uc3m | Universidad Carlos III de Madrid

Master Degree in Statistics for Data Science

Academic Year 2020-2021

Master Thesis

"Analysis of Deep Learning Strategies for Wind Energy Forecasting Applications"

Elianne Mora

Jenny Alexandra Cifuentes

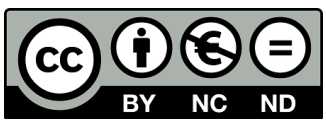
Geovanny Marulanda

Madrid, 2021

AVOID PLAGIARISM

The University uses the **Turnitin Feedback Studio** program within the Aula Global for the delivery of student work. This program compares the originality of the work delivered by each student with millions of electronic resources and detects those parts of the text that are copied and pasted. Plagiarizing in a TFM is considered a **Serious Misconduct**, and may result in permanent expulsion from the University.

un



ABSTRACT

Wind energy has been recognized as the most promising and economical renewable energy source, attracting increasing attention in recent years. Although wind energy has experienced steady growth worldwide throughout the last decade, the variability and uncertainty of wind energy pose challenges in the operation and planning of power systems. Therefore, accurate forecasting is crucial for wind power generation and operation systems, and to propel high levels of wind energy penetration within electricity markets.

In this work, a comparative framework is proposed where diverse deep learning architectures are implemented in order to address the existing gap and limitations of reported wind power forecasting methodologies. The methodology set forth implements a suite of long short-term memory (LSTM) recurrent neural networks (RNN) models inclusive of standard, bidirectional, stacked, convolutional, and autoencoder architectures. These integrated networks are implemented through an iterative process of varying hypeparameters to better assess their effect, and the overall performance of each architecture, when tackling one-hour to three-hours ahead wind power forecasting. It must be noted that some of the methodologies showcased in this work, such as the implementation of autoencoders for wind power applications, have not been explored in detail in the literature.

The proposed approach is validated through hourly wind power data from the Spanish electricity market, collected between 2014-2020. The proposed comparative error analysis shows that, overall, the models tend to showcase low error variability and better performance when the networks are able to learn in weekly sequences. Moreover, simpler architectures showcased better performance metrics and shorter implementation times. Overall, the model with the best performance forecasting one-hour ahead wind power is the standard LSTM implemented with weekly learning input sequences. In the case of three-hours ahead forecasting, the model with the best overall performance is the bidirectional LSTM implemented with weekly learning input sequences.

Keywords: Long Short-Term Memory, Deep Learning, Wind Power Forecasting

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The rising concerns over global warming, energy security, and the impact of Greenhouse gas (GHG) emissions on the global economy have increased interest in developing efficient renewable energy sources for the rapid replacement of fossil fuels [1]. Governments around the world have stepped up their climate ambitions, and by the end of 2020, 28 countries had issued "climate emergency" declarations, many of which were accompanied by plans and targets to transition to more renewable-based energy systems [2].

Wind power has emerged as the most promising and economical renewable energy source. The global wind power market expanded 19% in 2019 to 60 GW, for a total global capacity of 650 GW (621 GW onshore and the rest offshore) [2]. While economic factors are the driving force for the steady growth of wind power, the variability and uncertainty of wind energy pose challenges in the operation, scheduling, and planning of power systems, which ultimately increase generation costs [3]. Therefore, accurate forecasting is crucial for wind power generation systems, and to propel high levels of wind energy penetration.

According to schedulers, dispatcher and energy planners, among the most important was the need for a day-ahead wind power forecast for energy trading and the unit commitment schedule, followed by hourly forecasts (expressed in megawatts) inclusive of error bars and uncertainty intervals, and forecasts several days ahead for maintenance planning [4]. In order to meet these demands, numerous forecasting studies have been carried out focusing on wind speed and power forecasting, uncertainty forecasting, and ramp events forecasting in different time horizons [5], [6]. Throughout the years, wind energy forecasting has been tackled through different approaches such as physical or numerical weather prediction (NWP) models, statistical and probabilistic methods, intelligent forecasting models, and hybrid or ensemble methods. These models are mainly carried through four forecasting time horizons: very-short term (< 30 min.), short term (0.5-6 hrs.), medium term (6-24 hrs.), and long term (1-7 days) [7].

The physical method uses physical and weather information such as wind direction, roughness, obstruction, pressure, and temperature, to model wind power. Due to its high dependence on numerical weather forecast data, accuracy is greatly affected by the precision of numerical weather predictions [8]. On the other hand, statistical methods are mostly based on historical data, focusing on the time-varying relationships of given time sequences, often requiring high model orders to describe linear relationship of wind series. Some traditional statistical models are Kalman Filter (KF) and Autoregressive Integrated Moving Average model (ARIMA) [9]. However, in recent literature, research has moved towards the implementation of Machine Learning (ML) methods mainly due to the performance of such models in extracting robust features, which have reported a huge success in a wide range of applications [10]–[12]. ML models, such as Neural Networks

(NN) [13] and Support Vector Machines (SVM) [14], can establish a nonlinear mapping to accurately describe wind randomness, while Deep Learning (DL) and hybrid models better describe complex wind energy features [15], [16].

While ML and hybrid models may better capture the nonlinear dynamics of renewable generation processes, most models are non-trivial to tune and use in practice, depend on careful selection of input features and pre-treatment strategies, and have difficulties dealing with vanishing or exploding gradients [16], [17]. Due to the limitations and poor adaptability of traditional ML models, Deep Learning (DL) methods have been applied to wind power forecasting. Recurrent Neural Networks (RNN) have been specially highlighted to model the relationships among samples of time sequences. More specifically, Long Short-Term Memory (LSTM) networks have been successful in modeling long short-term dependencies [18], [19].

In the literature, different wind power forecasting strategies have been developed through the implementation of diverse LSTM models. In [20], medium-term wind power forecasting was performed using LSTM networks on historical wind power data and NWP data, where the latter data type was first analyzed by Principal Component Analysis (PCA) to narrow down the meteorological inputs. The study produced similar results for both LSTM and PCA-LSTM results; however, the latter hybrid model slightly outperformed the LSTM, yet both proved to be superior when compared to SVM models and back propagation (BP) neural networks performance metrics. In [21], short-term wind power forecasting was developed, for one to five steps ahead, based on discrete wavelet transform (DWT) and LSTM networks. Original wind power data from three different wind farms were decomposed by DWT into low-frequency and high-frequency sub-signals, such that independent LSTMs approximate the temporal dynamic behaviors of the sub-signals, respectively. The proposed DWT LSTM model proved to perform better when compared to RNN, and BP methods alone; however, when steps ahead increased, the performance metrics deteriorated. In [22], a combined model consisting of the variational mode decomposition (VMD), Convolutional LSTM network (ConvLSTM) and error analysis was conducted for short-term wind power forecasting. The proposed VMD-ConvLSTM-LSTM model was implemented on historical output data series from two wind turbines and two wind farms. The proposed model proved to outperform traditional forecasting algorithms; however, model parameters were selected by expertise given the large computational cost associated with estimation of optimal parameters. In [23], a multivariate stacked LSTM model is proposed. Historical wind speed, wind direction, temperature, humidity, pressure, dew point and solar radiation are used as inputs to predict future short-term wind speed. The study proposes a model with two LSTM layers stacked after the input layer, with 64 neurons each. The results proved to outperform multiple competing statistical methods, including Multiple Linear Regression, LASSO, and Ridge based strategies.

Although the verified performance of LSTM and hybrid LSTM models surpass statistical and ML methods, the above literature review highlights both the advancements and existing gaps in wind power forecasting. Few research has focused on the comparison and

evaluation among different DL models (LSTM, Bi-LSTM, Conv-LSTM, Stacked-LSTM, Autoencoder-LSTM) and to the best of our knowledge, a comparative analysis given the estimation of optimum hyperparameters for each DL model has not been applied to wind power forecasting. In addition although Bi-LSTM and Autoencoder-LSTM have been widely used to forecast meteorological time series, they have not been deeply analyzed in the case of wind power forecasting. Our aim is to fill these research gaps.

In this work, hourly wind power data from the Spanish electricity market is processed through a suite of LSTM RNN-based models, for single and multiple steps ahead (up to three) forecasting. A descriptive analysis of the time series data is performed to identify seasonal trends that may positively affect forecasting. Also, a comparative error analysis is produced given the diverse modeling strategies employed, where significant hyperparameters, such as the number of iterations, memory units, batch size, and size of the input sequence were implemented through an iterative process aiming to estimate the model with the best predictive capabilities.

The structure of the thesis is as follows. Chapter 2 focuses on detailed descriptions of the DL techniques and the definition of the evaluation metrics. Chapter 3 describes the dataset analyzed and the implementation methodology used in this work. Chapter 4 compares the results and metrics, and finally, in Chapter 5, conclusions are drawn and future work is discussed.

2. DEEP LEARNING STRATEGIES FOR TIME SERIES FORECASTING

Deep learning (DL) is a subset of machine learning (ML) that uses restructured multi-layered Artifical Neural Networks (ANNs) to deliver improved accuracy in diverse tasks, such as object detection, speech recognition, language translation, among many other applications. Given the success of DL models in tackling classification and regression problems, research continues to move towards the implementation of diverse DL model architectures for a plethora of applications.

A particular DL model architecture has gained popularity over the years given its success in handling long and short term time dependencies, and its ability to work with raw time series data without the need for extensive feature engineering nor data pre-processing. This DL model is known as Recurrent neural networks (RNNs).

2.1. LSTM RNNs based Models Description

Recurrent neural networks (RNNs) are dynamic systems that efficiently use the temporal information of a model's input sequences. For this reason, they are a widely used tool for time series forecasting. RNNs are derived from Feed-forward Neural Networks (FFNN), where the network is forward propagated according to the number of time steps per sample. RNNs are composed by a memory gate that allows the network to process sequential data, maintaining its previous inputs to predict the outputs. The RNN model updates its memory, also known as its recurrent hidden state h_t, as follows:

$$h_t = \sigma\left(W_x x_t + W_h h_{t-1} + b_t\right) \tag{2.1}$$

where $x = (x_1, x_2, ..., x_t)$ is a sequence of length t, σ is a nonlinear function such as sigmoid or rectified linear unit (ReLu), W_x and W_h are weight matrices, and b_t is a constant bias value [24]. Standard RNNs suffer from short-term memory and are limited to look back in time, such that the larger the size of the input learning sequence, the less it learns. Moreover, RNNs struggle with long sequential data, leading to exploding and vanishing gradients, where the former refers to the assignment of high importance to the weight matrices without any reason, and the latter refers to significantly small gradient values that prevent the RNN model from learning further [24].

LSTM networks, originally proposed in 1997 [25], address the short-term memory problem of RNNs that lead to vanishing and exploding back propagation error signals, that either grow or shrink with every time step in RNNs. In order to overcome this problem, gated units to control the information flow have been proposed. Specifically, LSTM units have reported successful results in different fields to forecast time series ([8], [12],

[19]). LSTM networks are capable of modeling long-term dependencies in time series, learning more than 1,000 time steps, whereas RNNs are limited to 10 steps [26]. Given the competency of LSTM networks in handling long and short term time dependencies, the study developed in this work utilizes a suite of LSTM-based models to better capture the real-time dynamics of wind energy.

2.1.1. Vanilla LSTM

The traditional structure of a LSTM network is composed by three gates: the forget gate, input gate, and output gate. The forget gate determines which information is valuable enough to be maintained within the network, assigning a value between 0 and 1, such that values close to 0 are discarded while values close to 1 are kept. The input gate decides what information is allowed to flow into the memory cell and be stored there.

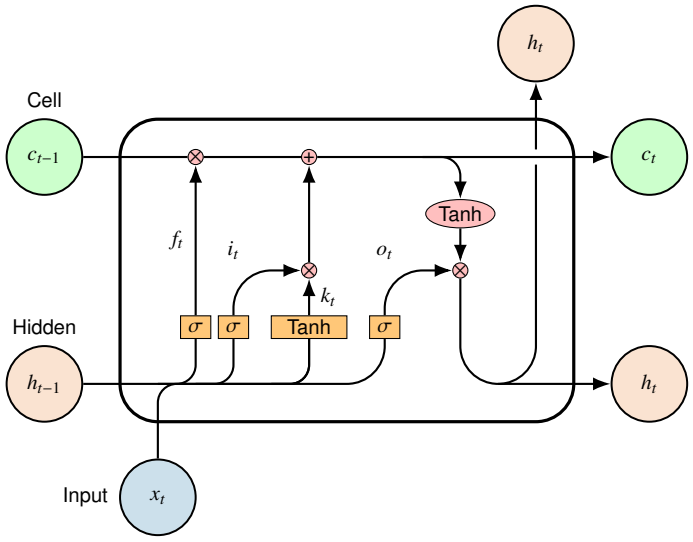


Fig. 2.1. LSTM Network Representation

Equations 2.3 and 2.4 below represent this decision, where k_t produces a vector of new values to be added to the memory cell based on the update signal given by i_t. The network is then updated as defined in Equation 2.5, where c_t (long-term state) considers whether to keep or forget the previous values and add new values. Lastly, the output of h_t (short-term state) is also composed by two layers. Here, o_t is the output value given by the σ activation, which determines what values from the memory cell to be considered as output, followed by a tanh layer with an output range [-1, 1] [27]. Equations 2.2-2.7 show how the network is updated, which was also illustrated in figure 2.1.

$$f_t = \sigma\left(W_f.[h_{t-1}, x_t] + b_f\right) \tag{2.2}$$

$$i_t = \sigma\left(W_i.[h_{t-1}, x_t] + b_i\right) \tag{2.3}$$

$$k_t = \tanh\left(W_k.[h_{t-1}, x_t] + b_k\right) \tag{2.4}$$

$$c_t = f_t \times c_{t-1} + i_t \times k_t \tag{2.5}$$

$$o_t = \sigma\left(W_o.[h_{t-1}, x_t] + b_o\right) \tag{2.6}$$

$$h_t = o_t \times \tanh(c_t) \tag{2.7}$$

where f_t, i_t, k_t, o_t are the output values of the forget gate, input gate, update signal, and output gate, respectively, and their input values are x_t at the current time t and the output value h_{t-1} at time $t-1$. $W_{f,i,k,o}$ are weight matrices and $b_{f,i,k,o}$ are bias vectors corresponding to the respective gates. σ is a nonlinear activation function, and c_t is the memory unit.

2.1.2. Bidirectional LSTM

While the Vanilla LSTM takes an unidirectional processing of the information, from left to right, the Bidirectional model learns the input time series sequences both forward and backwards, ultimately combining both interpretations. The information is processed through independent networks, where one network will process the time series data from left to right, while the other network will process the data from right to left. In this way, each network holds information at time t, while also holding information from either the past or the future.

We redefine the hidden state equation to incorporate the forward layers, backward layers and combined output. The forward layer's output sequence $\overrightarrow{h}$ is computed by going through the input sequence from left to right ($t-1$ to $t-n$), while the backward layer's output is computed from right to left using the reverse inputs. Ultimately, both outputs are computed as previously defined in equations 2.2-2.7.

Figure 2.2 exemplifies the incorporation of the direction for the forward and backward layers. The concatenated output is defined in Equation 2.8, where $W_{\overrightarrow{h}y}$ and $W_{\overleftarrow{h}y}$ are the weights of the forward and backward networks, respectively, and b_y is the bias of the output layer. Equation 2.8 can be generalized for multi-step forecasting problems such thet y_t would be otherwise computed given by y_{t+1}, y_{t+2}, and so on.

$$y_t = W_{\overrightarrow{h}y}\overrightarrow{h_t} + W_{\overleftarrow{h}y}\overleftarrow{h_t} + b_y \tag{2.8}$$

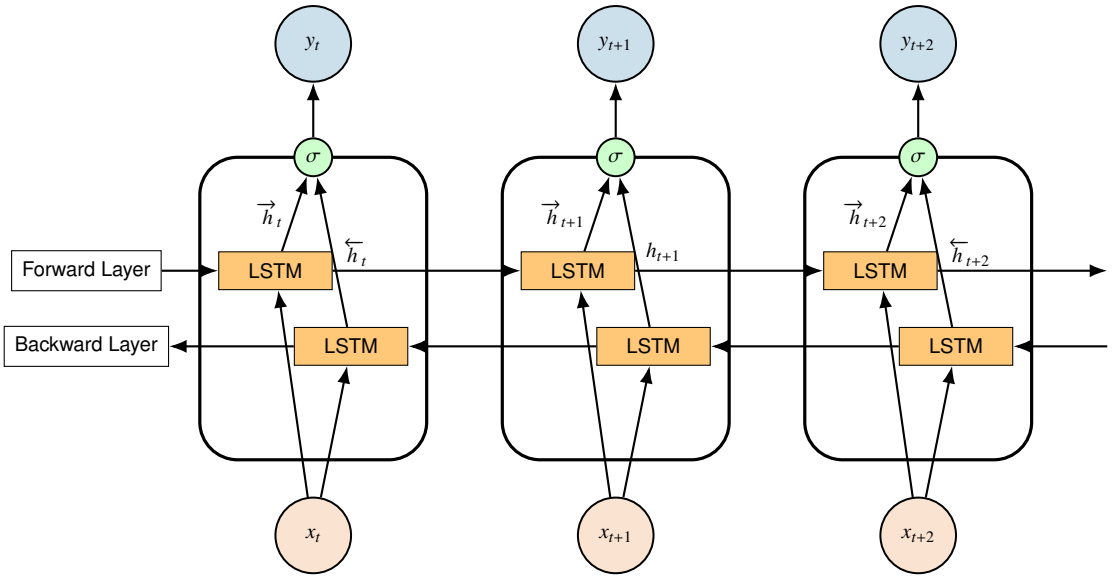


Fig. 2.2. Bidirectional LSTM Network Representation

2.1.3. Stacked LSTM

A stacked LSTM refers to a model architecture composed of multiple LSTM layers, making the model deeper. Under a standard LSTM model, the layer would receive and input sequence and output a value or a specified number of values to be predicted. However, under a stacked model, as it is shown in the Figure 2.3, the output of a hidden LSTM layer is not only propagated forward through time, but also used as one of the inputs for the next LSTM hidden layer. In this way, the hidden states are a function of all previous hidden states. Consequently, the l-th layer can be updated by the following equations:

$$f_t^l = \sigma\left(W_f^l.[h_{t-1}^l, h_t^{l-1}] + b_f^l\right) \tag{2.9}$$

$$i_t^l = \sigma\left(W_i^l.[h_{t-1}^l, h_t^{l-1}] + b_i^l\right) \tag{2.10}$$

$$k_t^l = \tanh\left(W_k^l.[h_{t-1}^l, h_t^{l-1}] + b_k^l\right) \tag{2.11}$$

$$c_t^l = f_t^l \times c_{t-1}^l + i_t^l \times k_t^l \tag{2.12}$$

$$o_t^l = \sigma\left(W_o^l.[h_{t-1}^l, h_t^{l-1}] + b_o^l\right) \tag{2.13}$$

$$h_t^l = o_t^l \times \tanh(c_t^l) \tag{2.14}$$

Based on this configuration, the input associated to the first layer is the raw time series:

$$h_t^0 = x_t, \tag{2.15}$$

while its respective output is an abstraction of the input, which is then fed as a hierarchical feature to the next LSTM layer. The final layer is composed by the number of neurons corresponding to the number of time steps to be predicted through the model. Different advantages have been reported for this methodology [28]. Some examples are:

- A stacked configuration allows the neural network to learn features from the raw time series in different components at each time step.
- The parameters associated to the neural network are spread across the whole model's space, which allows to accelerate the convergence of the algorithm and to fine-tune the nonlinear operations applied to the time series.

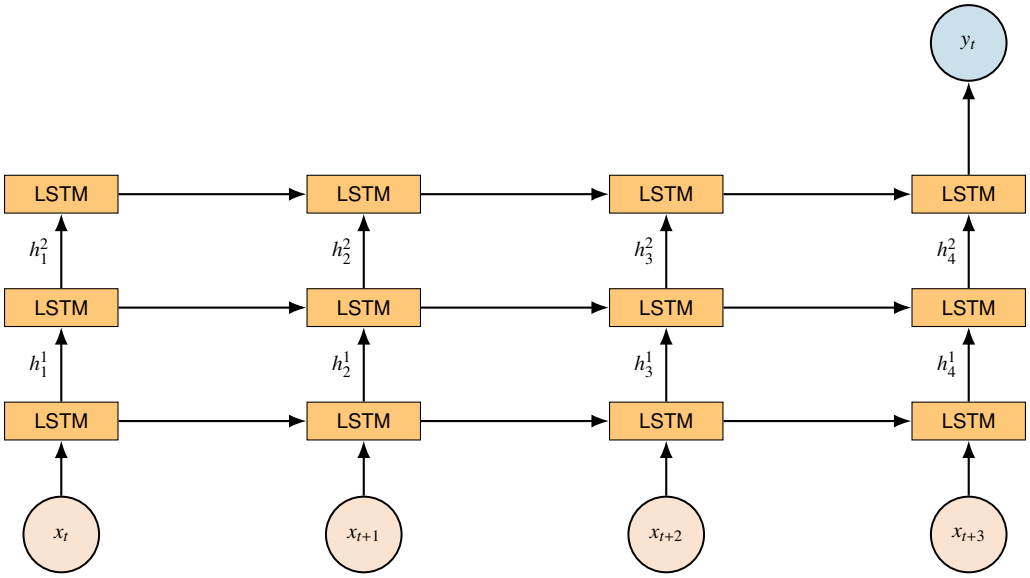


Fig. 2.3. Stacked LSTM architecture.

2.1.4. Convolutional LSTM

Convolutional Neural Networks (CNNs) have been widely used for feature extraction of data with grid-like configurations, such as images. The CNN architecture was originally developed for two-dimensional data, allowing the model to learn patterns hierarchically such that the patterns learned will be recognized throughout the network. Presently, one-dimensional CNNs have become more popular given their application to sequence data, such as text and time series, although their performance is suboptimal when compared to other models, like LSTMs [29].

CNNs are composed by three stacked layers: convolutional layer, pooling layer and fully connected layer. First, the input layer holds the input vector of values, followed by the convolutional layer where features are extracted through a filter or kernel, to be output into a fully connected layer [30]. The basic structure of CNNs is portrayed in figure 2.4. An LSTM network can be connected to the dense or fully connected layer of the CNN configuration as portrayed, such that the convolutional LSTM network would be updated in a similar way as in the Vanilla LSTM, replacing the matrix addition and dot product operators by convolutional operators.

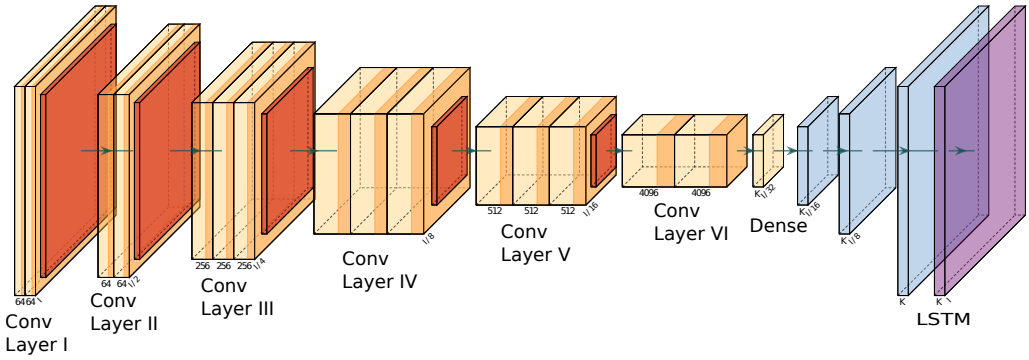


Fig. 2.4. Example of a CNN with six convolutional layers.

A Convolutional LSTM RNNs (ConvLSTM) incoporates the convolutional structures of the CNN in both the input-to-state and state-to-state transitions. The network is redefined such that the operators previously defined in 2.2-2.7 are replaced by the convolution operator ($*$) and the Hadamard product ($\odot$) as exemplified in equations 2.17-2.20, where the latter operator keeps the constant bias property of the cells. As it can be noted, these equations differ from the standard LSTM network in that in order to extend the LSTM network's capability to remember or forget information throughout the convolutional LSTM configuration, the information from the cell state is included in all output computations for each gate to take into account the analogous transitions between the states, which is exemplified in figure 2.5. Moreover, the weight matrices of the respective gates now include a convolution operator in order to incorporate the function of the filter. The network is updated as follows:

$$i_t = \sigma(W_{xi} * x_t + W_{hi} * h_{t-1} + W_{ci} \odot c_{t-1} + b_i) \tag{2.16}$$

$$f_t = \sigma(W_{xf} * x_t + W_{hf} * h_{t-1} + W_{cf} \odot c_{t-1} + b_f) \tag{2.17}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc} * x_t + W_{hc} * h_{t-1} + b_c) \tag{2.18}$$

$$o_t = \sigma(W_{xo} * x_t + W_{ho} * h_{t-1} + W_{co} \odot c_t + b_o) \tag{2.19}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.20}$$

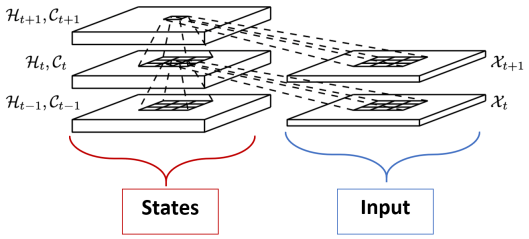


Fig. 2.5. Example of inner ConvLSTM structure.

2.1.5. Autoencoder LSTM

The Autoencoder (AE) architecture consists of three sequentially connected layers defined as the encoder, code, and the decoder. AEs extract features from input data in an unsupervised manner, and are often used as generative models [31], [32]. The AE architecture is illustrated in Figure 2.6.

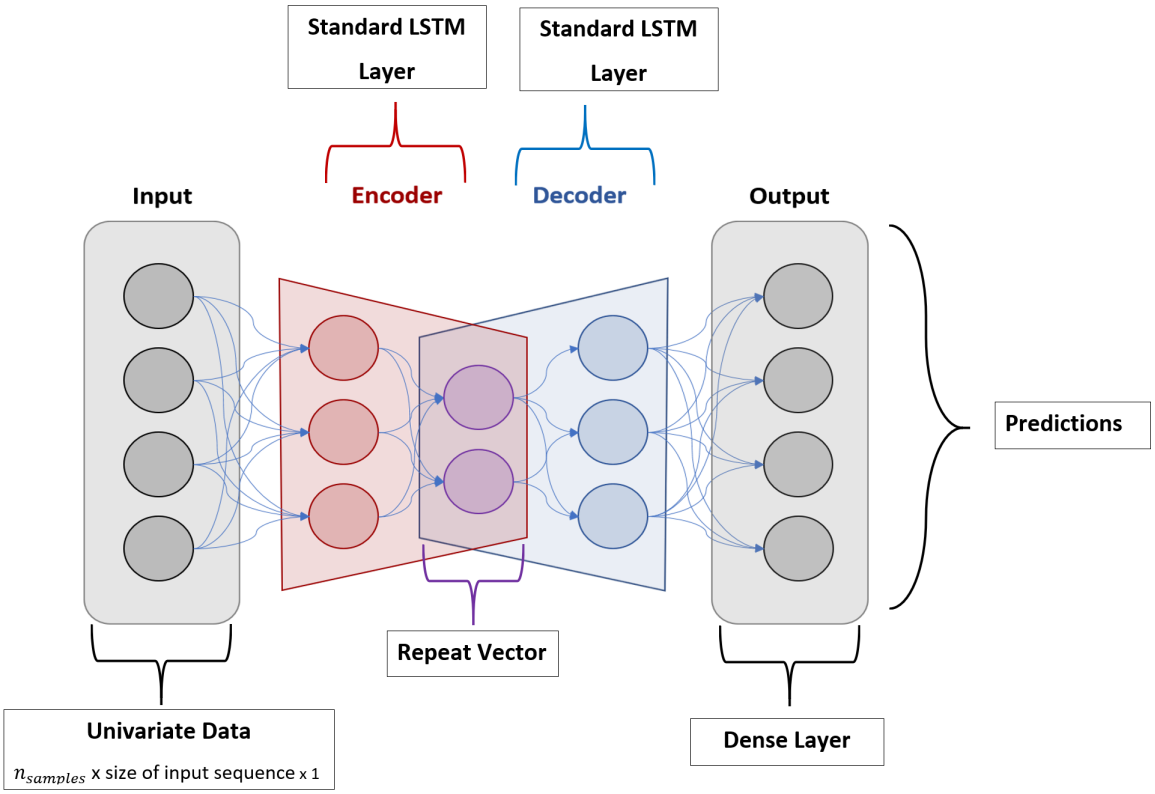


Fig. 2.6. Autoencoder LSTM architecture.

The encoder maps high-dimensional input data into a low-dimensional representation, where the decoder is then responsible for reconstructing the data. This architecture allows the decoder weights to be tuned first, although both the encoder and decoder are trained simultaneously, where ultimately the model minimizes the difference between the reconstructed output and the original input data. The encoded features e_t are given by:

$$e_i = \sigma(W_e x_t + b_e) \tag{2.21}$$

The decoded values given by the original input data x are given by:

$$\hat{x}_i = \sigma(W_d x_t + b_d) \tag{2.22}$$

where σ is the selected activation function, W_e is the encoder weight matrix, b_e is the encoder bias vector, W_d is the decoder weight matrix, and b_d is the decoder bias [33].

According to the architecture of AE models, LSTM units can be included in order to process time series, where multiple sequences are taken as input and later as output. This model presents special characteristics to deal with multi-step ahead forecasting, where the output predictions mapped represent several hours to be forecasted at a time.

2.2. Evaluation Metrics

The performance evaluation of the different LSTM models is performed by comparing the Root Mean Square Error (RMSE) and the Mean Absolute Percentage Error (MAPE), defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2} \tag{2.23}$$

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{x_i - \hat{x}_i}{x_i}\right| \times 100\% \tag{2.24}$$

where N is the number of predicted samples, and x_i and $\hat{x}_i$ are the real and predicted values, respectively. The selection of these metrics is based on their widely application in the literature to assess short-term forecasting models in wind power time series analysis [34].

3. WIND POWER FORECASTING

3.1. Data Set Description

The original dataset for this study covers approximately six years and a half of hourly wind power observations in Spain, collected from [35]. Specifically, the wind power time series data was recorded from January 1, 2014 to May 30, 2020, and reported in megawatts-hour (MWh), for a total of 56,231 observations. Figure 3.1 represents the time series wind power data processed and analyzed in this work.

Firstly, in order to remove innovational outliers, additive outliers or level shifts, the approach described in [36] was used for the automatic detection and removal of outliers in the time series. This strategy uses a spatial outlier detection by the implementation of the variogram method, where outliers are considered as those ones that considerably deviate from the majority of the data. This methodology assumes that the outliers are not correlated in time and space, and can be modeled using an α-stable distribution. The selection of this approach was based on its extensive use in the time series pre-processing [37]. Taking into account this approach, any outlier was found in the analysis of the whole time series. Initial observations on the data suggest that there are no changes in trend over the years. In addition, there are relevant peaks at the beginning of each year marking the annual seasonality of the wind power time series. These ideas are explored in more detail below.

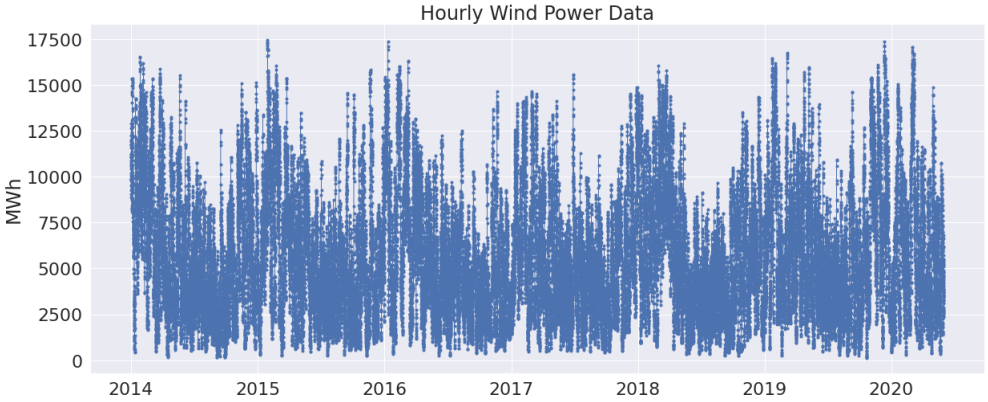
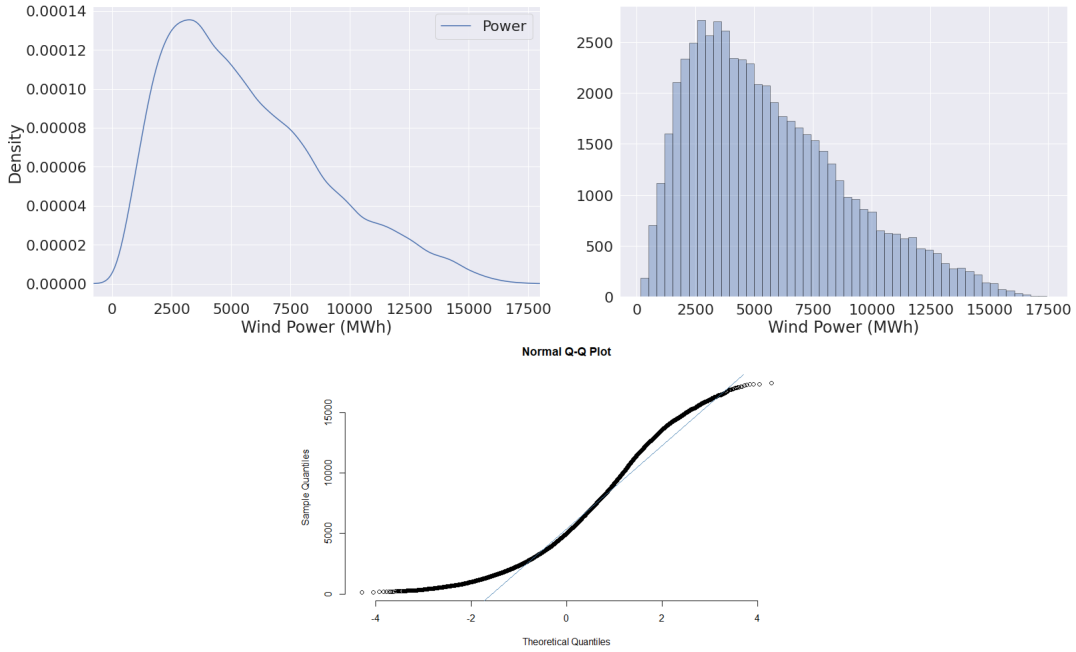


Fig. 3.1. Real wind power time series data from January, 2014 to May, 2020.

In order to provide a description of the wind power time series analyzed in this work, in Figure 3.2, the sample descriptive statistics, its corresponding histogram distribution and the Quantile-Quantile(QQ) plot are summarized. Based on the reported results, it can be seen that the skewness value describes a non-normal wind power time series with a right-skewed distribution. Also, the calculated Kurtosis value, being less than three, indicates the data is light-tailed and flatter than a normal distribution (platykurtic).



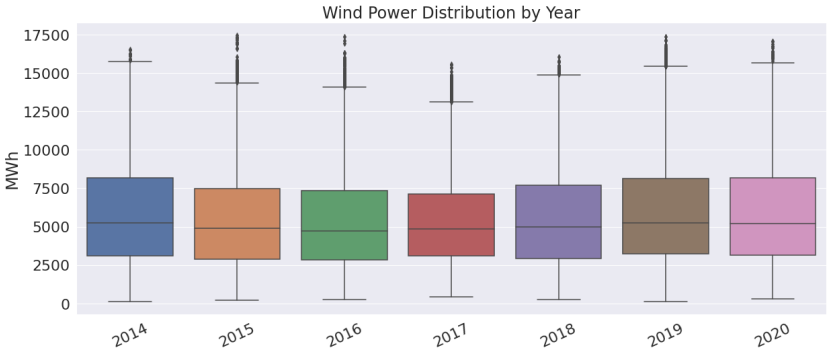
Mean	SD	Min	First Quantile	Median	Third Quantile	Max	Skewness	Kurtosis
5,634.18	3,317.08	143.00	3,023.50	4,985.83	7,680.25	17,435.83	0.7608	-0.0153

Fig. 3.2. Kernel density plot, histogram, QQ plot, and descriptive summary of time series data.

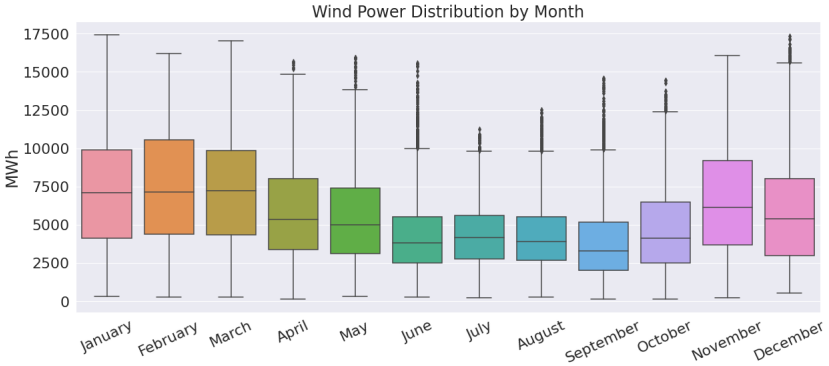
In Figure 3.3 the annual and monthly distributions of the data are represented through box-plots. It can be observed that there are no significant differences among the annual distributions, although 2017 did not reach as high power records compared to other years. Overall, the respective yearly medians oscillate between 4,741 and 5,259 MW. The overall trend of the time series data throughout the years does not show a significant upward movement, which may indicate that wind power penetration has remained practically constant within the Spanish electricity market during the studied period.

This dynamic is consistent with data from the Spanish Wind Energy Association (AEE) [38], which establishes that between 2014-2019, the total newly installed capacity increased by 2,796 MW, of which 2,243 MW were installed in 2019. Additionally, it can be observed that wind power data behaviour clearly varies by month. As such, there is a power drop during the summer months, leading to a mean wind power output as low as 3,872.11 MW, which picks up towards the end of fall and winter with a much higher mean of 7,448.48 MW in February.

A comparative illustration between the average wind power by month according to year is found in Figure 3.4. While the months of June through October experience low energy records and less variation, the opposite takes effect between January - March, November and December. The highest recorded averages are found in March, 2018 and November, 2019, with 10,343.29 and 9,959.37 MW, respectively. In addition, the descriptive Table 3.1 details the monthly wind power outputs. On average, the months with higher wind power generation are January, February and March, while the months with lower power are recorded from June through October.



(a)



(b)

Fig. 3.3. Boxplots of data by (a) year and (b) by month.

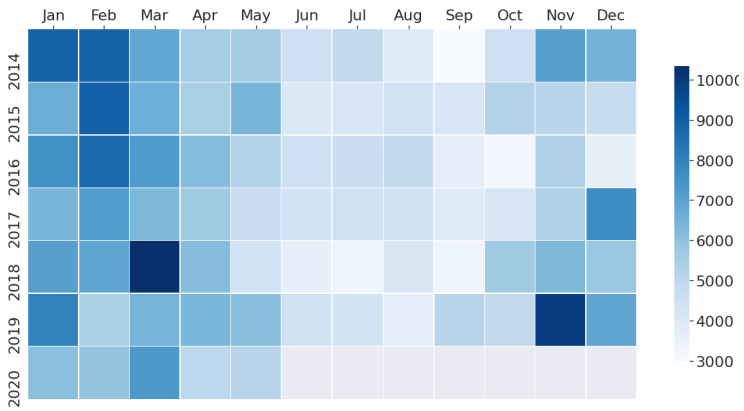


Fig. 3.4. Heat map of average wind power by month given year.

Month	Mean	SD	Min	First Quantile	Median	Third Quantile	Max	Total (MW)
January	7,262.30	3,727.32	300.67	4,131.79	7,086.25	9,898.13	17,435.83	37,822,051.96
February	7,448.48	3,754.99	270.83	4,393.54	7,135.83	10,533.88	16,176.00	35,395,195.16
March	7,328.58	3,678.42	253.67	4,362.88	7,244.58	9,847.54	17,033.83	38,167,240.18
April	5,791.54	3,037.59	147.83	3,370.96	5,365.33	8,006.96	15,686.67	29,189,348.65
May	5,373.89	2,818.41	333.50	3,113.83	4,981.50	7,410.92	15,952.67	27,981,824.96
June	4,246.53	2,340.90	274.17	2,525.50	3,811.67	5,528.87	15,576.67	18,344,994.69
July	4,306.48	1,960.89	250.50	2,767.88	4,158.42	5,591.21	11,268.17	19,224,130.36
August	4,203.30	2,101.57	254.67	2,664.63	3,890.00	5,524.67	12,502.00	18,763,535.64
September	3,872.11	2,485.50	162.83	2,029.92	3,305.50	5,180.46	14,592.83	16,727,524.82
October	4,632.41	2,653.91	143.00	2,520.13	4,130.42	6,482.67	14,448.00	20,679,057.15
November	6,543.35	3,637.15	234.50	3,687.96	6,121.83	9,202.04	16,071.33	28,267,284.17
December	5,881.10	3,456.98	525.00	2,968.21	5,379.67	8,025.08	17,345.50	26,253,221.02

Table 3.1. DESCRIPTIVE SUMMARY OF WIND POWER DATA BY MONTH.

The heat map shown in Figure 3.5 further exemplifies the behavior of the data by month, as it incorporates the time of the day. As it can be observed, the months with higher wind power generation are January, February and March, while lower power is experienced June through October. Also, it can be noted that, on average, the hours with decreased output lie between 06:00 and noon, specially during the summer months.

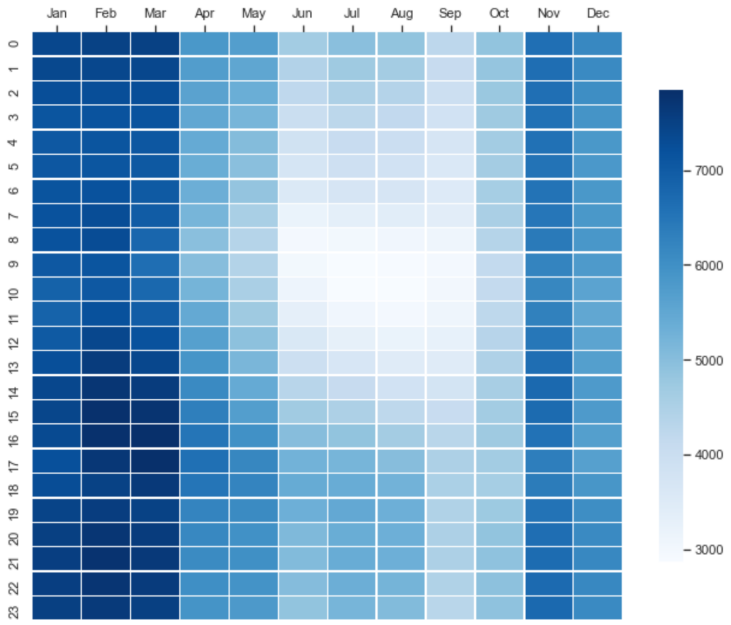


Fig. 3.5. Heat map of average wind power by month given time of the day.

Further exploration of the data is exemplified in Figure 3.6, where the behavior of wind power according to the day of the week and the time of the day is observed. On average, the times with the lowest reported power are found between 06:00 and noon, while there seems to be an increase between 16:00 - 0:00. Saturday and Sunday exhibit the highest power between the hours of 17:00 - 01:00.

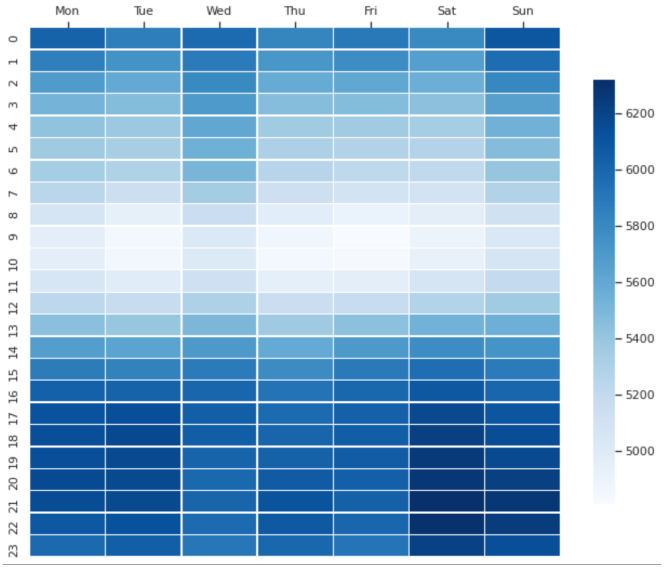


Fig. 3.6. Heat map of average wind power by week day given time of the day.

Given the previous analyses of the data portrayed at different time scales, there seems to be seasonal behaviour. In order to verify the observed seasonality of the data, the autocorrelation coefficients of the time series data (ACF- Autocorrelation Function) are computed to measure the linear relationship between the lagged values of the time series. The correlation of the data is calculated given the values of a series at previous times. The correlation function for a time series x of length T is given by Equation 3.1, where k is the time period being considered (lag):

$$Corr(x_t, x_{t-k}) = \frac{\sum_{t=k+1}^{T}(x_t - \bar{x})(x_{t-k-\bar{x})}}{\sum_{t=1}^{T}(x_t - \bar{x}^2)} \tag{3.1}$$

Figures 3.7 and 3.8 show the autocorrelation function (ACF) plots of the wind power data given different lag values. Figure 3.7 records the autocorrelation coefficients given by the first two years to grasp some of the annual behaviour. In addition, figure 3.8 portrays the coefficients given by the first 7 days where the hourly and daily behaviours are better visualized, which means that the seasonal period covers the entire sample of 168 observations. In concordance with the previous results, it is possible to observe higher correlations each 24 and 8760 hours showing a daily and an annual seasonality, respectively.

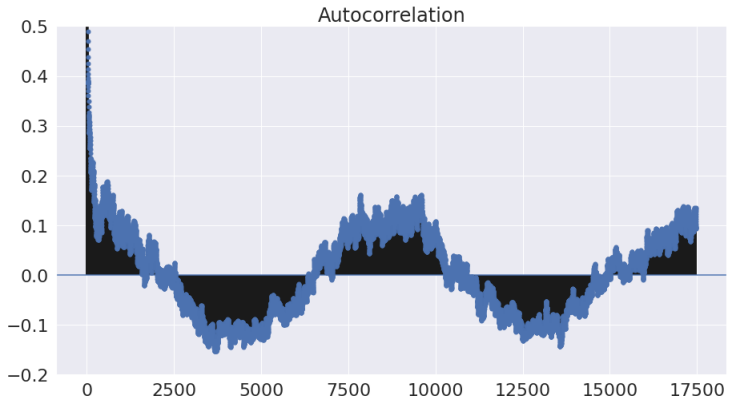


Fig. 3.7. Autocorrelation plot given a lag equivalent to two years (17,472 observations).

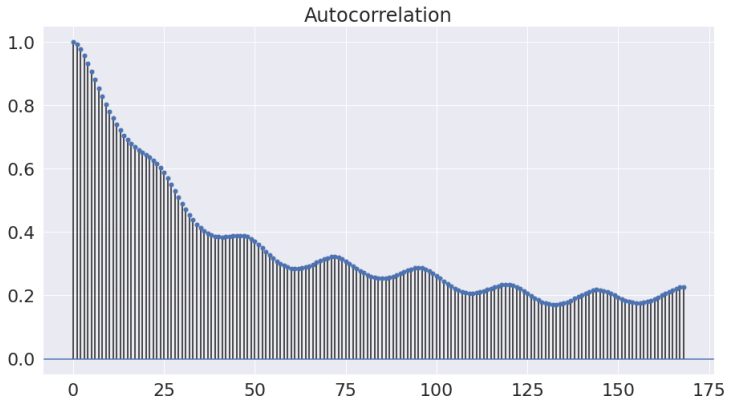


Fig. 3.8. Autocorrelation plot given a lag equivalent to 7 days (168 observations)

The preliminary results given by the ACF are insightful for the structure of the input learning sequences to be fed into the different model architectures, such that the networks are able to learn according to different time scales, such as weekly, monthly, and quarterly (considering several periods of the daily seasonality for the time series).

3.2. Implementation Methodology

Upon completion of the exploratory data analysis, all observations were scaled [0 to 1] for the sake of the networks' speed, learning rate and convergence. A data split of 70:30 for training and testing sets is implemented such that we are able to evaluate the models' predictive performance for up to two years (at least two periods of annual seasonality).

For the LSTM models to learn the mapping function of our sequence of historical wind power observations, the original sequence must be transformed into multiple time series samples from which the LSTM learns. The network maps these sequences as input to an output observation (one-step ahead) or to multiple output observations (multi-steps ahead). Given the previous analysis of the data, which shows different behaviours under weekly, monthly, and quarterly time scales, different input sequence sizes are considered

in this study. First, we consider a transformation where 168 time steps are used as input, such that the network learns by week. Then, 720 time steps are proposed, such that the models' input sequence is equivalent to roughly a month worth of observations. Lastly, we consider a quarterly transformation with 2,160 time steps used as input. It must be noted that a larger input sequence size of 8,760 steps (annual steps) is not considered due to implications the size of the time window has on the availability of data for validation of the models, as we would only very limited observations as part of the testing data set. Moreover, the annual input sequence would also require an increasing number of neurons in the hidden layers, which would positively affect the computational times of the networks, which makes the implementation in a timely manner unfeasible.

The models are first evaluated, given the different input learning sequences for one-step ahead forecasting, where an iterative training process is first implemented to find the DL-based model and configuration with the best performance. Then, the models and their respective configurations, yielding the lowest possible error rate, are selected to evaluate their performance when facing three-steps ahead forecasting. The iterative process is only completed for one-step ahead forecasting due to the exhaustive computational load that the forecasting of three steps entails, considering the different DL architectures. Lastly, we are able to conduct a comparative analysis to evaluate the performance of the three-steps ahead forecasting models against an AE-LSTM model, which is commonly used to tackle multi-step forecasts rather than one-step. The general implementation process is shown in the Figure 3.9.

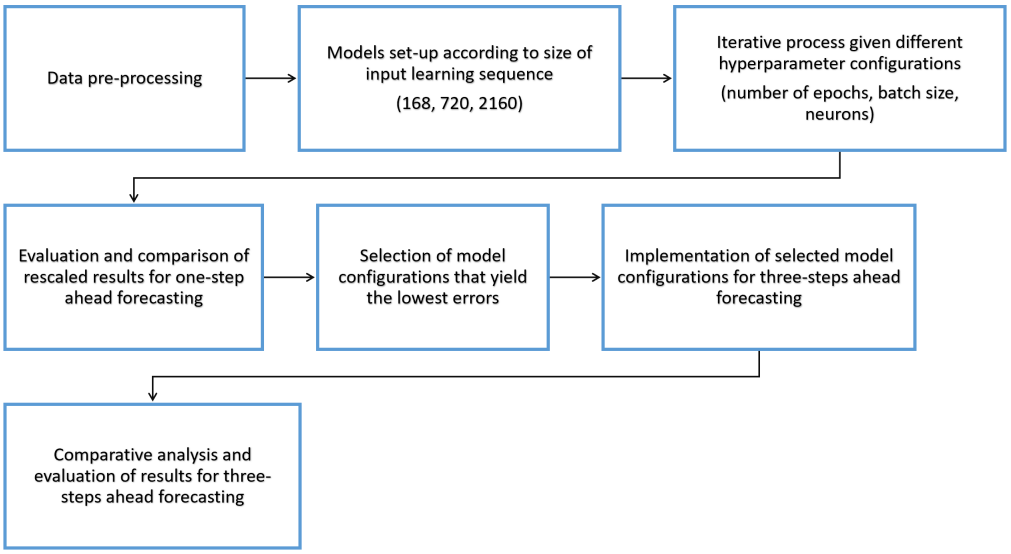


Fig. 3.9. Implementation process of the different DL models, for one-step and three-steps ahead forecasting.

Since batch size relates to the number of training samples to be considered at a time for updating the network, batch size may affect the network speed, learning rate and convergence. Therefore, different batch sizes are considered according to the number of samples obtained through the selected time windows, such that the samples are divisible

by the proposed batch sizes. As such, models set-up according to weekly and monthly learning sequences are implemented with batch sizes 12, 24, and 46, while models set-up through a quarterly learning sequence are implemented with batch sizes 12, 24, and 48. Likewise, we select different number of memory units for the LSTM layers. Since the initial model configurations are based off increasing learning sequence sizes, an increasing number of units is proposed to understand whether more complex models are able to better solve the problem at hand. Lastly, a varying number of training cycles is proposed (100 - 300 epochs). In this way, the iterative process will be performed such that the models' training cycles and batch sizes vary, so the networks are trained with different patterns.

Given all the possible hyperparameter configurations portrayed in Table 3.2, according to the number of steps in the input sequence there are at minimum 27 possible combinations for each, given the number of epochs, size of the batches, and number of neurons.

Input Sequence Size	**Epochs**	**Batch Size**	**Neurons**
Weekly (168 steps)	100	12	32
Monthly (720 steps)	200	24	64
Quarterly (2,160 steps)	300	46, 48	128

Table 3.2. POSSIBLE PARAMETER CONFIGURATIONS FOR ITERATIVE TRAINING OF VANILLA LSTM MODEL.

The activation function employed is ReLu, since its gradient does not tend to suffer from gradient vanishing or diffusion [33]. In addition, models are trained using the Adam optimization algorithm [39], where the default recommended parameters are used (learning rate=0.001, β_1=0.9, β_2=0.999, ϵ=1e-08), unless otherwise stated and required by the model. The selection of this model is based on the reported advantages of its implementation to support DL-based models [40]: straightforward implementation, few memory consumption, invariance to the diagonal re-scalation of gradients, suitable for models that involves large sizes of data and/or parameters, hyper-parameters involve an intuitive interpretation and generally default values have an adequate performance.

Finally, the Mean Square Error (MSE) is the selected loss function to be minimized, based on the successful results reported of this loss function in conjunction with the Adam optimizer [41]. This metric is defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2 \tag{3.2}$$

where N is the number of predicted samples, and x_i and $\hat{x}_i$ are the real and predicted samples, respectively. All of the code related to the implementation described in this study are publicly available at https://github.com/emora034/Wind_Power_DeepLearning .

4. EXPERIMENTAL RESULTS

Based on the methodology described in the previous chapter, the experiments for each model were carried out. It is important to note that every model was compiled in Python 3.8 with Tensor Flow 2.4.0 backend, and Intel(R) Core(TM) i7-8565U.

4.1. One-Step Forecasting

First, single-step ahead forecasting strategies were assessed, where single, bi-directional, stacked and convolutional LSTM RNNs were implemented according to the previously defined implementation process, varying the number of input neurons with only one output defined by the value to be predicted.

4.1.1. Vanilla LSTM

Given the hyperparameter configurations portrayed in Table 3.2, according to the number of steps in the input sequence, there are 27 possible combinations for each, given the number of epochs, size of the batches, and number of neurons in the hidden layer. The performance of each model is illustrated in Figures 4.1-4.3. Figure 4.1 portrays the reported MAPE for each of the 27 possible configurations implemented given 168 time steps, while figure 4.2shows the MAPE for each of the 27 configurations executed with an input sequence of 720 steps. Lastly, figure 4.3 illustrates the MAPE for each configuration given 2,160 time steps.

The smallest reported error of 4.30% was recorded under a configuration of 168 time steps corresponding to a weekly input learning, 200 iterations, batch size of 46, and 32 neurons in the hidden layer. The models implemented using monthly input sequences, produced errors between 4.61-8.49%, while the quarterly input sequence models reported errors between 4.84-12.21%. It seems that as the size of the input learning sequences increase, so do their corresponding error metrics.

To better grasp the performance of the Vanilla LSTM, given the hyperparameter configuration of the model that produced the lowest error, its performance is illustrated below. Figure 4.4 shows the full time series data (a), followed by 20,000 samples from the training set (b), and the entire testing set (c). As it can be observed the model successfully captures the behaviour of the data throughout the years. It can be noted that the overall model predictions closely follow the real wind power values, although it does not perfectly capture some maxima and minima values.

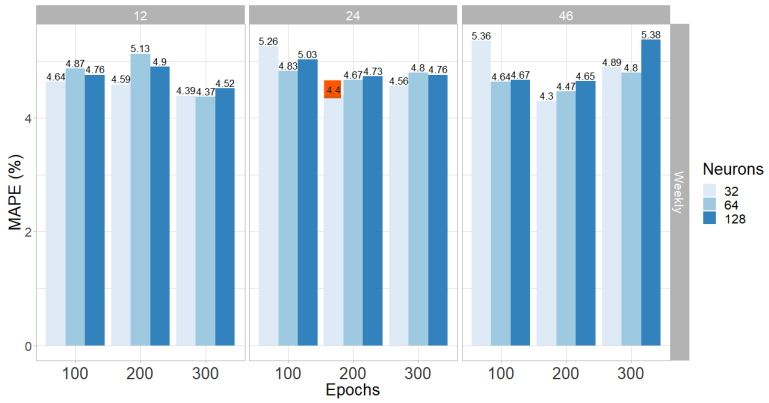


Fig. 4.1. MAPE for Vanilla LSTM given weekly (168) time steps, batch size (12, 24, 46), epochs (100, 200, 300), and number of neurons (32, 64, 128).

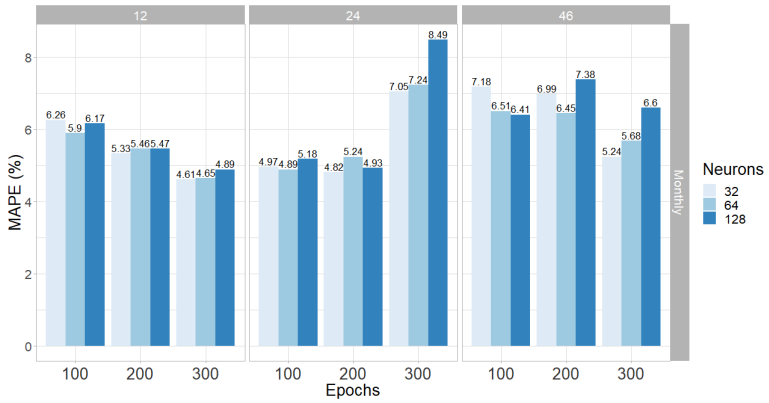


Fig. 4.2. MAPE for Vanilla LSTM given monthly (720) time steps, batch size (12, 24, 46), epochs (100, 200, 300), and number of neurons (32, 64, 128).

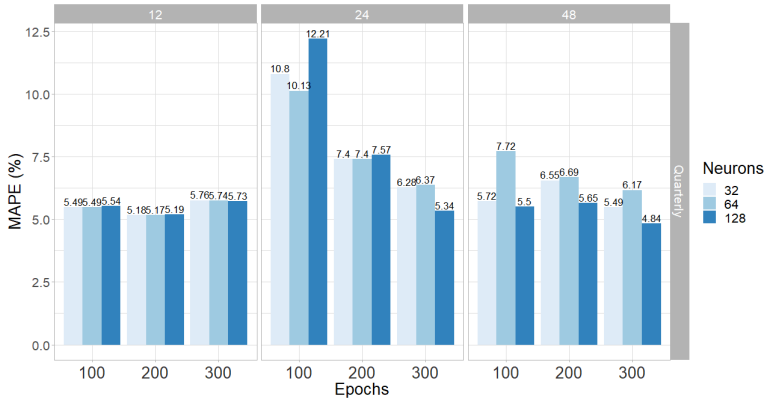


Fig. 4.3. MAPE for Vanilla LSTM given quarterly (2,160) time steps, batch size (12, 24, 48), epochs (100, 200, 300), and number of neurons (32, 64, 128).

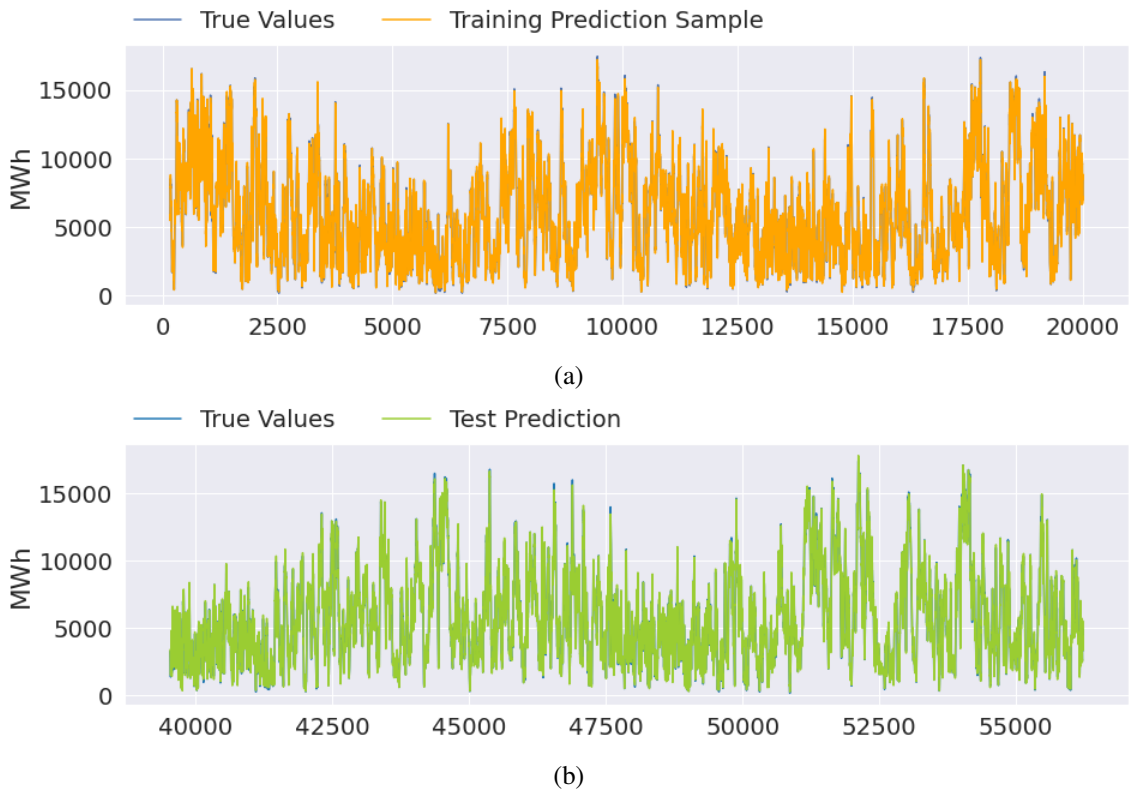


Fig. 4.4. (a)Plot of the first 20,000 observations of time series data against corresponding predicted values for the training, given 168 time steps, and hyperparameter configuration of 200 iterations, batch size of 46, and 32 neurons in the hidden layer. (b)Test set and corresponding model predictions.

4.1.2. Bidirectional LSTM

The Bidirectional LSTM models were implemented using all possible configuration parameters referenced in Table 3.2. Once again, the best performing model configuration, with a reported MAPE of 4.44%, is given by the model that learned by weekly time steps (168 steps) implemented with 200 epochs, batch size of 46, and 32 neurons. However, the highest reported error rates were found within the implementation with monthly time steps (720 steps), where the errors ranged between 4.60-8.91%, while the models implemented with quarterly time steps (2,160 steps) produced a 4.79-7.04% range. Overall, the models initiated with weekly time steps maintained values between 4.44-5.53%, resulting in lower variability among the reported metrics. This architecture maintained similar metrics for the weekly and monthly input sequence implementations, and it improved the resulting error range under the quarterly input steps; while the range for the vanilla for the quarterly input sequence (2,160 steps) was 4.84-12.21%, the Bidirectional reported a range of 4.79-7.04% resulting in lower variability.

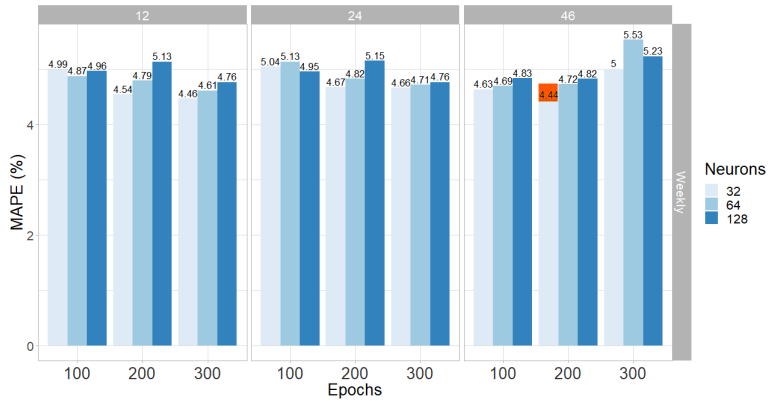


Fig. 4.5. MAPE for Bidirectional LSTM given weekly (168) time steps, batch size (12, 24, 46), epochs (100, 200, 300), and number of neurons (32, 64, 128).

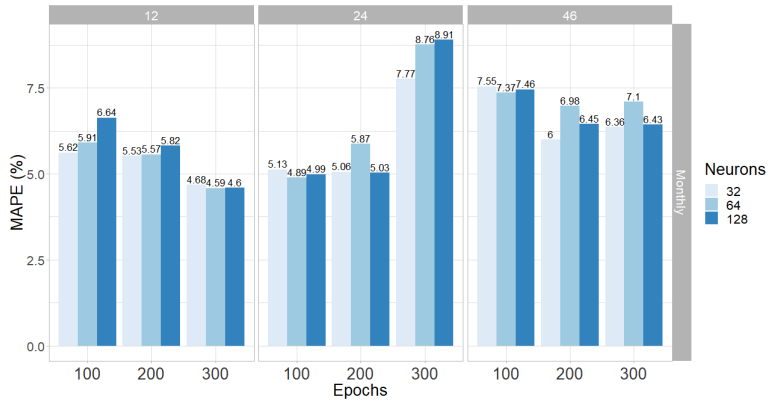


Fig. 4.6. MAPE for Bidirectional LSTM given monthly (720) time steps, batch size (12, 24, 46), epochs (100, 200, 300), and number of neurons (32, 64, 128).

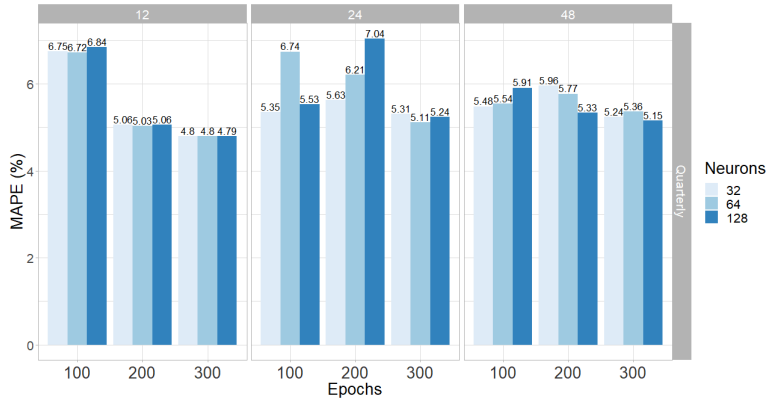


Fig. 4.7. MAPE for Bidirectional LSTM given quarterly (2,160) time steps, batch size (12, 24, 48), epochs (100, 200, 300), and number of neurons (32, 64, 128).

As it can be observed in the figure below, the model captures the overall dynamics of the wind power time series well. Although, it does not perfectly capture some minima and maxima values.

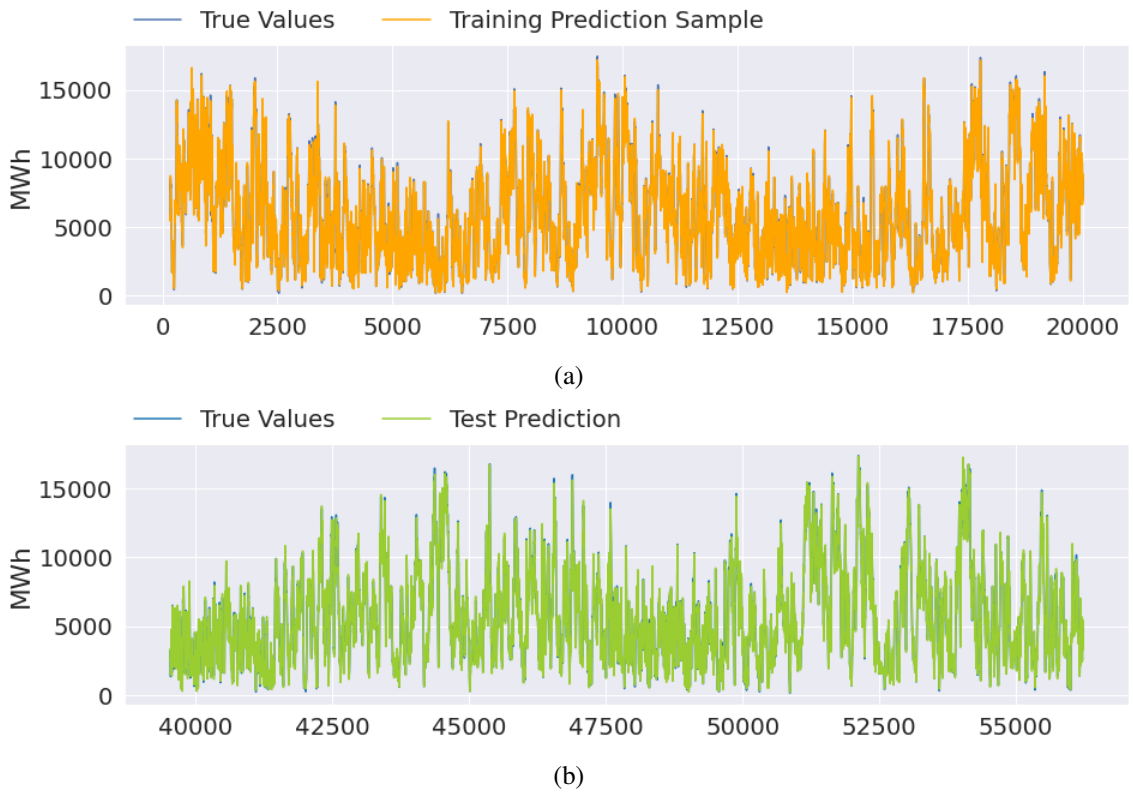


Fig. 4.8. (a) Plot of the first 20,000 observations of time series data against predicted values for the training set, given 168 time steps, and hyperparameter configuration of 200 iterations, batch size of 46, and 32 neurons in the hidden layer. (b) Test set and corresponding model predictions.

4.1.3. Stacked LSTM

A stacking architecture with two LSTM layers is implemented, where each layer is independent and the number of neurons in each layer may vary between 32 or 64 neurons, being these two values the most commonly employed under stacked architectures [42], [43] . Under this configuration, we exclude the implementation of 128 neurons for the sake of implementation times since higher number of neurons may prevent the models from finding a solution within an adequate time frame; moreover, we exclude this higher number of neurons given the additional combinations that would otherwise be available for implementation. The layers are then connected by a dense layer with one neuron, for a single time step output. Given all the possible parameter configurations portrayed in table 4.1, there are a total of 108 possible combinations given the size of the input sequence, iterations, size of the batches, and number of neurons in the first and second layers. The results are presented according to the number of neurons in the first layer, which are held constant for the sake of the analysis portrayed below. The results of the networks implemented with 32 neurons in the first layer are first presented, followed by the metrics produced given 64 neurons in the first layer.

Input Sequence Size	Epochs	Batch Size	Neurons (Layer 1)	Neurons (Layer 2)
Weekly (168 steps)	100	12	32	32
Monthly (720 steps)	200	24	64	64
Quarterly (2,160 steps)	300	46, 48		

Table 4.1. POSSIBLE PARAMETER CONFIGURATIONS FOR ITERATIVE TRAINING OF STACKED LSTM MODEL.

Figures 4.9-4.11 show the MAPE reported for stacked LSTM models with 32 neurons on the first layer and varying number of neurons on the second layer. The lowest recorded error of 4.37% was produced by the model implemented with 32 memory units in the first and second layers, 300 epochs, a batch size of 12, and a weekly input sequence (168 steps). The models implemented under the monthly input sequence yielded errors ranging from 4.68-8.82%, while the quarterly sequence approach produced errors ranging between 4.75-9.37%.

The results for the stacked model with 64 neurons on the first layer are portrayed below. Once again, the lowest recorded MAPE of 4.50% was produced by the model configured with an input sequence of size 168, 32 neurons in the second layer, 300 epochs, and batch size of 12. The models implemented under the monthly input sequence yielded errors ranging from 4.70-8.99%, while the quarterly sequence approach yielded errors ranging between 4.73-14.13%.

Overall, the models implemented with a weekly input sequence (168 steps) produced the lowest MAPEs ranging between 4.37-5.13%. The lowest reported error (4.37%) was produced by the model configured with 32 neurons on the first and second layers, 300 epochs, and a batch size of 12. This architecture achieves a slightly lower variability, in relation to weekly input sequence (168 steps) implementations, when compared to the Vanilla (4.30-5.38%) and Bidirectional (4.44-5.53%) models.

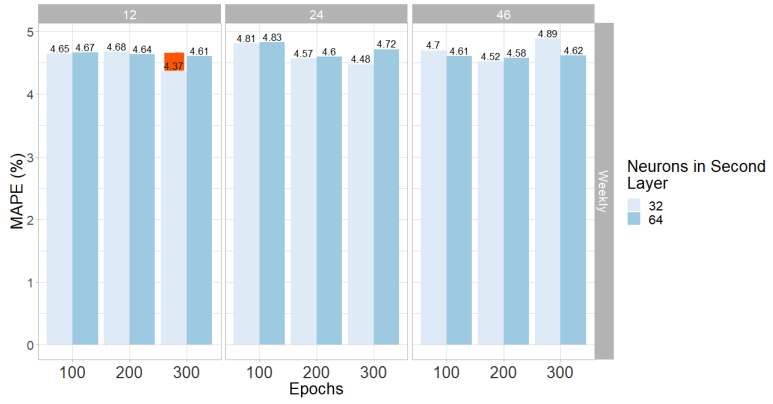


Fig. 4.9. MAPE for Stacked LSTM with 32 neurons in the first hidden layer, given weekly (168) time steps, batch size (12, 24, 46), epochs (100, 200, 300), and number of neurons in the second hidden layer (32, 64).

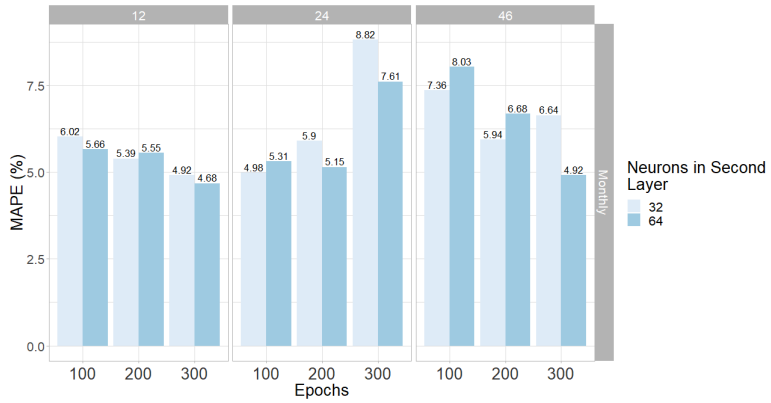


Fig. 4.10. MAPE for Stacked LSTM with 32 neurons in the first hidden layer, given monthly (720) time steps, batch size (12, 24, 46), epochs (100, 200, 300), and number of neurons in the second hidden layer (32, 64).

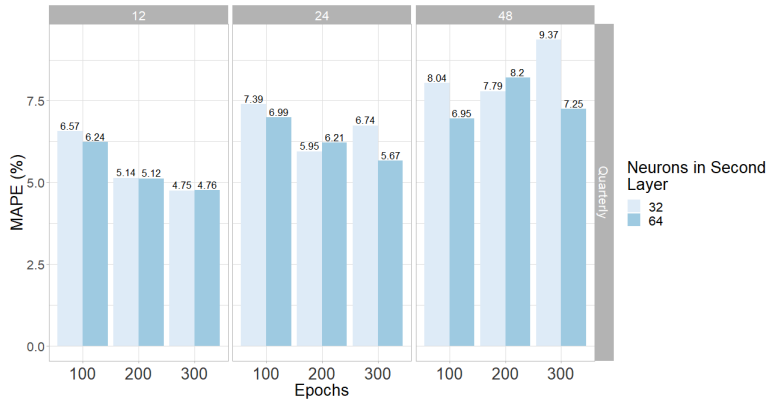


Fig. 4.11. MAPE for Stacked LSTM with 32 neurons in the first hidden layer, given quarterly (2,160) time steps, batch size (12, 24, 48), epochs (100, 200, 300), and number of neurons in the second hidden layer (32, 64).

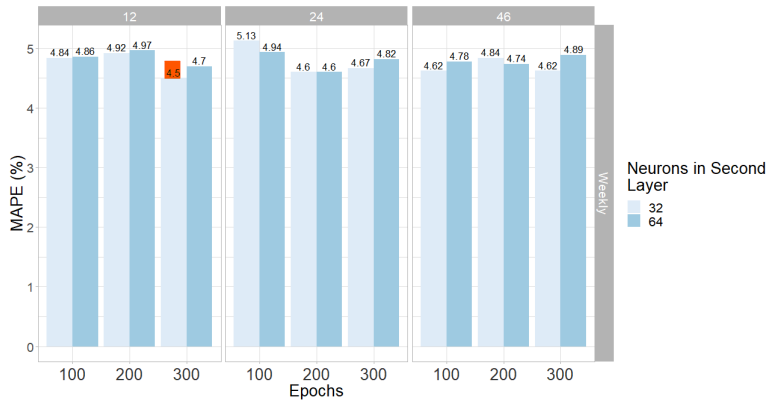


Fig. 4.12. MAPE for Stacked LSTM with 64 neurons in the first hidden layer, given weekly (168) time steps, batch size (12, 24, 46), epochs (100, 200, 300), and number of neurons in the second hidden layer (32, 64).

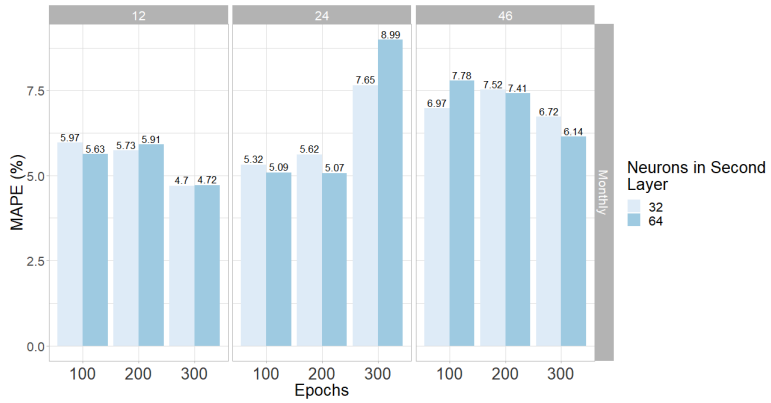


Fig. 4.13. MAPE for Stacked LSTM with 64 neurons in the first hidden layer, given monthly (720) time steps, batch size (12, 24, 46), epochs (100, 200, 300), and number of neurons in the second hidden layer (32, 64).

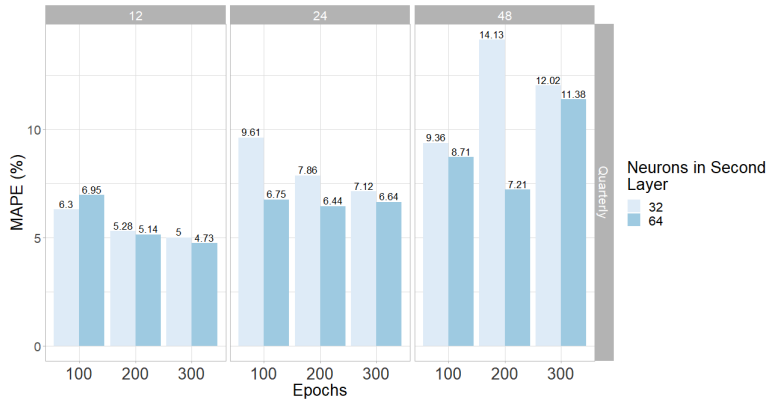


Fig. 4.14. MAPE for Stacked LSTM with 64 neurons in the first hidden layer, given quarterly (2,160) time steps, batch size (12, 24, 48), epochs (100, 200, 300), and number of neurons in the second hidden layer (32, 64).

Given the hyperparameter configuration of the model that produced the lowest error, its performance is better illustrated in figure 4.15, inclusive of real and predicted values, given by the training and testing sets. As it can be observed the model successfully captures the seasonality of the data throughout the years, although it does not capture maxima and minima, as is the case with the previously executed models.

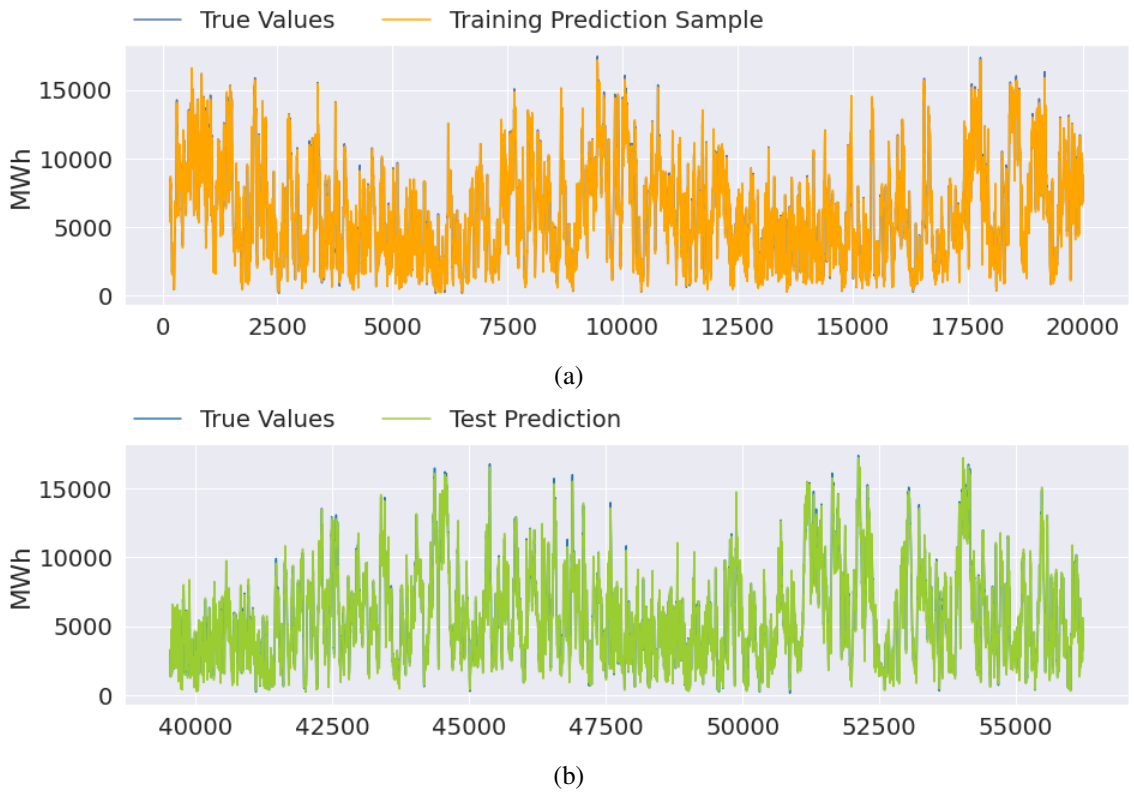


Fig. 4.15. (a) Plot of first 20,000 real time series data points against predicted values for the training set, given 168 time steps, and hyperparameter configuration of 300 iterations, batch size of 12, and 32 neurons in both hidden layers. (b) Test set and corresponding model predictions.

4.1.4. Convolutional LSTM

The Conv-LSTM implementation considers 64 filters in the convolutional layer, a kernel input size of 1, and a max-pooling kernel size of 2, as this parameter configuration has been previously employed in studies of short-term wind power forecasting [44]. This model required a further split of the input sequences into subsequences to take advantage of the feature extraction capability. Given the already established data splits (168, 720, 2,160 input sequence), each sample is then split into smaller samples. Our selection splits every sample by 24 hours for the model to read subsequences equivalent to daily data. In this way, the number of subsequences per sample, according to input sequences of sizes 168 (weekly), 720 (monthly) and 2,160 (quarterly), is 7, 30, and 90, respectively.

Only the models initialized with weekly input time steps (168 steps) were processed given all other possible configurations of the hyperparameters referenced in Table 3.2. This was due to the significant increase in computational times of this architecture, associated with the size of the monthly (720 steps) and quarterly (2,160 steps) input sequences. The results portrayed in figure 4.16 were produced in roughly 22 hours, using the full suite of possible configurations given a weekly input learning sequence (168 steps). However, the results shown in figures 4.17 and 4.18, given monthly and quarterly input time steps, implemented with 100 epochs only, took 27 and 57 hours, respectively. The complexity

of the Conv-LSTM model, along with the size of the input sequences, difficult the implementation of all parameter configurations, as it would approximately take 10 days to fully implement all the models, at minimum.

Under these Conv-LSTM configurations, the results range from 5.45-10.15%, 5.31-7.31%, and 5.87-6.81%, given weekly (168 steps), monthly (720 steps), and quarterly (2,160 steps) input learning sequences, respectively. While the previously analyzed architectures produced the lowest error variability under weekly input sequences, the Conv-LSTM produced the lowest error variability according to the quarterly input sequence implementations (5.87-6.81%). The lowest reported MAPE of 5.31% was found under monthly input time steps (720 steps), 100 epochs, batch size of size 12, and 128 neurons.

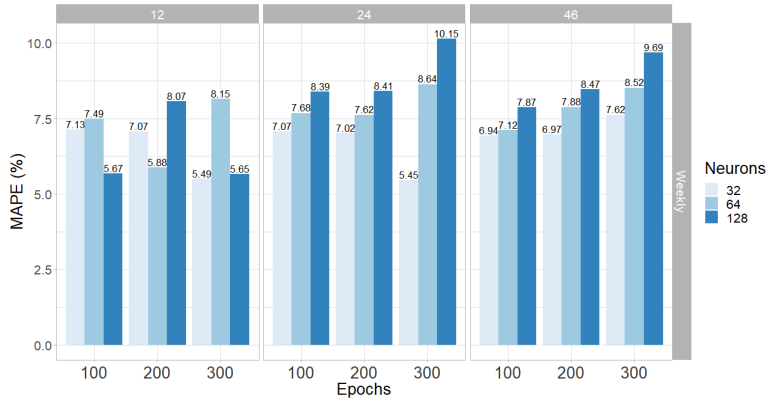


Fig. 4.16. MAPE for Convolutional LSTM given weekly (168) time steps, batch size (12, 24, 46), epochs (100, 200, 300), and number of neurons (32, 64, 128)

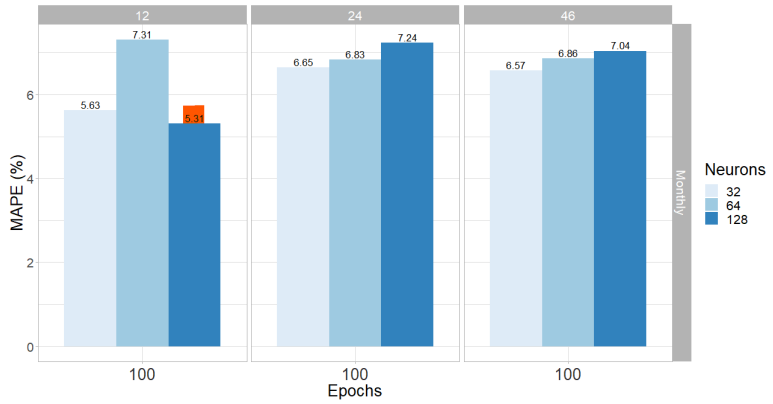


Fig. 4.17. MAPE for Convolutional LSTM given monthly (720) time steps, batch size (12, 24, 46), epochs (100), and number of neurons (32, 64, 128).

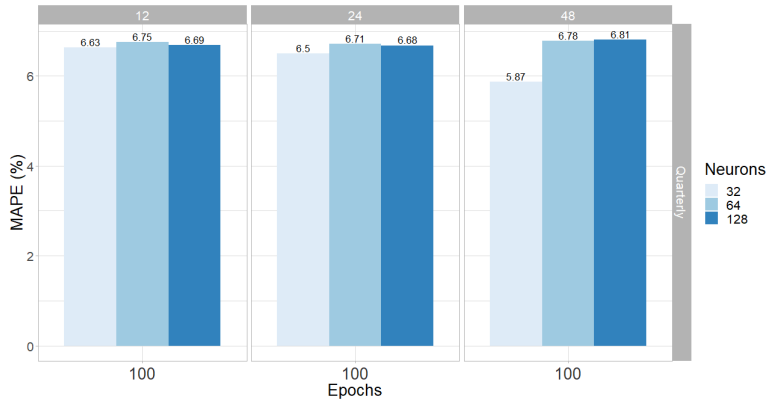


Fig. 4.18. MAPE for Convolutional LSTM given quarterly (2,160) time steps, batch size (12, 24, 48), epochs (100), and number of neurons (32, 64).

The performance of this model is assessed in figure 4.19 below, where it can be appreciated that the model mostly struggles with capturing maxima values and tends to over estimate some minima values around the annual peaks; although, it otherwise captures the overall behaviour of the data. However, the previous models have produced better results. In order to improve the overall performance of the model,and with respect to the reported metrics, the implementation of different activation functions on the output layer should be studied, to assess the potential improvements of the forecast.

Fig. 4.19. (a) Plot of first 20,000 real time series data points against predicted values for the training set, given 720 time steps, and hyperparameter configuration of 100 iterations, batch size of 12, and 128 neurons in the hidden layer. (b) Test set and corresponding model predictions.

4.1.5. Performance Summary

The Table 4.2 presents a summary of the performance of each DL model given the input sequence size, along with the configuration that produced the lowest MAPE. It must be noted that while the Vanilla and Bidirectional models were implemented using the full suite of hyperparameters presented in table 3.2, the Stacked and Convolutional models were not, mainly due to the additional number of configurations to be trained and the associated implementation times, respectively. As the complexity of the DL models increased, the implementation times increased substantially, making it difficult to implement the iterative process inclusive of all hyperparameter configurations. Therefore, some hyperparameters were fixed, according to the reported results of models previously studied for short-term wind power forecasting.

Table 4.2 shows that the models implemented given weekly time steps (168 steps) produced the lowest recorded error (4.30%), the lowest error variability on average (4.64-6.55%) and were implemented in the shortest amount of time (5-21 hours) when compared to the other models which took between 11-57 hours to be completed. As the size of the input time steps increased, so did the associated error variability and the implementation times. However, the increase in execution times is also positively related to the number of neurons, and the complexity of the models.

Model	Input Time Steps	Min MAPE (%)	Max MAPE (%)	Mean MAPE (%)	Implementation Time hh:mm:ss	Optimum Configuration by Lowest MAPE (%)
Vanilla LSTM	Weekly (168)	4.30	5.38	4.75	05:50:03	Time steps: 168 Neurons: 32 Epochs: 200 Batch Size: 46
	Monthly (720)	4.61	8.49	5.93	06:43:03	
	Quarterly (2,160)	4.84	12.21	6.56	11:09:53	
Bidirectional LSTM	Weekly (168)	4.44	5.53	4.85	08:04:21	Time steps: 168 Neurons: 32 Epochs: 200 Batch Size: 46
	Monthly (720)	4.59	8.91	6.19	13:18:53	
	Quarterly (2,160)	4.79	7.04	5.62	18:50:08	
Stacked LSTM	Weekly (168)	4.37	5.13	4.71	06:25:27	Time steps: 168 Neurons per layer: 32, 32 Epochs: 300 Batch Size: 12
	Monthly (720)	4.68	8.99	6.18	09:07:01	
	Quarterly (2,160)	4.73	14.13	7.22	16:23:17	
Convolutional LSTM	Weekly (168)	5.45	10.15	7.49	21:47:49	Time steps: 720 Neurons: 128 Epochs: 100 Batch Size: 12
	Monthly (720)	5.31	7.31	6.60	27:19:05	
	Quarterly (2,160)	5.87	6.81	6.60	57:39:11	

Table 4.2. PERFORMANCE SUMMARY FOR ALL MODELS EXECUTED ACCORDING TO TIME STEPS.

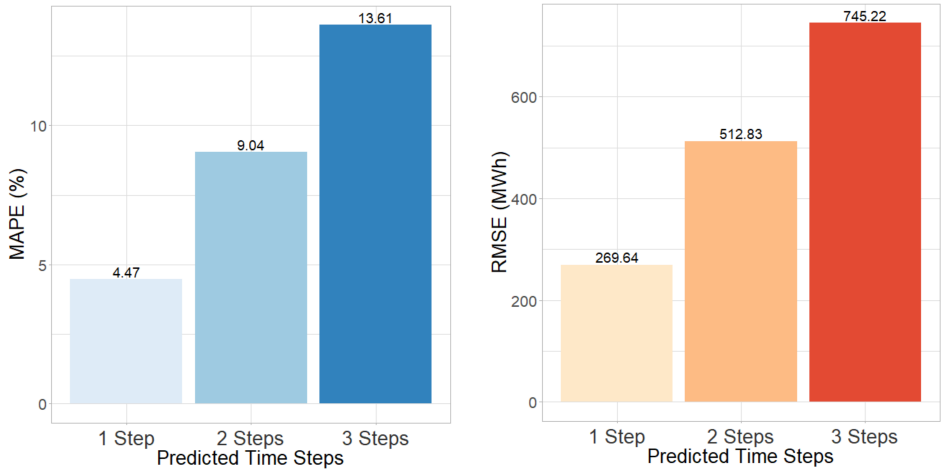
4.2. Three-Steps Forecasting

Given the results of the one-step forecasting models implemented in the previous section, we now select the model configurations that produced the lowest errors to carry a forecasting three-steps ahead. Lastly, and Autoencoder LSTM architecture is introduced for

this task in order to compare its performance against the previously studied models. This DL architecture is only presented for the three-steps ahead forecasting task due to its inner configurations, where multiple neurons are found at the input and output layers.

4.2.1. Vanilla LSTM

The one-step LSTM model that yielded the best predictive performance had a configuration of weekly input time steps (168 steps), 200 iterations, batch size of 46, and 32 neurons in the hidden layer, with an error rate of 4.30%. Based on this configuration, the model was updated such that there are three neurons in the output layer, instead of 1, to produce the three-steps ahead predictions. As it can be observed in the figure below, the reported MAPE and RMSE increase with each further time step predicted, as has been reported in the literature, ranging from 4.47-13.61% and 269.64-745.22 MW, respectively.

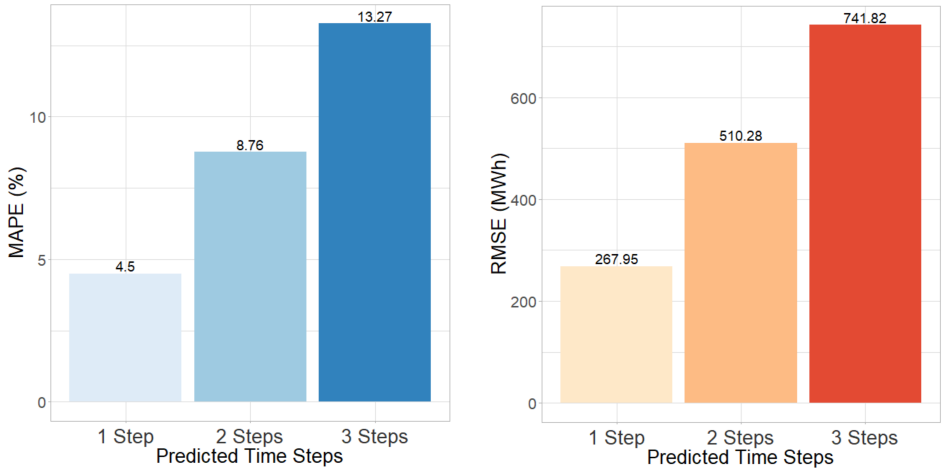


(a) (b)

Fig. 4.20. (a) Recorded MAPE (%) and RMSE (MWh) for each step-ahead predicted, given a configuration of 168 input time steps, 200 iterations, batch size of 46, and 32 neurons in the hidden layer.

4.2.2. Bidirectional LSTM

The best performing one-step BiLSTM model configuration, with a reported MAPE of 4.44%, was given by a configuration of 168 input time steps implemented with 200 epochs, batch size of 46, and 32 neurons. Based on this configuration, the model was updated such that there are three neurons in the output layer, instead of 1, to produce the three-steps ahead predictions. The three-steps ahead forecast implemented with this configuration produced almost identical results for the first hour predicted, yet it also proved that the errors increase with each additional hour forecasted, ranging from 4.50-13.27% which translates into 267.95-741.82 MWh.

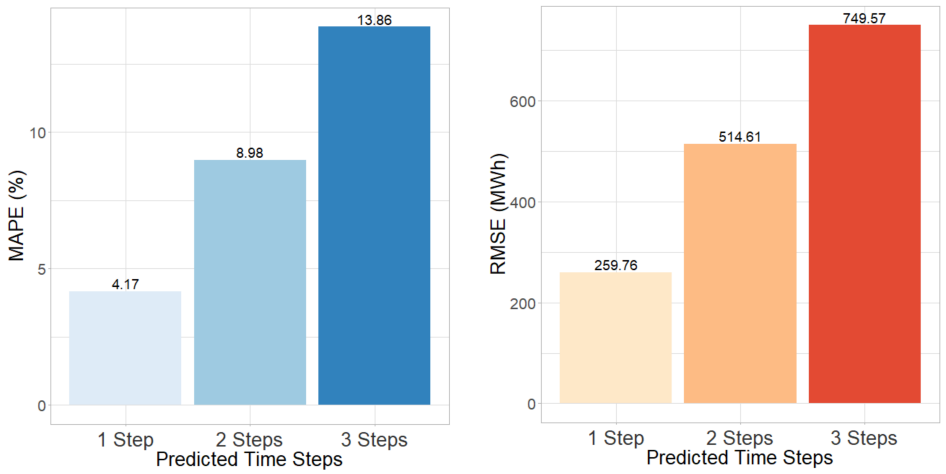


(a) (b)

Fig. 4.21. (a) Recorded MAPE (%) and RMSE (MWh) for each step-ahead predicted, given a configuration of 168 input time steps implemented with 200 epochs, batch size of 46, and 32 neurons.

4.2.3. Stacked LSTM

Under the lowest error yielding configuration referenced in table 4.2, the model was updated such that there are three neurons in the output layer, instead of 1, to produce the three-steps ahead predictions. The stacked architecture has produced a lower MAPE and RMSE for the first step predicted in comparison to the Vanilla and Bidirectional three-steps ahead models; however, these previous implementations showcase a better performance for the second and third hours predicted.

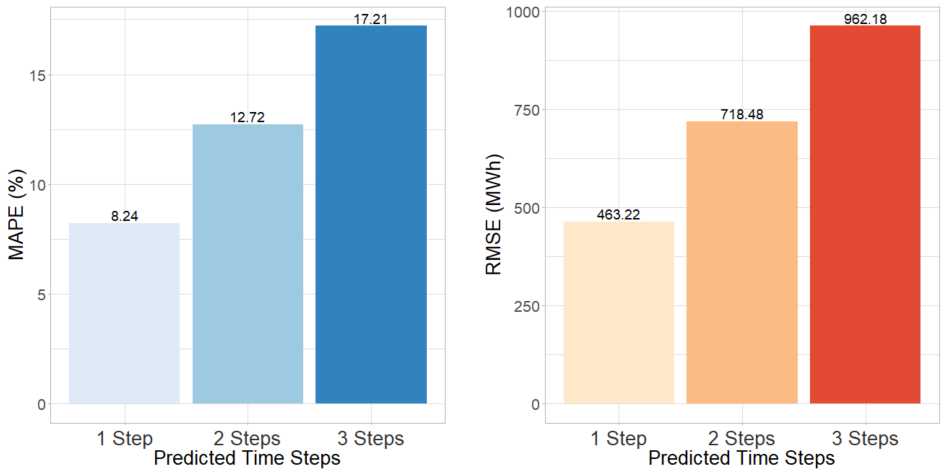


(a) (b)

Fig. 4.22. (a) Recorded MAPE (%) and RMSE (MWh) for each step-ahead predicted, given a hyperparameter configuration of 168 input steps, 32 neurons in both layers, 300 epochs, and batch size 12.

4.2.4. Convolutional LSTM

According to the lowest error yielding configuration referenced in table 4.2 produced by the Conv-LSTM, the model was updated such that there are three neurons in the output layer, instead of 1, to produce the three-steps ahead predictions. The model's performance for this task proves to be highest error yielding network, when compared to the previous architectures, as all error metrics for each step have virtually doubled under this configuration, ranging between 8.24-17.21% and 463.22-962.18 MWh.



(a) (b)

Fig. 4.23. (a) Recorded MAPE (%) and RMSE (MWh) for each step-ahead predicted, given a hyperparameter configuration of 720 input time steps, neurons, 100 epochs, and batch size 12.

4.2.5. Autoencoder LSTM

We introduce AE-LSTM networks given their encoder and decoder architecture, which is especially helpful in multi-step forecasting given their generative capabilities. Due to the extensive implementation times of this model, the implementations were only executed for 100 epochs.

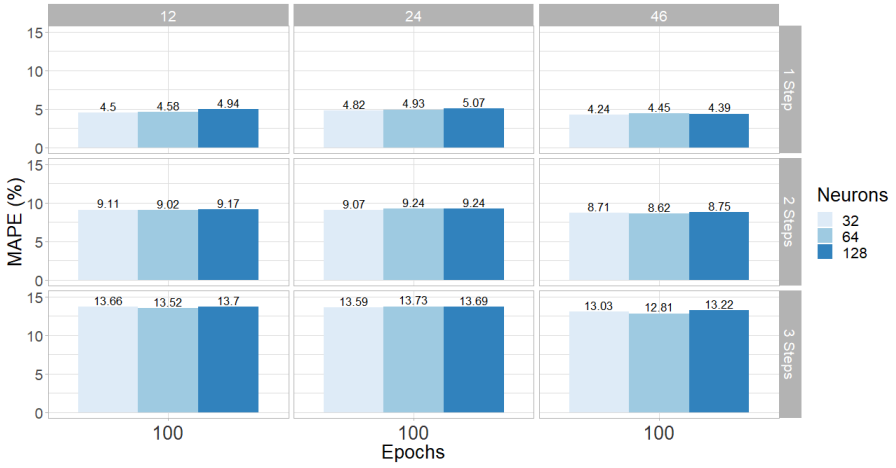
First, we consider the weekly learning input sequence (168 steps) where results are illustrated in figure 4.24. It must be noted that, after several attempts the Adam optimizer's default parameters had to be scaled for it to be able to tackle the prediction problem, since the network was not learning after the 85^{th} epoch. The specific parameter that was changed was ϵ, and it was increased until found a convergence in the algorithm. This change was based on the recommendation of Tensorflow documentation where it is stated that the default might not be a good default for every configuration [45]. For example, when training multidimensional more complex Deep Neural Networks, ϵ is increased.

The recorded metrics for this first implementation show that there is little variation among the models implemented with different number of neurons in the hidden layer. Overall, the hyperparameter configuration yielding the lowest MAPE is given by the

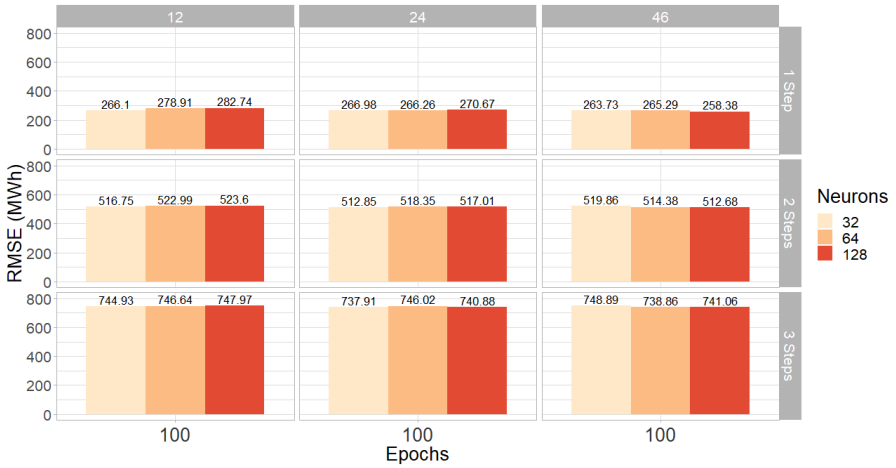
implementations set up with a batch of size 46 and 32 neurons, ranging between 4.24-13.03%, while the reported RMSE for this model produced negligible differences in terms of MWh.

For the execution of the models according to a monthly input sequence (720 steps), the exhaustive implementation times lead us to only run the models given 32 and 64 neurons. Once again, the model required an increase in ϵ, as with the previous implementation given weekly input steps. The performance metrics are illustrated in figure 4.25, where the lowest error-yielding configuration is given by batch size 24 and 32 neurons (4.24-12.87%) with negligible differences among the RMSE reported in MWh.

Lastly, the models with quarterly-sized input sequences (2,160 steps) also had to be trimmed down due to the extreme implementation time, such that only 32 neurons in the hidden LSTM layer were considered. Under this configuration, ϵ was increased to 1e-3 for batch sizes 12 and 24; however, a larger value (1e-2) was needed for the implementation of the configuration with a batch size of 48. Results are portrayed in figure 4.26. The lowest reported MAPE ranges betweent 4.19-12.60% and is related to the model with 32 neurons and a batch size of 12.

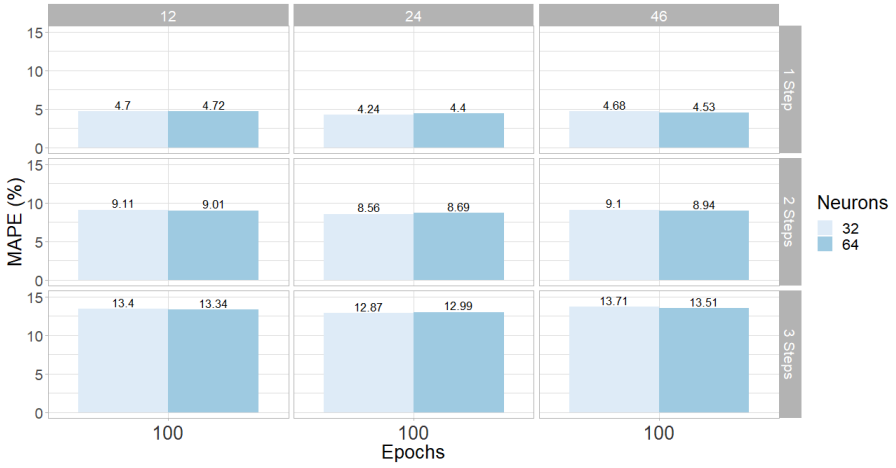


(a)

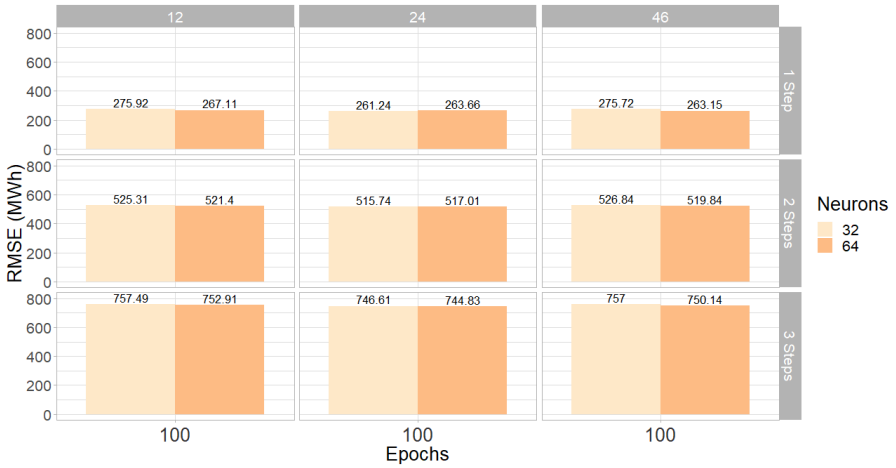


(b)

Fig. 4.24. (a) Recorded MAPE (%) given by 168 input time steps and 100 epochs, classified by batch size (12, 24, 46), neurons (32, 64, 128) and output time step (1, 2, 3). (b) RMSE (MWh) for each step-ahead predicted.

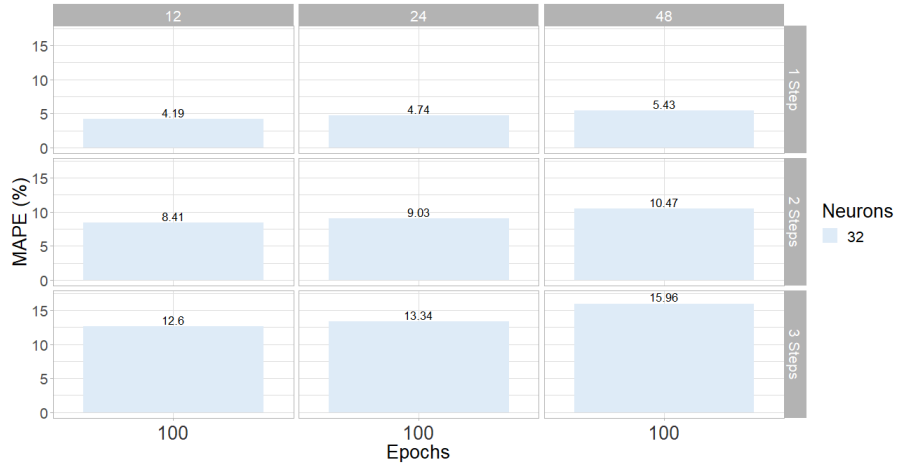


(a)

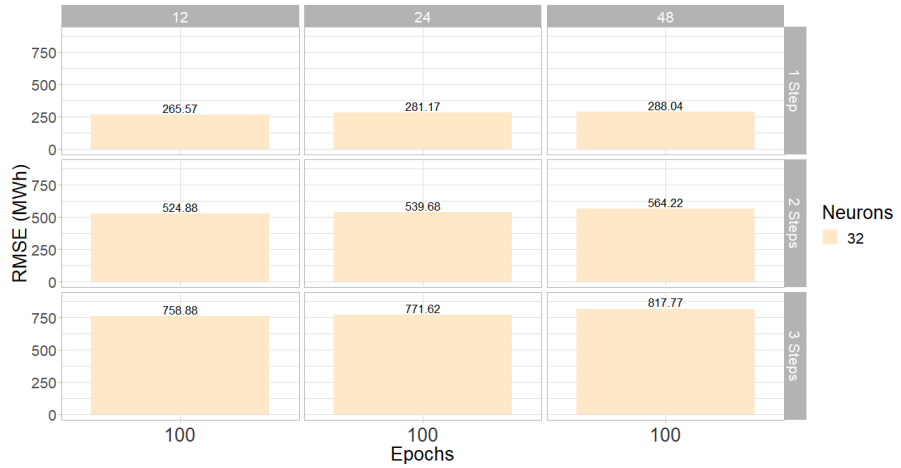


(b)

Fig. 4.25. (a) Recorded MAPE (%) given by 720 input time steps and 100 epochs, classified by batch size (12, 24, 46), neurons (32, 64) and output time step (1, 2, 3). (b) RMSE (MWh) for each step-ahead predicted.



(a)



(b)

Fig. 4.26. (a) Recorded MAPE (%) given by 2,160 input time steps and 100 epochs, classified by batch size (12, 24, 48), neurons (32) and output time step (1, 2, 3). (b) RMSE (MWh) for each step-ahead predicted.

4.2.6. Performance Summary

Table 4.3 shows the overall performance of the Vanilla, Bidirectional, Stacked, and Convolutional models, implemented given the configurations that yielded the lowest error reported in section 4.1. However, it must be noted that while the Vanilla en Bidirectional models were able to solve the three-steps ahead prediction task given the default recommended parameters of the Adam optimizer, this was not the case for the Stacked and Convolutional models. As referenced in their respective configurations below, the optimizer seemed incapable to solve the problem at hand with such a small value for ϵ. Therefore, several trial runs had to be completed in order to find the value of *epsilon* with which the model was able to perform.

Model	Forecasted Step	MAPE (%)	RMSE (MWh)	Implementation Time hh:mm:ss	Model Configuration
Vanilla LSTM	1	4.47	269.64	2:42:46	Time steps: 168 Neurons: 32 Epochs: 200 Batch Size: 46
	2	9.04	512.83		
	3	13.61	745.22		
Birectional LSTM	1	4.50	267.95	3:44:40	Time steps: 168 Neurons: 32 Epochs: 200 Batch Size: 46
	2	8.76	510.28		
	3	13.27	741.82		
Stacked LSTM	1	4.17	259.76	19:24:30	Time steps: 168 Neurons - Layer 1 & 2: 32, 32 Epochs: 300 Batch Size: 12 Epsilon= 1e-03
	2	8.98	514.61		
	3	13.86	749.57		
Convolutional LSTM	1	8.24	463.22	03:53:02	Time steps: 720 Neurons: 128 Epochs: 100 Batch Size: 12 Epsilon= 1e-05
	2	12.72	718.48		
	3	17.21	962.18		

Table 4.3. PERFORMANCE SUMMARY OF V-LSTM, BI-LSTM, LSTM-LSTM, CONV-LSTM MODELS EXECUTED FOR THREE-STEPS AHEAD FORECASTING.

Although some of the configurations of the AE-LSTM models did prove to perform better than the previous models, the differences are negligible, while the implementation times under this model architecture are extremely high. The results are referenced in table 4.4.

Model	Input Time Steps	Forecasted Steps	MAPE Range (%)	Mean MAPE (%)	RMSE Range (MWh)	Mean RMSE (MWh)	Implementation Time hh:mm:ss
Autoencoder LSTM	Weekly (168)	1	4.24 - 5.07	4.66	258.38 - 282.74	268.78	39:11:22
		2	8.62 - 9.24	8.99	512.68 - 523.60	517.61	
		3	12.81 - 13.73	13.44	737.91 - 748.89	743.68	
	Monthly (720)	1	4.24 - 4.72	4.55	261.24 - 275.92	267.80	60:11:42
		2	8.56 - 9.11	8.90	515.74 - 526.84	521.02	
		3	12.87 - 13.71	13.30	744.83 - 757.49	751.50	
	Quarterly (2,160)	1	4.19 - 5.43	4.79	265.57 - 288.04	278.26	121:40:19
		2	8.41 - 10.47	9.30	524.88 - 564.22	542.93	
		3	12.60 - 15.96	13.97	758.88 - 817.77	782.76	

Table 4.4. PERFORMANCE SUMMARY FOR AUTOENCODER LSTM FOR THREE-STEPS AHEAD FORECASTING.

5. CONCLUSIONS

The suite of LSTM RNN-based models presented in this study, along with the diverse patterns in which each architecture was implemented, allowed for a vast comparative analysis for one-step to three-steps ahead wind power forecasting. The significance of the work set forth translates into propelling wind power penetration within electricity markets, safe and efficient management of power systems, and optimum operation of power systems.

The work performed herein considered different LSTM-RNN architectures implemented in accordance with the size of the input learning sequence defined as weekly (168 time steps), monthly (720 time steps), and quarterly (2,160 time steps). The diverse model architectures were initially implemented given an iterative process, where different hyperparameters were varied, such that the models learned in different patterns given different input learning sequences. This implementation process was executed in order to tackle one-step and three-steps ahead predictions.

For one-step ahead wind power forecasting, according to the overall performance results recorded in table 4.2, we may conclude that the majority of the architectures produce better results when implemented with an input learning sequene of 168 steps (weekly time steps). Further, we may conclude that the trade-off between the error metric results and implementation times is not valuable enough as to choose a more complex nor deeper model (Stacked-LSTM or Conv-LSTM). While the Vanilla and Bidirectional models yielded MAPEs between 4.30-12.21%, the Stacked and Convolutional architectures produced errors between 4.37-14.13%. Additionally, Vanilla and Bidirectional models were able to implement all proposed hyperparameter configurations, referenced in table 3.2, in an approximately 23 and 40 hours, respectively. In contrast, the Stacked and Convolutional models had to be modified, limiting the variation of certain hyperparameters, for them to complete the task at hand in an adequate timeline; these limited implementations took a total of approximately 32 hours for the Stacked models, and 106 hours for the Conv-LSTM.

Likewise, the three-steps ahead forecasting executions suffered from extremely high implementation times, and the overall observed metrics variations did not justify the selection of deeper or more complex models. However, the vast majority of the AE-LSTM models produced similar results regardless of the input learning sequence size, and suffered less variations among the metrics, when compared to the reported performance of the models covered in 4.3.

5.1. Limitations and Future Work

While the study completed within this work fills in some of the research gaps with respect to DL comparative analysis, we have simply laid the blue print for future research focused in comparative analyses and optimization of LSTM-RNN architectures for short-term forecasting in the field of wind energy applications. Future works may consider a different strategy, taking into account the performance of the models with the lowest-yielding errors, for one-step and three-steps ahead forecasting. For instance, the consideration of an input learning sequence of 8,760 steps (equivalent to a year worth of data) may be explored. This input sequence size was not considered in this study due to the relatively short time series data available, which would limit the size of the validation data set. Moreover, the additional computational load of this input sequence size would not produce results in an adequate time frame. Following this larger input learning sequence, we may consider the use of Autoencoders to pre-process the data, followed by one of the LSTM architectures covered within this work. This framework would take into account the seasonality of the data throughout the year and could possibly aid in reducing computational times.

Moreover, our ongoing research will focus on the behavior of the wind power time series data to develop an ensemble model, where a time series Kmeans will be used to first extract, identify significant features and cluster patterns at diverse time scales. The results would then be combined with LSTM RNNs.

While some of the hyperparameters chosen for the iterative process were selected based on the reported effects these have had on the reported metrics when altered, others were selected due to their reported results and success when applied to short-term wind power forecasting. However, given the major set back that implementation times played, restricting our ability to conduct the experiments given identical configuration possibilities, future works may consider the incorporation of parallel or high performance computing (HPC) in order to efficiently solve the forecasting tasks at hand, consider more hyperparameters to be included in the iterative process, and assess deeper models.

Lastly, in order to complement the presented comparative analysis, it is important that future works conduct an exhausitive analysis inclusive of other time series forecasting techniques, such as statiscal models like ARIMA. However, this was out of the scope of the present work.

BIBLIOGRAPHY

[1] W. Tong, *Wind Power Generation and Wind Turbine Design*, 1st ed. 25 Bridge Street, Billerica, MA 01821, USA: WIT Press, 2010.

[2] *Renewables 2020 global status report*, https://ren21.net/gsr-2020/, 2020.

[3] L. Li *et al.*, "Review and outlook on the international renewable energy development," *Energy and Built Environment*, 2020. DOI: https://doi.org/10.1016/j.enbenv.2020.12.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2666123320301148.

[4] G. Giebel, G. Kariniotakis, and R. Brownsword, "The state-of-the-art in short term prediction of wind power from a danish perspective," *Proceedings of the 4th International Workshop on large scale integration of wind power and transmission networks for offshore wind farms*, Oct. 2003.

[5] P. Du, J. Wang, W. Yang, and T. Niu, "A novel hybrid model for short-term wind power forecasting," *Applied Soft Computing*, vol. 80, pp. 93–106, 2019.

[6] G. Marulanda, A. Bello, J. Cifuentes Quintero, and J. Reneses, "Wind power long-term scenario generation considering spatial-temporal dependencies in coupled electricity markets," *Energies*, vol. 13, p. 3427, Jul. 2020. DOI: 10.3390/en13133427.

[7] Y. Ren, P. Suganthan, and N. Srikanth, "Ensemble methods for wind and solar power forecasting—a state-of-the-art review," *Renewable and Sustainable Energy Reviews*, vol. 50, pp. 82–91, 2015. DOI: https://doi.org/10.1016/j.rser.2015.04.081. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032115003512.

[8] Y. Cao and L. Gui, "Multi-step wind power forecasting model using lstm networks, similar time series and lightgbm," in *2018 5th International Conference on Systems and Informatics (ICSAI)*, 2018, pp. 192–197. DOI: 10.1109/ICSAI.2018.8599498.

[9] A. Abdelaziz, M. Rahman, M. El-Khayat, and M. Hakim, "Short term wind power forecasting using autoregressive integrated moving average modeling," Dec. 2012.

[10] A. Essien and C. Giannetti, "A deep learning framework for univariate time series prediction using convolutional lstm stacked autoencoders," in *2019 IEEE International Symposium on INnovations in Intelligent SysTems and Applications (INISTA)*, 2019, pp. 1–6. DOI: 10.1109/INISTA.2019.8778417.

[11] J. Cifuentes, G. Marulanda, A. Bello, and J. Reneses, "Air temperature forecasting using machine learning techniques: A review," *Energies*, vol. 13, no. 16, p. 4215, 2020.

[12] M. A. Rodriguez, J. F. Sotomonte, J. Cifuentes, and M. Bueno-López, "A classification method for power-quality disturbances using hilbert–huang transform and lstm recurrent neural networks," *Journal of Electrical Engineering & Technology*, pp. 1–18, 2020.

[13] A. Eseye, J. Zhang, D. Zheng, and D. Shiferaw, "Short-term wind power forecasting using artificial neural networks for resource scheduling in microgrids," *International Journal of Science and Engineering Applications*, vol. 5, pp. 144–151, Apr. 2016. DOI: 10.7753/IJSEA0503.1005.

[14] J. Shi, Y. Liu, Y. Yang, and W. Lee, "Short-term wind power prediction based on wavelet transform-support vector machine and statistic characteristics analysis," in *2011 IEEE Industrial and Commercial Power Systems Technical Conference*, 2011, pp. 1–7. DOI: 10.1109/ICPS.2011.5890873.

[15] B. Yang *et al.*, "State-of-the-art one-stop handbook on wind forecasting technologies: An overview of classifications, methodologies, and analysis," *Journal of Cleaner Production*, p. 124 628, 2020. DOI: https://doi.org/10.1016/j.jclepro.2020.124628. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0959652620346722.

[16] Y. Chen, Y. Wang, D. Kirschen, and B. Zhang, "Model-free renewable scenario generation using generative adversarial networks," *IEEE Transactions on Power Systems*, vol. 33, no. 3, pp. 3265–3275, 2018. DOI: 10.1109/TPWRS.2018.2794541.

[17] C. Yildiz, H. Acikgoz, D. Korkmaz, and U. Budak, "An improved residual-based convolutional neural network for very short-term wind power forecasting," *Energy Conversion and Management*, vol. 228, p. 113 731, 2021. DOI: https://doi.org/10.1016/j.enconman.2020.113731. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196890420312553.

[18] G. Marulanda, J. Cifuentes, A. Bello, and J. Reneses, "Short-term wind power forecasting by a long short term memory ensemble approach," 2020.

[19] J. Cifuentes, P. Boulanger, M. T. Pham, F. Prieto, and R. Moreau, "Gesture classification using lstm recurrent neural networks," in *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, IEEE, 2019, pp. 6864–6867.

[20] Qu Xiaoyun, Kang Xiaoning, Zhang Chao, Jiang Shuai, and Ma Xiuda, "Short-term prediction of wind power based on deep long short-term memory," pp. 1148–1152, 2016. DOI: 10.1109/APPEEC.2016.7779672.

[21] Y. Liu *et al.*, "Wind power short-term prediction based on lstm and discrete wavelet transform," *Applied Sciences*, vol. 9, p. 1108, Mar. 2019. DOI: 10.3390/app9061108.

[22] Z. Sun and M. Zhao, "Short-term wind power forecasting based on vmd decomposition, convlstm networks and error analysis," *IEEE Access*, vol. 8, pp. 134 422–134 434, 2020. DOI: 10.1109/ACCESS.2020.3011060.

[23] S. Liang, L. Nguyen, and F. Jin, "A multi-variable stacked long-short term memory network for wind speed forecasting," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 4561–4564. DOI: 10.1109/BigData.2018.8622332.

[24] N. Tavakoli, "Modeling genome data using bidirectional lstm," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, 2019, pp. 183–188. DOI: 10.1109/COMPSAC.2019.10204.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 8 1997. DOI: 10.1162/neco.1997.9.8.1735.

[26] R. C. Staudemeyer and E. R. Morris, *Understanding lstm – a tutorial into long short-term memory recurrent neural networks*, 2019. arXiv: 1909.09586 [cs.NE].

[27] H. Zhen *et al.*, "A hybrid deep learning model and comparison for wind power forecasting considering temporal-spatial feature extraction," *Sustainability*, vol. 12, no. 22, 2020. DOI: 10.3390/su12229490. [Online]. Available: https://www.mdpi.com/2071-1050/12/22/9490.

[28] L. Yu, J. Qu, F. Gao, and Y. Tian, "A novel hierarchical algorithm for bearing fault diagnosis based on stacked lstm," *Shock and Vibration*, vol. 2019, 2019.

[29] A. Essien and C. Giannetti, "A deep learning model for smart manufacturing using convolutional lstm neural network autoencoder," 9, vol. 16, 2020.

[30] K. O'Shea and R. Nash, *An introduction to convolutional neural networks*, 2015. arXiv: 1511.08458 [cs.NE].

[31] A. Sagheer and M. Kotb, "Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems," *Scientific Reports*, vol. 9, 2019. DOI: https://doi.org/10.1038/s41598-019-55320-6.

[32] A. Essein and C. Giannetti, "A deep learning model for smart manufacturing using convolutional lstm neural network autoencoders," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, 2020.

[33] G. Liu, H. Bao, and B. Han, ""a stacked autoencoder-based deep neural network for achieving gearbox fault diagnosis"," *Mathematical Problems in Engineering*, vol. 2018, 2018. DOI: https://doi.org/10.1155/2018/5105709.

[34] H. H. Aly, "A novel deep learning intelligent clustered hybrid models for wind speed and power forecasting," *Energy*, vol. 213, p. 118 773, 2020.

[35] *Spanish peninsula - electricity demand tracking in real time*. [Online]. Available: https://demanda.ree.es/visiona/peninsula/demanda/.

[36] C. Chen and L.-M. Liu, "Joint estimation of model parameters and outlier effects in time series," *Journal of the American Statistical Association*, vol. 88, no. 421, pp. 284–297, 1993.

[37] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Transactions on Knowledge and data Engineering*, vol. 26, no. 9, pp. 2250–2267, 2013.

[38] *Installed power & wind generation*, 2020. [Online]. Available: https://www.aeeolica.org/sobre-la-eolica/la-eolica-espana/potencia-instalada-y-generacion, (accessed: 01.10.2021).

[39] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[40] ——, *Adam: A method for stochastic optimization*, 2014.

[41] V.-H. Nhu *et al.*, "Effectiveness assessment of keras based deep learning with different robust optimization algorithms for shallow landslide susceptibility mapping at tropical area," *Catena*, vol. 188, p. 104 458, 2020.

[42] M. A. Rodriguez, J. F. Sotomonte, J. Cifuentes, and M. Bueno-López, "Power quality disturbance classification via deep convolutional auto-encoders and stacked lstm recurrent neural networks," in *2020 International Conference on Smart Energy Systems and Technologies (SEST)*, IEEE, 2020, pp. 1–6.

[43] P. Wang, J. P. Rowe, W. Min, B. W. Mott, and J. C. Lester, "Interactive narrative personalization with deep reinforcement learning.," in *IJCAI*, 2017, pp. 3852–3858.

[44] H. Zhen *et al.*, "A hybrid deep learning model and comparison for wind power forecasting considering temporal-spatial feature extraction," *Sustainability*, vol. 12, no. 22, 2020. DOI: 10.3390/su12229490. [Online]. Available: https://www.mdpi.com/2071-1050/12/22/9490.

[45] *Tensorflow api documents*. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers, (accessed: 01.10.2021).