LSTM recurrent neural network for energy demand forecasting

1st Rodrigo G. Alarcón
Universidad Tecnológica Nacional (UTN)
Facultad Regional Reconquista (FRRQ), Argentina
ralarcon1493@comunidad.frrq.utn.edu.ar

2nd Martín A. Alarcón
Universidad Tecnológica Nacional (UTN)
Facultad Regional Reconquista (FRRQ), Argentina
malarcon@comunidad.frrq.utn.edu.ar

3rd Alejandro H. González
INTEC, CONICET, Universidad Nacional del Litoral (UNL)
Facultad de Ingeniería Química (FIQ), Argentina
alejgon@santafe-conicet.gov.ar

4th Antonio Ferramosca
Department of Management, Information
and Production Engineering University of Bergamos, Italy
antonio.ferramosca@unibg.it

***Abstract*—Recurrent Neural Networks (RNN) of the Long Short Term Memory (LSTM) type provide high accuracy in predicting sequential models in various application domains. As in most process control problems, their dynamics include non-manipulated variables that need to be predicted. This paper proposes using an LSTM neural network for energy demand forecasting, which applies to an Economic Model Predictive Control (EMPC) as a forecasting tool. For the training, data are taken from a three-phase intelligent power quality analyser located at the National Technological University, Reconquista Regional Faculty (Santa Fe, Argentina). A recursive strategy is used to update the state of the neural network and forecast over different prediction horizons. The accuracy achieved in training the neural network is measured using the root mean square error (RMSE) metric. Experimental results show that the proposed LSTM neural network has excellent generalisation capability.**

***Index Terms*—Recurrent neural network, Long short term memory, Forecasting, Energy demand, Economic model predictive control.**

I. Introduction

In recent years, the main emphasis has been put on the development and deployment of microgrids to meet energy demand efficiently and cost-effectively [1]. The case for microgrids is reinforced by the increase in renewable energy resources, such as wind and solar, as well as by the growing number of small-scale distributed generation systems. The control system plays a fundamental role in their implementation. Economic Model Predictive Control (EMPC) is a powerful and useful tool to handle complex multivariable systems, where the actions to achieve the objective are chosen from a feasible set, based on minimising some pre-established economic criterion over a finite time horizon. [2] proposes a management strategy using this approach for a microgrid connected to an electrical grid. In addition, allowing direct consideration of the disturbances present in the system. Energy demand is considered as a perturbation since there is no control over it and it should be considered since it modifies the behaviour of the system. Therefore, the planning and management of microgrids, while integrating renewable energies and distributed generation resources, require an accurate prediction of energy demand at different time horizons [3].

The information extracted from our representative historical energy demand data is an ordered sequence of values recorded at equal time intervals; which is known as a time series. Time series forecasting is a process that uses past values of a dependent variable to predict its future values. In other words, time series forecasting models attempt to understand the patterns found in time series. These patterns may include seasonality, trend and noise. The main drawback of forecasting time series is its complexity, since they add the complexity of a sequence dependence between the input variables.

Deep learning techniques, in particular neural networks, are currently gaining relevance for solving a large number of applications in multiple areas due to improvements in their computational capabilities. A powerful type of neural network designed to handle sequence dependence is called RNN (Recurrent Neural Networks), among which the most widespread are the LSTM (Long Short Term Memory), BiLSTM (Bi-directional long-short term memory) and the GRU (Gated recurrent unit) networks.

LSTM networks are widely used in time series forecasting problems because their design allows information to be remembered over long periods and facilitates the task of making future estimates using periods of historical records. This type of network is the one that yields the best results, given that the GRU network is a simplified version of the LSTM, improving only the computational burden of the network and without obvious better prediction results. On the other hand, the BiLSTM network hasn't application in prediction time series and it is largely used for image and data classification.

This work focuses on training an RNN-LSTM to serve as an energy demand forecasting tool to be implemented in an EMPC with different prediction horizons. The technique's performance is evaluated for the energy installation of the National Technological University, Reconquista Regional Faculty (Santa Fe, Argentina), using as input the data recorded by a three-phase intelligent power quality analyser. In addition, a hyperparameter search is performed by applying a Bayesian optimisation method, as opposed to [4] and [5], where the selection of hyperparameters is performed by testing different combinations and evaluated by validation methods.

The document is organized as follows. Section II presents the implemented methodology, where the model used, the processing and analysis of the database, and the learning algorithm to train the neural network. Section III describes the architecture of the implemented neural network and an alternative for the search for the hyperparameters with Bayesian optimisation methods. Section IV discusses the results, and Section V summarizes the main conclusions.

II. Methodology

A. Model description

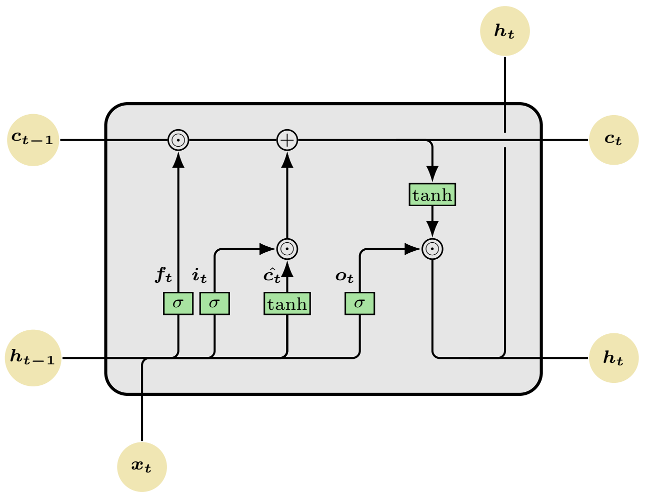

Figure 1. Basic architecture of the LSTM unit. $\boldsymbol{x_t}$ is the input at time t, $\boldsymbol{h_t}$ is the predicted output at time t, $\boldsymbol{h_{t-1}}$ is the predicted output at time $t-1$ (previous output), $\boldsymbol{c_{t-1}}$ is the information passed from the previous stage (memory) and $\boldsymbol{c_t}$ is the information passed to the next stage.

The LSTM neural network was first proposed by [6] and is widely used today due to its superior performance in accurately modelling both short- and long-term data dependencies.

Mathematically, suppose that there are h hidden units, the batch size is n, and the number of inputs is d. Each memory cell is equipped with an internal state and three multiplicative units known as logic gates: (i) input $\boldsymbol{i_t} \in \mathbb{R}^{n\times h}$, (ii) output $\boldsymbol{o_t} \in \mathbb{R}^{n\times h}$ and (iii) forget $\boldsymbol{f_t} \in \mathbb{R}^{n\times h}$. The operations performed on these gates can be explained by:

$$\boldsymbol{i_t} = \sigma\left(\boldsymbol{x_t}\,\boldsymbol{w_{xi}} + \boldsymbol{h_{t-1}}\,\boldsymbol{w_{hi}} + \boldsymbol{b_i}\right) \tag{1}$$

$$\boldsymbol{o_t} = \sigma\left(\boldsymbol{x_t}\,\boldsymbol{w_{xo}} + \boldsymbol{h_{t-1}}\,\boldsymbol{w_{ho}} + \boldsymbol{b_o}\right) \tag{2}$$

$$\boldsymbol{f_t} = \sigma\left(\boldsymbol{x_t}\,\boldsymbol{w_{xf}} + \boldsymbol{h_{t-1}}\,\boldsymbol{w_{hf}} + \boldsymbol{b_f}\right) \tag{3}$$

where $\sigma()$ is the sigmoid activation function, $\boldsymbol{x_t} \in \mathbb{R}^{n\times d}$ refer to the input variable at the time t, $\boldsymbol{h_{t-1}} \in \mathbb{R}^{n\times h}$ denotes the prediction output at the time $t-1$, $\boldsymbol{w_{xi}}, \boldsymbol{w_{xo}}, \boldsymbol{w_{xf}} \in \mathbb{R}^{d\times h}$ and $\boldsymbol{w_{hi}}, \boldsymbol{w_{ho}}, \boldsymbol{w_{hf}} \in \mathbb{R}^{h\times h}$ are weight parameters and $\boldsymbol{b_i}, \boldsymbol{b_o}, \boldsymbol{b_f} \in \mathbb{R}^{1\times h}$ are bias parameters.

The output of the network is indicated by the variable $\boldsymbol{c_t} \in \mathbb{R}^{n\times h}$, which is the memory cell at the time t and $\boldsymbol{h_t} \in \mathbb{R}^{n\times h}$ is the predicted variable at the time t. The outputs are given by:

$$\boldsymbol{\hat{c}_t} = \tanh\left(\boldsymbol{x_t}\,\boldsymbol{w_{xc}} + \boldsymbol{h_{t-1}}\,\boldsymbol{w_{hc}} + b_c\right) \tag{4}$$

$$\boldsymbol{c_t} = \boldsymbol{f_t} \odot \boldsymbol{c_{t-1}} + \boldsymbol{i_t} \odot \boldsymbol{\hat{c}_t} \tag{5}$$

$$\boldsymbol{h_t} = \boldsymbol{o_t} \odot \tanh\left(\boldsymbol{c_t}\right) \tag{6}$$

where $\odot$ is the Hadamard product operator, $\tanh()$ is the hyperbolic tangent activation function, $\boldsymbol{w_{xc}} \in \mathbb{R}^{d\times h}$ and $\boldsymbol{w_{hc}} \in \mathbb{R}^{h\times h}$ are weight parameters and $\boldsymbol{b_c} \in \mathbb{R}^{1\times h}$ bias parameter; and $\boldsymbol{\hat{c}_t} \in \mathbb{R}^{n\times h}$ is the temporal memory cell at time t. Figure 1 shows the architecture of the LSTM unit.

B. Acquisition and analysis of data

The database for training the network is built from representative historical samples recorded by an intelligent three-phase power quality analyser called ©Cloud Energy Meter, located at the faculty. It is integrated within a single device, an energy meter with harmonic analysis capability of the network with Wi-Fi and Bluetooth LE wireless connectivity. The Wi-Fi connectivity is used to access an internet connection, with which the device is linked to an associated web platform (called ©Field to Cloud) which allows the user to access historical and real-time data.

From the web platform, we can extract historical data of different electrical measurements, such as voltage, current, power factor, active power, reactive power and apparent power. Since the objective is to predict the energy demand, we extract the active power data. The platform gives us instantaneous and average power values, with sampling periods of 5 minutes.

Figure 2, at the top, shows the energy demand for a day when the complex has regular activity. Most of the activity in the Faculty, as seen in the figure in the areas where the greatest demand occurs, takes place in the afternoon and evening, i.e. from approximately 15:00 h until 23:00 h. On the other hand, the lower part shows a day with no activity, where only the indicated consumption of the different essential systems, such as servers and lighting, is being measured.

C. Descriptive analysis of the series

The measured data is interpreted as a univariate time series, where the study variable is the average active power computed over approximately one year, specifically between 01/01/2022 and 23/12/2022.

Observing the records for November, in Figure 3 (lower panel), a seasonal behaviour can be seen in the time series that is repeated periodically every week. During active days, energy consumption increases, contrary to the behaviour shows during inactive days (weekends and public holidays) when there is a notable drop in consumption. Analysing Figure 3 (upper panel), corresponding to data for July, it can be seen that in the intermediate region of the time series, there is reduced consumption of the installation. This is a consequence of the winter break, where the faculty is closed for two weeks. For

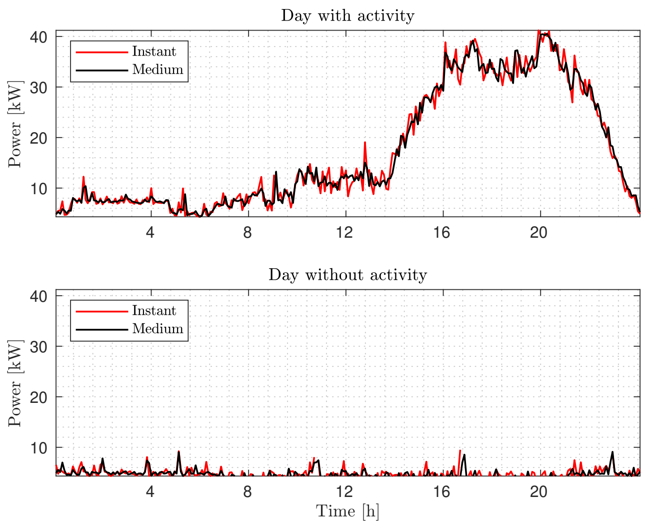

Figure 2. Energy demand for an active day and an inactive day. The upper part shows both the average active power (black line) and the instantaneous power (red line) for a day with normal activity in the faculty. The same applies to the lower part, but for a day without activity.

the remaining days, a similar behaviour to that presented for November is observed, but with lower power measurements. This is because air conditioning equipment are not used at this time of the year.

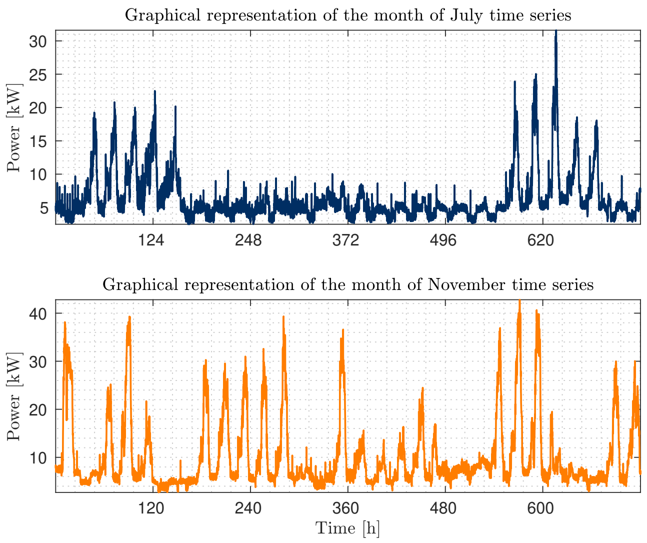

Figure 3. Graphical representation of the time series created with the average active power data. The upper part shows the time series for the month of July. The lower part shows the time series for the month of November.

D. Division of the data set

In general, the splitting of the dataset is done because the training algorithm learns from the data. They try to find or infer the pattern that allows them to predict the outcome for a new case. In order to determine whether a model works, we will need to test it on a different data set. Therefore, the data must be divided into three parts:

- dataTrain: Data set used to train the neural network. In other words, this is the set used for the neural network to learn patterns about the data.
- dataValid: Data set that tests the generalisation ability of the neural network at each training epoch. It serves to monitor the training, but does not intervene in any calculation.
- dataTest: Data set used to check how accurate the neural network is after training. It gives us a measure of generalising with a different dataset than the one used for training.

When dividing the data, it is necessary to ensure that the test set is large enough to generate statistically significant results, and that it is representative of the total set. In this case, 80 % of the initial records are reserved for training and the remaining 20 % for testing. From the training set 15 % is set aside for validation.

E. Pre-processing of data

To achieve a better fit and to prevent the training from diverging, it is necessary to perform a data scaling or standardisation process. Standardisation is a modification of the range of the original data so that all values are between 0 and 1. It is performed before the training process begins and makes the neural network less sensitive to the scale of the input values. In other words, normalisation ensures faster learning and that the convergence problem does not suffer from high variance, which makes optimisation possible.

The most commonly used normalisation is 'Linear function normalisation', which systematically changes the original data, assigns the result to the range of $[0, 1]$ and performs proportional scaling of the original data. To modify the data:

$$X_{\text{std}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{7}$$

where X is the original data and $X_{\max}$ and $X_{\min}$ are the maximum and minimum values in the data set, respectively.

F. Learning algorithm

The neural network is trained on a set of data in such a way that characteristic parameters are determined. The correct choice of parameters will make the predictions more or less accurate. It is posed as a numerical optimisation problem, the objective of which is to minimise a cost function responsible for quantifying the distance between the actual value and the value predicted by the neural network. The most commonly used cost function is the mean square error (MSE):

$$L(\boldsymbol{w}, \boldsymbol{b}) = \frac{1}{N_{\text{Train}}} \sum_{i=1}^{N_{\text{Train}}} (y_i - \hat{y}_i)^2 \tag{8}$$

where y is the actual value, $\hat{y}$ is the predicted value, N_{Train} is the number of elements in the training set and the notation $L(\boldsymbol{w}, \boldsymbol{b})$ refers to the fact that its value depends on the synaptic weights and biases that characterise the neural network, respectively. The optimisation problem is interpreting as the search for the parameters that minimise the function L, while penalizing the errors made by the neural network when making predictions.

A training algorithm is used to find the solution to the optimisation problem. The most commonly used algorithms are Stochastic Gradient Descent (SGD), Stochastic Gradient Descent with Momentum (SGDM), Root Mean Square Propagation (RMSProp), Adaptive Gradient Algorithm (AdaGrad) and Adaptive Moment Estimation (ADAM). In this paper, we use the *ADAM algorithm*, which was originally proposed by [7]. ADAM is an extension of SGD and a natural successor to AdaGrad and RMSProp that automatically adapts a learning rate for each input variable to the objective function and further smoothes the search process by using an exponentially decreasing moving average of the gradient to make updates of the variables. This algorithm was chosen because of its fast convergence and excellent results. Algorithms such as AdaGrad and RMSProp are similar to ADAM, and all perform very well in similar scenarios. However, the bias correction of the ADAM algorithm makes it faster and better than RMSProp when the gradient becomes sparse.

III. Implementation

A. Hyper-parameter optimisation

All neural network models have a set of parameters that cannot be learned from the data but have to be set after training. These parameters are often referred to as hyper-parameters. Since there is no way of knowing in advance what the appropriate values are, the only way to identify them is to try different combinations and evaluate them using resampling validation methods.

A different alternative is the hyperparameter search by means of Bayesian optimisation methods [8]. It belongs to a class of sequential model-based optimisation algorithms that allow us to use the results of our previous iteration to improve our sampling method for the next experiment.

The hyper-parameters selected for the optimisation process, with their corresponding ranges used for the experiment, are described in Table I. To take full advantage of the power of Bayesian optimisation, we perform 30 evaluations (maximum number of trials to run) with a different dataset than the one we will use for training. Once the trial is finished, we select the evaluation (iteration) with the lowest 'Training RMSE' and 'Validation RMSE' values. This optimisation process was performed with the app MATLAB® Experiment Manager®.

Based on the result obtained with Bayesian optimisation, training is performed by adopting the hyper-parameters shows in Table II.

B. Neural network architecture description

In order to predict future time step values of a sequence (time series), a 'Sequence to Sequence' regression LSTM neural network is trained. The responses are the training sequences with values shifted by one time step; i.e., at each time step of the input sequence, the neural network learns to predict the value of the next time step. To create the LSTM neural network architecture, MATLAB® Deep Network Designer® was used. This app allows us to create, visualise and edit all types of neural networks.

Table I
Description of the Hyper-parameters adopted for the optimisation. The interval $[\backsim, \backsim]$ or category $[" \backsim ", " \backsim "]$ of analysis is specified.

Hyper-parameters	Range	Description
InitialLearnRate	$[0.01, 0.0001]$	Initial learning rate used for training
LearnRateDropPeriod	$["25", "50"]$	Number of epochs for dropping the learning rate by a factor of 0.1.
NumHiddenUnits	$[50, 300]$	Number of neurons in the hidden layer.
MaxEpoch	$[50, 150]$	Maximum number of epochs to use for training.
MiniBatchSize	$["64", "128", "256"]$	Size of the mini-batch to use for each training iteration.

Table II
Value of the hyper-parameters adopted for training.

Hyper-parameters	Value
InitialLearnRate	0.01
LearnRateDropPeriod	50
NumHiddenUnits	98
MaxEpoch	59
MiniBatchSize	128

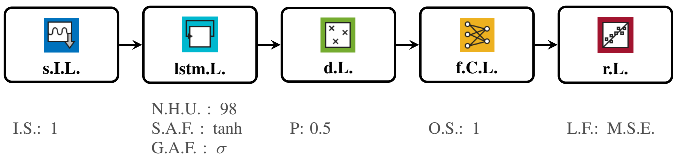

Figure 4. Architecture of the LSTM neural network used, designed for 'sequence to sequence' regression tasks. The first layer is the sequential input, the second is the LSTM unit, the third is the dropout layer, the fourth is the multiple connection layer and the fifth is the regression layer.

Figure 4 shows the neural network architecture used. The first block 'sequencelInputLayer (s.I.L.)' represents the sequential input layer, where the training data is stored. This data has only one input feature, hence 'InputSize (I.S.) = 1'.

The second block is the LSTM unit 'lstmLayer (lstm.L.)'. It is also known as the hidden unit of the neural network to be modelled and is the one that will learn long term dependencies between time steps in time series and sequence data. In it, we can configure some features, such as the number of hidden neurons 'NumHiddenUnits (N.H.U.) = 98', the activation function to update the hidden cells and state 'StateActivationFunction (S.A.F.) = hyperbolic tangent'; and the activation function for logic gates 'GateActivationFunction (G.A.F.) = sigmoid'.

The third block 'dropoutLayer (d.L.)' is called the dropout layer. Its function is to shut down some neurons during training. This forces the neurons to be robust and not rely on the activity of other specific neurons. The purpose of this block is to mitigate the possible occurrence of the phenomenon known as overfitting. It is set with a probability of occurrence of 0.5 'Probability (P) = 0.5'.

The fourth block is the multiple connection layer 'fully-ConnectedLayer (f.C.L.)', which consists of a fully connected layer that multiplies the input by a matrix of sympathetic weights and then adds a bias vector. The output data, like the input data, has a unique output characteristic, i.e. 'OutputSize (O.S.) = 1'.

In the fifth and last block is the regression layer 'regressionLayer (r.L.)', which shows the value resulting from the prediction at a given time. In addition, it uses to evaluate the neural network performance, it uses a loss function (Mean Square Error) 'LossFunctions (L.F.) = M.S.E'.

C. Training Options

In order to achieve adequate training of the proposed neural network, it is possible to create a set of training options [9]. These options are parameterised according to the characteristics of the neural network, as shown next.

Training Options: Configurable options for LSTM network training.

```
options=trainingOptions('adam', ...
    'VerboseFrequency',25, ...
    'ValidationData',{XValid,YValid}, ...
    'ValidationFrequency',10, ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',59, ...
    'MiniBatchSize',128, ...
    'ExecutionEnvironment','cpu', ...
    'SequenceLenght','shortest', ...
    'GradientThreshold',1, ...
    'GradientThresholdMethod','l2norm', ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropFactor',0.1, ...
    'LearnRateDropPeriod',50, ...
    'Plots','training-progress');
```

IV. Results

There are different techniques defined in [10] for the problem of multi-step prediction over time. The recursive strategy involves adding the last prediction of the last time interval as input to the next prediction; in this way we define a single output model and a recursive prediction system up to the fixed limit. In other words, it is necessary to update the state of the network after each prediction. Since we have access to the actual (observed) values, we can update the network state with these values. This will allow us to prevent previous predictions from affecting new predictions. Before making any predictions, we must initialise the state of the neural network. For this, we use the training data. To corroborate the level of generalisation of the neural network, as explained above, we use the dataTest set corresponding to 20 % of the total data set.

For the first test of the neural network, we want to predict the electricity consumption for a whole month. Figure 5 shows the forecast made for October. The red curve shows the prediction made by the neural network, while the grey curve represents the observed values.

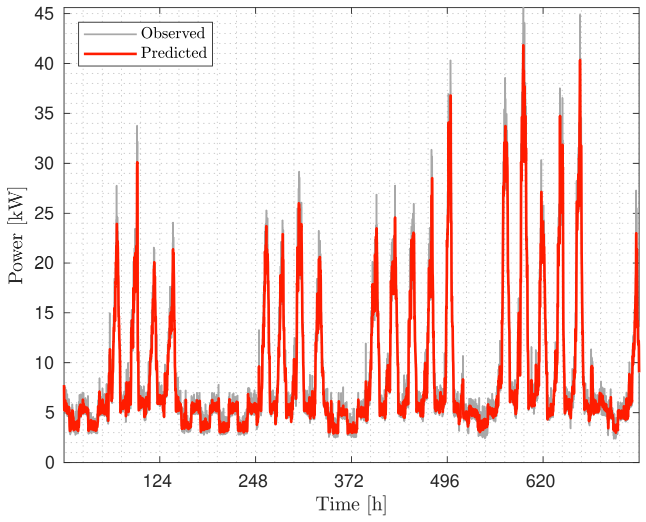

Figure 5. Prediction made for October. The red curve represents the prediction made by the neural network and the grey curve the observed values.

A. Forecasting with different horizons

Since the objective is to train a neural network to serve as an energy demand forecasting tool to be implemented in an EMPC with different prediction horizons, simulations are carried out for three possible scenarios.

The root means that square error (RMSE) metric is used for measuring the neural network's performance. In addition, box and whisker plots are employed to improve the interpretation and visualisation of the RMSE.

1) Predictions with 12 h horizon: Figure 6 shows the result obtained considering a 12 h horizon. It can be seen that the average RMSE error is 0.99 kW (green point), and it is above the limit corresponding to the third quartile. This is because of outliers (blue crosses), which disturb the mean. On the other hand, if we focus on the values in the third quartile (75 % of the data), we see that the RMSE values are below 0.86 kW. Consequently, there is a close match between the observed data (dataTest) and the prediction.

2) Predictions with 24 h horizon: Figure 7 shows the prediction made considering a 24 h horizon. We can notice that we only have a single outlier. In this context, the outliers represent an erroneous prediction made by the neural network. This can be due to false measurement in the observed data, caused by an outlier event that may have taken place in the faculty facilities. For this particular situation, we have an average RMSE error of 1.82 kW and in 75 % of the cases the error does not exceed 2.26 kW.

3) Predictions with 48 h horizon: Figure 8 shows the prediction made for a 48 h horizon. In this case, the average RMSE error is 1.82 kW and in 75 % of the cases the error

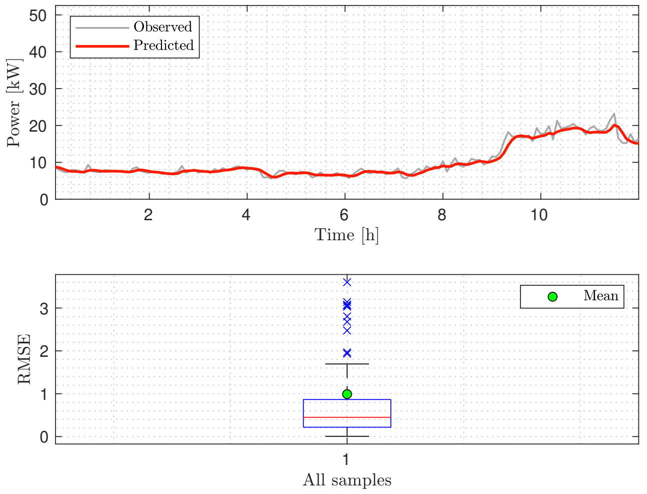

Figure 6. Predictions with 12 h horizon. The upper part shows the prediction made (red curve) compared to the test data (grey curve). The lower part shows the box and whisker plot corresponding to the RMSE error.

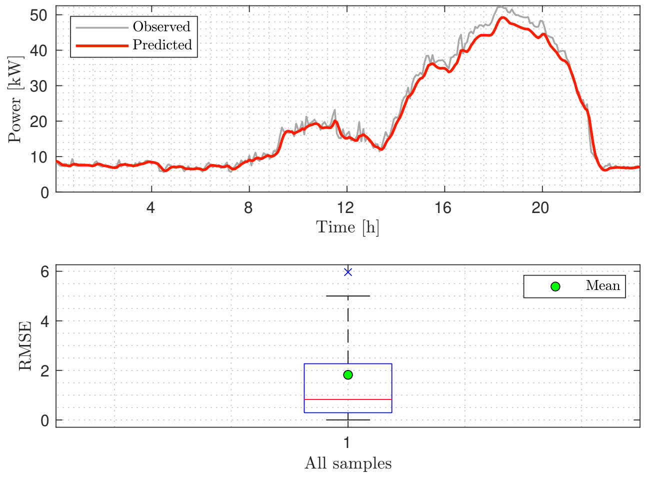

Figure 7. Predictions with 24 h horizon. The upper part shows the prediction made (red curve) compared to the test data (grey curve). The lower part shows the box and whisker plot corresponding to the RMSE error.

does not exceed 2.06 kW. Given the results obtained in the previous scenarios, it is possible to infer that the prediction horizon does not affect the quality of the forecast. This is due to the way the neural network state is updated with the observed values instead of the predicted values. Consequently, predictions are more accurate and are not adversely affected by a long prediction horizon.

V. Conclusions

In this paper, we propose a Recurrent Neural Network (RNN) of the Long-Term Memory (LSTM) type for energy demand forecasting over multiple prediction horizons, which is applicable as a forecasting tool to be implemented in an Economic Model Predictive Control (EMPC). A database was built with historical samples recorded by an intelligent three-phase power quality analyser located in the same facilities of the National Technological University, Reconquista Regional

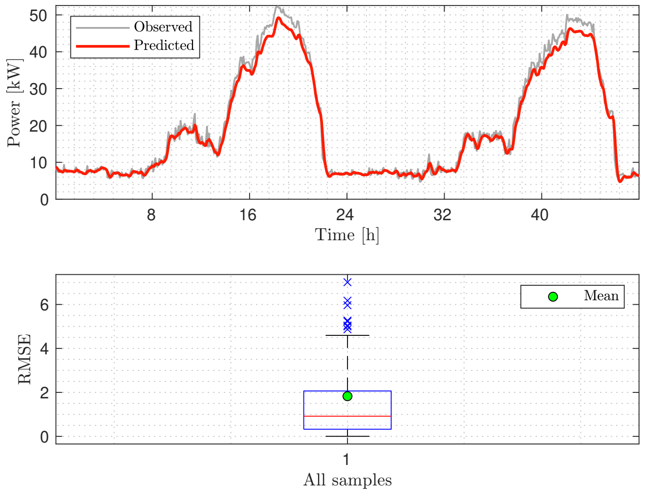

Figure 8. Predictions with 48 h horizon. The upper part shows the prediction made (red curve) compared to the test data (grey curve). The lower part shows the box and whisker plot corresponding to the RMSE error.

Faculty. With MATLAB Deep Network Designer® we created a LSTM neural network to predict values of future time steps of a sequence. In addition, we performed a hyperparameter search of the neural network by applying a Bayesian optimisation method, using a dataset independent of the one used for training, testing and validation. Finally, we use the Test dataset to evaluate the predictions made by the neural network with different prediction horizons (12 h, 24 h and 48 h). The results obtained show a good performance of the trained neural network. Depending on the prediction horizon adopted, the average RMSE error may vary. In the three case studies analysed the error does not exceed 2 kW.

References

[1] B. Lasseter, "Microgrids [distributed power generation]," in *2001 IEEE power engineering society winter meeting. Conference proceedings (Cat. No. 01CH37194)*, vol. 1. IEEE, 2001, pp. 146–149.
[2] M. A. Alarcón, R. G. Alarcón, A. H. Gonzalez, and A. Ferramosca, "Economic model predictive control for energy management of a microgrid connected to the main electrical grid," *Journal of Process Control*, vol. 117, pp. 40–51, 2022.
[3] C. Deb, F. Zhang, J. Yang, S. E. Lee, and K. W. Shah, "A review on time series forecasting techniques for building energy consumption," *Renewable and Sustainable Energy Reviews*, vol. 74, pp. 902–924, 2017.
[4] S. Arslan, "A hybrid forecasting model using lstm and prophet for energy consumption with decomposition of time series data," *PeerJ Computer Science*, vol. 8, p. e1001, 2022.
[5] S. Zagrebina, V. Mokhov, and V. Tsimbol, "Electrical energy consumption prediction is based on the recurrent neural network," *Procedia Computer Science*, vol. 150, pp. 340–346, 2019.
[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[8] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, "Hyperparameter optimization for machine learning models based on bayesian optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019.
[9] T. M. Inc., "Deep learning toolbox version: 9.9.0 (r2020b)," 2020.
[10] G. Bontempi, S. Ben Taieb, and Y.-A. L. Borgne, "Machine learning strategies for time series forecasting," in *European business intelligence summer school*. Springer, 2012, pp. 62–77.