Forecasting Spare Part Reordering using Artificial Intelligence

Muhamad Hairil Fizy Md Dzahir[1], Choi Kai Sheng[1], Mohammad Ikmal Abdul Amir[1], Muhamad Syafiq Taqiuddin Shafeai[1], Farah Izzati Abdul Hamid[1], Mohd. Hafis Izran Ishak[1*]

[1]Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, Malaysia

Corresponding author* email: hafis@fke.utm.my
Available online 30 December 2024

ABSTRACT

This project is about forecasting spare parts for a company. Artificial Intelligence is used to make a prediction for the spare part and programming language used was Python and framework used was TensorFlow. TensorFlow is a framework mainly used for Artificial Intelligence since it contains library for it. By using Python and TensorFlow, a forecast model was created to predict the number of spare parts need to buy in the next few months.

Keywords: Long Short-Term Memory, Python, Recurrent Neural Network, Tensor Flow.

1. Introduction

Artificial Intelligence (AI) is an approach to make a computer, a robot or a device mimic on how humans think. AI is the study of how human brain think, learn, decide and work when it tries to solve some problems. AI is a tool that helps human simplify things and solve problems. Even though AI is an advanced thing, not all situations we can use AI since AI is only suitable to use when there is a lot of data. One of the applications of AI is used for prediction of something. A pool of data is used to train an AI and make a prediction from it.

2. Recurrent Neural Network (Long Short-Term Memory)

2.1 Introduction

Recurrent neural networks (RNNs) are a connectionist model containing a self-connected hidden layer. One advantage of the recurring connection is that a 'memory' of previous inputs remains in the internal state of the network, allowing it to use past context. Another significant benefit of recurrency is that the rate of change of the internal state can be precisely modulated by the recurrent weights, which builds in robustness to localised fluctuations of the input data [2]. Unfortunately, it is well known that recurrent networks are difficult to train, and thus the full potential of recurrent models is unlikely to be demonstrated. The Long Short-Term Memory (LSTM) neural network architecture addresses these issues. [1]

The range of contextual information that can be accessed by standard RNNs is quite limited in practice. The problem is that the effect of a given input on the hidden layer and therefore on the output of the network either decay or exponentially blows up as it loops through the repeated connections of the network. This shortcoming, vanishing gradient problem, makes it difficult for an RNN to bridge gaps between relevant input and target events of more than about 10 time steps. The vanishing gradient problem is illustrated schematically in Figure 1.

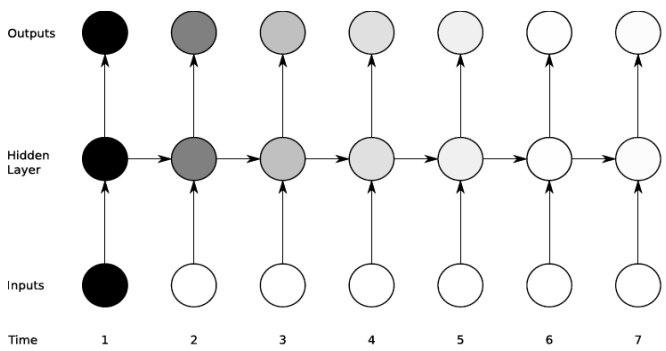

Figure 1. The vanishing gradient problem illustration

Long Short-Term Memory is an RNN architecture specifically designed to address the vanishing gradient problem. An LSTM hidden layer consists of memory blocks. Memory blocks are recurrently connected subnets. Three multiplicative gates control activation of each block that contains a set of internal units, or cells. These three multiplicative gates are the input gate, forget gate and output gate.

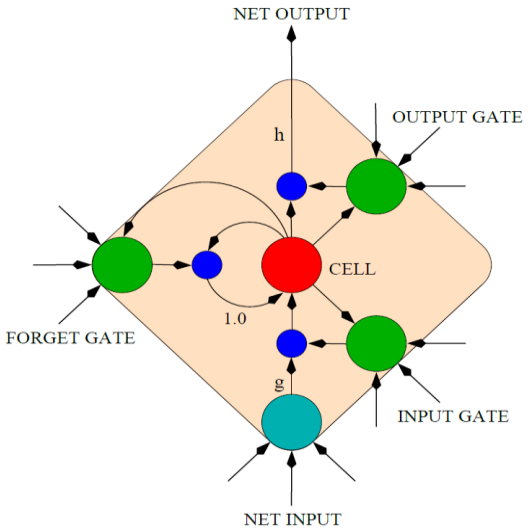

Figure 2. LSTM memory block with a single cell provides a detailed illustration of an LSTM memory block with one cell

The gates allow the cells to store and access information for long periods of time. For example, the activation of the cell will not be overwritten by the new inputs as long as the input gate remains closed. Similarly, the cell activation is available to the rest of the network only when the output gate is open, and the recurrent connection of the cell is switched on and off by the forget gate.

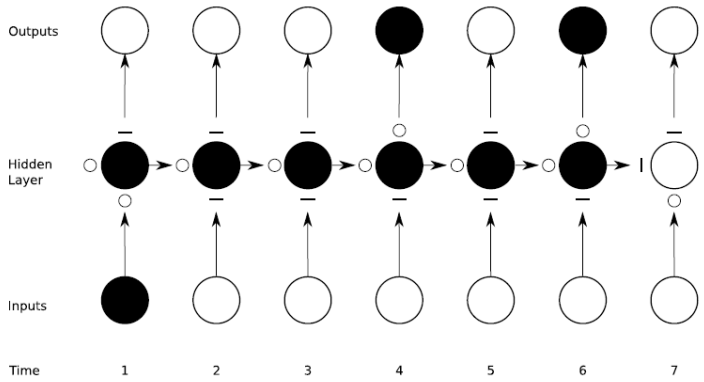

Figure 3. Preservation of gradient information by LSTM

Figure 3 illustrates how an LSTM block maintains gradient information over time. When the forget gate is open and the input gate is closed, the dependency is 'carried' by the memory cell. The output dependency can be switched on and off by the output gate without affecting the hidden cell. [2]

2.2 Models

The application that is used in this work is the time series prediction method and Recurrent Neural Network (RNN) structures. These structures are effective to compare and predict the value of the next moment [3]. It makes more sense to expect an intelligent system to predict weather or energy changes over a period of time. The time series prediction has attracted wide attention in the research community [4]. For this work, this application is used to get the current spare part and make a long-term prediction for the spare part. It can easier the user to make reordering for the future. RNN methods commonly focus on solving one aspect of dynamic Spatio-temporal relationships. Hence, these methods are impossible to achieve accurate and robust long-term predictions of multivariate time series [5]. Moreover, attention-based RNNs are

used to effectively represent and learn temporal-spatial correlations in time series, but these methods are only successfully applied in single-step prediction and short-term prediction.

Figures 4, 5 and 6 is the illustration of our approaches. Figure 1 shows the overall framework of the proposed Dual-stage two-phase (DSTP)-RNN model. Figure 5 shows the overall framework of the proposed DTSP-RNN -II model. Figure 6 shows the model details within a time window as an example. From the figures below, the red box represents the spatial attention with the original series (e.g., X), the combined series (e.g., X2), or the weighted series (e.g., Z) as input. Then, the blue box represents the temporal-spatial attention with a weighting operation for the encoder hidden state (e.g., HX). DeepAttn stacks only one more spatial attention unit based on DTSP-RNN, with the weighted data of the second phase attention as input. The notation meaning and details of vector mapping will be explained later.

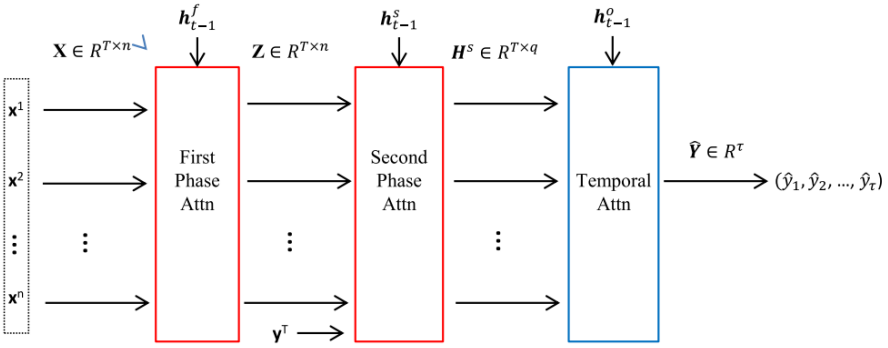

Figure 4. The overall framework of the proposed Dual-stage two-phase (DSTP)-RNN model

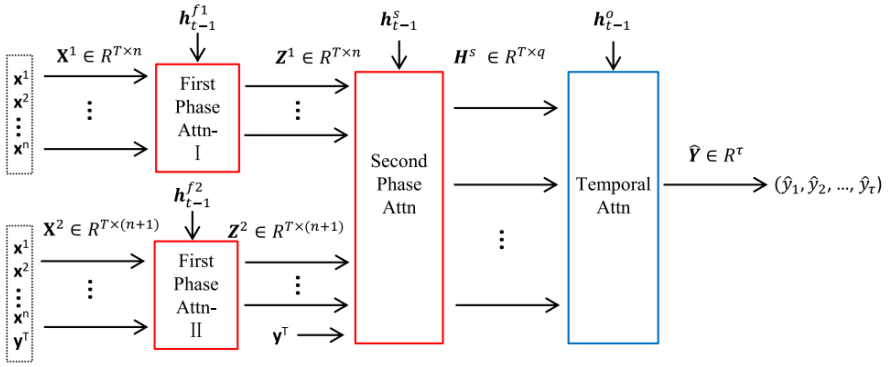

Figure 5. The overall framework of the proposed DTSP-RNN -II model

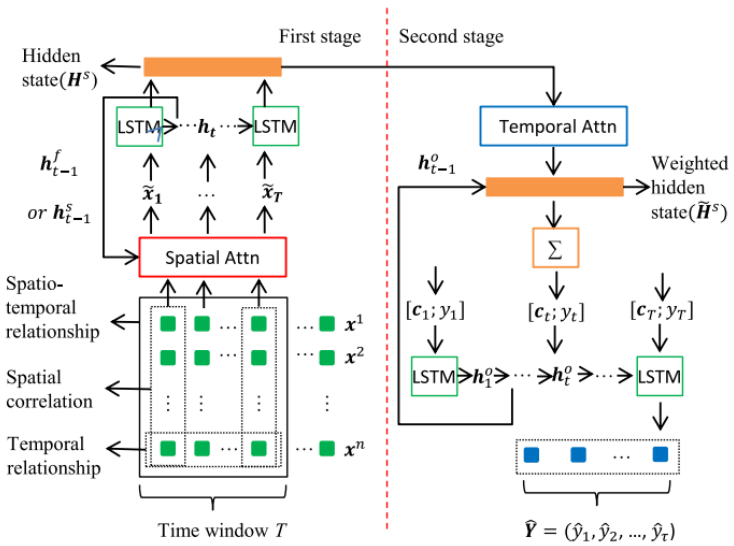

Figure 6. The model details within a time window as an example

2.3 Spatial Attention

The first and second phases of the attention mechanism belong to the spatial attention mechanism. This stage is to train spatial correlations between exogenous series and target series. More specifically, the spatial correlation of exogenous series is to learn at first phase and in the second phase, it was used to learn weighted features again. In short, the stable and robust of the learned spatial correlations is ensured by the two-phase spatial mechanism.

2.4 First phase attention

In the first phase, only an attention module employed for exogenous series to learn the spatial correlations between these exogenous attributes. A variant is to use an independent attention module to learn the spatial correlations between target series and exogenous series, which is the ideal source for two parallel attention structures in DSTP-RNN-II. In fact, the main difference between these two parallel attention structures, first phase attention-I and first phase attention-II (Figure 5), is whether the exogenous series is concatenated with the target series at the corresponding time (i.e., $\dot{x}_t^k = [x_t^k; h_t^k]$). Typically, given the k-th attribute vector of any exogenous series at time t (i.e., xk), we employ the following attention mechanism:

$$f_t^k = v_f^T tanh\,(W_f[h_{t-1}^f; s_{t-1}^f)]+ U_f x^k + b_f \qquad (1)$$

$$\alpha_f^k = \frac{exp\,(f_t^k)}{\sum_{j=1}^{n} exp\,(f_t^j)} \qquad (2)$$

Where [*;*] is a concentration operation, and vf, bf ∈ RT, Wf ∈ RTx2m, Uf ∈ RTxT are parameter to train. Here, h_{t-1}^f ∈ Rm and s_{t-1}^f∈ Rm is the hidden state and the cell state of the encoder, respectively. m is the hidden size in this attention module. The attention weights are determined by historical hidden state and current input in the encoder, which represents the impact of each attribute in time series prediction. Since any attribute value at any time has its corresponding weight, the output after the first phase attention weighting is defined as follows:

$$\tilde{x}_t = (\alpha_t^1 x_{t,}^1\, \alpha_t^2 x_t^2, \ldots., \alpha_t^n x_t^n)\mathrm{T} \qquad (3)$$

2.5 Second phase attention

The input of the second phase attention is a vector that catenates the value and output of the first phase of the target series. Through this learning of spatial correlations, more stationary and concentrated spatial correlations can be obtained through this second learning. All the past values of target series with exogenous series at the corresponding time are combined. The output of the parallel attention structure in the first phase attention as the input to the second phase attention also concatenated to learn the difference between the target series and exogenous series whereby it performed in DSTP-RNN-II. As for the final input vector of this module, the concatenated corresponding to the target variable yk to the k-th attribute $\tilde{x}^k$ to form a new vector zk, i.e., $z^k = [\,\tilde{x}^k; y^k] \in$ R(n+1)xT, and the attention weight is learned as follows:

$$s_t^k = v_f^T tanh\,(W_s[h_{t-1}^s; s_{t-1}^s)]+ U_s[\tilde{x}^k; y^k]+b_s \qquad (4)$$

$$\beta_f^k = \frac{exp\,(s_t^k)}{\sum_{j=1}^{n+1} exp\,(s_t^j)} \qquad (5)$$

where vs, bs ∈ RT, Ws ∈ RT × 2q, Us ∈ RT × T are parameters to learn, and h_{t-1}^s ∈ Rq and s_{t-1}^s ∈ Rq is the hidden state and the cell state of the encoder, respectively. q is the hidden size in this attention module. For each spatial attention module with target series (e.g., first phase attention-II), we employ the above operation independently. The output after the second phase attention is as follows:

$$\tilde{z}_t = (\beta_t^1 z_{t,}^1\, \beta_t^2 z_t^2, \ldots., \beta_t^n z_t^n)\mathrm{T} \qquad (6)$$

2.6 Temporal attention

The method of maintaining time dependency in the spatial attention includes the Spatio-temporal relationships in the encoder and the concatenation vector between target series and exogenous series, both of which reflect the temporal relationships within a fixed window. Since the time dependency in a fixed window is not enough, it is still necessary to employ the attention mechanism to select the hidden state of the encoder to learn more robust temporal relationships. Specifically, the long-term dependency of time series can be learned by weighting the hidden state in the encoder that is most related to the target value. The Spatio-temporal relationships learn through joint training of the spatial attention and temporal attention. For each i-th hidden state from the second phase attention, the temporal relationships can be learned by an attention mechanism as follows:

$$d_t^i = v_d^T tanh\,(W_d[h_{t-1}^0; s_{t-1}^0)]+ U_d h_i^s + b_d \quad (7)$$

$$\gamma_t^i = \frac{exp\,(d_t^i)}{\sum_{j=1}^{T} exp\,(d_t^j)} \quad (8)$$

Then, the weighted hidden state $\tilde{h}_t^s$ and the context vector are defined as follows

$$\tilde{h}_t^s = (\gamma_t^1 h_1^s, \gamma_t^2 h_2^2, \ldots., \gamma_T^i h_t^s)\mathrm{T} \quad (9)$$

$$c_t \sum_{j=1}^{T} \quad \gamma_t^j \tilde{h}_j^s \quad (10)$$

where vd, bd ∈ Rp, Wd ∈ Rq × 2p, Ud ∈ Rp × p are parameters to learn. $h_{t-1}^0 \in$ Rp and $s_{t-1}^0 \in$ Rp is the hidden state and the cell state of the decoder, respectively. p is the hidden size in this attention module. $h_i^s \in$ Hs is the i-th encoder hidden state at the second (or last) stage attention module. The context vector ct means the converged information of all hidden states in the encoder, which represents the Spatio-temporal relationship within each time window.

2.7 Training method

The single layer of LSTM is used in both first phase attention and second phase attention. It used to encode all series as feature representations of the hidden state. The feature below is learned from input data Xt at time t:

$$h_t^f = f_e(h_{t-1}^f, x_t) \quad (11)$$

where $h_t^f \in$ Rm is the hidden state of the first phase attention, m is the hidden size in the first phase, and f_e is an LSTM unit. Note that all attention modules follow the above basic mapping in both the two-phase spatial attention stage and the temporal attention stage. The difference between different attention modules is the different input series and the independent LSTM units, e.g., f_e in Eq. (11) and f_d in Eq. (13), are independent. The data mapping process of the specific input series and output series is noted in Figure 4.

Once the weighted and summed context vector c_t in the decoder is achieved. The context vector c_t and the target series Y will be combined at the corresponding time.

$$\tilde{y}_t = \tilde{w}^T[y_t; c_t] + b^{\check{}} \quad (12)$$

Where $\tilde{w}^T \in$ Rq+1 and $b^{\check{}} \in$ R are the parameters that map the concatenation to the size of the decoder hidden states. The target is aligning in series with context vectors to easier the maintaining of temporal relationships, and the result is used to update the decoder hidden state:

$$h_t^0 = f_d(h_{t-1}^0, \tilde{y}_{t-1}) \quad (13)$$

where hot ∈ Rp is the hidden state used in the decoder, and f_d is also an independent LSTM unit. Finally, the context vector c_t is concatenate with the hidden state h_t^0, where it is used as the new hidden state to make the final-step prediction:

$$\tilde{y}_t, \tilde{y}_{T+1}, \ldots. \tilde{y}_{T-1} = v_y^T\left(W_y\left[h_t^0; c_t\right] + b_y\right) + b'_y \quad (14)$$

where $W_y \in R^{\,p\,x\,(p+q)}$ and $b_y \in R^{\,p}$ map the concatenation $[h_t^0; c_t] \in R^{\,p+q}$ to the size of the decoder hidden states. The linear function with weights $v_y \in R^{\,\tau \times p}$ and bias $b'_y \in R^{\,\tau}$ yields the final prediction result.

2.8 Parameters Concern during Training

Training a model is to determine good values for all the bias and the weight from labelled examples. Before training a neural network, it is important to scale features. A common way of doing this scaling is standardization by subtracting the mean and dividing by the standard deviation of each feature. This method rescales the values into a range of 0 to 1. While training the neural network, the training loss function and validation loss function should be analysed to optimize the neural network algorithm. Loss is the result of a bad prediction. The loss is calculated on training and validation, which interprets how well the model is doing in these two sets. In other words, it is the sum of error for each example in validation or training sets. The performance of a model is implied by the loss value after each iteration of optimization. Prediction is bad if the value of loss is high for any model. Figure 7 shows the graph representing a high loss whereas Figure 8 shows graph with low loss.

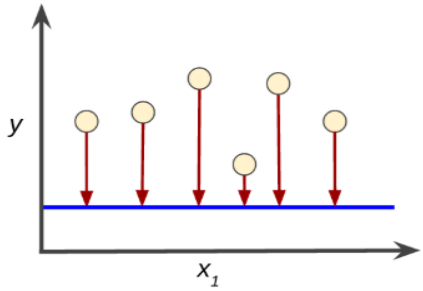

Figure 7. Graphic illustration of high loss

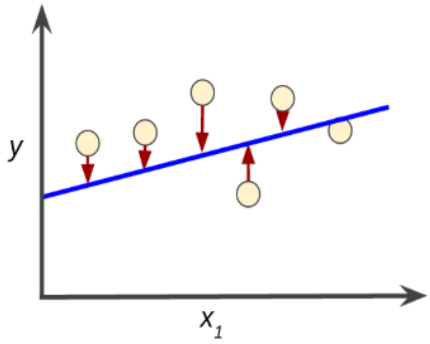

Figure 8. Graphic illustration of low loss

From the figures above, the loss is represented by the arrows whereas the blue lines indicate prediction. The values of validation loss and training loss obtained during training could be used to indicate whether the prediction is overfitting, underfitting or just right. Overfitting occurs when the noise of the data is captured by the neural network algorithm, which is usually a result of an overly complicated model. Fitting multiple models and using cross-validation or validation to compare their predictive accuracies on test data could prevent overfitting. Underfitting occurs when the underlying trend of the data could not be captured by the neural network algorithm. In other words, the algorithm does not fit the data well enough, which cause underfitting. This is due to the result of an excessively simple model. Both underfitting and overfitting lead to poor predictions on new data sets. The validation loss and training loss values are compared to determine overfitting or underfitting. If training loss value is less than validation loss value, the algorithm is overfitting whereas if training loss value is more than validation loss value, the algorithm is underfitting. The algorithm has a good prediction if the training loss value is approximately the same as validation loss value.

After training the models, accuracy is one metric to evaluate the neural network model. Accuracy is the ratio of correct predictions and the total number of predictions which follows the following definition:

$$\text{Accuracy} = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ prediction} \tag{15}$$

The accuracy determines how well the model behaves in prediction. However, it does not indicate the overall behaviour with a class-imbalanced dataset where there is a significant disparity between the number of positive and negative values.

2.9 Related Applications of RNN and LSTM

Besides prediction of stock, RNN and LSTM are used in various applications. In [6], RNN is trained by Back Propagation training algorithm and using normal data to monitor atypical activity to detect any violation that impedes computer systems security. Besides that, a novel single layer and static RNN is used to the recognition of dynamic patterns in [7]. The learning performance of the patterns is improved with the feedback connection. The networks are applied to the recognition of Korean spoken digit. Furthermore, the specialized layer of RNN called LSTM is used to predict depression from EEG signal in [8]. Based on the extracted EEG signals, LSTM models are used in prediction of trends of depression for the next instants. The results show that LSTM predictor model works best for he prediction of trends of depression. Moreover, in [9], LSTM model is combined with Kalman filtering for the prediction of time series data with short-term and long-term characteristics of air quality. The information contained in the pre-ordered data is stored by the LSTM-Kalman model by using the unique memory feature of LSTM. The results show that LSTM model combined with Kalman filter has better performance in air quality prediction compared to a regular LSTM model. Based

on the discussed applications, LSTM is the best approach to predict stock values as the prediction is in time-based. LSTM is also proven to have a good prediction based on the reviewed paper.

3. Implementation

Python language was used to code an algorithm for this project. The reason behind it because Python quite famous nowadays and lots of reference we can get from open source. Using other languages also can but the number of references is limited compared when using Python as most of advanced researchers and engineers use this language for their development.

Tensor Flow used in this project. There are lots of libraries there prepared to be used for Artificial Intelligence development. Python code can be used with library from TensorFlow to make an AI. Then, to implement this model, not suitable to use personal laptop as the processing speed is not good enough. To train prediction for one item will take too much time even though using GPU from laptop so, recommended to use a server.

4. Conclusion

We have introduced approaches on how to forecast spare parts for reordering purposes, using recurrent neural network. The network's key features are the bidirectional Long Short-Term Memory architecture that provides long-range access, bidirectional contextual information, and the connectionist temporal classification output layer that enables the network to be trained on unsegmented sequence data. By using tools mentioned, a forecast model successfully created to predict the number of spare parts needed to buy in the next few months.

References

[1] Sundermeyer, Martin / Schlüter, Ralf / Ney, Hermann (2012): "LSTM neural networks for language modeling", In *INTERSPEECH-2012*, 194-197

[2] Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H. and Schmidhuber, J. (2008). A Novel Connectionist System for Unconstrained Handwriting Recognition.

[3] Liu, Y., Gong, C., Yang, L., Chen, Y. DSTP-RNN: A dual-stage two-phase attention-based recurrent neural network for long-term and multivariate time series prediction. Expert Systems with Applications. 143. 2020: Article number 113082

[4] Ribeiro De Miranda, A., De Andrade Barbosa, T.M.G., Scolari Conceicao, A.G., Soares Alcala, S.G. Recurrent neural network based on the statistical recurrent unit for remaining useful life estimation (2019) Proceedings - 2019 Brazilian Conference on Intelligent Systems, BRACIS 2019, art. no. 8923724, pp. 425-430.

[5] Jeong, J., Park, E., Chen, H., Kim, K.-Y., Shik Han, W., Suk, H. Estimation of groundwater level based on the robust training of recurrent neural networks using corrupted data (2020) Journal of Hydrology, 582, art. no. 124512.

[6] N. Chowdhury and M. A. kashem, "A comparative analysis of Feed-forward neural network & Recurrent Neural network to detect intrusion," 2008 International Conference on Electrical and Computer Engineering, Dhaka, 2008, pp. 488-492.
doi: 10.1109/ICECE.2008.4769258

[7] J. K. Ryeu, H. Y. Tak, N. W. Heo and H. S. Chung, "Recognition of Korean spoken digit using single layer recurrent neural networks," Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan), Nagoya, Japan, 1993, pp. 267-270 vol.1.
doi: 10.1109/IJCNN.1993.713908

[8] S. D. Kumar and D. Subha, "Prediction of Depression from EEG Signal Using Long Short Term Memory(LSTM)," 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2019, pp. 1248-1253.
doi: 10.1109/ICOEI.2019.8862560

[9] X. Song, J. Huang and D. Song, "Air Quality Prediction based on LSTM-Kalman Model," 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 2019, pp. 695-699.
doi: 10.1109/ITAIC.2019.8785751