

Article

An Enhanced Algorithm of RNN Using Trend in Time-Series

Dokkyun Yi ¹, Sunyoung Bu ² and Inmi Kim ^{1,*}

¹ DU College, Daegu University, Kyungsan 38453, Korea

² Department of Liberal Arts, Hongik University, Sejong 04066, Korea

* Correspondence: inmikim@gmail.com

Received: 29 May 2019; Accepted: 10 July 2019; Published: 12 July 2019



Abstract: The concept of trend in data and a novel neural network method for the forecasting of upcoming time-series data are proposed in this paper. The proposed method extracts two data sets—the trend and the remainder—resulting in two separate learning sets for training. This method works sufficiently, even when only using a simple recurrent neural network (RNN). The proposed scheme is demonstrated to achieve better performance in selected real-life examples, compared to other averaging-based statistical forecast methods and other recurrent methods, such as long short-term memory (LSTM).

Keywords: time series; trend; machine learning; RNN; LSTM

1. Introduction

In the modern age, a tremendous amount of time-series data, such as stock market fluctuations and average temperature per month per region, are generated and saved with each passing second. This trend is accelerating, showing no signs of slowing and, thus, many tools have been developed and used to extract useful information from such data for various purposes, such as for profit, estimation, detection, and future prediction. In particular, the prediction of information has been researched extensively and many statistical forecasting methods have been developed [1–10]. With growing interest in machine learning, similar developments in time-series-related recurrent neural networks (RNN) have been made, as can be found in [11–14].

As the recurrent neural network (RNN) has evolved, several branches, such as long short-term memory (LSTM) [15,16] or the gated recurrent unit (GRU), have been developed and implemented in many research areas and real-life applications. As LSTM and GRU reduce the weaknesses of RNN, such as the vanishing or exploding gradient problems [17–21], they have become widely used methods; however, even these new methods are not without their own problems, such as complexity of calculation. Much research on combinations of techniques, such as hybrid schemes [22], has been carried out and some preliminary studies [23–25] have utilized the concept of trend to make predictions. However, [24] assumed that the trend sets were already given and, thus, did not provide a specific method to obtain them.

The aim of this paper is to provide a new technique for the prediction of more accurate future trends, using a single-layer RNN structure for a given time-series. For this, first, a trend vector is derived from a given set of data whose components are length (duration), slope, and an adjustment part. Second, a non-trend data part is defined, which is equivalent to the difference between the values of the original data and the trend value defined in the previous step. Additionally, instead of using the original time-series as the input of the learning and training mechanism, the previously defined 1 by 3 vector-valued trend and the non-trend parts are used as the input for the learning process. Note that, for the most part of the paper, the simple RNN is the default training method, which was chosen to

emphasize the strength of trend methods; that is, even with one of the simplest learning schemes, the forecasting results are better than some variance of RNN learning methods, which is a motivation and strength of this work. This is also demonstrated by the examples and numerical experiments, in which the use of both trend and non-trend data parts results in better prediction performance. A brief explanation of this process is shown in Figure 1.

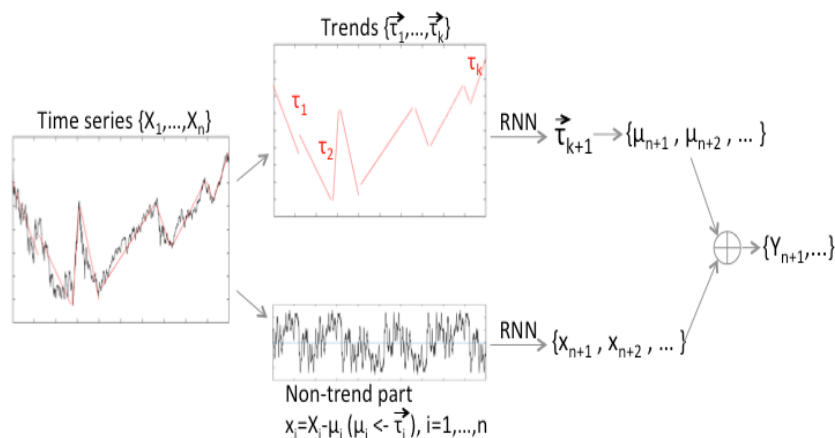


Figure 1. Trend Training method process. If the time i is included in the domain of j -th trend $\vec{\tau}_j$, then μ_i , the trend value at time i , is determined from the elements of $\vec{\tau}_j$.

2. Obtaining the Trend from Training Data

In this section, we explain a new approach to finding linear trends in a given time-series.

2.1. Definition of Trend

From the given data of a total learning data set $TLS = \{X_1, X_2, \dots, X_n\}$, where n is the length of the data, a set of linear trends $\{\vec{\tau}_1, \vec{\tau}_2, \dots, \vec{\tau}_k\}$, ($k < n$) is taken, where each trend $\vec{\tau}_j$ is a vector with three components, given as

$$\vec{\tau}_j = \langle \ell_j, s_j, \varepsilon_j \rangle, \quad (1)$$

where

$$\sum_{j=1}^k (\ell_j - 1) = n - 1. \quad (2)$$

and ℓ_j represents the duration of the j -th trend $\vec{\tau}_j$ in time domain; that is,

$$\ell_j = t_{\tau_{j+1}} - t_{\tau_j} + 1, \quad (3)$$

where t_{τ_k} represents the time value at which the k -th trend starts and s_j represents the (linear) slope of $\vec{\tau}_j$,

$$s_j = \arctan\left(\frac{X_{\tau_{j+1}} - X_{\tau_j}}{\ell_j}\right) \quad (4)$$

where X_{τ_k} is the first data value of the k -th trend. The set of data TLS is represented as k subsets LS_j s as

$$\cup_{j=1}^k LS_j = TLS, \quad (5)$$

where the intersection of LS_j and LS_{j+1} (i.e., $LS_j \cap LS_{j+1}$) is a single element on the boundary of the two trend lines. The elements of LS_j are data values with respect to the time domain $\{1 + \sum_{t=1}^{j-1} (\ell_t - 1), \dots, 1 + \sum_{t=1}^j (\ell_t - 1)\}$.

2.2. Defining the Trend Value μ_i

In detail, the durations $\ell_1, \ell_2, \dots, \ell_k$ and the slopes $s_1, s_2, \dots, s_k$ of all trends are calculated by the following technique: First, in the total learning set TLS (see Equation (5)), pick a maximum-valued element and a minimum-valued element and label them as X_{max_1} and X_{min_1} , respectively. If there are more than one occurrences of maximum or minimum, just choose one. In our algorithm, we chose the first one (i.e., at the smallest time index). Assume that $max_1 < min_1$ (note that the other case works in the same manner). Then, link the first data X_1 to X_{max_1} , X_{max_1} to X_{min_1} , and X_{min_1} to the last data X_n in the graph of the time-series TLS . From this first process, we have three subsets $\{LS_1^1, LS_2^1, LS_3^1\}$, where $LS_1^1 = \{X_1, \dots, X_{max_1}\}$, $LS_2^1 = \{X_{max_1}, \dots, X_{min_1}\}$ and $LS_3^1 = \{X_{min_1}, \dots, X_n\}$ as shown in Figure 2. The linear lines produced by linking the end points of LS_j^1 ($j = 1, 2, 3$) are the trends, with one dividing process.

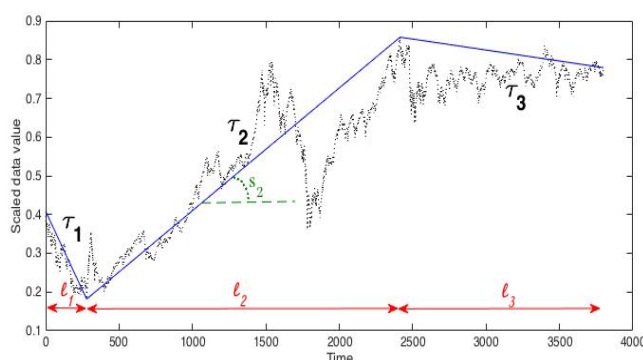


Figure 2. First step of calculating trends from data. The three linear lines represent the three trends τ_1 , τ_2 , and τ_3 . The first two components of a trend, the duration and the slope (defined by (7) and (8), respectively), are marked in the figure.

In Figure 2, Korea Composite Stock Price Index (Kospi) data (<http://kr.investing.com>) with a data length of 3800 was used as an example for showing how the trends are calculated. If the first or the last data is one of the maximum or minimum values, there are only one or two trends. The next step is to divide each LS_j^1 ($j = 1, 2, 3$) into one, two, or three subsets, through a similar process.

From now, we will change the data value into $\{X_i - \mu_i\}_{i=1, \dots, n}$ where μ_i is the value of the linear trend at $t = i$. For example, in $LS_1^1 = \{X_1, \dots, X_{max_1}\}$, μ_i is a trend value at time i for $i = 1, \dots, max_1$ and is determined from a linear function which connects X_1 and X_{max_1} ; that is, μ_i is calculated as

$$\mu_i = \begin{cases} X_1 & \text{if } i = 1 \\ \frac{X_{max_1} - X_1}{max_1 - 1}(i - 1) + X_1 & \text{if } 1 < i < max_1 \\ X_{max_1} & \text{if } i = max_1. \end{cases} \quad (6)$$

Note that μ_i is changed when dividing the subsets of LS .

Then LS_1^1 produces subsets LS_1^2, LS_2^2 and LS_3^2 with respect to the new indices max_2, min_2 , where max_2, min_2 represents the indices of the maximum and minimum data value of $\{X_i - \mu_i\}_{i=1, \dots, max_1}$, and so on. To stop reproducing subsets and complete the making the trend data set, a reasonable threshold is needed. This procedure keeps repeating until the difference between the maximum and the minimum value of a modified data subset is smaller than the threshold. In our experiments the threshold is chosen depending on the data set. If the threshold is not small enough, the trend value might not be able to represent the flow of the time-series data well and also the number of trends is too small to execute the learning with the trend set. Thus, to obtain a suitable number of trends, we need to have a smaller threshold. Figure 3 shows an example of using a threshold. A threshold of 0.5 was used for a Kospi data subset with length 3800. Finding linear trends, the blue lines, is done after scaling data by dividing by the maximum value of the total data set so the threshold

is less than 1. It is shown that with the threshold 0.5, the number of the trends of this data is 5 which is too small for the number of learning data set and not enough to catch the non-ignorable bumps in the time-series. For finding a reasonable threshold, more detailed explanation is described in the next section.

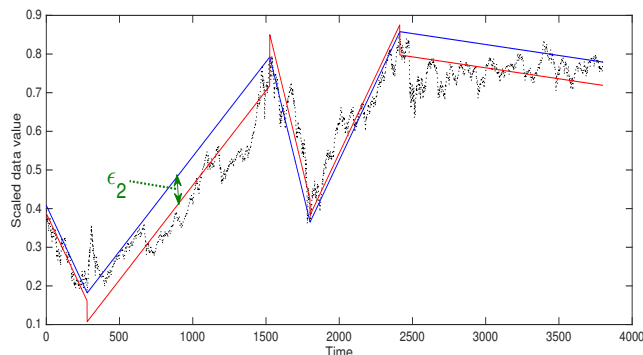


Figure 3. Obtaining trends from data (blue line) and creating new trends (red line) by adding the adjustment part ε_j for the j -th trend (given by (9)).

After obtaining the final set of linear trends, the j -th trend can be expressed as its duration ℓ_j and the slope s_j of the line for $j = 1, \dots, k$ where k is the number of trends.

2.3. Defining ℓ_j and s_j

Note that the trend between X_p and X_q lies on the time interval $[p, q]$. (i.e., $q - p + 1$ is the number of trend data) and the corresponding duration ℓ_j is defined as

$$\ell_j = q - p + 1. \quad (7)$$

Furthermore, the slope s_j of the trend can be defined as the angle from the slope $(X_q - X_p)/(q - p)$, where p and q are the indices of the endpoints of the trend. We use the radian value of the angle from the inverse tangent of the slope; that is,

$$s_j = \arctan\left(\frac{X_q - X_p}{q - p}\right), \quad (8)$$

which is a real number between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$. Examples of ℓ_j and s_j are shown in Figure 2.

2.4. Defining ε_j

The next step is to adjust and to update each trend set from a vector of two elements, ℓ_j and s_j , to a vector of three elements with the following rule: In the j -th trend (for example), for a time-series data value in the trend set $X_i \in LS_j$, calculate the expectation value of all $X_i - \mu_i$ and denote it by ε_j , where X_i is a data value in j -th trend τ_j . That makes

$$\sum_{X_i \in LS_j} (X_i - \mu_i - \varepsilon_j) = 0. \quad (9)$$

Define the j -th trend as

$$\vec{\tau}_j = \langle \ell_j, s_j, \varepsilon_j \rangle, \quad (10)$$

where $j = 1, \dots, k$. Now, this is equivalent to a linear function with slope s_j with duration ℓ_j from the first point of LS_j , and with ε_j subtracted; that is, define an adjusted trend value $\mu'_i = \mu_i + \varepsilon_j$, represented by $\vec{\tau}_j$ where $\{\vec{\tau}_i\}_{i=1, \dots, k}$ satisfies

$$\mathbb{E}(X_i - \mu'_i) = 0 \quad (11)$$

for all $X_i \in LS_j$. These are shown as red line segments in Figure 3.

Now, it remains to train a RNN on the two different time-series, in order to predict the next unknown data. One is the adjusted trend set $\{\vec{\tau}_1, \vec{\tau}_2, \dots, \vec{\tau}_k\}$ and the other is the non-trend data $\{x_i = X_i - \mu'_i\}_{i=1, \dots, n}$. By learning the trend set, we can predict the length and duration of the next trend. The expected value of the non-trend part is trained. The algorithm, so far, can be expressed as the following.

3. Finding a Threshold

In this section, we describe a method to obtain a reasonable threshold for Algorithm 1. As mentioned in the previous section, subsets of given data are divided until the biggest difference between the divided trend and the data is less than the threshold.

Algorithm 1: Obtaining the Trend Set

1. Set an index set as $Index = \{1, n\}$, where n is the number of the training data.
 2. Find max_1, min_1 such that $\max(X_1, \dots, X_n) = X_{max_1}, \min(X_1, \dots, X_n) = X_{min_1}$ and add these to the $Index$ set (i.e., $Index = \{1, max_1, min_1, n\}$). Get rid of duplicate indices and sort the set.
 3. For $LS_i^1 = \{X_{Index(i)}, \dots, X_{Index(i+1)}\}$ for $i = 1, \dots, (\#ofIndex - 1)$, link $X_{Index(i)}$ and $X_{Index(i+1)}$ and let $\mu^1(j)$ be the value of the linked linear line.
 4. For each $\{X_j - \mu_j\}, j = 1, \dots, n$, find max and min indices and repeat Steps 2 and 3.
 5. Repeat Step 4 until $|X_{max} - X_{min}| \leq threshold$ for a pre-chosen threshold.
-

Before obtaining trends of real data, such as a financial series or temperature data, let us consider some examples of data. As periodicity is an important factor among the many characteristics of data, we consider periodic and non-periodic example data to figure out suitable thresholds for each type of data. For the periodic example, consider $X_i = \sin(\frac{i}{30})$ for $i = 1, \dots, n$ and, for the non-periodic one, consider $X_i = \exp(-\frac{i}{900})$ for $i = 1, \dots, n$. Note that the data is used after scaling by the maximum absolute value of the data, such that it is bounded in $[-1, 1]$. After applying the dividing domain methods mentioned in the previous section with a very small threshold (say, 0.00001), we obtain the LS_i s; the subsets of the total learning set $TLS = \{X_i\}_{i=1, \dots, n}$.

By varying the different number of data n between 1000, 2000, 3000, and 4000, we can make observations about the variance of the difference between data and the trend per division of the trends. If μ'_i ($i = 1, \dots, n$) represents an adjusted trend value after a certain amount of trend division, we define the variance as

$$variance = \frac{1}{\#TLS} \sum_{X_i \in TLS} (X_i - \mu'_i)^2, \quad (12)$$

where X_i is an element at time i in the total learning set and μ_i is the corresponding trend value at time i .

Figure 4 plots the variance of two example data per trend division number as the number of data was increased by 1000. As the trigonometric function had a proportional number of extremes to the total length data, the trend division number was also proportional to the total length. We can observe that the first variance (i.e., the variance after the first trend division) was similar after increasing the size of data and, also, that the final variance barely changed, even when the length of the data was increased. However, for non-periodic data (the exponential function, in our example), the first variance increased as the size of the data increased. It took a lot of divisions to approach the goal after getting near the final variance (with a small threshold 0.00001) for the non-periodic data, while the periodic data stopped trend division shortly after getting near the final variance.

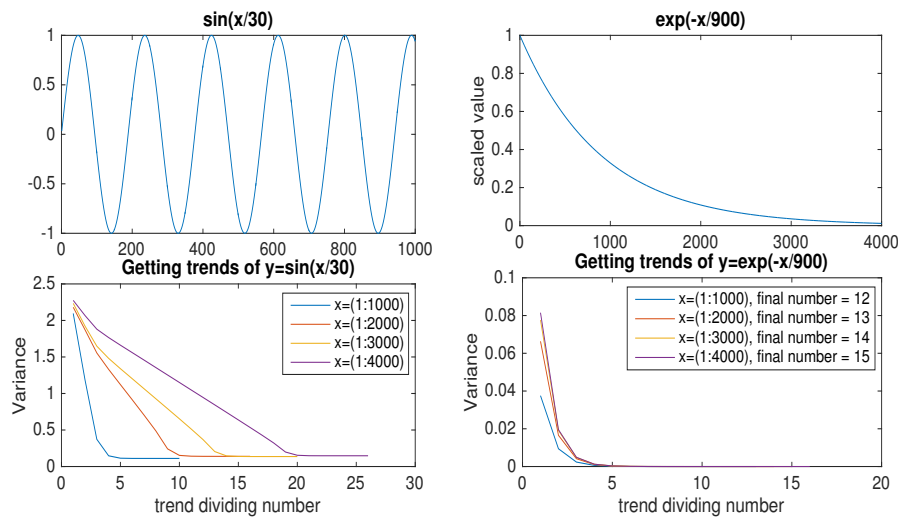


Figure 4. Sine and exponential function data and calculation results of variance of the difference between the data and the trend value per trend division. In the sine function data, the trend division process stops when the next size start to converges. In the exponential function data, data of different size converges quickly in a similar amount of time but took a long time to finish the division process.

From this observation, we can get a hint if the given data is periodic or not by calculating variance per trend division with different sized sets of the data (i.e., we can do a pre-processing to get the periodicity of a data). After pre-processing, if a data is periodic or periodic-like, we can choose a small portion of the data and choose the threshold depending on the amount of data difference one wants to ignore when setting trends.

The above method suggests a way to find a suitable threshold; however, changing threshold results in different trend lengths which may affect interpretation of various aspects of the data. The length of the trend does not affect the accuracy of learning, so using different thresholds with various goals of forecasting is possible.

4. Learning the Trend

In this section, we want to train a set $\{\vec{\tau}_1, \vec{\tau}_2, \dots, \vec{\tau}_k\}$, where $\vec{\tau}_j$ is a 1×3 vector, for $j = 1, \dots, k$, where k is the number of trends (see Equation (10)). As we have a 1×3 vector $\vec{\tau}_t$ as the t -th input of the simple RNN process, the dimension of $\vec{h}_t$ and $\vec{y}_t$, which are the t -th hidden state and the output value of trend, respectively, can be set as vectors of size 1×3 , as shown in the following equations:

$$\vec{y}_t = W_y^T \vec{h}_t + \vec{b}_y^T, \quad (13)$$

$$\vec{h}_t = \tanh(W_h^T \vec{h}_{t-1} + W_\tau^T \vec{\tau}_t + \vec{b}_h^T), \quad (14)$$

where W_y^T, W_h^T and W_τ^T are in $\mathbb{R}^{3 \times 3}$ and $\vec{b}_y^T$ and $\vec{b}_h^T$ are in $\mathbb{R}^{1 \times 3}$. Note that we need to calculate the matrix–vector multiplications in the opposite order; for example, we calculate $W_y^T \vec{h}_t$ as $\vec{h}_t \cdot W_y^T$ to match the dimension of the result vector. The goal of this training is to get an approximated set of weights and biases which minimise the cost function

$$cost = \frac{1}{train^\tau - seq^\tau} \sum_{i=seq^\tau}^{train^\tau-1} \|\vec{y}_i - \vec{\tau}_{i+1}\|_2^2 \quad (15)$$

$$= \frac{1}{train^\tau - seq^\tau} \sum_{i=seq^\tau}^{train^\tau-1} \|W_y^T \tanh(W_h^T \vec{h}_{i-1} + W_\tau^T \vec{\tau}_i + \vec{b}_h^T) + \vec{b}_y^T - \vec{\tau}_{i+1}\|_2^2, \quad (16)$$

where $\|\cdot\|_2$ is the ℓ_2 -norm, $train^\tau$ is the number of trends for training (which is k , in this case), and seq^τ is the sequence length for trend training. The cost is defined as the average value of all differences squared, as shown in (15). We use the Adam optimisation method to minimise this cost and get the optimum weight and bias sets [26].

Once a trend set is well-trained, a prediction of the next trend $\vec{\tau}_{k+1} = \langle \ell_{k+1}, s_{k+1}, \varepsilon_{k+1} \rangle$ can be generated, which is close to the real future data.

5. Learning the Non-Trend Part

In the previous section, we obtained a prediction of trend which adequately predicts the flow of the data. Now, the remaining oscillating, noise-like part is the non-trend part of the data. There are two different methods of forecasting suggested in this paper. One is considering the non-trend data as a time-series with same length as the original data and using the same scheme, simple RNN, with trend training. The other is calculating the expected bounds of non-trend part from the minimum and the maximum value of non-trend part in each trend.

5.1. Simple RNN Method

First, the non-trend part can be also trained using the simple RNN process and produce a set of prediction values $\{y_t\}$ by the following equations,

$$y_t = w_y h_t + b_y, \quad (17)$$

$$h_t = \tanh(w_h h_{t-1} + w_x x_t + b_h), \quad (18)$$

and this result can be added to the obtained trend. It is identical to the simple RNN process, but the input data is given as the original scaled data with the trend prediction from the previous process subtracted; that is,

$$x_t = X_t - \mu'_t, \quad (19)$$

where μ'_t ($t = 1, \dots, n$) represents the data value of the corresponding trend $\vec{\tau}_i$ in the trend set $\{\vec{\tau}_i\}_{i=1, \dots, k}$. Note that the number of predictions of the non-trend part is limited to the length of the predicted trend, since we are only interested in the movement of the data in that trend. For instance, in the previous section, after the trend training, the $(k+1)$ -th trend $\langle \ell_{k+1}, s_{k+1}, \varepsilon_{k+1} \rangle$ is obtained. So, only ℓ_{k+1} number of predictions $\{y_{n+1}, y_{n+2}, \dots, y_{n+\ell_{k+1}}\}$ are obtained from the non-trend training.

Combining this non-trend part and the trend training result from the previous subsection, the final prediction is obtained as

$$Y_t = \mu'_t + y_t, \quad (20)$$

where μ'_t is the data value of $\vec{\tau}_{k+1}$ and y_t is the non-trend prediction result for $t = 1, \dots, \ell_{k+1}$. The algorithm, so far, can be expressed by the following algorithm (Algorithm 2). Note that, in Step 1, in the form of the trend, the third element ε_i is a value obtained to make the average of the data with the trend subtracted zero in each trend. Also, note that, in Step 3, it uses the optimisation scheme with three variables.

Algorithm 2: Forecasting by Learning Trend and Non-trend Parts

1. Take a trend set of the data with a form of $\langle \text{length}, \text{slope}, \text{error} \rangle$ (i.e., $\vec{\tau}_i = \langle \ell_i, s_i, \varepsilon_i \rangle, i = 1, \dots, k$ for some k).
2. Obtain the hidden layer values and output values after the simple RNN process (as in Equation (13)).
3. Perform the Adam optimisation scheme with suitable iteration number to get a minimum of the trend cost (Equation (15)).
4. From Step 3, get a prediction of upcoming trend after the training data set $\vec{\tau}_{k+1} = \langle \ell_{k+1}, s_{k+1}, \varepsilon_{k+1} \rangle$.
5. Take a non-trend set of the data by subtracting the original data by the trend value from Step 1.
6. Get hidden layer values and output values after simple RNN process with one variable input.
7. Perform the Adam optimisation scheme with suitable iteration number to get a minimum of the non-trend cost.
8. From Step 7, get predictions of the non-trend values. Note that the number of predictions is equivalent to the length ℓ_{k+1} of the predicted trend $\vec{\tau}_{k+1}$.
9. Combine the results from Steps 4 and 8.

5.2. Bound Training Method

Another method for dealing with the non-trend part is by training the bounds of the non-trend value in each trend. First, we calculate the maximum and minimum values of $X_j - \mu'_j$ in each trend for $j = 1, \dots, n$ where X_j is the data value and μ'_j represents the trend value for a trend length n . Then, there are k pairs of maximum and minimum values (i.e., $\{(max_i, min_i)\}_{i=1, \dots, k}$, where k is the number of trends of the data). This can be considered as a time-series as well, so the RNN can be used again with this series of vectors with two elements. The hidden state vector $\vec{y}^{nt}_t$ and the output value vector $\vec{h}^{nt}_t$ for non-trend part are both 1×2 vectors and satisfy the following equations

$$\vec{y}^{nt}_t = W_y^{nt} \vec{h}^{nt}_t + \vec{b}_y^{nt}, \quad (21)$$

$$\vec{h}^{nt}_t = \tanh(W_h^{nt} \vec{h}^{nt}_{t-1} + W_{n\tau}^{nt} \vec{\tau}_t + \vec{b}_h^{nt}), \quad (22)$$

where the weight matrix for non-trend part W_y^{nt} , W_h^{nt} , and $W_{n\tau}^{nt}$ are in $\mathbb{R}^{2 \times 2}$ and the bias vector for the non-trend part $\vec{b}_y^{nt}$ and $\vec{b}_h^{nt}$ are in $\mathbb{R}^{1 \times 2}$ for $t = 1, \dots, k$. As a result of optimisation of the above equations, predictions for the maximum and the minimum for the non-trend values are produced.

6. Numerical Experiments

In this section, some numerical results of the proposed schemes—RNNs for trend, RNNs for non-trend, and the combination of these two—are shown through two examples of practical data.

6.1. Stock Market Index Data

The first example is the Korea Composite Stock Price Index (Kospi) closing price data, which was used in Section 2. In this experiment, the number of the data was over 4200. As shown in Figure 5, pre-processing of this data gave the flow of variance of the non-trend part by trend division per length of data. The variance value converged fast and remained near convergence for over 70 percent of the trend division time; this behavior of convergent time was the same for all lengths of regularly chopped data. This is similar to the result for the exponential function data in Section 3, so the Kospi data can be considered to be a non-periodic data type. Thus, the choice of a threshold for the trend division should be small (i.e., 0.00001), in order to catch the number of trends.

The number of trends with threshold 0.00001 was 274 and the threshold which gave the closest number of trends was 0.05. So, for this experiment, the training data length was 4000 and the threshold

was 0.05. In Figure 6, the data and its trend are given in the first graph and the (decreasing) trend of learning error in the second graph. This shows that the trend learning worked well. With this threshold, the number of trends was 178 and (as we used 1/10 of the number of trends) 17 was the sequence length. A total of 2800 iterations were used for the learning of this trend set.

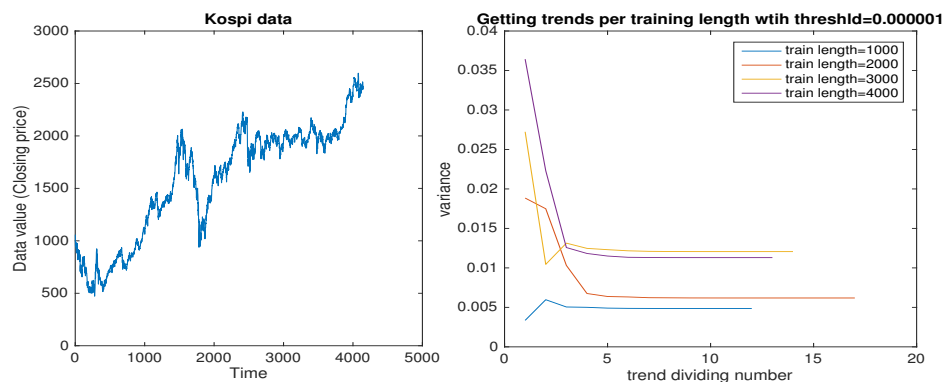


Figure 5. Kospi data showing non-periodic variance pattern in its convergence speed.

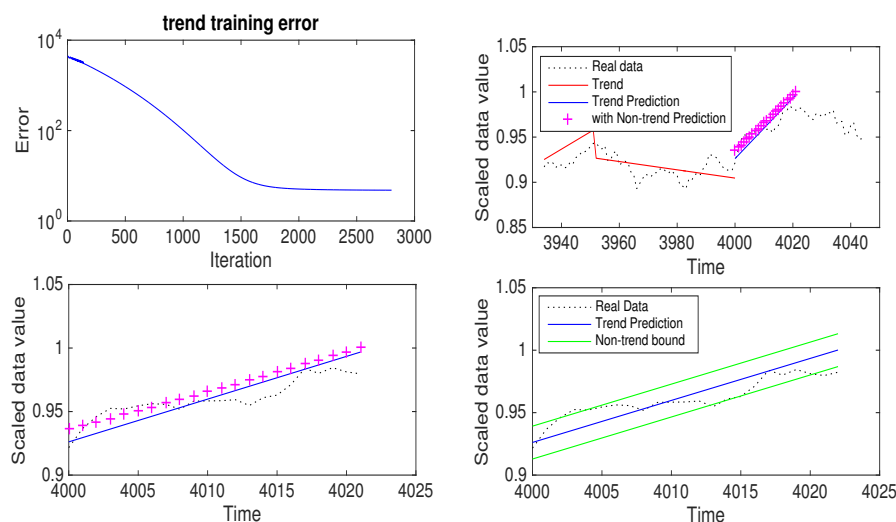


Figure 6. Training error decrease for trends in the Kospi data and prediction of the trend and non-trend parts. The third graph is a close-up of a section of the second graph. The last graph shows the forecasted bounds from the second scheme for the non-trend forecasting method.

After learning the trend, it predicted the next trend which turned out to be 22, 0.003364, and 0.004562 as the duration, slope, and adjustment, respectively. See the third and fourth graphs in Figure 6 for closer look at the prediction part. We can see that this scheme worked well in finding the next trend. For the non-trend part, the training length was the same as the total data number used to obtain the trends. The sequence length of the non-trend part was decided as the floor function of the length of the training data divided by the length of the trend obtained from the previous step. So, for this example, the sequence length of the non-trend training was $\lfloor 4000/178 \rfloor = 22$. The number of iterations for the non-trend part learning was 3000.

6.2. Temperature Data

The next data set was temperature data. It is the monthly average temperature measured in Seoul, South Korea and the data length was over 24,000. Similar to the the previous data, Figure 7 shows the variance information of the non-trend part as changing trends per length of regularly chopped data. From this graph, the convergent time of variance for each data length varied and was

proportional to the length of the training data. Given this pre-processing result, the temperature data was determined to be periodic/periodic-like and, so, the threshold for obtaining trends was chosen as needed. In this experiment, a threshold of 0.5 suited our purposes. Thus, a training data length of 23,000 and a threshold of 0.5 were used. Then, 988 trends were obtained with this threshold and, so, the sequence length for trend learning was set to 98, calculated similarly to the previous case. In Figure 8, the data and its trends are shown in the first graph and the decreasing trend learning error in the second graph. We can see that the trend learning worked well. With a sequence length of 98 and 23,000 iterations, the trend learning gave that the duration, slope, and adjustment of the next trend were 23, -0.012443 , and -0.000828 respectively. In Figure 8, this result is shown visually, where the fourth graph shows a closer look of the prediction part. For the non-trend part, 5000 iterations were used.

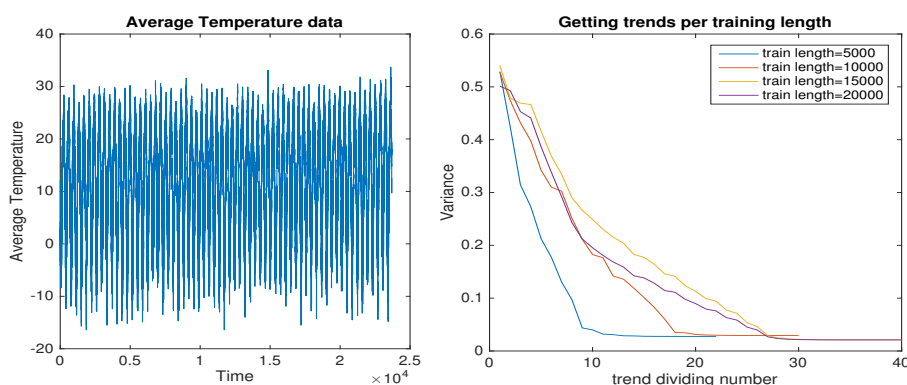


Figure 7. The average temperature data of Seoul, South Korea shows a periodic variance pattern.

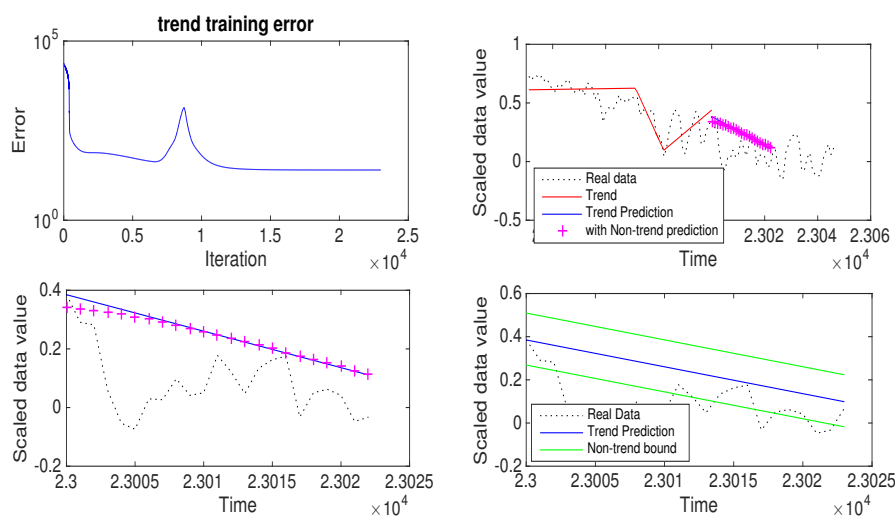


Figure 8. Training error decrease for trends in the average temperature data of Seoul, South Korea and prediction of the trend and non-trend parts. The third graph is a close-up of a section of the second graph. The last graph shows the forecasted bounds from the second scheme for the non-trend forecasting method.

7. Comparison with Other Methods in the Numerical Experiments

In this section, we will compare the detection results of our scheme with other existing forecasting methods.

7.1. Comparison with Other Trend Methods

There are many statistical methods and techniques for the forecasting of time-series. Among these methods, averaging-based methods are typically used to compare forecasting results. The first method used was the single exponential smoothing method [27]. The smoothed value is given by

$$S_{t+1} = \alpha y_{origin} + (1 - \alpha)S_t, 0 < \alpha \leq 1, t \geq 3, \quad (23)$$

where y_{origin} remains constant for all future forecast calculations, which is known as bootstrapping [28]. The parameter α is determined by minimizing the mean squared error (MSE). We used this bootstrapping forecasting method for comparison. See Figures 9 and 10 for the comparison forecasting results of the Kospi and temperature data, respectively. The thin pink line is the data smoothed with the single exponential method and the thick pink line represents the forecast value using the bootstrap method.

The next averaging method we used was double exponential smoothing [29]. It uses two constants and is known to handle trends more efficiently. The equation is obtained as follows

$$S_t = \alpha y_t + (1 - \alpha)(S_{t-1} + b_{t-1}) \quad (24)$$

$$b_t = \gamma(S_t - S_{t-1}) + (1 - \gamma)b_{t-1}, \quad (25)$$

where α and γ are determined using non-linear optimisation techniques. The m -period-ahead forecast is obtained by

$$F_{t+m} = S_t + mb_t. \quad (26)$$

In Figures 9 and 10, the thin blue line represents the double exponential smoothed data and the thick blue line shows the linear forecasting result F_{t+m} .

In both data examples, the red line shows the predicted trend with our proposed scheme and the series of points marked with + indicates the trend prediction added by the non-trend part predictions. It can be observed, in both figures, that our proposed scheme detected the real data more accurately. See Table 1 for the calculated mean squared errors of each method, for both Kospi and temperature data sets, in Figures 9 and 10.

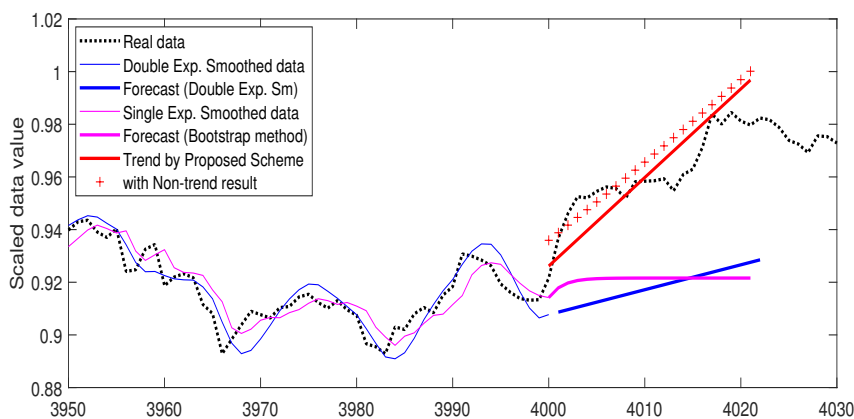


Figure 9. Comparison with smoothing methods: Kospi data.

Table 1. Comparison of mean squared errors (MSEs) with exponential smoothing methods.

	Kospi	Temperature
Single exponential smoothing method	0.0018	0.1057
Double exponential smoothing method	0.0020	0.1958
Proposed scheme	1.2329×10^{-4}	0.0372

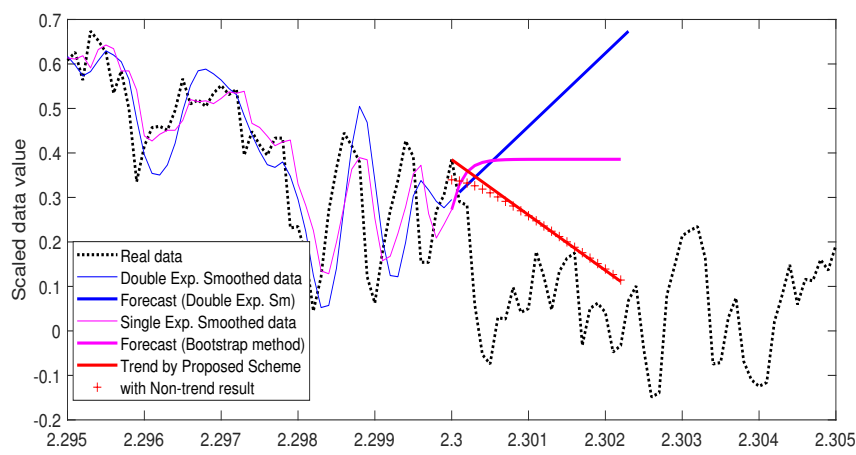


Figure 10. Comparison with smoothing methods: Temperature data.

7.2. Comparison with Other Learning Methods

In this sub-section, we compare our proposed method with other learning schemes, such as simple RNN and simple LSTM (both with a single layer). We used a simple version of these schemes on the Kospi data, and the prediction results are shown in Figure 11. With the same training number (4000), the result of LSTM popped up and down, back to around 1.35 at the time 4007, whereas the result of RNN stayed around the real data value. Thus, our result (the red line in the Figure 11), which was closest to the real data, gave the best prediction among these methods. See Table 2 for a comparison of the mean squared errors of each learning method. A comparison using the temperature data gave a similar result, as this is from the simple RNN variances which have some issues (such as vanishing gradient).

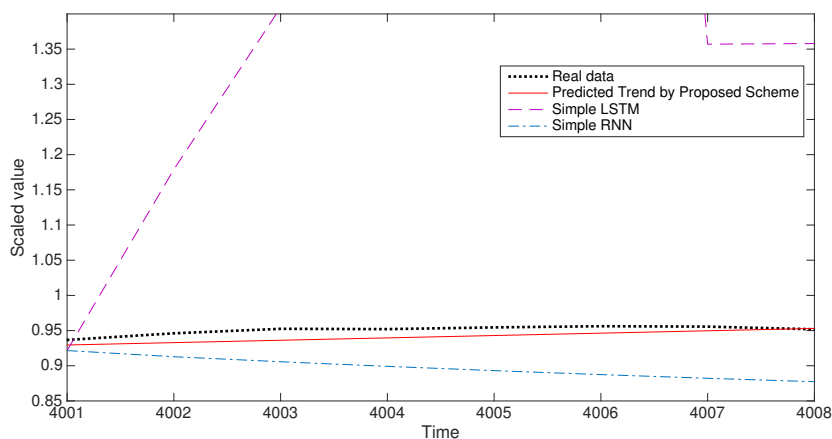


Figure 11. Comparison with other learning methods: Kospi data.

Table 2. Comparison of MSEs with other learning methods.

	Kospi
Simple RNN	0.0105
Simple LSTM	0.4024
Proposed scheme	1.2329×10^{-4}

8. Conclusions and Future Works

In this paper, we introduced a new algorithm to predict future data from given time-series data by finding trends in the data set which fit into the given data and, then, applying existing simple recurrent

neural network learning to them. Here, we propose a technique to collect and find the trend set with three pieces of information—slope, duration, and adjustment. Once the trends set has been collected, it is trained by a simple RNN, such that future trends can be predicted. Additionally, to increase the accuracy of the prediction, we considered the non-trend part in the time-series. A simple RNN was also trained on the non-trend parts and the results were added to the prediction of the trends.

Through several numerical simulations, it was shown that the collected trends can fit the given time-series data and are also well-trained, such that the prediction of future trends is accurate, compared to the real data. Furthermore, the accuracy could be increased by adjusting the trend prediction by training on the non-trend part. Note that all training simulations were executed by a simple, single-layer RNN, due to hardware limitations. Hence, in order to fully explore the efficiency of the proposed algorithm, we will combine the proposed algorithm with LSTM or GRU, instead of using only a simple RNN, since those variants of RNN are well-known to be better predictors in RNN structures [30,31].

Author Contributions: Conceptualization, D.Y.; methodology, I.K.; software, I.K.; validation, D.Y., S.B., and I.K.; resources, S.B.; data curation, S.B. and I.K.; writing—original draft preparation, I.K.; writing—review and editing, S.B.; supervision, D.Y.; project administration, D.Y.; funding acquisition, D.Y. and S.B.

Funding: This research was funded by the basic science research program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (grant number NRF-2017R1E1A1A03070311). The second author Bu was partly supported by the basic science research program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (grant number 2019R1F1A1058378).

Acknowledgments: The authors would like to express our sincere gratitude to Peter Jeonghun Kim of Samsung Electronics for invaluable comments about the new testing schemes and improving the use of English in the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. 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. [\[CrossRef\]](#)
2. Box, G.; Jenkins, G.; Reinsel, G. *Time Series Analysis: Forecasting and Control*; Prentice Hall: Upper Saddle River, NJ, USA, 1994.
3. Dangelmayr, G.; Gadaleta, S.; Hundley, D.; Kirby, M. Time series prediction by estimating markov probabilities through topology preserving maps. In *Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation II*; International Society for Optics and Photonics: Bellingham, WA, USA, 1999; Volume 3812, pp. 86–93.
4. Graves, A.; Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Netw.* **2005**, *18*, 602–610. [\[CrossRef\]](#) [\[PubMed\]](#)
5. Hosseini, S.; Khaled, A.A.; Vadamani, S. Hybrid imperialist competitive algorithm, variable neighborhood search, and simulated annealing for dynamic facility layout problem. *Neural Comput. Appl.* **2014**, *25*, 1871–1885. [\[CrossRef\]](#)
6. Jin, Z.; Zhou, G.; Gao, D.; Zhang, Y. EEG classification using sparse Bayesian extreme learning machine for brain—Computer interface. *Neural Comput. Appl.* **2018**, 1–9. [\[CrossRef\]](#)
7. Keogh, E.J. A decade of progress in indexing and mining large time series databases. In Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, 12–15 September 2006; pp. 1268–1268.
8. Khaled, A.A.; Hosseini, S. Fuzzy adaptive imperialist competitive algorithm for global optimization. *Neural Comput. Appl.* **2015**, *26*, 813–825. [\[CrossRef\]](#)
9. Zhang, X.; Yao, L.; Wang, X.; Monaghan, J.; McAlpine, D.; Zhang, Y. A Survey on Deep Learning based Brain Computer Interface: Recent Advances and New Frontiers. *arXiv* **2019**, arXiv:1905.04149.
10. Zhang, Y.; Wang, Y.; Zhou, G.; Jin, J.; Wang, B.; Wang, X.; Cichocki, A. Multi-kernel extreme learning machine for EEG classification in brain-computer interfaces. *Expert Syst. Appl.* **2018**, *96*, 302–310. [\[CrossRef\]](#)
11. Elman, J.L. Finding structure in time. *Cognit. Sci.* **1990**, *14*, 179–211. [\[CrossRef\]](#)

12. Mueen, A.; Keogh, E. Online discovery and maintenance of time series motifs. In Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 25–28 July 2010; pp. 1089–1098.
13. Schmidhuber, J. A Local Learning Algorithm for Dynamic Feedforward and Recurrent Networks. *Connect. Sci.* **1989**, *1*, 403–412. [[CrossRef](#)]
14. Werbos, P.J. Generalization of backpropagation with application to a recurrent gas market model. *Neural Netw.* **1988**, *1*, 339–356. [[CrossRef](#)]
15. Gers, F.A.; Schraudolph, N.N.; Schmidhuber, J. Learning Precise Timing with LSTM Recurrent Networks. *J. Mach. Learn. Res.* **2002**, *3*, 115–143.
16. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [[CrossRef](#)] [[PubMed](#)]
17. Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **1994**, *5*, 157–166. [[CrossRef](#)] [[PubMed](#)]
18. Pascanu, R.; Mikolov, T.; Bengio, Y. On the difficulty of training recurrent neural networks. In Proceedings of the 30th International Conference on Machine Learning (ICML 2013), Atlanta, GA, USA, 16–21 June 2013; pp. 1310–1318.
19. Rohwer, R. The moving targets training algorithm. In *Advances in Neural Information Processing Systems 2*; Touretzky, D.S., Ed.; Morgan Kaufmann: San Matteo, CA, USA, 1990; pp. 558–565.
20. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [[CrossRef](#)]
21. Schmidhuber, J. A Fixed Size Storage $O(n^3)$ Time Complexity Learning Algorithm for Fully Recurrent Continually Running Networks. *Neural Comput.* **1992**, *4*, 243–248. [[CrossRef](#)]
22. Xu, X.; Ren, W. A Hybrid Model Based on a Two-Layer Decomposition Approach and an Optimized Neural Network for Chaotic Time Series Prediction. *Symmetry* **2019**, *11*, 610. [[CrossRef](#)]
23. Afolabi, D.; Guan, S.; Man, K.L.; Wong, P.W.H.; Zhao, X. Hierarchical Meta-Learning in Time Series Forecasting for Improved Inference-Less Machine Learning. *Symmetry* **2017**, *9*, 283. [[CrossRef](#)]
24. Lin, T.; Guo, T.; Aberer, K. Hybrid Neural Networks for Learning the Trend in Time Series. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17), Melbourne, Australia, 19–25 August 2017; pp. 2273–2279.
25. Wang, P.; Wang, H.; Wang, W. Finding semantics in time series. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, Athens, Greece, 12–16 June 2011; pp. 385–396.
26. Kingma, D.P.; Ba, J.L. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference for Learning Representations (ICLR 2015), San Diego, CA, USA, 7–9 May 2015.
27. Brown, R.G. *Smoothing, Forecasting and Prediction*; Prentice Hall: Upper Saddle River, NJ, USA, 1962.
28. Efron, B.; Tibshirani, R. *An Introduction to the Bootstrap*; Chapman & Hall/CRC: Boca Raton, FL, USA, 1993.
29. Prajakta, S.K. *Time Series Forecasting Using Holt-Winters Exponential Smoothing*; Kanwal Rekhi School of Information Technology: Mumbai, India, 2004; pp. 1–3.
30. Cho, K.; Merriënboer, B.V.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv* **2014**, arXiv:1406.1078.
31. Cho, K.; Merriënboer, B.V.; Bahdanau, D.; Bengio, Y. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *arXiv* **2014**, arXiv:1409.1259.

