AMAZON. The $K = 70$ window size demonstrated enhanced performance for APPLE compared to alternative window sizes. Table 1 shows the performance results for APPLE stock for each shift of the artificial neural network, taking into account the number of nodes in the hidden layer, the random *k*-fold value, and the $K = 70$ window size. The $K = 40$ window size demonstrated enhanced performance for GOOGLE compared to the other window sizes. Table 2 displays the performance data for GOOGLE stock associated with each shift of the artificial neural network, considering the nodes in the hidden layer and the random *k*-fold value along with the $K = 40$ window size. Similarly, the $K = 60$ window size demonstrated superior performance for AMAZON compared to other window sizes. Table 3 shows the performance results for AMAZON stock based on each shift in the artificial neural network, taking into account the nodes in the hidden layer and the random *k*-fold value within the $K = 60$ window size.

Figure 6 demonstrates the dynamic architecture of the ANN with the highest accuracy of folds in each window shift, where the number of nodes in the hidden layer changes

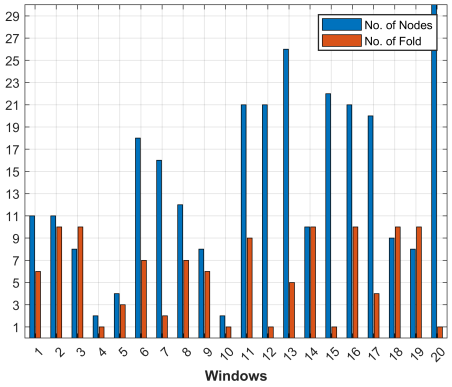
ranging from 1 to 30. This adaptability allows the model to effectively capture temporal patterns and fluctuations in stock prices. The results also demonstrate that incorporating the optimal number of nodes with high accuracy folds in the hidden layer significantly enhances the accuracy of predictions. This shows how important it is to fine-tune model parameters in financial forecasting.



(a) AMAZON



(b) APPLE



(c) GOOGLE

Figure 6. In each window shift, an ANN changes dynamically the number of nodes in the hidden layer and the number of random *k*-folds to (i) forecast opening and closing prices and (ii) predict the correct trend for the next day.

Table 1. Performance metrics across different window shifts, number of hidden layer nodes, and number of random folds for APPLE for $K = 70$-day window shift.

Window Shift	Nodes in Hidden Layer	Random k-Fold	rTrain	rVal	rTest	rAll	rmseTrain	rmseVal	rmseTest	rmseAll	maeTrain	maeVal	maeTest	maeAll
1	6	3	0.9830	0.8718	0.5091	0.9476	1.1746	3.0729	4.1236	2.1074	0.9249	1.5382	1.8375	1.1740
2	6	2	0.9621	0.8227	0.8340	0.9471	1.8720	3.3692	5.5221	2.8208	1.0246	1.7217	2.0924	1.4217
3	20	8	1.0000	0.9019	0.7041	0.9839	0.0000	3.1538	4.4780	1.9683	0.0521	1.5659	1.8739	0.8821
4	26	6	1.0000	0.9279	0.7808	0.9748	0.0000	4.4489	6.1927	2.7414	0.0323	1.8643	2.3201	1.0731
5	22	4	1.0000	0.9622	0.1743	0.9871	0.0005	3.8980	4.6546	2.1903	0.0569	1.7534	1.8883	0.9332
6	11	8	1.0000	0.9747	0.5385	0.9922	0.0000	3.0213	3.8339	1.7585	0.0000	1.5182	1.7802	0.8444
7	22	10	1.0000	0.9665	0.8147	0.9859	0.0085	3.0583	4.1433	1.8526	0.0822	1.5277	1.7521	0.8424
8	9	6	0.9966	0.9417	0.6432	0.9877	0.8719	3.1345	2.8860	1.7183	0.7938	1.5082	1.5582	1.0397
9	9	3	0.9915	0.9038	0.8818	0.9870	1.5391	3.1654	2.2111	1.9329	1.0940	1.4896	1.2984	1.1826
10	29	7	1.0000	0.9458	0.7701	0.9882	0.0468	4.3363	2.8941	1.9063	0.1792	1.7959	1.4564	0.8557
11	9	2	0.9965	0.9714	0.5131	0.9855	1.0548	3.2386	3.5833	1.9667	0.9116	1.5337	1.6641	1.1324
12	6	6	0.9961	0.9794	0.6841	0.9858	0.9897	2.6305	3.7645	1.8562	0.8611	1.4489	1.6968	1.0933
13	5	5	0.9924	0.9781	0.7344	0.9859	1.2147	1.9064	3.1267	1.6763	0.9804	1.2442	1.6170	1.1174
14	6	8	0.9931	0.9643	0.7599	0.9782	1.1944	3.1048	4.0405	2.1020	0.9621	1.6301	1.6267	1.1742
15	12	5	1.0000	0.9709	0.6563	0.9844	0.0000	3.1356	4.0787	1.8524	0.0000	1.6180	1.8204	0.8799
16	20	1	1.0000	0.8408	0.5250	0.9810	0.0297	3.4762	3.6275	1.8185	0.1422	1.7323	1.7313	0.8957
17	26	5	1.0000	0.8923	0.4915	0.9442	0.0205	3.2045	5.2711	2.2103	0.1107	1.5723	2.1171	0.9520

Notes: rTrain, rVal, rTest, rAll: Pearson correlation coefficients for training, validation, testing, and combined datasets. Values closer to 1 indicate strong correlation. rmseTrain, rmseVal, rmseTest, rmseAll: root mean squared error for training, validation, testing, and overall datasets. A lower value means better performance. maeTrain, maeVal, maeTest, maeAll: mean absolute error for training, validation, testing, and overall datasets. Another evaluation metric where a lower value indicates a better fit. This table presents the performance of the proposed model for APPLE stock using a 70-day sliding window shift. Each row corresponds to a different configuration of the model, varying by the number of hidden layer nodes, the number of folds by using random k-fold cross-validation, and the window shift index. The metrics include correlation coefficient (r) for training, validation, and overall, along with rmse (root mean squared error) and mae (mean absolute error) values for each model architecture.

Table 2. Performance metrics across different window shifts, number of hidden layer nodes, and number of random folds for GOOGLE for $K = 40$-day window shift.

Window Shift	Nodes in Hidden Layer	Random k-Fold	rTrain	rVal	rTest	rAll	rmseTrain	rmseVal	rmseTest	rmseAll	maeTrain	maeVal	maeTest	maeAll
1	11	6	0.9996	0.8019	0.6680	0.9377	0.1785	2.7597	3.3368	1.7783	0.3298	1.5120	1.6453	0.9432
2	11	10	0.9835	0.4648	0.2800	0.9479	1.1027	2.5097	3.0030	1.8402	0.8459	1.4393	1.5720	1.1088
3	8	10	1.0000	0.9387	0.8139	0.9586	0.0000	1.9568	2.7263	1.3950	0.0000	1.2270	1.4554	0.7774
4	2	1	0.9799	0.8899	0.6486	0.9476	0.9604	3.2273	5.7366	2.9084	0.8400	1.6101	2.1601	1.3130
5	4	3	0.9975	0.9438	0.3591	0.9890	0.6212	2.5195	2.6049	1.5431	0.6155	1.4340	1.4904	0.9740
6	18	7	0.9982	0.9743	0.5093	0.9881	0.7916	1.9599	3.4582	1.8108	0.7870	1.2019	1.6350	1.0626

Table 2. *Cont.*

Window Shift	Nodes in Hidden Layer	Random k-Fold	rTrain	rVal	rTest	rAll	rmseTrain	rmseVal	rmseTest	rmseAll	maeTrain	maeVal	maeTest	maeAll
7	16	2	0.9997	0.9133	0.8079	0.9819	0.2336	3.1081	2.5383	1.5763	0.4408	1.5264	1.4248	0.9043
8	12	7	0.9993	0.6739	0.5953	0.9219	0.2230	3.2635	3.1647	1.8207	0.4137	1.6388	1.6676	0.9974
9	8	6	0.9938	0.5482	0.7932	0.9250	0.3980	2.7715	2.0986	1.3821	0.5274	1.5366	1.3203	0.9061
10	2	1	0.9038	0.6437	0.4686	0.8038	1.9132	2.1354	3.0609	2.2159	1.2349	1.3278	1.5758	1.3211
11	21	9	0.9823	0.8803	0.7045	0.9452	0.6986	3.0247	2.3540	1.5931	0.7113	1.6476	1.3192	1.0089
12	21	1	1.0000	0.8855	0.8850	0.9439	0.0012	2.1513	2.9578	1.5183	0.0299	1.3815	1.6039	0.8626
13	26	5	0.9972	0.8102	0.0909	0.9532	0.3075	1.7599	2.7274	1.3870	0.4387	1.2074	1.4525	0.8531
14	10	10	0.9899	0.9679	0.5921	0.9556	0.8048	1.4260	4.0472	1.9901	0.7656	1.0906	1.8584	1.1100
15	22	1	1.0000	0.9721	0.6071	0.9916	0.0289	1.9288	1.8748	1.0724	0.1394	1.2305	1.2565	0.7146
16	21	10	1.0000	0.9696	0.4630	0.9862	0.0142	1.9255	2.2231	1.1973	0.0925	1.2240	1.3800	0.7527
17	20	4	0.9846	0.7408	0.7795	0.9335	0.8987	3.3132	4.3512	2.3776	0.8370	1.7115	1.8786	1.2384
18	9	10	0.9825	0.9287	0.7515	0.9411	0.9462	2.6783	2.5463	1.6632	0.8320	1.4594	1.3543	1.0455
19	8	10	0.9881	-0.1981	0.3653	0.9222	0.7978	3.0065	2.7782	1.7496	0.8060	1.3602	1.5150	1.0596
20	30	1	1.0000	0.2057	0.6186	0.9388	0.0198	2.6710	2.9098	1.5968	0.1030	1.4235	1.4814	0.8302

Notes: rTrain, rVal, rTest, rAll: Pearson correlation coefficients for training, validation, testing, and combined datasets. Values closer to 1 indicate strong correlation. rmseTrain, rmseVal, rmseTest, rmseAll: root mean squared error for training, validation, testing, and overall datasets. A lower value means better performance. maeTrain, maeVal, maeTest, maeAll: mean absolute error for training, validation, testing, and overall datasets. Another evaluation metric where a lower value indicates a better fit. This tablepresents the performance of the proposed model for GOOGLE stock using a 40-day sliding window shift. Each row corresponds to a different configuration of the model, varying by the number of hidden layer nodes, the number of folds by using random *k*-fold cross-validation, and the window shift index. The metrics include correlation coefficient (*r*) for training, validation, testing, and overall, along with rmse (root mean squared error) and mae (mean absolute error) values for each model architecture.

Table 3. Performance metrics across different window shifts, number of hidden layer nodes, and number of random folds for AMAZON with a $K = 60$-day window shift.

Window Shift	Nodes in Hidden Layer	Random k-Fold	rTrain	rVal	rTest	rAll	rmseTrain	rmseVal	rmseTest	rmseAll	maeTrain	maeVal	maeTest	maeAll
1	23	2	1.0000	0.9155	0.7070	0.9708	0.0001	4.2528	3.5339	2.0272	0.0013	1.9030	1.6235	0.9177
2	6	7	0.9507	0.9425	0.6550	0.9220	2.4938	2.9255	9.6783	4.3604	1.3632	1.4206	2.5213	1.5879
3	2	9	0.9770	0.8713	0.6412	0.9843	2.3846	4.1438	7.8081	3.8807	1.3356	1.7996	2.6360	1.6458
4	15	7	1.0000	0.9539	0.7680	0.9883	0.0658	6.1101	6.0553	3.1688	0.2167	2.1491	2.2607	1.1654
5	2	1	0.9874	0.9872	0.5924	0.9730	3.7013	3.1698	10.5128	5.2020	1.6438	1.5474	2.7606	1.8344
6	6	5	0.9992	0.9955	0.3366	0.9944	0.8923	2.4659	6.1146	2.5890	0.8360	1.3937	2.2803	1.2254

Table 3. *Cont.*

Window Shift	Nodes in Hidden Layer	Random k-Fold	rTrain	rVal	rTest	rAll	rmseTrain	rmseVal	rmseTest	rmseAll	maeTrain	maeVal	maeTest	maeAll
7	15	5	1.0000	0.9798	0.6795	0.9939	0.1271	4.0509	3.3589	1.9322	0.3183	1.7610	1.6064	0.9171
8	19	4	1.0000	0.9384	0.4742	0.9633	0.0001	5.4808	6.1400	3.0410	0.0084	2.0688	2.2174	1.1193
9	8	7	0.9935	0.8459	0.6343	0.9896	1.0432	3.6586	3.8642	2.1557	0.9103	1.7160	1.8277	1.2081
10	15	3	1.0000	0.9678	0.6576	0.9855	0.0001	3.7542	5.1327	2.3613	0.0001	1.7793	2.0605	1.0068
11	26	4	1.0000	0.9729	0.0627	0.9864	0.0001	3.1701	4.5313	2.0556	0.0006	1.5901	1.8997	0.9169
12	10	4	0.9902	0.9759	0.7244	0.9806	1.8837	2.8553	3.9303	2.4166	1.1919	1.5237	1.8189	1.3439
13	23	8	1.0000	0.9517	0.6145	0.9839	0.0134	4.3055	2.9516	1.9048	0.0952	1.8298	1.5628	0.8866
14	23	7	1.0000	0.9068	0.3747	0.9793	0.0003	4.1147	3.9275	2.0929	0.0146	1.8187	1.7477	0.9283
15	7	5	0.9916	0.9430	0.8540	0.9775	1.4169	4.0533	6.2570	3.0279	1.0357	1.9103	2.2464	1.4041
16	6	5	0.9944	0.9602	0.1182	0.9864	1.4033	3.9595	4.2563	2.4573	1.0359	1.8686	1.7771	1.2969
17	21	9	1.0000	0.9443	0.5080	0.9912	0.1229	3.6950	3.4038	1.8497	0.2964	1.8173	1.6913	0.9472
18	15	6	0.9995	0.9507	0.4649	0.9900	0.4474	3.2109	3.7319	1.8604	0.5351	1.5512	1.7002	0.9649

Notes: rTrain, rVal, rTest, rAll: Pearson correlation coefficients for training, validation, testing, and combined datasets. Values closer to 1 indicate strong correlation. rmseTrain, rmseVal, rmseTest, rmseAll: root mean squared error for training, validation, testing, and overall datasets. A lower value means better performance. maeTrain, maeVal, maeTest, maeAll: mean absolute error for training, validation, testing, and overall datasets. Another evaluation metric where a lower value indicates a better fit. This tablepresents the performance of the proposed model for AMAZON stock using a 60-day sliding window shift. Each row corresponds to a different configuration of the model, varying by the number of hidden layer nodes, the number of folds by using random *k*-fold cross-validation, and the window shift index. The metrics include correlation coefficient (*r*) for training, validation, testing and overall, along with rmse (root mean squared error) and mae (mean absolute error) values for each model architecture.

4. Results

In this study, we focus on the following analysis: (i) measuring the accuracy of the predicted pattern trend to the actual pattern trend and (ii) actual versus forecasted opening and closing price by optimal K-day window size for the abovementioned stocks. For the first analysis, we define the actual pattern trend as the difference between the day's closing price and the opening price, and the predicted pattern trend as the difference between the next day's predicted closing price and the predicted opening price. If this difference is positive, then we call it an "upward trend," and its sign will be positive; otherwise, we call it a "downward trend," and its sign will be negative. The results of this comparison will allow us to evaluate the efficacy of our dynamic neural network and discover any differences between our forecasts and the actual short-term market trend behavior.

Let us define the following terms to understand the pattern trend analysis as follows:

1. Predicted trend of the $(t+1)$ day on the day t, defined as:
   (PT) = Predicted closing price of $(t+1)$ day $-$ Predicted opening price of $(t+1)$ day.
2. Actual trend of the $(t+1)$ day, defined as:
   (AT) =Actual closing price of the $(t+1)$ day $-$ Actual opening price of the $(t+1)$ day.
3. Correct pattern prediction (CP):
   (a) Case 1: Same direction: If $\text{sign}(AT) = \text{sign}(PT)$, then:
   $$CP = 1 \quad (\text{Correct prediction})$$
   (b) Case 2: Opposite direction, within tolerance (T) level: If $\text{sign}(AT) \neq \text{sign}(PT)$ and $AT \neq 0$ and $(|PT| + |AT|) \leq T$, then:
   $$CP = 1 \quad (\text{Correct prediciton})$$
   (c) Case 3: Opposite direction, beyond tolerance (T): If $\text{sign}(AT) \neq \text{sign}(PT)$ and $AT \neq 0$ and $(|PT| + |AT|) > T$, then:
   $$CP = 0 \quad (\text{Incorrect prediction})$$
   (d) Case 4: AT is zero: If $AT = 0$, then if $PT = 0$ then $CP = 1$ else $CP = 0$.
4. Total number of predictions (TP) =
   $$\text{Total number of correct predictions}(CP = 1)$$
   $$+$$
   $$\text{Total number of incorrect predictions}(CP = 0).$$

With the above four conditions, we will find the accuracy of the correct prediction trend one day before the actual trend as follows:

$$\%\text{ of accuracy for correct predicting trend} = \frac{\text{Total number of } CP = 1}{TP} \times 100.$$

By analyzing these discrepancies, investors could create buying and selling strategies for $(t+1)$-th day on the t-th day. As an example, if the difference between the next day's predicted closing price and predicted opening price is significantly large (upward trend), then the investor can buy stocks in bulk at the next day's market opening time and book a profit at the end of the day. If the difference is considerably negative (downward trend), avoid the deal for the following day. Here, we set the tolerance to 2.5%, which is equal to

±1.25% of the closing price of the previous day. It indicates that if the trade price does not exhibit a substantially larger movement (sideways trend), it will not cause a major loss or profit for the investor. This strategy allows investors to minimize risk while maximizing potential gains. For this, we are analyzing the accuracy of correct predicted pattern trends in % for every *K*-day window size while considering the abovementioned four cases.

Table 4 presents the accuracy (in percentage) level of a dynamic ANN to predicting the next day's pattern trend for three different stocks: AMAZON, GOOGLE, and APPLE. The accuracy is measure for all *K*-window sizes to predict correct pattern trend for next day. For the AMAZON stock, $K = 60$ days window size predicts correct predicted trend with highest accuracy of 71.11%. For the GOOGLE stock, $K = 40$ days window size predicts correct predicted trend with highest accuracy of 73% and lastly, for the APPLE stock, $K = 70$ days window size predicts correct predicted trend with highest accuracy of 75.29%.

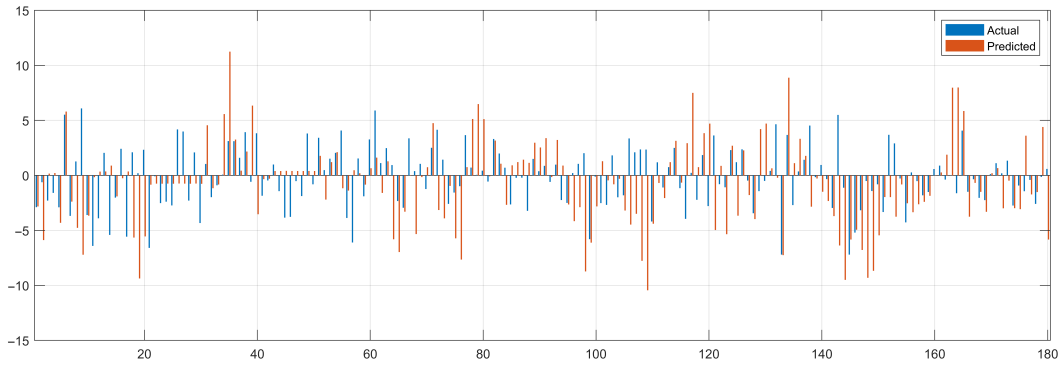
Table 4. Accuracy in % of correct predicted pattern trend for next day with analyzing all *K*-window sizes with the dynamic ANN with L-M algorithm.

K-Window Size	AMAZON	GOOGLE	APPLE
30	66.67	71.90	70.00
40	60.50	73.00	70.00
50	64.21	69.47	73.16
60	71.11	63.89	70.00
70	69.42	72.33	75.29
80	68.75	68.12	73.75

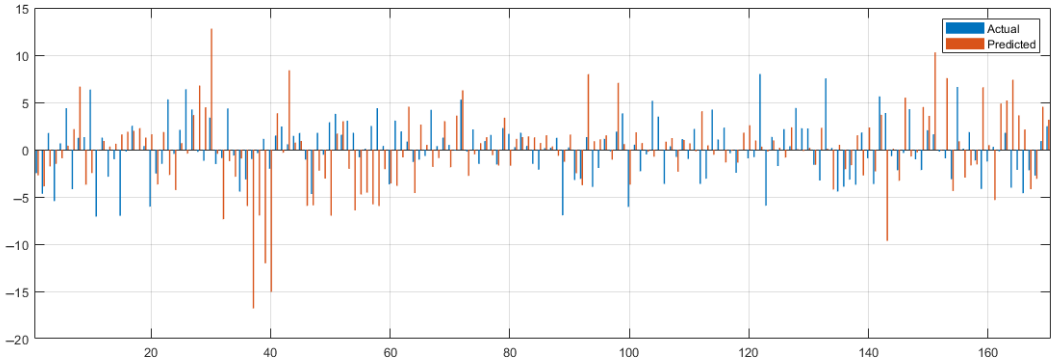
Figure 7 demonstrates how the actual versus predicted trends match each other for the abovementioned stocks. Figure 7a for AMAZON represents the actual vs. predicted trend with the window size $K = 60$ and 71.11% accuracy. Similarly, Figure 7b,c represent, for APPLE and GOOGLE stock, the actual trend vs. the predicted trend of window size $K = 70$ and $K = 40$ with 75.29% and 73.0% accuracy, respectively.

Figure 8 illustrates the actual versus predicted (i) opening prices (Figure 8a,c,e) and (ii) closing prices (Figure 8b,d,f) for the abovementioned stocks across the dataset. The analysis employs window sizes *K* = 60, 70, and 40 days for AMAZON, APPLE, and GOOGLE, respectively, as these sizes yield the most accurate predictions for the subsequent day. Quantitative metrics such as rmse and regression line (R) are crucial for evaluating time series forecasting models; nevertheless, they often provide an aggregated and potentially incomplete representation of a model's predictive capacity. This metric is crucial for assessing error magnitude; however, they can mask significant performance trends and lack the ability to pinpoint precise forecasts. Regression line plots, in particular, show the correlation between the model's predicted and actual values in forecasting evaluation.

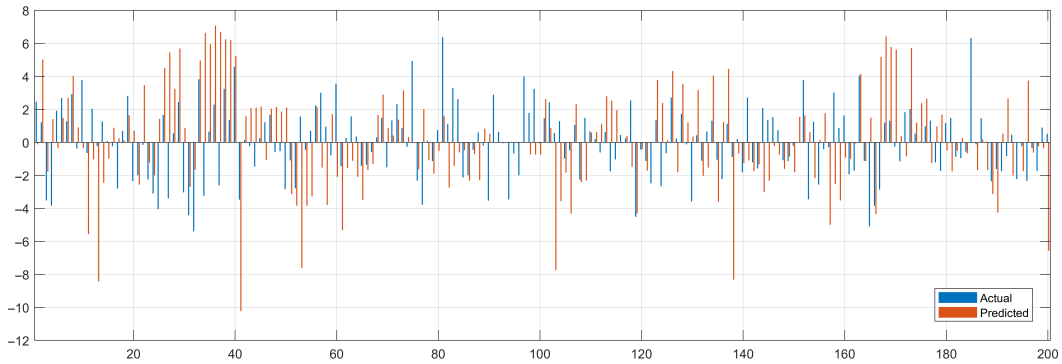
Figure 9 represents the regression line for the actual versus predicted variable for opening prices (Figure 9a,c,e) and for closing prices (Figure 9b,d,f) across the dataset. The analysis employs window sizes *K* = 60, 70, and 40 days for AMAZON, APPLE, and GOOGLE, respectively, as these window sizes yield the most accurate predictions for the subsequent day. This graph aligns the model's output with actual data on a point-by-point basis, bypassing scalar error metrics. A model exhibiting a slope about equal to 1 and an intercept close to 0 can be considered unbiased. On the other hand, if the model consistently overestimates or underestimates values, and there is a lot of variation in the points, it shows where the model does not accurately represent the time series data. Regression line plots can reveal localized performance patterns that aggregated error measures fail to capture. A model with a low mean squared error might seem effective overall, but a regression line plot can show specific points where the model's predictions often differ from the real values.



(**a**) AMAZON

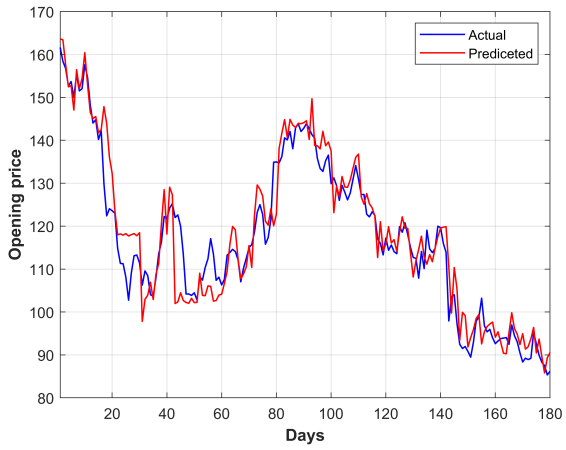


(**b**) APPLE

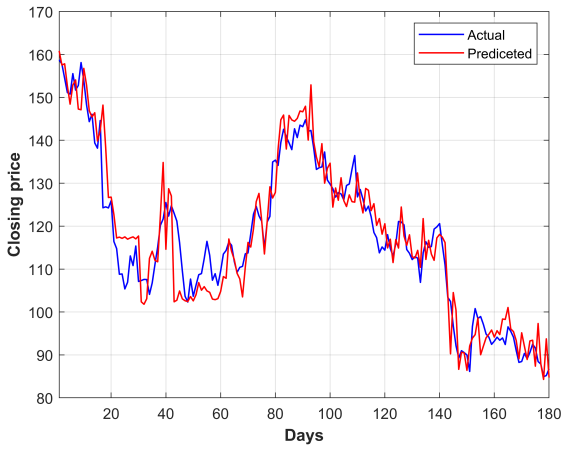


(**c**) GOOGLE

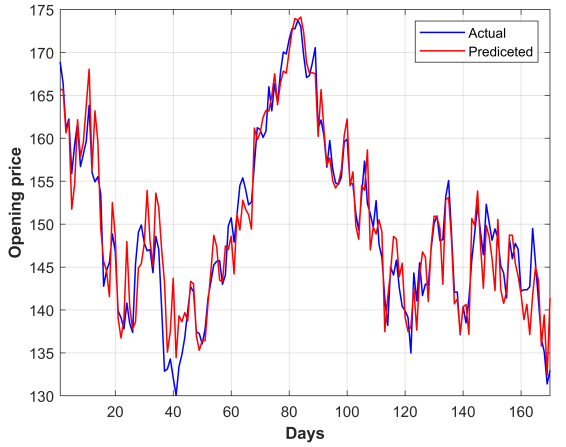
Figure 7. Comparison of the predicted next-day trend with the actual next-day trend for (**a**) AMAZON with $K = 60$ window size, (**b**) APPLE with $K = 70$ window size, and (**c**) GOOGLE with $K = 40$ window size.



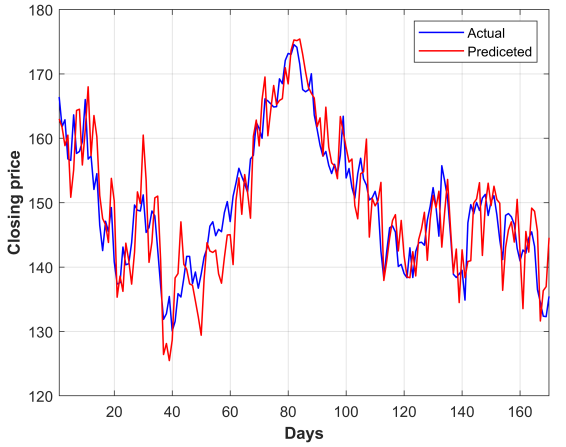
(a) AMAZON



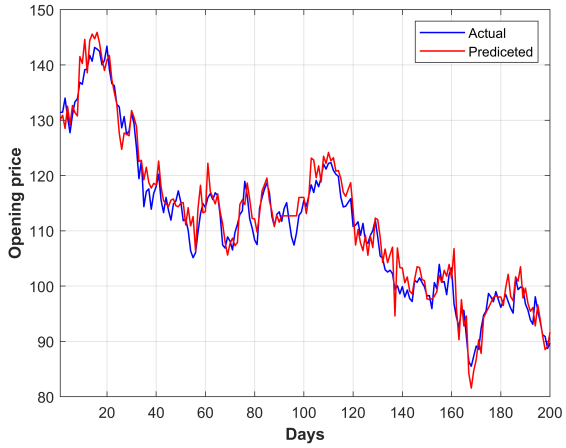
(b) AMAZON



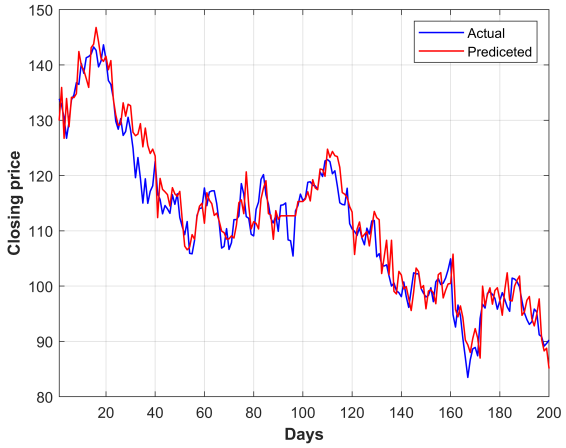
(c) APPLE



(d) APPLE

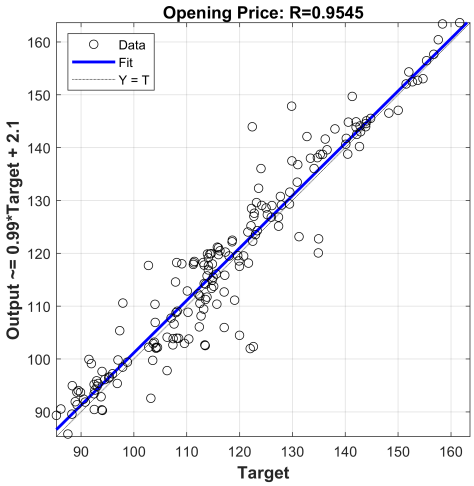


(e) GOOGLE

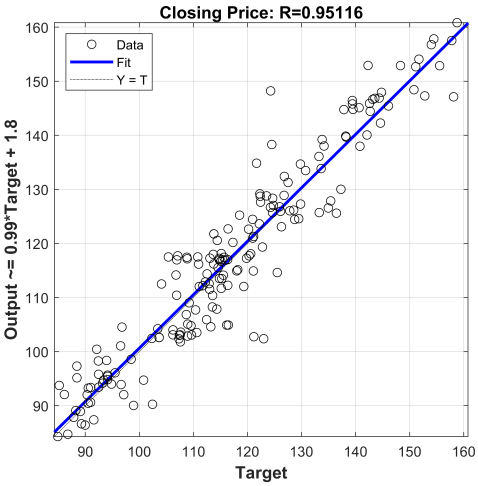


(f) GOOGLE

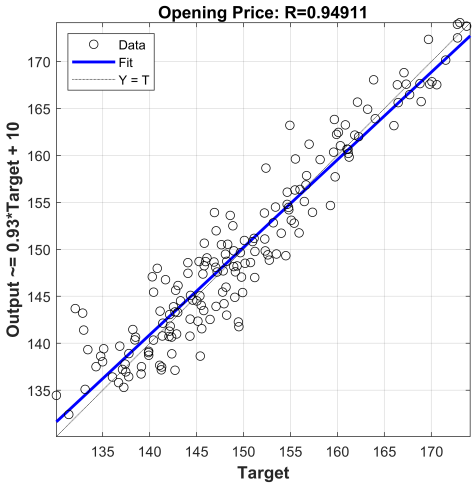
Figure 8. Actual versus predicted forecasting for (**a**,**c**,**e**) opening and (**b**,**d**,**f**) closing price for above-mentioned stocks using dynamic ANN with L-M algorithm and K = (**a**) 60-, (**b**) 70-, (**c**) 40-, (**d**) 60-, (**e**) 70-, and (**f**) 40-day sliding window sizes.



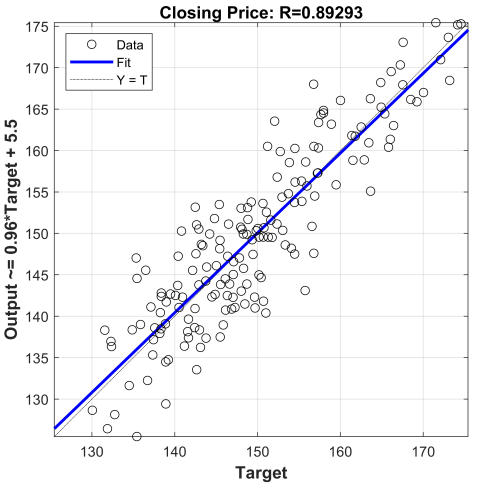
(a) AMAZON



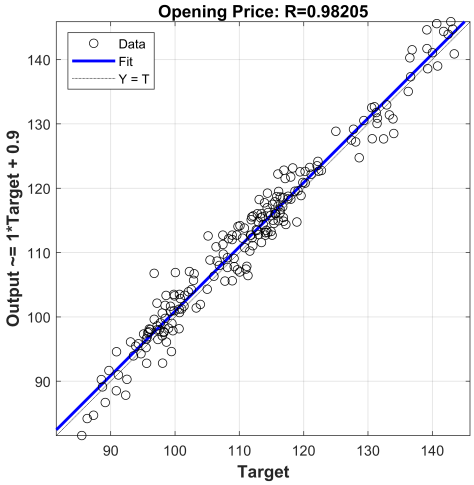
(b) AMAZON



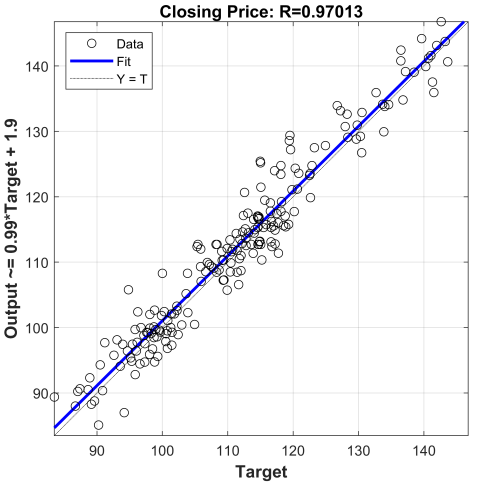
(c) APPLE



(d) APPLE



(e) GOOGLE



(f) GOOGLE

Figure 9. Plot regression line for (**a**,**c**,**e**) opening and (**b**,**d**,**f**) closing price for abovementioned stocks using dynamic ANN with L-M algorithm and K = (**a**) 60-, (**b**) 70-, (**c**) 40-, (**d**) 60-, (**e**) 70-, and (**f**) 40-day sliding window sizes.

5. Discussion

After a well-tuned refining process that included data normalization, variable *K*-day sliding window shifts, and randomized *k*-fold validation for each fold with ANN training using the L-M algorithm, the proposed model shows better accuracy over existing models such as LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) by consistently generating significantly lower rmse and mae values as per the Table 5. Also, Table 4 shows that the proposed dynamic model achieved the highest accuracy for APPLE with a window size of $K = 60$, GOOGLE with $K = 40$, and Amazon with $K = 70$. Upon determining appropriate *K*-window sizes for the respective stocks and applying them to unseen data, the model yields accurate predictions for the next day's price trend: APPLE with 71.11%, GOOGLE with 73%, and AMAZON with 75.29%. Apart from the low rmse and mae values, the high R values for AMAZON, APPLE, and GOOGLE are 0.9545, 0.9491, and 0.9820, respectively, indicating a strong correlation between the predicted and actual values.

Table 5. Rmse and mae value comparison of the proposed model with existing LSTM and GRU models.

Stock	LSTM		GRU		Proposed Model	
	rmse	**mae**	**rmse**	**mae**	**rmse**	**mae**
APPLE	7.4849	6.1997	6.7798	5.3719	2.0282	1.0291
GOOGLE	6.9836	5.6475	6.8672	5.0158	1.7208	0.9892
AMAZON	8.9498	6.7350	9.4086	6.7741	2.6880	1.1843

NOTE: LSTM = Long Short-Term Memory, GRU = Gated Recurrent Unit, rmse = root mean square error, mae = mean absolute error. Lower numbers indicate better performance.

Despite the abovementioned strengths, the current model relies solely on historical OCHLV (open, close, high, low, and volume) data as input parameter, which, while valuable for capturing internal market dynamics, fails to account for external factors that often drive stock price movements. Critical influences such as political developments, economic announcements, earnings reports, and news sentiment are excluded. These exogenous and often unstructured variables may have a significant nonlinear effect on price trends. The absence of such contextual data limits the model's adaptability during sudden or sentiment-driven market shifts. To address this limitation, future work could involve the integration of alternative data sources such as financial news sentiment, macroeconomic indicators (e.g., interest rates, CPI), and technical indicators such as RSI and MACD. Since the proposed model provided a certain level of accuracy in forecasting trends, it also provides a glimpse into future possibilities for integrating the dynamic model with more complex optimizers.

Practical Feasibility of Deploying the Model

Although the proposed model involves dynamic architectural adjustments and randomized *k*-fold cross-validation, the overall computational load remains manageable. We conducted this experiment using MATLAB R2024a on an Apple M1 chip with 8GB RAM, typically completing each window shift in a shorter time. Importantly, once the model is trained, real-time prediction requires only forward propagation, which is computationally lightweight. The model maintains a computational structure that is practical for real-time or near real-time deployment, as forecasts can be performed efficiently post market close, while retraining can be scheduled during off-market hours, ensuring uninterrupted forecasting service with minimal latency. Furthermore, real-time stock market data can be readily accessed through public APIs such as Yahoo Finance, enabling seamless integration of the model into a real-time forecasting pipeline. This integration, coupled with

the model's efficient forecast and retraining strategy, supports its practical deployment in operational environments.

6. Conclusions

In comparison to the current LSTM and GRU models on the study of under-researched stocks, the proposed dynamic model provides the highest accuracy in terms of the rmse and mae values following the well-tuned refining process, which includes data normalization, variable K-sized sliding windows, and randomized k-fold validations on each fold to determine the ideal number of nodes in the hidden layer of the ANN using the L-M algorithm. This approach not only enhances the model's predictive performance but also ensures robustness against overfitting. As a result, the dynamic model demonstrates superior capabilities in capturing the correct price trend for the next day.

So, as future work, the integration of dynamic ANNs with the sliding window technique demonstrates broad applicability across various sectors, including finance, gas, oil, commodity, and currency exchanges. Additionally, transitioning to hybrid or ensemble architectures like ANN-LSTM, ANN-GRU, or ANN-XGboost would allow the model to better capture both short-term fluctuations and long-term dependencies in time series while also incorporating memory of past external events. We can extend this innovative approach to forecast cyberattack moments in security defense systems, as well as in the healthcare system. Future efforts will focus on improving the proposed technique and expanding its application across these areas.

Author Contributions: Conceptualization, methodology, writing—original draft preparation, formal analysis, investigation, writing—review and editing by J.I.P.; visualization, supervision, validation, software by R.D. Both authors have contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets used and/or analyzed during the current study were obtained from Yahoo Finance website (https://finance.yahoo.com/) accessed on 2 March 2025 and are available from the corresponding author upon reasonable request.

Acknowledgments: The authors express gratitude to the Department of Mathematics, School of Advanced Sciences, Vellore Institute of Technology, Vellore, India, for facilitating the research at the FIST Lab (SR/FST/MS-II/2023/139), which is funded by the Department of Science and Technology (DST), Government of India. Furthermore, the authors extend gratitude to the anonymous reviewers and editors for their constructive feedback and insightful comments, which have significantly improved this manuscript. Their expertise and dedication have significantly enhanced our work.

Conflicts of Interest: The authors have equally contributed and given their consent for publication. Furthermore, the authors affirm that they have no competing interests. The authors additionally state that they possess no identifiable competing financial interests or relationships that may have seemed to influence the work presented in this study.

Appendix A

Appendix A.1. Steepest Descent Algorithm (S-D)

The steepest descent algorithm (S-D) is categorized as a first-order partial derivative algorithm. It utilizes the first-order derivative of the total loss function for determining the

minima within the loss space. The gradient g is frequently characterized as the first-order derivative of the total loss function in (1):

$$g = \frac{\partial L(x, w)}{\partial w} = \left[\frac{\partial L}{\partial w_1} \quad \frac{\partial L}{\partial w_2} \quad \cdots \quad \frac{\partial L}{\partial w_{\mathcal{N}}} \right]^T \tag{A1}$$

Utilizing the concept of gradient g in (A1), the iterative rule for the steepest descent method can be defined as

$$w_{k+1} - w_k = -\beta g_k \tag{A2}$$

The learning constant, denoted by β, is the step size.

Appendix A.2. Newton's Method

Newton's method presupposes that all gradient components $g_1, g_2, \cdots, g_{\mathcal{N}}$ are functions of the weights and that all weights are linearly independent.

$$\begin{cases} g_1 = \mathcal{F}_1(w_1, w_2, \cdots, w_{\mathcal{N}}) \\ g_2 = \mathcal{F}_2(w_1, w_2, \cdots, w_{\mathcal{N}}) \\ \quad \vdots \\ g_{\mathcal{N}} = \mathcal{F}_{\mathcal{N}}(w_1, w_2, \cdots, w_{\mathcal{N}}) \end{cases} \tag{A3}$$

where $\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_{\mathcal{N}}$ represent nonlinear correlations between weights and their corresponding gradient vector.

Expand each $g_i (i = 1, 2, \cdots, \mathcal{N})$ in Equations (A3) using a Taylor series first-order approximation:

$$\begin{cases} g_1 \approx g_{1,0} + \frac{\partial g_1}{\partial w_1}\Delta w_1 + \frac{\partial g_1}{\partial w_2}\Delta w_2 + \cdots + \frac{\partial g_1}{\partial w_{\mathcal{N}}}\Delta w_{\mathcal{N}} \\ g_2 \approx g_{2,0} + \frac{\partial g_2}{\partial w_1}\Delta w_1 + \frac{\partial g_2}{\partial w_2}\Delta w_2 + \cdots + \frac{\partial g_2}{\partial w_{\mathcal{N}}}\Delta w_{\mathcal{N}} \\ \quad \vdots \\ g_{\mathcal{N}} \approx g_{\mathcal{N},0} + \frac{\partial g_{\mathcal{N}}}{\partial w_1}\Delta w_1 + \frac{\partial g_{\mathcal{N}}}{\partial w_2}\Delta w_2 + \cdots + \frac{\partial g_{\mathcal{N}}}{\partial w_{\mathcal{N}}}\Delta w_{\mathcal{N}} \end{cases} \tag{A4}$$

By incorporating the definition of the gradient vector g in (A1), it can be determined that

$$\frac{\partial g_i}{\partial w_j} = \frac{\partial \left(\frac{\partial L}{\partial w_i} \right)}{\partial w_j} = \frac{\partial^2 L}{\partial w_i \partial w_j} \tag{A5}$$

By substituting Equation (A5) into Equation (A4):

$$\begin{cases} g_1 \approx g_{1,0} + \frac{\partial^2 L}{\partial w_1^2}\Delta w_1 + \frac{\partial^2 L}{\partial w_1 \partial w_2}\Delta w_2 + \cdots + \frac{\partial^2 L}{\partial w_1 \partial w_{\mathcal{N}}}\Delta w_{\mathcal{N}} \\ g_2 \approx g_{2,0} + \frac{\partial^2 L}{\partial w_2 \partial w_1}\Delta w_1 + \frac{\partial^2 L}{\partial w_2^2}\Delta w_2 + \cdots + \frac{\partial^2 L}{\partial w_2 \partial w_{\mathcal{N}}}\Delta w_{\mathcal{N}} \\ \quad \vdots \\ g_{\mathcal{N}} \approx g_{\mathcal{N},0} + \frac{\partial^2 L}{\partial w_{\mathcal{N}} \partial w_1}\Delta w_1 + \frac{\partial^2 L}{\partial w_{\mathcal{N}} \partial w_2}\Delta w_2 + \cdots + \frac{\partial^2 L}{\partial w_{\mathcal{N}}^2}\Delta w_{\mathcal{N}} \end{cases} \tag{A6}$$

In contrast to the steepest descent approach, the second-order derivatives of the total loss function must be computed for each element of the gradient vector. To achieve the

minima of the total loss function L, each component of the gradient vector must equal zero. Consequently, the left sides of Equations (A6) are all null, and then

$$\begin{cases} 0 \approx g_{1,0} + \dfrac{\partial^2 L}{\partial w_1^2}\Delta w_1 + \dfrac{\partial^2 L}{\partial w_1 \partial w_2}\Delta w_2 + \cdots + \dfrac{\partial^2 L}{\partial w_1 \partial w_\mathcal{N}}\Delta w_\mathcal{N} \\ 0 \approx g_{2,0} + \dfrac{\partial^2 L}{\partial w_2 \partial w_1}\Delta w_1 + \dfrac{\partial^2 L}{\partial w_2^2}\Delta w_2 + \cdots + \dfrac{\partial^2 L}{\partial w_2 \partial w_\mathcal{N}}\Delta w_\mathcal{N} \\ \vdots \\ 0 \approx g_{\mathcal{N},0} + \dfrac{\partial^2 L}{\partial w_\mathcal{N} \partial w_1}\Delta w_1 + \dfrac{\partial^2 L}{\partial w_\mathcal{N} \partial w_2}\Delta w_2 + \cdots + \dfrac{\partial^2 L}{\partial w_\mathcal{N}^2}\Delta w_\mathcal{N} \end{cases} \tag{A7}$$

By equating Equation (A1) with (A7)

$$\begin{cases} -\dfrac{\partial L}{\partial w_1} = -g_{1,0} = \approx \dfrac{\partial^2 L}{\partial w_1^2}\Delta w_1 + \dfrac{\partial^2 L}{\partial w_1 \partial w_2}\Delta w_2 + \cdots + \dfrac{\partial^2 L}{\partial w_1 \partial w_\mathcal{N}}\Delta w_\mathcal{N} \\ -\dfrac{\partial L}{\partial w_2} = -g_{2,0} = \approx \dfrac{\partial^2 L}{\partial w_2 \partial w_1}\Delta w_1 + \dfrac{\partial^2 L}{\partial w_2^2}\Delta w_2 + \cdots + \dfrac{\partial^2 L}{\partial w_2 \partial w_\mathcal{N}}\Delta w_\mathcal{N} \\ \vdots \\ -\dfrac{\partial L}{\partial w_\mathcal{N}} = -g_{\mathcal{N},0} = \approx \dfrac{\partial^2 L}{\partial w_\mathcal{N} \partial w_1}\Delta w_1 + \dfrac{\partial^2 L}{\partial w_\mathcal{N} \partial w_2}\Delta w_2 + \cdots + \dfrac{\partial^2 L}{\partial w_\mathcal{N}^2}\Delta w_\mathcal{N} \end{cases} \tag{A8}$$

There exist $\mathcal{N}$ equations corresponding to $\mathcal{N}$ parameters, enabling the calculation of all Δw_i. The solutions enable iterative updates to the weight space. Equation (A8) may alternatively be expressed in matrix form:

$$\begin{aligned} \begin{bmatrix} -g_1 \\ -g_2 \\ \vdots \\ -g_\mathcal{N} \end{bmatrix} &= \begin{bmatrix} -\frac{\partial L}{\partial w_1} \\ -\frac{\partial L}{\partial w_2} \\ \vdots \\ -\frac{\partial L}{\partial w_\mathcal{N}} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial L^2}{\partial w_1^2} & \frac{\partial L^2}{\partial w_1 \partial w_2} & \cdots & \frac{\partial L^2}{\partial w_1 \partial w_\mathcal{N}} \\ \frac{\partial L^2}{\partial w_2 \partial w_1} & \frac{\partial L^2}{\partial w_2^2} & \cdots & \frac{\partial L^2}{\partial w_2 \partial w_\mathcal{N}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial L^2}{\partial w_\mathcal{N} \partial w_1} & \frac{\partial L^2}{\partial w_\mathcal{N} \partial w_2} & \cdots & \frac{\partial L^2}{\partial w_\mathcal{N}^2} \end{bmatrix} \begin{bmatrix} \Delta w_1 \\ \Delta w_2 \\ \vdots \\ \Delta w_\mathcal{N} \end{bmatrix} \end{aligned} \tag{A9}$$

where the square matrix can be denoted as Hessian matrix:

$$\mathcal{H} = \begin{bmatrix} \frac{\partial L^2}{\partial w_1^2} & \frac{\partial L^2}{\partial w_1 \partial w_2} & \cdots & \frac{\partial L^2}{\partial w_1 \partial w_\mathcal{N}} \\ \frac{\partial L^2}{\partial w_2 \partial w_1} & \frac{\partial L^2}{\partial w_2^2} & \cdots & \frac{\partial L^2}{\partial w_2 \partial w_\mathcal{N}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial L^2}{\partial w_\mathcal{N} \partial w_1} & \frac{\partial L^2}{\partial w_\mathcal{N} \partial w_2} & \cdots & \frac{\partial L^2}{\partial w_\mathcal{N}^2} \end{bmatrix} \tag{A10}$$

Adding Equation (A1) and (A10) with Equation (A9), we have

$$-g = \mathcal{H}\Delta w \tag{A11}$$

Therefore,

$$\Delta w = -\mathcal{H}^{-1} g \tag{A12}$$

So, the update rule for Newton's technique can be written as

$$w_{k+1} - w_k = -\mathscr{H}_k^{-1} g_k \tag{A13}$$

The Hessian matrix $\mathscr{H}$, representing the second-order derivatives of the total loss function, offers a precise evaluation of the gradient vector's variation. Comparing Equations (A2) and (A13) provides that suitably aligned step sizes are generated by the inverse of the matrix $\mathscr{H}$.

Appendix A.3. Gauss–Newton Method

Utilizing Newton's method for weight adjustment necessitates the calculation of the matrix $\mathscr{H}$, which involves determining the second-order derivatives of the total loss function, a process that may be rather complex. To simplify the calculation procedure, the Jacobian matrix $\mathbb{J}$ is defined as

$$\mathbb{J} = \begin{bmatrix} \frac{\partial l_{1,1}}{\partial w_1} & \frac{\partial l_{1,1}}{\partial w_2} & \cdots & \frac{\partial l_{1,1}}{\partial w_\mathcal{N}} \\ \frac{\partial l_{1,2}}{\partial w_1} & \frac{\partial l_{1,2}}{\partial w_2} & \cdots & \frac{\partial l_{1,2}}{\partial w_\mathcal{N}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial l_{1,\mathcal{M}}}{\partial w_1} & \frac{\partial l_{1,\mathcal{M}}}{\partial w_2} & \cdots & \frac{\partial l_{1,\mathcal{M}}}{\partial w_\mathcal{N}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial l_{\mathcal{P},1}}{\partial w_1} & \frac{\partial l_{\mathcal{P},1}}{\partial w_2} & \cdots & \frac{\partial l_{\mathcal{P},1}}{\partial w_\mathcal{N}} \\ \frac{\partial l_{\mathcal{P},2}}{\partial w_1} & \frac{\partial l_{\mathcal{P},2}}{\partial w_2} & \cdots & \frac{\partial l_{\mathcal{P},2}}{\partial w_\mathcal{N}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial l_{\mathcal{P},\mathcal{M}}}{\partial w_1} & \frac{\partial l_{\mathcal{P},\mathcal{M}}}{\partial w_2} & \cdots & \frac{\partial l_{\mathcal{P},\mathcal{M}}}{\partial w_\mathcal{N}} \end{bmatrix} \tag{A14}$$

Combining (1) and (A1), components of the g can be evaluated as:

$$\begin{aligned} g_i &= \frac{\partial L}{\partial w_i} \\ &= \frac{\partial \left(\frac{1}{2} \sum_{p=1}^{\mathcal{P}} \sum_{m=1}^{M} l_{p,m}^2 \right)}{\partial w_i} \\ &= \sum_{p=1}^{\mathcal{P}} \sum_{m=1}^{\mathcal{M}} \left(\frac{\partial l_{p,m}}{\partial w_i} l_{p,m} \right) \end{aligned} \tag{A15}$$

By adding Equations (A14) and (A15), the correlation between the matrix $\mathbb{J}$ and g is established as follows:

$$g = \mathbb{J} l \tag{A16}$$

where the loss vector l can be given as:

$$l = \begin{bmatrix} l_{1,1} \\ l_{1,2} \\ \cdots \\ l_{1,\mathcal{M}} \\ \cdots \\ l_{\mathcal{P},1} \\ l_{\mathcal{P},2} \\ \cdots \\ l_{\mathcal{P},\mathcal{M}} \end{bmatrix} \tag{A17}$$

The element located in the j-th column and i-th row of the Hessian matrix $\mathscr{H}$ could be determined by substituting Equation (1) into Equation (A10):

$$\begin{aligned} \hbar_{i,j} &= \frac{\partial^2 L}{\partial w_i \partial w_j} \\ &= \frac{\partial^2 \left(\frac{1}{2} \sum_{p=1}^{\mathscr{P}} \sum_{m=1}^{\mathscr{M}} l_{p,m}^2 \right)}{\partial w_i \partial w_j} \\ &= \sum_{p=1}^{\mathscr{P}} \sum_{m=1}^{\mathscr{M}} \frac{\partial l_{p,m}}{\partial w_i} \frac{\partial l_{p,m}}{\partial w_j} + \mathcal{S}_{i,j} \end{aligned} \tag{A18}$$

where $\mathcal{S}_{i,j}$ is

$$\mathcal{S}_{i,j} = \sum_{p=1}^{\mathscr{P}} \sum_{m=1}^{\mathscr{M}} \frac{\partial^2 l_{p,m}}{\partial w_i \partial w_j} l_{p,m} \tag{A19}$$

Given that the core idea of Newton's technique postulates that $\mathcal{S}_{i,j}$ tends to zero, the correlation between the matrix $\mathscr{H}$ and $\mathbb{J}$ may be reformulated as

$$\mathscr{H} = \mathbb{J}^T \mathbb{J} \tag{A20}$$

The revised formula for the Gauss–Newton technique can be derived from incorporating Equations (A13), (A16), and (A20):

$$w_{k+1} = w_k - (\mathbb{J}_k^T \mathbb{J}_k)^{-1} \mathbb{J}_k^T l_k \tag{A21}$$

It is clear that in comparison to Newton's technique (A13), the Gauss–Newton technique is better because it does not need to calculate higher-order derivatives for the total loss function. Instead, it uses the matrix $\mathbb{J}$. The Gauss–Newton technique encounters the same invertibility issue as the Newton technique. From a mathematical perspective, the issue can be viewed as the potential invertibility of the matrix $\mathbb{J}^T \mathbb{J}$.

References

1. Bulkowski, T.N. *Encyclopedia of Chart Patterns*; John Wiley & Sons: Hoboken, NJ, USA, 2021.
2. Tealab, A. Time Series Forecasting Using Artificial Neural Networks Methodologies: A Systematic Review. *Future Comput. Inform. J.* **2018**, *3*, 334–340. [CrossRef]
3. Smith, J.S.; Wu, B.; Wilamowski, B.M. Neural Network Training with Levenberg–Marquardt and Adaptable Weight Compression. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *30*, 580–587. [CrossRef]
4. Han, H.; Liu, Z.; Barrios Barrios, M. Time series forecasting model for non-stationary series pattern extraction using deep learning and GARCH modeling. *J. Cloud Comput.* **2024**, *13*, 2. [CrossRef]
5. Zhang, Y. Deep learning models for price forecasting of financial time series: A review of recent advancements: 2020–2022. *WIREs Data Min. Knowl. Discov.* **2024**, *14*, e1519. [CrossRef]
6. Zhan, Z.; Kim, S.K. Versatile Time-Window Sliding Machine Learning Techniques for Stock Market Forecasting. *Artif. Intell. Rev.* **2024**, *57*, 209. [CrossRef]
7. May, B. A soft landing may be in sight, but expect turbulence. *Econ. Outlook* **2024**, *48*, 29–32. [CrossRef]
8. Jahan, I.; Sajal, S.Z.; Nygard, K.E. Prediction Model Using Recurrent Neural Networks. In Proceedings of the 2019 IEEE International Conference on Electro Information Technology (EIT), Brookings, SD, USA, 20–22 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–6. [CrossRef]
9. Pai, P.F.; Liu, C.H. Predicting Vehicle Sales by Sentiment Analysis of Twitter Data and Stock Market Values. *IEEE Access* **2018**, *6*, 57655–57662. [CrossRef]
10. Yang, F.; Chen, Z.; Li, J.; Tang, L. A Novel Hybrid Stock Selection Method with Stock Prediction. *Appl. Soft Comput.* **2019**, *80*, 820–831. [CrossRef]
11. Chouhan, S.S.; Kaul, A.; Singh, U.P. Image Segmentation Using Computational Intelligence Techniques: Review. *Arch. Comput. Methods Eng.* **2019**, *26*, 533–596. [CrossRef]

12. Adebiyi, A.A.; Adewumi, A.O.; Ayo, C.K. Comparison of ARIMA and Artificial Neural Networks Models for Stock Price Prediction. *J. Appl. Math.* **2014**, *2014*, 614342. [CrossRef]
13. Kristjanpoller, W.; Minutolo, M.C. A hybrid volatility forecasting framework integrating GARCH, artificial neural network, technical analysis and principal components analysis. *Expert Syst. Appl.* **2018**, *109*, 1–11. [CrossRef]
14. Sakhre, V.; Singh, U.P.; Jain, S. FCPN Approach for Uncertain Nonlinear Dynamical System with Unknown Disturbance. *Int. J. Fuzzy Syst.* **2017**, *19*, 452–469. [CrossRef]
15. Gu, S.; Kelly, B.; Xiu, D. Empirical Asset Pricing via Machine Learning. *Rev. Financ. Stud.* **2020**, *33*, 2223–2273. [CrossRef]
16. Zhou, X.; Zhou, H.; Long, H. Forecasting the Equity Premium: Do Deep Neural Network Models Work? *Mod. Financ.* **2023**, *1*, 1–11. [CrossRef]
17. Atsalakis, G.S.; Valavanis, K.P. Forecasting stock market short-term trends using a neuro-fuzzy based methodology. *Expert Syst. Appl.* **2009**, *36*, 10696–10707. [CrossRef]
18. Box, G.; Jenkins, G.; Reinsel, G.; Ljung, G. *Time Series Analysis: Forecasting and Control*; John Wiley & Sons: Hoboken, NJ, USA, 2015.
19. D'Angelo, G.; Tipaldi, M.; GlieL-Mo, L.; Rampone, S. Spacecraft autonomy modeled via Markov decision process and associative rule-based machine learning. In Proceedings of the 4th IEEE International Workshop on Metrology for AeroSpace, MetroAeroSpace 2017—Proceedings, Padua, Italy, 21–23 June 2017; pp. 324–329. [CrossRef]
20. Roodschild, M.; Gotay Sardiñas, J.; Will, A. A new approach for the vanishing gradient problem on sigmoid activation. *Prog. Artif. Intell.* **2020**, *9*, 351–360. [CrossRef]
21. Cui, L.; Yang, S.; Chen, F.; Ming, Z.; Lu, N.; Qin, J. A survey on application of machine learning for Internet of Things. *Int. J. Mach. Learn. Cybern.* **2018**, *9*, 1399–1417. [CrossRef]
22. Rong, G.; Mendez, A.; Assi, E.B.; Zhao, B.; Sawan, M. Artificial Intelligence in Healthcare: Review and Prediction Case Studies. *Engineering* **2020**, *6*, 291–301. [CrossRef]
23. Tang, Y.; Chau, K.-Y.; Li, W.; Wan, T. Forecasting Economic Recession through Share Price in the Logistics Industry with Artificial Intelligence (AI). *Computation* **2020**, *8*, 70. [CrossRef]
24. Manohar, B.; Das, R. Artificial Neural Networks for Prediction of COVID-19 in India by Using Backpropagation. *Expert Syst.* **2023**, *40*, e13105. [CrossRef]
25. Marquardt, D.W. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *SIAM J. Appl. Math.* **1963**, *11*, 431–441. [CrossRef]
26. Golovachev, S. Forecasting the U.S. Stock Market via Levenberg-Marquardt and Haken Artificial Neural Networks Using ICA&PCA Pre-processing Techniques. In *Pattern Recognition and Machine Intelligence*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 460–467. [CrossRef]
27. Manohar, B.; Das, R.; Lakshmi, M.; Prajapati, J.I.; Reddy, K.M. Artificial Neural Network-Based Stock Price Prediction Using Levenberg-Marquardt Algorithm. In *Artificial Intelligence Based Solutions for Industrial Applications*; CRC Press: Boca Raton, FL, USA, 2024; pp. 315–337.
28. Selvamuthu, D.; Kumar, V.; Mishra, A. Indian Stock Market Prediction Using Artificial Neural Networks on Tick Data. *Financ. Innov.* **2019**, *5*, 16. [CrossRef]
29. Goel, H.; Som, B.K. Stock market prediction, COVID-19 pandemic and neural networks: An SCG algorithm application. *Bus. Perspect. Res.* **2023**, *11*, 271–287. [CrossRef]
30. Zhou, R.; Wu, D.; Fang, L.; Xu, A.; Lou, X. A Levenberg–Marquardt Backpropagation Neural Network for Predicting Forest Growing Stock Based on the Least-Squares Equation Fitting Parameters. *Forests* **2018**, *9*, 757. [CrossRef]
31. Nawi, N.M.; Khan, A.; Rehman, M.Z. CSLM: Levenberg Marquardt based Back Propagation Algorithm Optimized with Cuckoo Search. *J. ICT Res. Appl.* **2013**, *7*, 103–116. [CrossRef]
32. Tiwari, A.K.; Jana, R.K.; Das, D.; Roubaud, D. Informational Efficiency of Indian Stock Market: Evidence from High-Frequency Data Using LSTM Model. *Phys. A Stat. Mech. Its Appl.* **2019**, *516*, 114–124. [CrossRef]
33. Dey, P.; Hossain, E.; Hossain, M.I.; Chowdhury, M.A.; Alam, M.S.; Hossain, M.S.; Andersson, K. Comparative Analysis of Recurrent Neural Networks in Stock Price Prediction for Different Frequency Domains. *Algorithms* **2021**, *14*, 251. [CrossRef]
34. Wei, W.; Wang, J. Stock Price Prediction Based on Information Entropy and Artificial Neural Network. *PeerJ Comput. Sci.* **2022**, *8*, e915. [CrossRef]
35. Yao, J.; Tan, C.L. A Case Study on Using Neural Networks to Perform Technical Forecasting of Forex. *Neurocomputing* **2000**, *34*, 79–98. [CrossRef]
36. Kara, Y.; Boyacioglu, M.A.; Baykan, Ö.K. Predicting Direction of Stock Price Index Movement Using Artificial Neural Networks and Support Vector Machines: The Sample of the Istanbul Stock Exchange. *Expert Syst. Appl.* **2011**, *38*, 5311–5319. [CrossRef]
37. Garrido, D.R.; Lorenzo, M.S. Application of the Sliding Window Method to the Short Range Prediction System for the Correction of Precipitation Forecast Errors. *Environ. Sci. Proc.* **2022**, *19*, 53. [CrossRef]

38. Thu, H.G.T.; Thanh, T.N.; Le Quy, T. A Neighborhood Deep Neural Network Model Using Sliding Window for Stock Price Prediction. In Proceedings of the 2021 IEEE International Conference on Big Data and Smart Computing (BigComp), Jeju Island, Republic of Korea, 17–20 January 2021; pp. 69–74. [CrossRef]
39. Wang, J.; Wang, J.; Zhang, Z.; Guo, S. Forecasting Stock Indices with Back Propagation Neural Network. *Expert Syst. Appl.* **2012**, *38*, 14346–14355. [CrossRef]
40. Atsalakis, G.S.; Valavanis, K.P. Surveying Stock Market Forecasting Techniques – Part II: Soft Computing Methods. *Expert Syst. Appl.* **2009**, *36*, 5932–5941. [CrossRef]
41. Ballings, M.; Van den Poel, D.; Hespeels, N.; Gryp, R. Evaluating Multiple Classifiers for Stock Price Direction Prediction. *Expert Syst. Appl.* **2015**, *42*, 7046–7056. [CrossRef]
42. Fischer, T.; Krauss, C. Deep Learning with Long Short-Term Memory Networks for Financial Market Predictions. *Eur. J. Oper. Res.* **2018**, *270*, 654–669. [CrossRef]
43. Teixeira, D.M.; Barbosa, R.S. Stock Price Prediction in the Financial Market Using Machine Learning Models. *Computation* **2025**, *13*, 3. [CrossRef]
44. Nelson, D.M.; Pereira, A.C.M.; de Oliveira, R.A. Stock Market's Price Movement Prediction with LSTM Neural Networks. *Expert Syst. Appl.* **2017**, *84*, 67–78. [CrossRef]
45. Cao, J.; Li, Z.; Li, J. Financial Time Series Forecasting Model Based on CEEMDAN and GRU. *Appl. Soft Comput.* **2021**, *101*, 107049. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.