Article citation info:
Kerboua A, Kelaiaia R. Recurrent neural network optimization for wind turbine condition prognosis. Diagnostyka. 2022;23(3):2022301.
https://doi.org/10.29354/diag/151608

DIAGNOSTYKA, 2022, Vol. 23, No. 3

e-ISSN 2449-5220
DOI: 10.29354/diag/151608

RECURRENT NEURAL NETWORK OPTIMIZATION FOR WIND TURBINE CONDITION PROGNOSIS

Adlen KERBOUA [1, *], **Ridha KELAIAIA** [2]

[1] Faculty of Technology, Université 20 août 1955-Skikda, Algeria
[2] LGMM Laboratory, Faculty of Technology, Université 20 Août 1955-Skikda, Algeria
[*]Corresponding author, e-mail: ad.kerboua@univ-skikda.dz

Abstract

This research focuses on employing Recurrent Neural Networks (RNN) to prognosis a wind turbine operation's health from collected vibration time series data, by using several memory cell variations, including Long Short Time Memory (LSTM), Bilateral LSTM (BiLSTM), and Gated Recurrent Unit (GRU), which are integrated into various architectures. We tune the training hyperparameters as well as the adapted depth and recurrent cell number of the proposed networks to obtain the most accurate predictions. Tuning those parameters is a hard task and depends widely on the experience of the designer. This can be resolved by integrating the training process in a Bayesian optimization loop where the loss is considered as the objective function to minimize. The obtained results show the effectiveness of the proposed method, which generates more accurate recurrent models with a more accurate prognosis of the operating state of the wind turbine than those generated using trivial training parameters.

Keywords: forecasting, recurrent networks, optimization, hyperparameters, loss.

1. INTRODUCTION AND RELATED WORK

Due to its efficiency, artificial intelligence is becoming a key tool in the development of numerous industries, particularly in the manufacturing industry [1, 2], where several issues are open. Recently, time series have become a general subject of tremendous practical interest. because it allows us to deduce the future values of a series from its past values with a margin of error. Numerous successful applications in several domains, such as engineering [3, 4] and manufacturing, energy production and management [5, 6], and other fields, have been documented in the appropriate literature. The use of deep learning tools for forecasting time series data trends is relatively recent, and continues to attract the attention of the scientific community in different fields of technology.

Several frameworks are reported in the literature to resolve this issue. For instance, wind energy [7], which is among the sources of energy that see the fastest growth in the world. Nevertheless, the failure to detect the breakdown of turbine parts can be exceptionally exorbitant [8]. Consequently, defining and building models for the predictive maintenance of wind turbines [9] is a crucial task. It is noted that preventive maintenance can be done at regular intervals to prevent common problems. In this context, the authors of [10] employed two different methodologies using Nonlinear Auto-Regressive eXogenous input (NARX) and Adaptive Neuro-Fuzzy Inference System (ANFIS) to estimate the health of a wind turbine gearbox using vibration trend index. The experimentation was conducted on two datasets and demonstrated that the two proposed prediction methods perform well at estimating the fluctuations of the monitoring indices.

The authors proposed a new method based on deep feature modelling and a LSTM neural network for predicting the remaining useful life of rolling bearings in [11]. They carried out experiments using IEEE PHM Challenge 2012 data sets. The obtained results demonstrate the importance of the suggested method's performance improvement in both prediction accuracy and numerical stability. In the same context, Liu ZH et al. [12] proposed an approach that combines elastic net with LSTM. The E-LSTM algorithm is made up of an elastic mesh and LSTM, utilizing the LSTM to anticipate the remaining useful life using temporal-spatial correlation. In their work [13], the researchers proposed a Convolutional Neural Network or CNN-LSTM model based on an attention mechanism for wind turbine fault prediction. The model is trained using the Semantic Sensor Network (SSN) ontology-annotated icing fault and yaw fault datasets of wind turbines. The trained model can accurately anticipate the probability of a wind turbine fault. The experimental results, according to the authors, suggest that the proposed model is more effective

and superior than some of the current mainstream models, based on real wind turbine test data.

The authors in [14] proposed an end-to-end fault diagnosis approach. They employ similar temporal sequences as input to a Convolutional LSTM (CLSTM) to detect the bearing defect with high accuracy in a small duration of time. Authors noticed that the method can achieve good accuracy without performing pre-processing. The fault diagnosis method's effectiveness and feasibility are demonstrated by comparing the results to those of existing intelligent fault detection systems using benchmark experimental vibration datasets. The authors [15] employed the LSTM to anticipate traffic flow using traffic sensor data that was available as a big data set. Then, they compared the results obtained by LSTM to the traditional statistical methods of traffic prediction. It is found that the LSTM-based model performs better. All of these studies train networks, including basic versions of LSTM-like memory cells, and they try to improve the performance of prediction by varying the hyperparameters only.

Even if the efficiency of the LSTM is proven, however, it is not the best choice all the time. Depending on the nature and the availability of the data to be processed, variants of the LSTM model offered by the literature as well as their combination in deeper networks can improve the performance of prediction tasks. Also, using the optimum training hyperparameters improves significantly the performance of prediction.

It is important to study the internal structure of a memory cell to provide the robustness of the predictions before starting to design the models. Recurrent networks based on memory cells can handle the restrictions of conventional time series forecasting methodologies and deliver state-of-the-art results on temporal data by adding nonlinearities to a given dataset. Each memory cell has its own time step and transmits its output to the next cell until the last one, which gives the expected sequential output.

The recurrent network can be designed in several ways. By using only one recurrent layer that can comport a well-defined number of memory cells, we can choose to design a deeper network by stacking more recurrent layer cells, which can learn in a more sophisticated way the data trends. We can also use a lot of training hyperparameters. Judiciously tuning those parameters can have a significant impact on the final prediction performance. This can be hard and depends widely on the experience of the designer.

In this work, we investigate various RNN based models to forecast time series data for wind turbine prognosis purposes using vibration. We first introduce the LSTM cell, which learns long-term relationships among time steps in time series data by maintaining an internal state, and we use it as well as two modified versions of this architecture, BiLSTM and GRU units, to predict the trends of the degradation of the bearing inside the wind turbine using vibration time series data. During the design phase of the proposed models, special attention was paid to the adjustment of all parameters, such as the number of cells in each recurrent layer, the depth of the network, and even the training hyperparameters, which have a direct impact on prediction performance.

Tuning those parameters can be hard and depends widely on experience. Also, RNN can be relatively quick to train because it allows one to integrate the training process in a Bayesian optimization loop by considering the loss as the objective function to minimize. This loop is designed to compute the best fitting parameters. The results obtained show the effectiveness of the proposed method which generates more accurate recurrent models than those generated using trivial chosen training parameters. The Bayesian optimization loop is ideal for tuning the training hyperparameters as well as the adapted depth and recurrent cell number of the networks. The objective function of the optimization process is defined as the loss obtained by the RNN by computing the gap between the experimental and the predicted data.

The key contributions of this work are threefold:

- ✓ investigate several RNN for wind turbine prognosis using vibration.
- ✓ optimize training hyperparameters used to enhance the prediction power of the model.
- ✓ optimize the model depth by finding the best number of recurrent layers to stack.

Our paper follows the steps below: An overview of the proposed method is presented in section 2. The datasets and the pre-processing steps are detailed in section 3. The recurrent models employed in this research and their optimization are described in section 4. We exhumed and discussed the experimental results in section 5, and we introduced the conclusion and future work in section 6.

2. OVERVIEW OF THE PROPOSED METHOD

Industrial system prognosis is the process of projecting how long a machine will likely operate before it requires predictive maintenance to maximize operating efficiency and reduce unplanned downtime. Because of this, one of the most important goals of predictive maintenance, Prognostics, and Health Management (PHM) systems is to make an accurate assessment of the operational state.

We use a standard prognostic methodology in this study, which includes data import and analysis, feature extraction, model design, training and forecasting, and performance assessment. We investigate several sequence-to-sequence regression networks based on RNN memory cells such as LSTM to estimate the operating state of a wind turbine. The trained models are able to detect significant degradation trends and update their internal states when a new observation becomes available. The entire prognosis workflow is

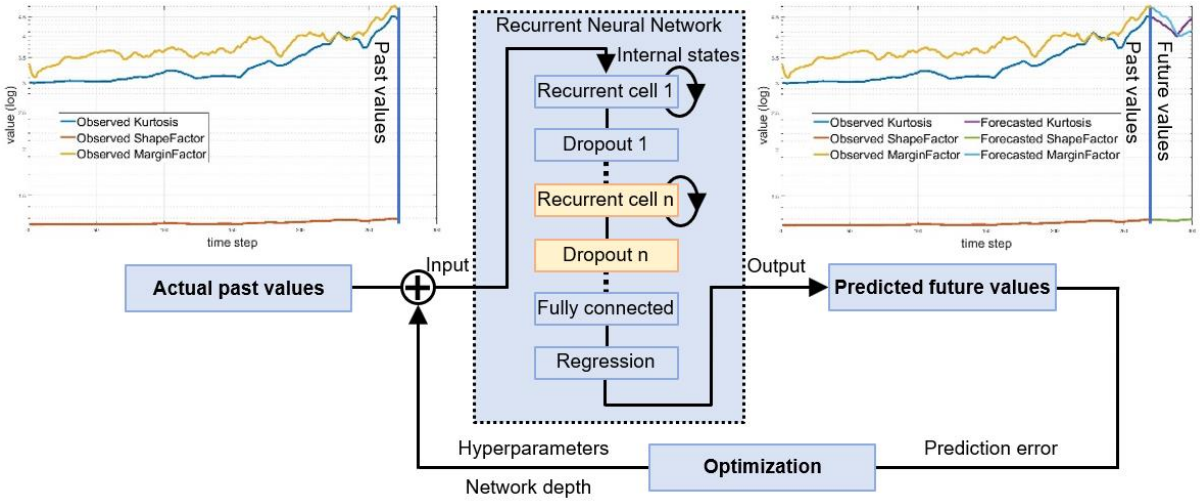

Fig. 1. Overview of the proposed optimization loop

integrated in an optimization loop (Fig. 1) designed to tune the RNN attributes like training hyperparameters, cell number, and network depth.

Thus, we investigate several LSTM-like RNN predicted performances, such as LSTM, BiLSTM, and GRU.

By keeping a hidden state, every one of those recurrent cells learns long-term relations among time steps in a time series, conducting additive interactions during training, which can aid in enhancing gradient flow across longer sequences using gradient clipping calculations that prevent gradient vanishing/explosion by stabilizing the training event at higher learning rates and/or in the presence of outliers in the training data [16]. However, these recurrent techniques use different mechanisms to remember or forget time dependencies using gates. We also explore different combinations of these recurrent layers by making the RNN deeper by inserting extra recurrent layers. This technique makes it possible to memorize time dependencies in a more sophisticated way. We include dropout layers after each recurrent one to avoid overfitting.

3. PREPROCESSING DATA

Before the detailed explanation of the proposed framework, we present time series data contained in the used dataset, collected and presented by the authors of [17] from the high-speed shaft of a 2MW wind turbine actuated by a 20-tooth pinion gear. On each of the 50 days, 50 vibration and tach signals lasting 6 seconds were recorded at a sampling frequency of 97656 Hz. During the 50-day timeframe, an inner race defect occurred, causing the bearing to fail. We investigate the data in a time domain thought to be good for prognosis purpose. First, the vibration signals are shown in the time domain. by plotting the 50 vibration and tachometer signals stacked one after each other.

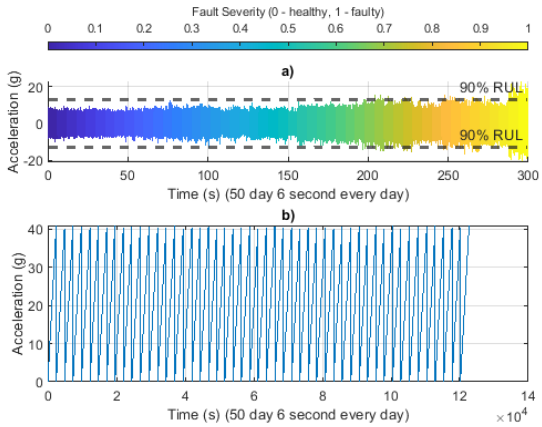

Fig. 2. Signals in the dataset, over 50 days with 6 second per day, a) cumulative vibration and b) tachometer

In Fig. 2 we can see the fault severity indicated by the progression of color from blue to yellow, which reflects the progression of the defect. We also discover that the vibration data is noisy and exhibits a rising trend in signal impulsivity over time steps. The extracted time-domain features from each time step, such us kurtosis, form factor, and margin factor, can measure the signal impulsive behavior. These are promising features for predictive the future of wind turbine [18].

As mentioned above, the dataset was collected over 50 days with 6 seconds per day, giving us initially 50 time steps, which is insufficient for the networks to understand the data trend. To resolve this issue, we consider each second of data as one time step, in order to have enough data to train and test the proposed RNNs, which results in 300-time steps. We proceed by extracting time domain features. First, we compute the kurtosis of a time series sequence X of length T (represented by Fig. 3 a) using the following formula:

$$k = E(X-\mu)^4/\sigma^4, \tag{1}$$

Where μ is the T mean, and σ is the standard deviation of X, and $E(t)$ represents the expected value of the vibration signal in time $t \in [1:T]$. The kurtosis function computes a sample version of this population value. Then we compute the second time domain feature, which is the shape factor (Fig. 3 b) using the formula:

$$s = rms(X)/mean(|X|), \tag{2}$$

Lastly, we use the following formula to compute the margin factor as illustrated in Fig. 3 c:

$$m = \max(X)/mean(|X|), \tag{3}$$

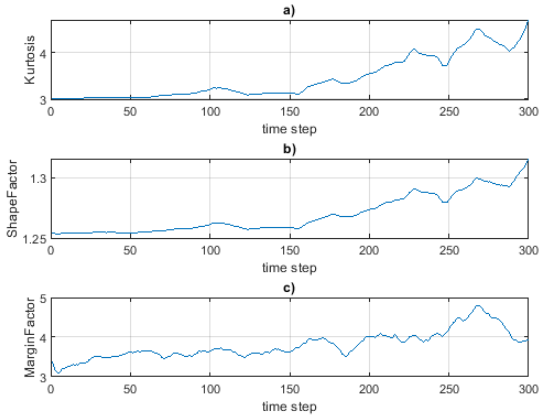

Fig. 3. Extracted time domain features, a) Kurtosis, b) shape factor and c) Margin factor

3.1. Standardize data

We normalize the data for training to have zero mean and unit variance for a better match and to avoid the training from diverging during training. Normalization, also known as z-scoring, is a common approach to enhance network performance. We apply the same standardization parameters as training data to the test data.

The training process can be divergent and instable when a network is trained using data distributed over a large interval [19]. This is the reason why we proceed by standardizing the data by computing μ and standard deviation σ from the training data to standardize both training and testing data as follows:

$$\sigma = \sqrt{\frac{1}{X-1}\sum_{t=1}^{T}|X_t - \mu|^2}, \tag{4}$$

Where μ is the mean of the sequence, and X_t the time series data in time $t \in [1:T]$.

$$\mu = \frac{1}{X}\sum_{t=1}^{T} X_t, \tag{5}$$

Then we can compute the standardized time series data X_s as follow:

$$X_s = (X - \mu)/\sigma, \tag{6}$$

3.2. Prepare predictors and responses

In the forecasting tasks, to test the effectiveness of the method, the standard protocol is to split available data into two distinct subsets. The first one, composed of the starting part of the data, is used to train the network. The networks can establish long-term relationships across training data and fit the data trends during the training phase. The second subset is used as metric data to compute the gap between this actual subset and the predicted values. We decided to use the first 90% of the time steps for the training and the last 10% for the testing.

A sequence-to-sequence regression technique is used to predict the values of future time steps in time series data. So, the output (label) vector is the same as the training sequence vector but with values pushed forward one time step. That is, at each time step of the input sequence, the RNNs learn to predict the value of the next time step. And we perform the same process with testing data.

$$X_s = X_s(1: length(X_s) - 1), \tag{7}$$

$$label = X_s(2: length(X_s)), \tag{8}$$

The main idea is to forecast the values of future time steps in a sequence using training sequences with values shifted by one time step as replies. As a result, the RNN may learn to anticipate the value of the next time step for each time step of the input sequence and modify the network state with fresh incoming data for each time step of the input sequence.

4. DEFINE THE RNNs ARCHITECTURES

In this work, we investigate prediction accuracies obtained by several types of RNNs, trained using optimized hyperparameters. So, we choose to design an LSTM [20] based RNN. An LSTM unit can learn dependencies in long time series data, to a deeper version of LSTM obtained by stacking two or more LSTM layers [21] to effectively deal with more long-term temporal dependencies inside the data. To prevent overfitting, we insert dropout layers after each of the LSTM layers.

We also implemented the Bidirectional LSTM or BiLSTM [22], and a deeper version of BiLSTM, which, in the same way as classic LSTM, can remember dependencies between long sequences but in both directions, from the beginning of the sequence to the end and vice versa. BiLSTM is an enhanced version of LSTM by encapsulating two LSTM cells. The first cell learns from the original input sequence, while the second one uses the sequence in reversed order. This technique can give more information about the context of the sequence.

Similar to the LSTM cell, we also introduced the GRU cell in [23]. Like the two first models, the GRU cell allows to learn temporal structure in time series but ureses a simpler mechanism to update and forget dependencies between time steps.

4.1. Mathematical description of the used RNNs

As mentioned above, both BiLSTM and GRU are modified versions of the basic LSTM model. In this section we survey the mathematical fundaments of these methods and how the memory cells can choose information to remember and those to forget.

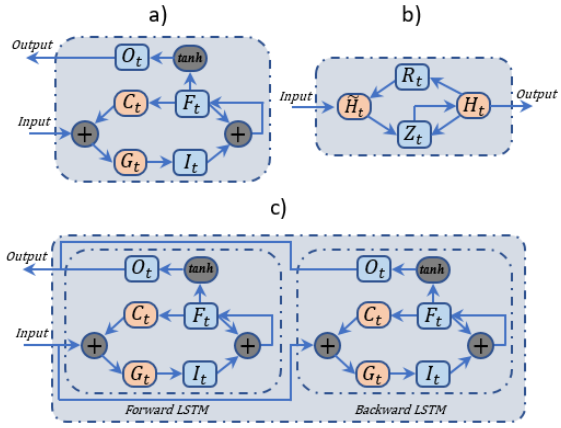

Fig. 4. Main recurrent cells internal structures, a) LSTM, b) GRU and c) BiLSTM

a) LSTM

The LSTM cell can keep long-term dependencies among time steps in time series data by storing information learnt from prior time steps in its internal state. At each time step, the state is updated by adding or removing information using gates.

The internal construction of an LSTM cell is depicted in Fig. 4 a, showing the three gates: the input, the output, the forget gate, and the hidden state cell. The three gates are responsible for regulating the flow of data and deciding to store significant information while forgetting unimportant information. LSTM performs well for time series prediction. The advent of LSTM has solved two major problems that RNNs face: vanishing and exploding gradients. By applying the next equation to X_t the time series data, where $t \in [1:T]$, we can determine the hidden cell state C_t and the current output vector H_t of the LSTM cell using an input sequence X_t of length T.

$$C_t = F_t \odot C_{t-1} \odot G_t, \tag{9}$$

$$H_t = O_t \odot \tanh(C_t), \tag{10}$$

To update the hidden state, we use the tanh activation function, where $\odot$ represents the element-wise product:

$$\tanh(X) = \frac{\sinh(X)}{\cosh(X)} = \frac{e^{2X}-1}{e^{2X}-1}, \tag{11}$$

And we apply sigmoid activation function to the gates:

$$\sigma(x) = (1 + e^{-X})^{-1}, \tag{12}$$

The input gate, forget gate, cell input, and output gate are denoted by I, F, G and O, respectively, for each element at time step t, the following equation is used:

The input gate I_t that control level of update:

$$I_t = \sigma(W_i . X_t + R_i H_{t-1} + b_i), \tag{13}$$

The forget gate F_t that control the data to forget:

$$I_t = \sigma(W_i . X_t + R_i H_{t-1} + b_i), \tag{14}$$

The input G_t that add information to keep:

$$G_t = \sigma(W_g . X_t + R_g H_{t-1} + b_g), \tag{15}$$

We update the output state by adding a part of data of the cell state defined by the output gate $O_t$as fellow:

$$O_t = \sigma(W_o . X_t + R_o X_{t-1} + b_o), \tag{16}$$

The learnable parameters of the LSTM cell are optimized during the training process, comprising weights W, the recurrent weights R and the bias b of each gate.

b) GRU

Even the GRU cells (Fig. 4 b) are similar to LSTM in that they use gates, but they have fewer parameters than LSTM since they don't have an output gate. GRU's performance on music and voice modelling tasks was found to be equal to that of LSTM. On some smaller datasets, GRU has been found to perform even better. As a result, it is a widely used and simplified LSTM cell. The following equation is used to update the GRU cell's hidden state:

$$H_t = (1 - Z_t) \odot \widetilde{H}_t + Z_t \odot H_{t-1}, \tag{17}$$

To update the gate, we use the following equation, which determines how much the GRU unit is updated:

$$Z_t = \sigma(W_z . X_t + R_z H_{t-1} + b_z), \tag{18}$$

The reset gate is calculated in the same way as the update gate, with the following expression:

$$R_t = \sigma(W_r . X_t + R_r H_{t-1} + b_r), \tag{19}$$

By applying the hyperbolic tan function to the reset gate, which is defined by the following function, a new remember gate is made:

$$\widetilde{H}_t = \tanh(W_h . X_t + (R_t \odot H_{t-1}) R_h + b_h), \tag{20}$$

When, the time series of input X_t weights W, the recurrent weights R and the bias b of each component Z, R and $\widetilde{H}$, are respectively, the learnable weights of the GRU cell, optimized over network training.

c) BiLSTM

We use BiLSTM to understand data in two directions, from the beginning to the end, and vice versa, because it runs the inputs in two directions, one from past to future and the other from future to past. What distinguishes this approach from unidirectional is that the LSTM that runs backwards preserves information from the future, whereas the BiLSTM encapsulates two cells, which can remember information from both directions of time at any point in time. We can see in Fig. 4 c the BiLSTM cell internal structure. The forward, backward, and output sequences are managed using the following formulas:

$$\vec{H}_t = \sigma(W_{\vec{h}} . X_t + \vec{H}_{t-1} R_{\vec{h}} + b_{\vec{h}}), \tag{21}$$

$$\overleftarrow{H}_t = \sigma(W_{\overleftarrow{h}} . X_t + \vec{H}_{t-1} R_{\overleftarrow{h}} + b_{\overleftarrow{h}}), \tag{22}$$

$$Y_t = \vec{H}_t . W_y + \overleftarrow{H}_t W_y + b_y), \tag{23}$$

Where σ represent the sigmoid function. The BiLSTM is useful in cases when data context is

required. It has been frequently utilized in classification and in forecasting [24].

4.2. Used loss function

As mentioned above, forecasting time series data is mainly a sequence-to-sequence regression. The most adequate loss function used in regression problems is defined as the Mean Absolute Error (MAE) loss. This is the average of the absolute error of the forecasted values compared to actual values over the mini-batch for each time step. This loss function is used to update learnable weights from all layers of the network during the learning process by backpropagation:

$$\text{MAE} = \frac{1}{T}\sum_{t=1}^{T}|X_t - Y_t|, \tag{24}$$

Where X_t is the true values and Y_t is the predicted values, T is the length of the considered sequence.

The learnable parameters are initialized randomly at the beginning of the training, but currently the state of the art in the matter offers several methods to do it. The choice of the initialization method can have an important impact on how well the network trains. Hence, we choose to initialize weights and bias in all learnable layers using the method described in [25]. This gives the best results. In this method, the authors resolve the problem of exploding/vanishing gradient in the learnable layer by multiplying weights obtained from random initialization by a specific factor, as:

$$W^l = \text{random}(size^l, size^{l-1})\sqrt{\frac{2}{size^{l-1}}}, \tag{25}$$

4.3. Studied RNNs

As cited above, in this study we will explore the three main recurrent cell types, which are LSTM, BiLSTM, and GRU. Each type is presented in a simple form with a single recurrent layer, which will be used to memorize the temporal dependencies between time steps, as well as a deeper version with the stacking of two recurrent layers, serving to memorize more sophisticated relations between the time steps. Initially, we study the following six architectures in Table 1:

Table 1. Six studied recurrent architectures

LSTM	BiLSTM	GRU	Deep LSTM	Deep BiLSTM	Deep GRU
input	input	input	input	input	Input
recurrent cell1	recurrent cell1	recurrent cell1	recurrent cell1	recurrent cell1	recurrent cell1
fully connected	fully connected	fully connected	recurrent cell2	recurrent cell2	recurrent cell2
mae regression	mae regression	mae regression	dropout	dropout	Dropout
			fully connected	fully connected	fully connected
			mae regression	mae regression	mae regression

In a second step, this configuration is reproduced to build deeper networks inside the Bayesian optimization loop by stacking several recurrent layers separated by dropout layers in order to avoid overfitting. A dropout layer sets random elements from the input to zero with some probability [26].

4.4. Training hyperparameters

For the first time, to highlight the prediction improvement by using optimal hyperparameters, we start by exploring the forecasting of six networks that contain only one or two LSTM, BiLSTM, and GRU layers. Trained using default hyperparameters, we specify fixed values for all the training hyperparameters. We set the training optimizer to ADAM (ADAptive Moment estimation) [27], which is a type of Stochastic Gradient Descent with Momentum (SGDM) using an automatically adapted learning rate by a specific momentum term, and we train for 200 epochs. The learning rate can have a direct impact on the convergence and the speed of the network. we decided to set it to 0.05 and we plan to drop this value by a factor of 0.2 during the training progress to keep stability of the training. We summarize the hyperparameters used to perform the training process, as well as the properties of recurrent cell numbers, in the following Table 2.

Table 2. Recurrent cells number and training hyperparameters

Parameter	Value
training optimizer	ADAM
recurrent cell1	10
recurrent cell2	20
dropout	0.2
training epochs	200
initial learning rate	0.05
L2Regularization	10-4
approach with a gradient threshold	L2norm
gradient threshold	1
drop learning rate epochs	100

To monitor the effectiveness of the training process, we plot the training loss of every network. We extract the training loss stacked during the training process for every epoch. For each of the networks, we plot the epoch numbers against the validation loss.

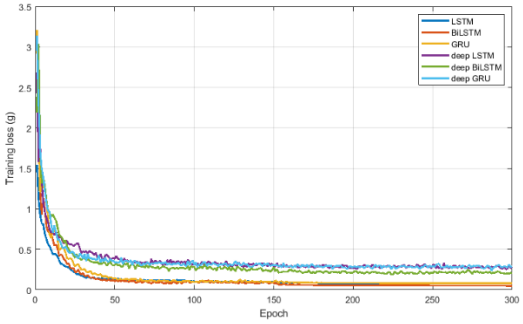

Fig. 5. Validation loss over epochs of the proposed RNNs

Fig. 5 shows the evolution of the loss function of shallow and deep LSTM, BiLSTM, and GRU networks during the training stage as a function of the number of epochs. It can be observed that all of the models converge quickly, with the GRU being the fastest, followed by deep GRU. This is mostly

due to the GRU being a basic model with fewer computational parameters than LSTM and BiLSTM.

5. RESULTS AND DISCUSSION

The most important elements for evaluating a forecasting system's performance are twofold: first, we can observe the activations inside a recurrent cell to evaluate the robustness of the training process; and second, we can use metrics to quantify these performances; the most common is the Root Mean Square Error (RMSE). It shows how well the observed data matches the expected values. Lower RMSE values imply better fit.

5.1. Visualize network activations

We can see the network's learnt features by computing the recurrent layer's activations for each time step of the sequences. we note that before the training the parameters were initialized randomly.

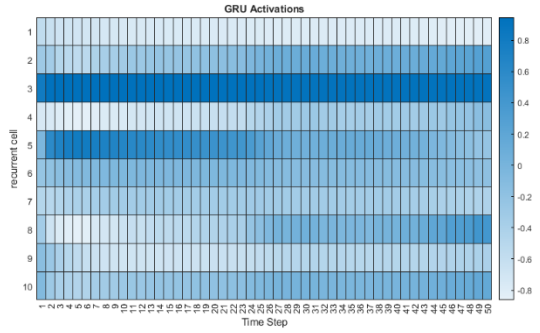

Fig. 6. Activation visualization of one trained RNNs

The heatmap in Fig. 6 represents the activation of the GRU-based neural network with a single recurrent layer. We can notice that the activations fluctuate over time, implying that the cell weights are adjusted in response to data fluctuations.

5.2. Errors metrics of the recurrent model

To compare the performances of the trained models, we must quantify the gap between the real and predicted values. We compute two quantitative measurements, which are Root Mean Square Error (RMSE), Mean Absolute Error (MAE). And the qualitative one, which is the Mean Absolute Percentage Error (MAPE), by comparing the testing dataset by comparing unstandardized prediction data.

$$\mathrm{RMSE} = \sqrt{\sum_{t=1}^{T} \frac{(X_t - Y_t)^2}{T}}, \tag{26}$$

Where X_t is the true values and Y_t is the predicted values, T is the length of the considered sequence.

Unlike RMSE, MAE described by formula (24) allows to reduce the effects of outliers or incorrectly forecasted points. While the absolute error metrics RMSE and MAE presented above are on the same scale as the data (case number), MAPE quantifies the relative percentage error and can provide a clearer view on each network's performance. MAPE is computed by taking the mean of the individual forecast point errors divided by the actual value.

$$\mathrm{MAPE} = \frac{100}{T} \sum_{t=1}^{T} \left| \frac{X_t - Y_t}{X_t} \right|, \tag{27}$$

Absolute and relative metrics are computed from the unstandardized testing data compared to the best predictions. We can forecast the next time step for each prediction by utilizing the actual data from the previous time step. Before computing the metrics, we unstandardized the test dataset and the predictions by using the same standardization factors that we had already found from training subset.

5.3. Discussion of results

The quality of the prediction will now be evaluated using the testing dataset that was not included in the training process. The testing data encompasses 10% of the total data, which includes the three time-domain features extracted from the vibration data. In Table 3 and Fig. 7, we compute the RMSE of the six proposed models for each of the three classes in the dataset. As a first remark, we can notice the models containing a single recurrent layer perform better than those with a deeper architecture. This is mainly due to the limited data sequence used for training the networks. Also, the GRU-based models perform better due to their simple forecasting mechanism, well adapted to short data sequences.

Table 3. RMSE of the RNNs

	LSTM	BiLSTM	GRU	Deep LSTM	Deep BiLSTM	Deep GRU
Kurtosis	0.1369	0.2246	**0.0805**	0.2117	0.2935	0.1181
Shape factor	**0.0043**	0.0104	0.0053	0.0089	0.0127	0.0067
Margin factor	0.1318	**0.0946**	0.1135	0.0471	0.1113	0.1133
Sum	0.2730	0.3296	**0.1993**	0.2677	0.4175	0.2380

Table 4. MAE of the RNNs

	LSTM	BiLSTM	GRU	Deep LSTM	Deep BiLSTM	Deep GRU
Kurtosis	0.1158	0.1597	**0.0580**	0.1385	0.1813	0.0757
Shape factor	0.0038	0.0072	**0.0037**	0.0055	0.0080	0.0043
Margin factor	0.1114	0.0837	0.0930	**0.0399**	0.0971	0.0877
Sum	0.2309	0.2506	**0.1547**	0.1838	0.2864	0.1677

Table 5. MAPE of the RNNs (in %)

	LSTM	BiLSTM	GRU	Deep LSTM	Deep BiLSTM	Deep GRU
Kurtosis	2.6794	3.6482	**1.3196**	3.1452	4.1052	1.7123
Shape factor	0.2889	0.5524	**0.2834**	0.4175	0.6117	0.3264
Margin factor	2.5969	2.0330	2.2730	**0.9447**	2.3447	2.1843
Sum	5.5652	6.2336	**3.8760**	4.5074	7.0616	4.2230

These tendencies are confirmed using the MAE and the MAPE metrics illustrated in Table 4 and Table 5 each (graphically in Fig. 8 and Fig. 9). With 3.8760% of prediction errors, the GRU is in the first position, followed by the deep GRU with 4.2230%. We note that although the fact of increasing the number of recurrent layers in a network should normally give a better result given the possibility of memorizing long sequences with more precision, we

see that this is not the case in this experiment. In our opinion, this degradation in the performance of the deep models is due to the trivial choice of the training hyperparameters as well as the number of cells that each layer must contain.

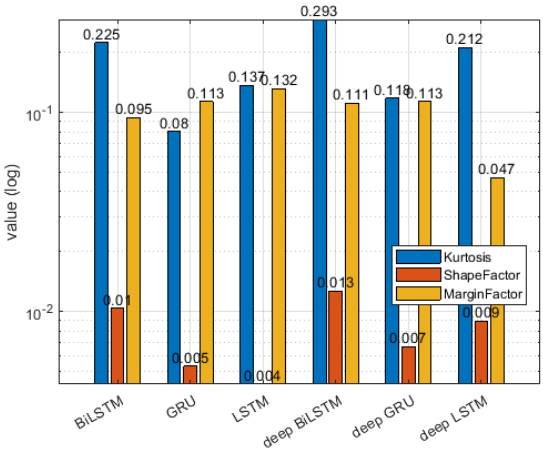

Fig. 7. RMSE comparison (values)

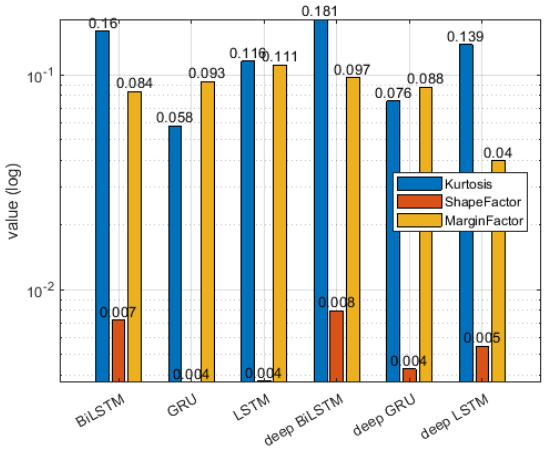

Fig. 8. MAE comparison (values)

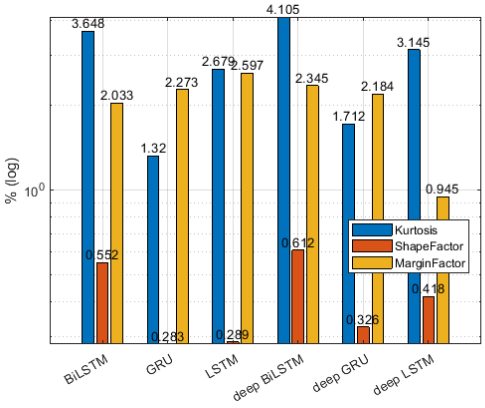

Fig. 9. MAPE comparison (%)

In Fig. 10, we compare side by side the actual values and the predicted values. We note that all the models have been able to capture the trend of the data. To examine in a more precise way the learning of the dynamics of the data, we consult Fig. 11-16, which show an overlay of the actual data and the predicted ones, with a calculation of the prediction errors for each time step. The trends observed previously are confirmed. The network based on GRU learns the dynamics of the data. We also note that the LSTM model is also efficient, while the deep networks are all below expectations.

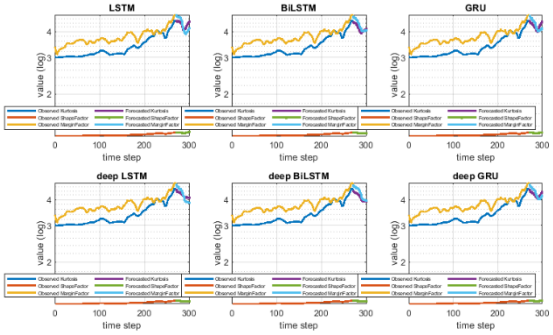

Fig. 10. The training data with the forecasted values

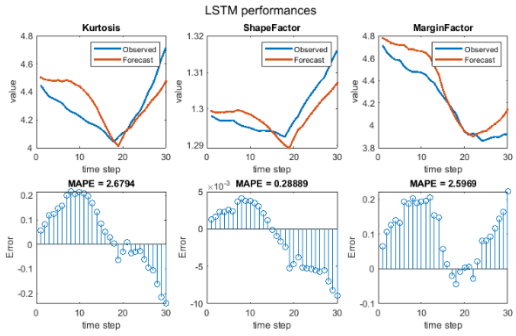

Fig. 11. Values predicted by LSTM, as well as testing data and the related MAPE

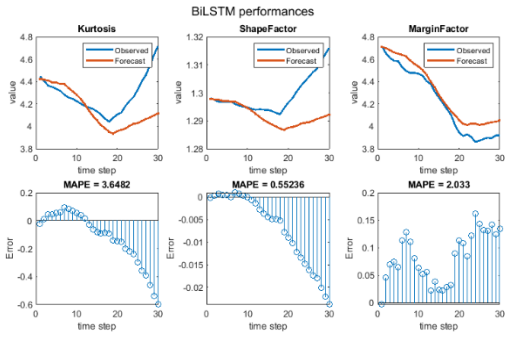

Fig. 12. Values predicted by BiLSTM, as well as testing data and the related MAPEE

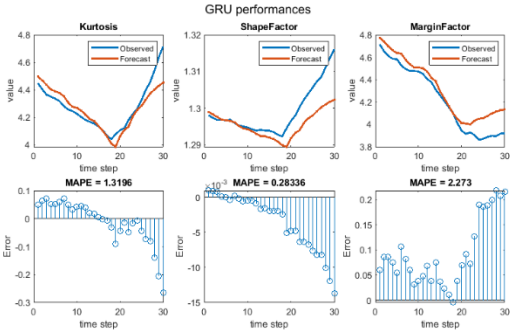

Fig. 13. Values predicted by GRU, as well as testing data and the related MAPE

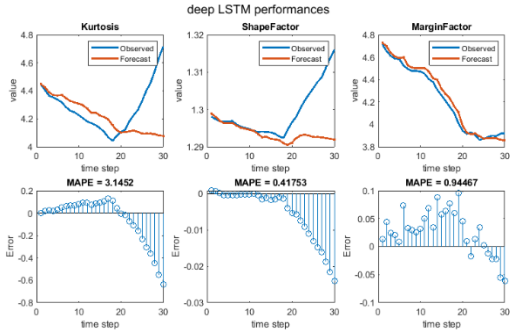

Fig. 14. Values predicted by deep LSTM, as well as testing data and the related MAPE

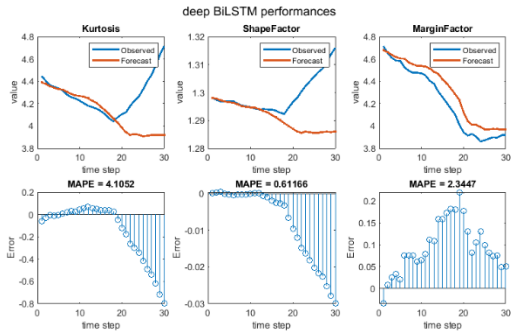

Fig. 15. Values predicted by deep BiLSTM, as well as testing data and the related MAPE

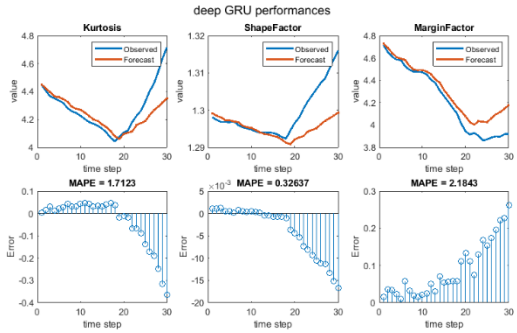

Fig. 16. Values predicted by deep GRU, as well as testing data and the related MAPE

In comparison to the other evaluated models, the GRU-based method delivers improved predictions, as seen in Fig. 13. The metrics MAE, RMSE, and MAPE values calculated based on the prediction and reported in Table 3, 4 and 5 were used to evaluate the obtained results quantitatively and qualitatively. It is clear that the GRU model with one recurrent layer outperforms the others in terms of predicting performance, with reduced RMSE, MAE, and MAPE error metrics.

5.4. Network parameters optimization

We must provide the architecture (depth) and also the training hyperparameters while training a RNN. Adjusting these settings is a challenging ongoing process that requires the designer's expertise. The optimization process can help us find the best fit parameters. In this context, we find that Bayesian optimization is well adapted to optimize black box functions that are time-consuming to evaluate, like the deep learning training process. The optimization algorithm keeps track of the objective function's Gaussian process model and trains it with objective function evaluations.

Because the evolution of the objective function is stochastic, the Bayesian method considers it as a random function and applies a prior to it. Which can encode the function's behavior after accumulating the function evaluations, which are treated as data. Then, the prior is modified to create the posterior distribution over the objective function. This posterior distribution is utilized to generate an acquisition function that specifies the query point for the next iteration. There are several approaches to defining the prior/posterior distribution over the objective function. Gaussian processes are used in the most prevalent two approaches. It creates a surrogate for the objective function, utilizes a Bayesian machine learning approach called Gaussian process regression to quantify the uncertainty in that surrogate, and then utilizes an acquisition function created from the surrogate to select wherever to sample. The Bayesian optimization method uses Gaussian process regression and three common acquisition functions: predicted improvement, entropy search, and knowledge gradient.

We choose to optimize three parameters; one training hyperparameter, which is the learning rate; and two parameters relative to the design of the network. We optimize the number of the stacked recurrent layers (depth) and the number of recurrent cells in each one.

The network has three sections. The first one is composed of a sequence input layer. The second contains a recurrent layer followed by a dropout layer. This section is used to control the network depth by repeating it several times according to depth value. The last section has a fully connected layer. At the top there is a regression layer to compute loss.

Table 6. Parameter optimization ranges and search space

Parameter	Range	Search space
Initial learning rate	0.001-0.5	log
Cell number	10-60	integer
Network depth	1-6	integer
Objective evaluation iteration	1200	/

Then, we define the objective function to be minimized, this function is the sum of the MAPE predictions errors of the three classes defined as follow:

$$f(cell, depth, learningrate) = \underset{cell,depth,learningrate}{\arg\min} \sum_{i=1}^{3} MAPE_i, \quad (28)$$

Were $i \in \{kurtosis, \text{shapefactor}, \text{marginfactor}\}$, $cell \in \{10,11,\dots,60\}$, $depth \in \{1,2,\dots,6\}$, $learningrate \in [0.001,0.5]$,

In Table 6, we define the searching ranges, the types of the variables (integer, categorical, or real) and the search space transformation (linear or logarithmic) for the three optimization variables. The optimization process is performed for 1200 iterations.

In Table 7 and in Fig. 17, we can find the results of the optimization process after performing 134 iterations. Get by the best scored RNN is composed of 2 recurrent layers, using 34 cells in the first one and 14 in the second layer. With learning rate set to 0.0369, we can get the best objective function of 1.8505% which is the sum of relative errors in all classes. This prediction result is significantly more accurate than the MAPE given by the best non optimized GRU network, with 3.8760% of prediction errors as shown in the previous section.

Table 7. Values offering the best objective function compared to the best non optimized network

Network	Depth	Cell1	Cell2	Initial learning rate	Best fit (MAPE)
Optimized	2	34	14	0.0369	1.8505%
GRU	1	10	/	0.05	3.8760%

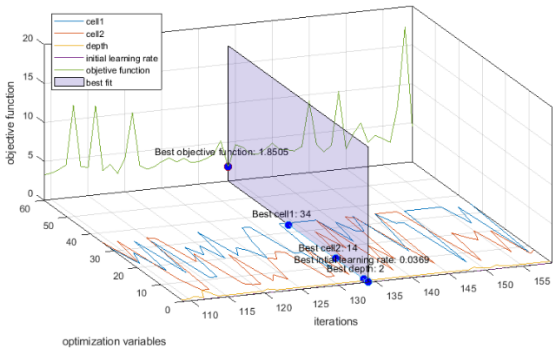

Fig. 17. Optimization parameters and best fit

6. CONCLUSION AND FUTURE WORK

In this work, we further studied the use of recurrent neural networks in the prediction and prognosis of the operational state of a wind turbine. We mainly based our study on the models derived from LSTM, all of which includes a cell memorizing long-term temporary dependencies. This cell is updated using gates. This investigation showed us the power of these tools in predicting time series data trends. We also explored architectures comprising several stacked recurrent layers able to learn more complex data dependencies over time.

Even though there isn't a lot of data available and it doesn't cover a long period of the time, the proposed models still manage to capture the dynamics and trends of the wind turbine's operating condition getting worse. We find that the most accurate predictions are made by GRU-based networks, which, despite having a simpler decision system then traditional LSTMs, are better at learning and capturing data trends.

Before training a network, the design and definition of hyperparameters is a very important step which will directly impact the predictive performance of the model. This phase is often problematic and depends on the experience of the designer. Using Bayesian optimization loop to effectively tune those parameters gives significant enhancement of the accuracy of the prediction. In light of this fact, we plan to conduct further studies on the optimization of network architecture parameters like network depth and the training hyperparameters using more complex optimization algorithms like particle swarm or genetic optimization. Taking the optimal parameters can significantly improve the performance of the network.

Another very promising avenue is to include feature extraction steps in network layers to link an end-to-end architecture and thus avoid bottlenecks in data processing. This type of layer can integrate several types of features, such as the Fourier transform, the spectrogram, and the wavelet.

Author contributions: *research concept and design, A.K., R.K.; Collection and/or assembly of data, A.K., R.K.; Data analysis and interpretation, A.K., R.K.; Writing the article, A.K., R.K.; Critical revision of the article, A.K., R.K.; Final approval of the article, A.K., R.K.*

Declaration of competing interest: *The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.*

REFERENCES

1. Lipinski D, Majewski M. System for monitoring and optimization of micro- and nano-machining processes using intelligent voice and visual communication. In Lecture Notes in Computer Science; Springer: Berlin, Germany, 2013;8206:16-23.
2. Majewski M, Kacalak W. Smart control of lifting devices using patterns and antipatterns. Advances in intelligent systems and computing. In Artificial Intelligence. Trends in Intelligent Systems; Springer: Cham, Switzerland, 2017;573:486-493. https://doi.org/10.1007/978-3-319-57261-1_48.
3. Ganga D, Ramachandran V. Adaptive prediction model for effective electrical machine maintenance. Journal of Quality in Maintenance Engineering 2020; 26(1):166-180. https://doi.org/10.1108/JQME-12-2017-0087.
4. Demidova L, Marchev D. Development of the forecasting model for the complex technical systems' failures time during the proactive maintenance using the recurrent neural networks' technology. 2nd International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA). 2020:370-374. https://doi.org/10.1109/SUMMA50634.2020.9280781.
5. Cherif H, Benakcha A, Laib I, Chehaidia S, Menacer A, Soudan B, Olabi A-G. Early detection and localization of stator inter-turn faults based on discrete wavelet energy ratio and neural networks in induction

motor. Energy. 2020;212:118684. https://doi.org/10.1016/j.energy.2020.118684.
6. Rosato A, Araneo R, Andreotti A, Succetti F, Panella M. 2-D Convolutional Deep Neural Network for the Multivariate Prediction of Photovoltaic Time Series. Energies. 2021;14:2392. https://doi.org/10.3390/en14092392.
7. Azzouzi M, Diarra R, Popescu D. Fault diagnosis of sensors, actuators and wind turbine system. Diagnostyka. 2018;19(4):3-10. https://doi.org/10.29354/diag/93846.
8. Tarek K, Abdelaziz L, Zoubir C, Kais K, Karim N. Optimized multi layer perceptron artificial neural network based fault diagnosis of induction motor using vibration signals. Diagnostyka. 2021;22(1):65-74. https://doi.org/10.29354/diag/133091.
9. Mana M, Piccioni E, Terzi L. Wind turbine fault diagnosis through temperature analysis: an Artificial Neural Network approach. Diagnostyka. 2017;18(1): 9-16.
10. Sajid H, Hossam AG. Vibration analysis and time series prediction for wind turbine gearbox prognostics. International Journal of Prognostics and Health Management; Special Issue on Wind Turbine PHM. 2013;4(3). https://doi.org/10.36001/ijphm.2013.v4i3.2144.
11. Mao W, He J, Tang J, Li Y. Predicting remaining useful life of rolling bearings based on deep feature representation and long short-term memory neural network. Advances in Mechanical Engineering 2018; 10(12). https://doi.org/10.1177/1687814018817184.
12. Liu, ZH., Meng, XD., Wei, HL, Chen L, Lu BL, Wang ZH, Chen L. A Regularized LSTM method for predicting remaining useful life of rolling bearings. Int. J. Autom. Comput. 2021;18:581-593. https://doi.org/10.1007/s11633-020-1276-6.
13. Xie Y, Zhao J, Qiang B, Mi L, Tang C, Li L. Attention mechanism-based CNN-LSTM model for wind turbine fault prediction using SSN ontology annotation. Wireless Communications and Mobile Computing. 2021:627588. https://doi.org/10.1155/2021/6627588.
14. Khorram A, Khalooei M, Rezghi M. End-to-end CNN + LSTM deep learning approach for bearing fault diagnosis. Applied Intellifence 2021;51:736-751. https://doi.org/10.1007/s10489-020-01859-1.
15. Baskar P-K, Kaluvan H. Long short-term memory (LSTM) recurrent neural network (RNN) based traffic forecasting for intelligent transportation. AIP Conference Proceedings 2022;2435:020039. https://doi.org/10.1063/5.0083590.
16. Pascanu R, Mikolov T, Bengio Y. On the difficulty of training recurrent neural networks. International conference on machine learning PMLR 2013:1310-1318.
17. Bechhoefer E, Van Hecke B, He D. Processing for improved spectral analysis. Annual Conference of the Prognostics and Health Management Society 2013.
18. Jaouher B, Saidi L, Harrath S, Bechhoefer E, Benbouzid M. Online automatic diagnosis of wind turbine bearings progressive degradations under real experimental conditions based on unsupervised machine learning. Applied Acoustics 2017;133:167-181. https://doi.org/10.1016/j.apacoust.2017.11.021.
19. Brownlee J. How to scale data for long short-term memory networks in Python. 2017. https://machinelearningmastery.com/how-to-scale-data-for-long-short-term-memory-networks-in-python/.
20. Hochreiter S, Schmidhuber J. Long short-term memory. Neural computation. 1997;9(8):1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735.
21. Kerboua A, Metatla A, Kelaiaia R, Batouche M. Real-time safety monitoring in the induction motor using deep hierarchic long short-term memory. Int J Adv Manuf Technol. 2018;99:2245–2255. https://doi.org/10.1007/s00170-018-2607-4.
22. Schuster M, Paliwal K. Bidirectional recurrent neural networks. IEEE transactions on Signal Processing. 1997;45(11):2673-2681. https://doi.org/10.1109/78.650093.
23. Cho K, Van Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv. 2014. https://doi.org/10.48550/arXiv.1406.1078.
24. Wang S, Wang X, Wang S, Wang D. Bi-directional long short-term memory method based on attention mechanism and rolling update for short-term load forecasting. International Journal of Electrical Power & Energy Systems. 2019;109(2):470-479. https://doi.org/10.1016/j.ijepes.2019.02.022.
25. He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. Proceedings of the IEEE international conference on computer vision. 2015; 1026-1034. https://doi.org/10.48550/arXiv.1502.01852.
26. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A Simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 2014;\ 15:1929-1958.
27. Kingma D-P, Jimmy B. Adam: A method for stochastic optimization. arXiv 2014:1412.6980. https://doi.org/10.5555/2627435.2670313.

Received 2022-04-21
Accepted 2022-06-25
Available online 2022-06-27

Adlen KERBOUA
Received his MSc (2004) and magister (2012) in computer sciences, PhD (2018). Thesis: improving industrial safety using intelligent system. Currently, he is associate professor in the University of Skikda, Algeria.

Ridha KELAIAIA
Received the magister in Robotics and Ph.D. degree in 2012. His current position is a full professor at the University of Skikda. His research interests are oriented towards Robotics, Electromechanics and Engineering Applications of Artificial Intelligence.