UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESIGN AND IMPLEMENTATION OF AN HVAC CONSUMPTION PREDICTION SYSTEM BASED ON LSTM NEURAL NETWORKS

Author: Rafael Sendra Arranz

Tutor: Álvaro Gutiérrez Martín

2018

Abstract

This Thesis designs and implements an artificial neural network based predictor to forecast the power consumption of an HVAC system. The featured HVAC system is situated at the MagicBox. The MagicBox is a self-sufficient solar house with a monitoring system located at the Escuela Técnica Superior de Ingenieros de Telecomunicación (ETSIT) of the Universidad Politécnica de Madrid (UPM). The presented predictor makes short term predictions, more precisely it forecasts one day ahead of the power consumption. In this Thesis, three multi step prediction models, based on LSTM neural networks, are proposed. In addition, suitable data preprocessing and arrangement techniques are set to adapt the raw dataset. The models provide outstanding results in terms of errors and correlation between the temporal behaviour of the predictions and targets.

Keywords: Energy consumption prediction, HVAC systems, short term forecast, artificial neural networks, recurrent neural networks, LSTM layers, Mean squared error, Pearson correlation coefficient.

Resumen

En este Trabajo Fin de Grado se ha diseñado e implementado un predictor basado en redes neuronales artificiales para la predicción del consumo de potencia de un sistema HVAC. El sistema HVAC en cuestión está situado en una casa solar llamada MagicBox. La MagicBox es una casa solar autosuficiente equipada con un sistema de monitorización y localizada en Escuela Técnica Superior de Ingenieros de Telecomunicación (ETSIT) de la Universidad Politécnica de Madrid (UPM). El predictor presentado realiza estimaciones a corto plazo de la potencia consumida, concretamente predice el día siguiente de consumo. En este Trabajo Fin de Grado tres modelos de predicción de múltiples instantes de tiempo son presentados. Además, se exponen diversas técnicas de preprocesado de los datos con el fin de adaptar el dataset original. Los modelos presentan resultados destacables en referencia al computo de errores y medidas de la correlación entre las series temporales de las predicciones y los datos reales.

Palabras clave: Predicción del consumo de energía, sistemas HVAC, predicción a corto plazo, redes neuronales artificiales, redes neuronales recurrentes, capas LSTM, error cuadrático medio, coeficiente de correlación de Pearson.

Agradecimientos

En primer lugar, quiero mostrar mi agradecimiento a mi tutor, Dr. Álvaro Gutiérrez, por toda la ayuda y asesoramiento prestado durante el desarrollo de este Trabajo Fin de Grado así como por haber creído en mi todo momento.

También quisiera agradecer a mis padres y hermanos por todo el apoyo y paciencia que siempre me han proporcionado.

Contents

List of Figures

Index of Tables

List of Acronyms

PV: Photovoltaic.

BESS: Battery Energy Storage System.

HVAC: Heating, ventilation and air conditioning.

ANN: Artificial Neural Network.

RNN: Recurrent Neural Network.

LSTM: Long-Short Term Memory.

BBTT: Back-Propagation Through Time.

OSPM: One-Step Prediction Model.

MSPM: Multi-Step Prediction Model.

SGD: Stochastic Gradient Descent.

Chapter 1

Introduction and objectives

1.1 Context

Nowadays, there is an increasing concern in the society for the correct consumption and use of energy. This phenomenon has motivated the development of new technologies turning into real scientific challenges. One of the main problems in this context is to find a suitable equilibrium between the energy consumption and its generation. This is a complex task because of different reasons as for instance: the consumption is not equally distributed within a day, due to the existence of peaks of aggregated consumption in the activity hours, and valleys at inactivity periods (such as the night); the consumption remarkably depends on the season of the year; it is strongly altered by the human habits, etc.

To solve these problems Demand Side Management (DSM) techniques have been studied and applied. DSM can be defined as the set of technologies and procedures that modify the aggregated energy consumption in order to fulfill an imposed goal such as the balance and flattening of the demand side energy consumption curve.

Furthermore, in the recent years, Smart Grids have emerged to aid the DSM purposes among other utilities. Smart Grids can be seen as the combination of the regular energy grid system, renewable energies, battery energy storage system (BESS), and intelligent systems aiming to modify the use of the consumption elements in order to enhance the energy efficiency. These intelligent grids allow not only the energy flow but also the data flow to control the demand response.

Several authors have addressed the problem by means of these techniques. Asghar et al. (2015) proposes a Generic DSM model based on a genetic algorithm for residential users to reduce the Peak-to-Average Ratio, the total energy cost and the waiting time of appliances. In addition, in Ullah et al. (2013) several residential load controlling techniques are described. It is based on the scheduling and time shifting of the operation of the loads in order to smooth the energy demand curve. It was shown how those methods reduced the energy consumption cost and the Peak-to-Average Ratio. Furthermore, in Castillo-Cagigal et al. (2011a) the authors present simulated and real experiments integrating BESS and Photovoltaic(PV) generation along with Active Demand side management (ADSM) in a grid connected self sufficient house to maximize the PV energy self consumption.

The problem described above is too wide to be treated globally. One may consider

the study for separate consumptions of different machines to be integrated jointly in a later phase of the treatment of the problem. In this work we focus on the consumption of heating, ventilation and air-conditioning (HVAC) systems. These systems represent a great percentage of the energy consumption in residential buildings. Thus, its study can be considered an important task to be developed. A second important issue of the effective treatment of DSM techniques is to enable the prediction of the future behaviour of the aggregated consumption. Thus, the energy consumption forecast of HVAC systems could lead to an enhancement of the used DSM with controlling and scheduling techniques.

To approach the prediction, different techniques have been utilized as, for instance, linear regression models, Autoregressive, Moving Average and Autoregressive Integrated Moving Average models, Support Vector Machines and Artificial Neural Networks, among others.

In Solano et al. (2017), a predictor of the power consumption of an HVAC system was implemented to aid the operation of two control strategies. These strategies aim to increase the PV self-consumption and grid-peak shaving respectively and were developed and assessed in the frame of a self-sufficient solar house, called MagicBox, with integrated BESS, PV generation and monitoring systems. The HVAC predictor was based on a *linear regression* model and, as stated by the authors, it was left as future research the design of more accurate forecasting techniques. The current Thesis can be seen as a continuation in that direction. Indeed, in this work, a more complex prediction system of the HVAC power consumption is designed, implemented and assessed under the same self sufficient house mentioned above. Below in this section, a more extensive description of the contribution is presented.

In order to develop the design of a time series predictor, such as the power consumption, several variables have to be taken into account. Firstly, one considers the time series technique to make the forecast. In addition to the linear regression model discussed above, Deng and Jirutitijaroen (2010) compared *Autoregressive*, *Moving Average* and *Autoregressive Integrated Moving Average* for short term load forecasting.

Alternatively, *Support Vector Machines* were utilized as the prediction technique in Hou and Lian (2009) and Ceperic et al. (2013). Finally, *Artificial Neural Networks* (ANN) have been considered in Beccali et al. (2008) and Park et al. (1991). Moreover, the use of Recurrent Neural Networks (RNN) is strongly recommended because they are able to retain and consider the temporal variations of the time series throughout their feedback connections. González and Zamarreño (2005) takes advance of these particular type of ANNs to develop a model to forecast hourly energy consumption. In the research presented here, Long-Short Term Memory (LSTM) neural networks (see Section 2.3) are utilized. LSTM neural networks are a type of RNNs presented in Hochreiter and Schmidhuber (1997) to solve the vanishing gradient issue of regular RNNs.

The second aspect to be considered during the design of a time series predictor is the horizon of the desired forecasts. Normally, short and long term forecasts are the most known time horizons. As said above, the predictor designed here is thought

to aid the DSM techniques such as the task scheduling of the consuming loads. Therefore short term forecasting is the desirable time horizon for this application. More precisely, a day ahead prediction is discussed (see Section 3.5.2). Lusis et al. (2017) and Gajowniczek and Zabkowski (2014) compare different prediction models to forecast short term load consumption. On the other hand, long term forecasting is studied in Rahman et al. (2018) with the use of an encoder–decoder based LTSM ANN. However, although the techniques used in the paper are similar to those of this Thesis, it is important to take into account that long term horizon forecast essentially corresponds a different problem to the one consider in this Thesis.

1.2 Objectives and contribution

After the study of the state of the art in the frame of the problem to solve, the objectives of this Thesis are described. The main goal is to design and implement a predictor with the aim of performing the forecast of the energy consumed by the HVAC system of a self-sufficient solar house, MagicBox, located at the ETSIT-UPM. Furthermore, the designed predictor should make forecasts of the next day in order to have the potential to be integrated in a real application. Morevoer, the system's predictions should provide sufficiently low errors and correlation with the targeted time series.

In the following, the contributions of this Thesis are stated. A set of models to perform short term forecasts of the power consumption of an HVAC system are proposed. The predictor model consists of an stacked LSTM ANN trained by several data obtained from the aforementioned MagicBox. Firstly, a single step ahead predictor is stated as an introductory approach. Subsequently, three models are designed and implemented with the aim to forecast short term multiple steps ahead. More precisely, the RNNs will output the next day when the previous day's sequence is provided. The multiple steps ahead predictors show a high performance results, highlighting test pearson correlation coefficients around 0.97 and normalized root mean square error (NRMSE) of 0.049 for the most accurate model (see Section 4.2 for an explanation of those metrics).

1.3 Document layout

Chapter 1 has briefly described the structure of the overall document, commenting the most important aspects treated in each of the chapters.

- Chapter 2 provides a basic description of the theoretical concepts and techniques that will be relevant and used throughout this work. More precisely, it gives an insight of the concept and mathematics behind artificial neural networks, focusing on recurrent neural networks. In addition, an introduction of one of the most known and utilized algorithms for performing the training of ANNs, namely the backpropagation algorithm, is provided. Finally, LSTM neural networks are explained, describing its behaviour, basic concepts behind its units and exposing the main reasons of its creation.

- Chapter 3 starts presenting the nature of the utilized dataset, and introduce the aforementioned self-sufficient house where the data was measured. Afterwards, the preprocessing and arrangement techniques for the data set, selected with the aim of adapting the data to suitably feed the RNN, is described. In addition, the main architecture of the RNN models is stated. This architecture will be inherited with all the subsequent models with the proper modification to perform the single or multiple step ahead predictions. Finally, the single step model and the three multi step architectures are presented and analysed.

- Chapter 4 is devoted to the assessment of the designed models in Chapter 3. It started by introducing the utilized tools and the programming environment in which the RNNs where implemented and verified. Secondly, the metrics used to measure the performance of the models are arranged and explained. Afterwards, a selection of some of the most relevant parameters and algorithms used to complete the RNN structure and the training process is performed. The followed method to choose this parameters is based on an empirical procedure. To conclude the chapter, the verification of the models with the selected parameters is accomplished. In addition, in this chapter, the reader can consult a comparison between the models a summary of the metrics measuring the effectiveness of the designed systems.

- Chapter 5 ends the Thesis with the conclusions on the results, that can be observed throughout the whole document. In addition, possible future research lines, in the frame of the treated problem, are exposed.

Chapter 2

Theory

This section is devoted to summarize the main ideas from the existing theory that will be used throughout this document. For further details we refer to Haykin (2009), Goodfellow et al. (2016) and Demuth et al. (2014).

Firstly a brief insight of the concepts and computations made by the most basic artificial neural network, namely the perceptron neural network (and it's multilayer variation), is provided. This network will lead to the basic recurrent neural network which is described as well.

In addition, the backpropagation algorithm, which is one of the most known machine learning algorithms in the frame of training artificial neural networks, is explained and analysed. Moreover, to support the application of this algorithm to the recurrent neural networks, the Backpropagation Through Time (BPTT) variation is compactly introduced. This variation of the backpropagation method exposes one of the main reasons of utilizing Long-Short Term Memory (LSTM) neural networks. LSTM neural networks are the last sort of ANN's described in this chapter.

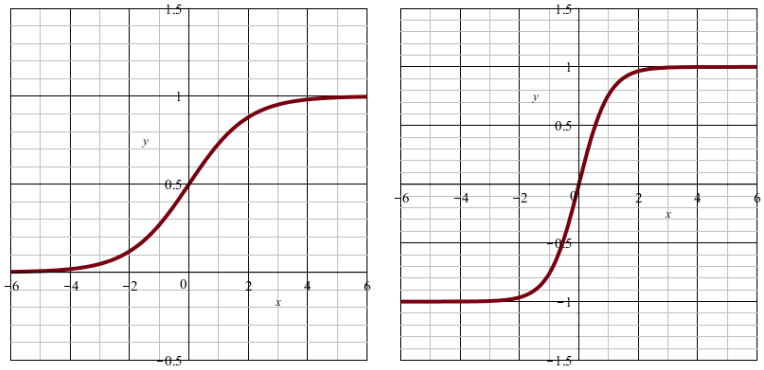
2.1 Perceptron and Recurrent neural networks

Recurrent neural networks (RNN) are a type of artificial neural networks with a feedback connection between the output and the input with unit delays. This feedback connection permits the RNN to maintain the temporal dependencies. These sort of artificial neural networks excel when the treated problem involves input and output data with sequential nature. Some examples of the application of recurrent neural networks are the machine translation (Cho et al. (2014a)), time series forecast (Giles et al. (2001)) or speech recognition (Graves et al. (2013)). In this document a time series prediction problem is stated. Thus a recurrent neural network, more precisely an LSTM based architecture, will be proposed to deal with our problem.

We start introducing the mathematical notation for describing the basic theory of the ANNs while they will be appearing in the text. First, let $\varphi(\cdot)$ denote the arbitrary activation function of the recurrent neural network. Then $\mathbf{\Phi}(\mathbf{v})$, where $\mathbf{v} = (v_1, \ldots, v_J)$, is defined as:

$$\mathbf{\Phi}(\mathbf{v}) = \begin{bmatrix} \varphi(v_1) \\ \varphi(v_2) \\ \vdots \\ \varphi(v_J) \end{bmatrix} \tag{2.1.1}$$

This function, known as the activation vector function of a neural network's layer, is usually set to a sigmoid or a hyperbolic tangent (see Figs. 2.1(a) and 2.1(b)) in order to fix non linear decision boundaries.



(a) (b)

Figure 2.1: Plot of the sigmoid (a) and hyperbolic tangent (b) functions.

Thus, with the previous definitions in mind, we may introduce the simplest neural network, namely the perceptron, as follows

$$\mathbf{y}_k = \mathbf{\Phi}(\mathbf{W}\mathbf{x}_k + \mathbf{b}). \tag{2.1.2}$$

where $\mathbf{x}_k$ and $\mathbf{y}_k$ are the input and output vectors at the time step k. In addition, let $\mathbf{W}$ be the weight matrix that apply a linear transformation to $\mathbf{x}_k$ and let $\mathbf{b}$ be a vector, called the bias, that will be used to apply an affine transformation to the linear transformation $\mathbf{W}$.

This network only has a layer and it is obviously non-recursive. Similarly, Eq. 2.1.3 describes de behaviour of a single layer recurrent neural network

$$\mathbf{h}_k = \Phi(\mathbf{W}\mathbf{x}_k + \mathbf{U}\mathbf{h}_{k-1} + \mathbf{b}) \tag{2.1.3}$$

where $\mathbf{h}_k$ and $\mathbf{h}_{k-1}$ are the RNN's output at the current and previous time step. Let $\mathbf{U}$ be the weight matrix that applies the linear transformation to $\mathbf{h}_{k-1}$.

It can be observed how the previous output of the network is fed back as an input of the layer.

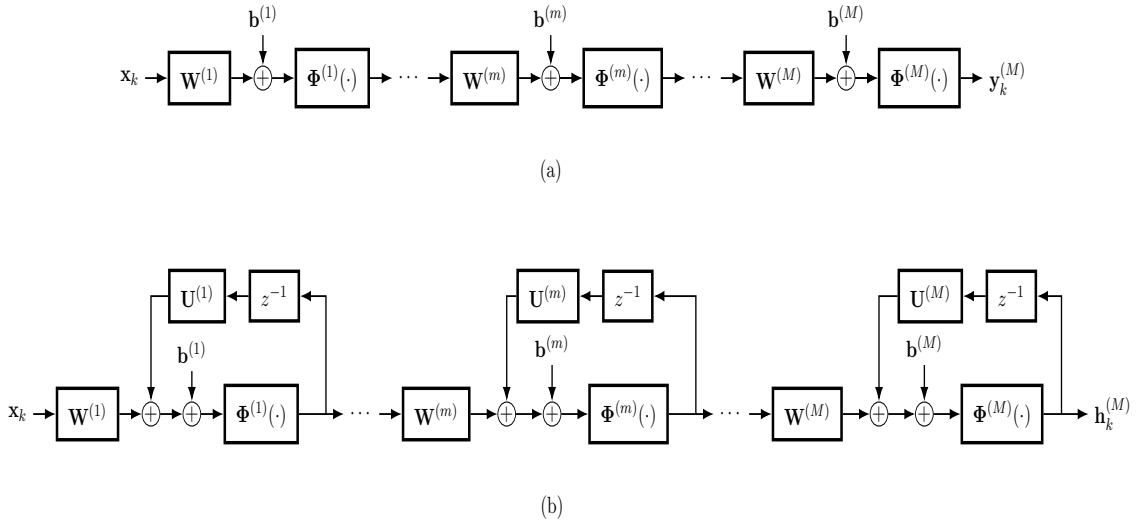


Figure 2.2: (a) Computational graph of a multilayer perceptron. (b) Computational graph of a multilayer RNN.

The networks in Eq. 2.1.2 and Eq. 2.1.3 can be extended to the case of multi layers. Let M be the number of layers and $\mathbf{U}^{(m)}, \mathbf{W}^{(m)}, \mathbf{b}^{(m)}$ the weight matrices and biases at the m-th layer. Moreover, let $\mathbf{\Phi}^{(m)}$ be the activation vector function at the m-th layer. Also, let $\mathbf{h}_k^{(m)}$ be the output vector at the m-layer. Then, the multilayer perceptron is described as (see Fig. 2.2 a)

$$\begin{cases} \mathbf{y}_k^{(m)} &= \mathbf{\Phi}^{(m)}\left(\mathbf{W}^{(m)}\mathbf{y}_k^{(m-1)} + \mathbf{b}^{(m)}\right) \text{ with } m \in \{2, \dots, M\} \\ \mathbf{y}_k^{(1)} &= \mathbf{\Phi}^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x}_k + \mathbf{b}^{(1)}\right) \end{cases} \tag{2.1.4}$$

It can be noticed how in the multilayer perceptron, the time indexing variable k could be omitted due to the absence of a feedback connection. Finally, the multilayer RNN works as follows (see also Fig. 2.2 b)

$$\begin{cases} \mathbf{h}_k^{(m)} &= \mathbf{\Phi}^{(m)}\left(\mathbf{W}^{(m)}\mathbf{h}_k^{(m-1)} + \mathbf{U}^{(m)}\mathbf{h}_{k-1}^{(m)} + \mathbf{b}^{(m)}\right) \text{ with } m \in \{2, \dots, M\} \\ \mathbf{h}_k^{(1)} &= \mathbf{\Phi}^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x}_k + \mathbf{U}^{(1)}\mathbf{h}_{k-1}^{(1)} + \mathbf{b}^{(1)}\right) \end{cases} \tag{2.1.5}$$

2.2 Machine learning algorithm

The machine learning techniques focus on training artificial neural networks by minimizing certain error or loss function $\mathcal{L}$. For this purpose, in a supervised learning, a collection of inputs $\mathcal{I}$ and outputs or targets $\mathcal{T}$ is required (see Chapter 3.1). Then, the error function measures the distance between the elements of $\mathcal{T}$ and the predictions generated by the network when receives inputs from $\mathcal{I}$.

The back-propagation algorithm Rumelhart et al. (1988) is the most utilized supervised machine learning algorithm to train artificial neural networks, under the assumption that the activation functions are differentiable enough. This method is based on the gradient descent algorithm, aiming to adjust the weights and biases of the ANN to minimize $\mathcal{L}$.

In order to describe the method, we treat first the perceptron case to afterwards extend the method to the recurrent case. The gradient descent algorithm iteratively reaches a local minimum of the loss function $\mathcal{L}$ with respect to the artificial neural network parameters. The parameters of the ANN are the entries of all the weight matrices $\mathbf{W}^{(m)}$ and all the biases $\mathbf{b}^{(m)}$ (see Eq. 2.1.4). Nevertheless, the precise definition of the real function $\mathcal{L}$ is postponed till Eq. 2.2.3. Firstly, the main iteration of the method, assuming that $\mathcal{L}$ is sufficiently differentiable, is presented. This process is performed by computing the gradient of $\mathcal{L}$ to minimize and update the network parameters with a proportion of that gradient.

Let $\mathbf{\Theta}^{(m)}$ be the parameters involved in the m-th layer of the ANN, where $m \in \{1, ..., M\}$. The gradient descent algorithm uses the following equation to update the parameters $\mathbf{\Theta}^{(m)}$,

$$\mathbf{\Theta}^{(m)}(r+1) = \mathbf{\Theta}^{(m)}(r) + \alpha \nabla_{\mathbf{\Theta}^{(m)}} \mathcal{L}(\mathbf{\Theta}^{(m)}(r)) \tag{2.2.1}$$

where r represents the index of the iteration process, $\nabla_{\mathbf{\Theta}^{(m)}}$ represents the gradient with respect to the variables $\mathbf{\Theta}^{(m)}$, and α is the learning rate constant that establishes the proportion of the gradient used to update the parameters. A wise selection of the learning rate is crucial to perform the training successfully.

Now, we introduce the loss function. Let $\mathbf{t}(n)$ denote the target examples and $\mathbf{y}(n)$ the predictions made by the system for the sample n. Then, we introduce the vector $\boldsymbol{\ell}(n)$ as

$$\boldsymbol{\ell}(n) = \mathbf{t}(n) - \mathbf{y}(n). \tag{2.2.2}$$

In this situation, we introduce the loss function as

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \|\boldsymbol{\ell}(n)\|_2^2 \tag{2.2.3}$$

where $n \in \{1, ..., N\}$ denotes the sample n, and $\|\cdot\|_2$ denotes the Euclidean 2-norm. In addition, N represents the number of samples taken into account to compute the next descent in the algorithm with Eq. 2.2.1. If an online training is performed, then $N = 1$, because a parameter update is made with every sample provided. In contrast, the batch training can be used. This mode adjusts the weights and biases after $N > 1$ samples are fed.

In this situation, the back-propagation algorithm consists of the following three main steps:

1. The first stage of the algorithm is the forward-propagation. This process feeds the input data into the ANN in order to compute the loss function $\mathcal{L}$ with the system's output.

2. Once the loss function is calculated, the back-propagation step is performed. This stage aims to compute the gradient of $\mathcal{L}$ to update the ANN parameters as established in Eq.2.2.1. Then the gradient $\nabla_{\mathbf{\Theta}^{(m)}} \mathcal{L}(\mathbf{\Theta}^{(m)}(r))$ can be computed as,

$$\nabla_{\mathbf{\Theta}^{(m)}} \mathcal{L}(\mathbf{\Theta}^{(m)}(r)) = \boldsymbol{\delta}^{(m)}(r) \mathbf{y}^{(m-1)}(r) \tag{2.2.4}$$

with $\mathbf{y}^{(m-1)}(r)$ being the output of the $(m-1)$-th layer at the iteration r (see Eq. 2.1.4), and $\boldsymbol{\delta}^{(m)}(r)$ being the local gradient or sensibility of the m-th layer. Moreover, $\boldsymbol{\delta}^{(m)}$ can be computed as follows:

$$\begin{cases} \boldsymbol{\delta}^{(M)}(r) &= -2(\boldsymbol{\ell}(r) \circ \dot{\boldsymbol{\Phi}}^{(M)}(\mathbf{v}^{(M)}(r))) \\ \boldsymbol{\delta}^{(m)}(r) &= \left(\dot{\boldsymbol{\Phi}}^{(m)}(\mathbf{v}^{(m)}(r)) \circ (\boldsymbol{\Theta}^{(m+1)}(r))\right)^T \boldsymbol{\delta}^{(m+1)}(r) \ \ \forall m \neq M \end{cases} \tag{2.2.5}$$

where $\circ$ denotes the Hadamard product of matrices (i.e. $(A_{ij}) \circ (B_{ij}) = (A_{ij}B_{ij})$), $\dot{\boldsymbol{\Phi}}$ is the derivative of $\boldsymbol{\Phi}$ with respect to the variable of φ (see Eq. 2.1.1) and

$$\mathbf{v}^{(m)}(r) = \mathbf{W}^{(m)}(r)\mathbf{y}^{(m-1)}(r) + \mathbf{b}^{(m)}(r) \tag{2.2.6}$$

with $\mathbf{y}^{(0)}(r) = \mathbf{x}(r)$. In addition, A^T denotes the transpose of the matrix A.

3. The final stage simply applies the results obtained by the back-propagation process to compute the next step in the gradient descent. The substitution of Eq. 2.2.4 into Eq. 2.2.1 leads to the final equation of the back-propagation algorithm

$$\boldsymbol{\Theta}^{(m)}(r+1) = \boldsymbol{\Theta}^{(m)}(r) + \alpha\boldsymbol{\delta}^{(m)}(r)\mathbf{y}^{(m-1)}(r) \tag{2.2.7}$$

The described back-propagation algorithm addresses the training process of feedforward (non-recurrent) artificial neural networks. In order to train a recurrent neural network, a variation of this algorithm is required. One of the possible solutions is the back-propagation-through-time (BPTT) algorithm. To understand the BPTT algorithm a brief insight of the unfolding of a recurrent neural network is needed.

An unfolding of a RNN consists of converting the recursive neural network into a feedforward network along a finite number of time steps. Eq. 2.2.8 denotes the unfolding of 3 time steps of a recurrent neural network described by the formula of Eq. 2.1.3.

$$\mathbf{h}_k = \boldsymbol{\Phi}(\mathbf{W}\mathbf{x}_k + \mathbf{U}(\underbrace{\boldsymbol{\Phi}(\mathbf{W}\mathbf{x}_{k-1} + \mathbf{U}(\underbrace{\boldsymbol{\Phi}(\mathbf{W}\mathbf{x}_{k-2} + \mathbf{U}\mathbf{h}_{k-3} + \mathbf{b})}_{\mathbf{h}_{k-2}} + \mathbf{b}))}_{\mathbf{h}_{k-1}} + \mathbf{b}) \tag{2.2.8}$$

Fig. 2.3 depicts the unfolding process computed in Eq. 2.2.8. An analogous treatment can be applied to the case of multi layers RNN. Chapter 3.5 harnesses this temporal unrolling process in order to develop multiple step forecasting models.

Thus, BPTT is an extension of the back-propagation algorithm that performs supervised machine learning of recurrent neural networks. The basic idea of this variation is to unfold the RNN $\tau < \infty$ time steps to convert the recursion into a multi layer perceptron. Therefore the regular back-propagation method can be applied to the unfolded RNN.

There are few details that should be pointed out about BPTT. The first of them is that along the unrolled RNN the same parameters (i.e. the entries of the weight

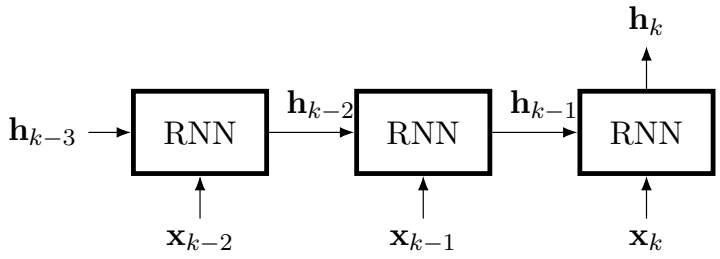


Figure 2.3: Illustration of the unfolding process described in Eq. 2.2.8

matrices and biases of the RNN) are applied and optimized. In addition, the back-propagation of the errors is just performed along the τ time steps. This leads to the requirement of the arrangement of the samples in the shape of $[\tau, \dim(\mathbf{x_k})]$.

Let us finish this subsection commenting some of the disadvantages of the BPTT, namely the fact that BPTT can trigger the vanishing of the gradient phenomenon. The vanishing of the gradient means that during the computation of the gradient in the back-propagation (see Eq. 2.2.4) it tends to an extremely small value. This causes the learning process to be unfeasible. The vanishing of the gradient happens when the number τ, of time steps, is too large. Then, the unfolded network is a multi layer perceptron with $\tau + 1$ layers. Moreover, there is a composition of $\tau + 1$ activation functions that strengthen the vanishing gradient, for instance when these functions are either sigmoid or hyperbolic tangent functions. Therefore the regular recurrent neural networks have strong limitation when long term dependencies are considered. This is the motivation of Section 2.3 where a solution to this problem is described.

2.3 LSTM neural networks

The vanishing of the gradient problem described above motivates the use of a particular type of RNN, namely LSTM neural network, that solves this issue. Long short-term memory (LSTM) neural networks are recurrent neural networks able to maintain both short and long time dependencies through its states. It is composed by gates that control the information that is stored in the states by filtering the input and output flows of data. These gates are called forget, input and output gates. An LSTM unit contains two different states, the hidden state ($\mathbf{h}_k$) which is analogous to the state of regular RNN described in Eq. 2.1.3. This state represents the output value of the LSTM layer set by the output gate. In contrast, the cell state ($\mathbf{c}_k$) is the actual memory that is controlled by the forget and input gates. The cell state is the responsible of maintaining the temporal dependencies.

Eq. 2.3.1 states the set of computations performed by an LSTM layer with multiple units. In addition, the computational graph of the formulas is depicted in Fig. 2.4.

$$
\left.
\begin{aligned}
\mathbf{f}_k &= \sigma(\mathbf{W}_f \mathbf{x}_k + \mathbf{U}_f \mathbf{h}_{k-1} + \mathbf{b}_f) \\
\mathbf{i}_k &= \sigma(\mathbf{W}_i \mathbf{x}_k + \mathbf{U}_i \mathbf{h}_{k-1} + \mathbf{b}_i) \\
\mathbf{o}_k &= \sigma(\mathbf{W}_o \mathbf{x}_k + \mathbf{U}_o \mathbf{h}_{k-1} + \mathbf{b}_o) \\
\mathbf{c}_{i_k} &= \mathbf{i}_k \circ \tanh(\mathbf{W}_c \mathbf{x}_k + \mathbf{U}_c \mathbf{h}_{k-1} + \mathbf{b}_c) \\
\mathbf{c}_k &= \mathbf{f}_k \circ \mathbf{c}_{k-1} + \mathbf{c}_{i_k} \\
\mathbf{h}_k &= \mathbf{o}_k \circ \tanh(\mathbf{c}_k)
\end{aligned}
\right\} \tag{2.3.1}
$$

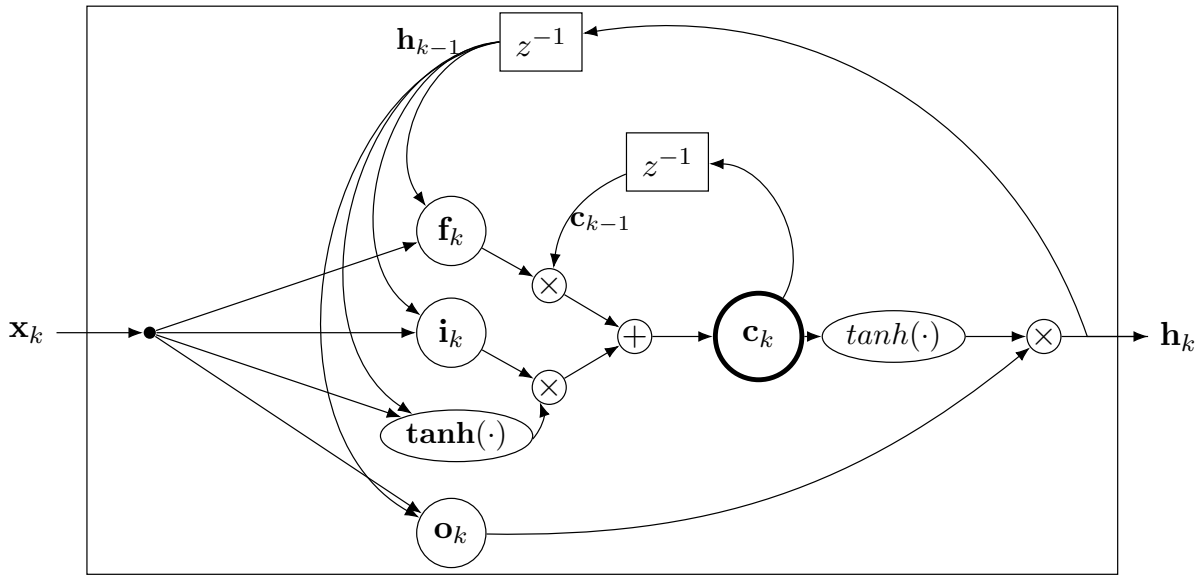


Figure 2.4: Computational graph of an LSTM layer.

In both, Eq. 2.3.1 and Fig. 2.4 $\mathbf{f}_k$, $\mathbf{i}_k$ and $\mathbf{o}_k$ are the forget, input and output gates respectively. σ and tanh represent the sigmoid and the hyperbolic tangent activation functions and $\circ$ denotes the Hadamard product.

It can be noticed how all the gates ($\mathbf{f}_k$, $\mathbf{i}_k$ and $\mathbf{o}_k$) receive the input vector ($\mathbf{x}_k$) at the current time step and the previous hidden state ($\mathbf{h}_{k-1}$). However each gate has its particular set of parameter matrices ($\mathbf{W}$ and $\mathbf{U}$) and its own bias vector ($\mathbf{b}$). Thus, the number of training parameters highly increases with respect to the normal RNN. Firstly, $\mathbf{f}_k$ denotes the forget gate. This gate represents the amount of information that will be forgotten or vanished from the cell state at time step k. In addition, the input gate controls the extent of new data that is added to the cell state. In Eq. 2.3.1 $\mathbf{c}_{i_k}$ denotes the new update added to the cell state to acquire the new information at time state k. Finally, $\mathbf{c}_k$ is updated by the elimination of old memory, by multiplying the forget gate vector and the previous cell state, and the addition of the new insights acquired in $\mathbf{c}_{i_k}$.

The output gate is responsible of controlling the amount of data that is forwarded to the hidden state and thus the output value of the layer.

Chapter 3

Design Process

The aim of this chapter is the description of the steps followed in the design process of an ANN for the prediction of the power consumption of a heating, ventilation and air conditioning (HVAC) in a real and self sufficient solar house. Firstly, the different features contained in the data set will be described and visualized. In addition, the real house from where the data was measured will be introduced and referenced.

Secondly, a set of suitable data preprocessing and arrangement techniques will be proposed. The adaptation of the data set to feed a neural architecture is a crucial design stage that can significantly improve the performance of the system.

Finally, several ANNs are designed to make power consumption forecasts. One of the models will be designed in order to make single step predictions. In contrast, three alternative models will be dedicated to the multiple step ahead predictions. All the models are modification of a main neural architecture that generalizes all of them.

3.1 Data acquisition system

In order to develop the training process of the ANN through the back-propagation algorithm, a dataset, containing the power consumption records, and several inputs with a remarkable correlation with the consumption, is required. To assess this task, a dataset was extracted from a real solar house called MagicBox.

MagicBox is a self-sufficient solar house (see Fig. 3.1) located at the Escuela Técnica Superior de Ingenieros de Telecomunicación (ETSIT) of the Universidad Politécnica de Madrid (UPM). The house integrates sustainable elements based on renewable energies, self-sufficiency energetic methods, bioclimatic architecture and recycled construction materials. In addition, it includes Information and Communication Technologies to monitor and control the house power flow.

Figure 3.1: Frontal view of the MagicBox.

MagicBox was originally designed to participate in the Solar Decathlon 2005 contest, being the first house from an European university to take part in this event (see Caamaño-martín et al. (2005) and Calvo-Fernández et al. (2005)).

Multiple studies have been developed on the MagicBox. In Castillo-Cagigal et al. (2011c) an heterogeneous collaborative sensor network designed to manage the energy performance of the magicBox was described. In addition, Castillo-Cagigal et al. (2011b) presents the operation of a semi-distributed electrical demand-side management system with the PV generation in order to improve the self-consumption. Moreover, the optimization of the self-consumption in a system with the PV generation coupled to a battery energy storage system and connected to the grid was studied and tested in Solano et al. (2017).

In this Thesis, only the HVAC system and the sensors that measure the different variables were utilized. The consideration of other features of the MagicBox, such as the ones mentioned above, in combination with the designed and implemented neural predictor of the HVAC system's consumption, is left as future research work.

Due to the nature of the problem, the input data set is strongly related to the weather. On one hand, one considers outdoor measurement variables such as the outdoor temperature [ºC], the relative humidity [%], the irradiance $[\frac{W}{m^2}]$ and the $C0_2$ measure. These input variables will provide the machine learning system with an insight of the physical behaviour of the weather. On the other hand, two indoor variables are considered: the indoor temperature of the house, where the HVAC system is located, and the reference temperature set by the user to fix the desired indoor comfort temperature. In conclusion, the ANN will receive the 6 aforementioned

input variables and output the forecasted power consumption, which is contained in the data set as well. This output variable will be provided to the ANN in the format of target examples of the desired output values during the learning process, as stated in Section 2.2.

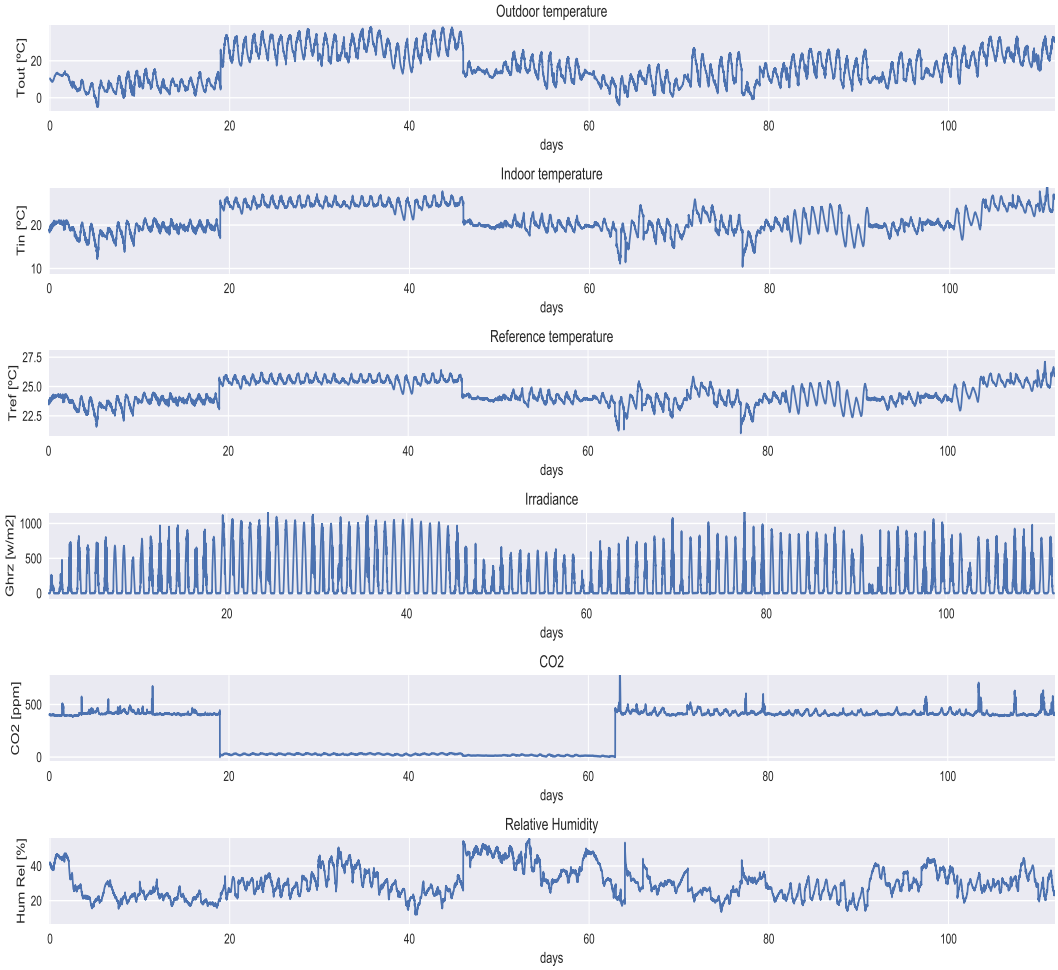


Figure 3.2: Representation of the different measured input features.

An important issue regarding the data set variables is the time dependency. The system will not explicitly receive any time variable. Due to the fact that RNNs are utilized, the temporal behaviour will be implicitly inferred by the model. In order to enable this insight to be acquired by the RNN, both input and targets should be ordered with an equal time difference between samples. Fig. 3.2 shows the different input time series. In contrast, the power consumption of the HVAC system can be visualized in Fig. 3.3. The data was measured from December 2nd 2016 to August 8th 2017, with 1 minute time separation between samples. This makes 7 time series, each with a total amount of 161281 samples.

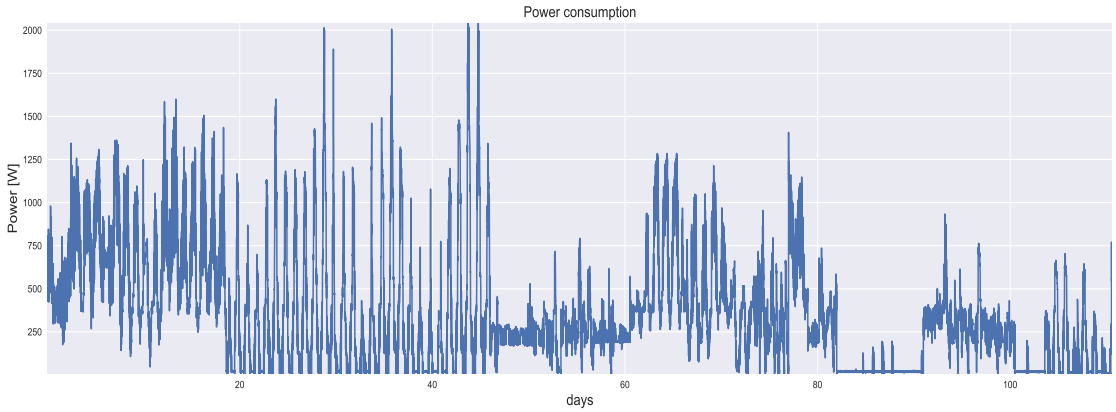


Figure 3.3: Representation of the power consumption of the HVAC system to be forecasted.

Observing the power graphic in Fig. 3.3, different behaviours of the power consumption can be noticed along the time axis depending on the season of the year. In addition, the power consumption data was fixed into multiples of 60 W during the measure process. This leads into a lost of resolution that cause a quantification error of this variable and the actual consumption.

3.2 Data preprocessing

The data, as mentioned in Section 3.1, was obtained from several sensors by an automated data acquisition system. This measured data is in a raw format. Thus, in order to train the RNN, an appropriate set of preprocessing techniques should be applied to the original data.

Data preprocessing consists of a selected set of transformations that provides a suitable conditioning of the raw data set such that the feeding of the RNN improves the training performance. In addition, the preprocessing transformations should be wisely selected depending on the problem to be solved and the learning system to be used.

In this section several preprocessing techniques are proposed and described in the frame of preparing time series data to feed an RNN. For this purpose, in the sequel, we will use the following notation. The RNN receives, as input, r time series ($r = 7$) of data, and generates, as output, s time series ($s = 1$); each time series is a vector of length ℓ (representing the number of samples). For $i \in \{1, \ldots, r + s\}$,

$$\mathbf{x}_i(k) = \begin{bmatrix} x_i(1) \\ \vdots \\ x_i(\ell) \end{bmatrix} \tag{3.2.1}$$

represents the corresponding time series, and k denotes the discrete time variable, with a sampling period of 1 minute. In Fig. 3.4 the reader may see a description of the data preprocessing process together with the data arrangement process to be described in the next section.

3.2.1 Normalization

The first considered preprocessing technique is the so called data normalization. Data set normalization consists of transforming the original range of the time series into a range between 0 and 1. More precisely, the preprocessing works as follows: let $\mathbf{x}_i^{\min}$ denote the minimal value that $\mathbf{x}_i(k)$ takes when the time k moves from 1 to ℓ. That is,

$$\mathbf{x}_i^{\min} = \min\{x_i(k) \,|\, k \in \{1, \dots, \ell\}\}.$$

Similarly,

$$\mathbf{x}_i^{\max} = \max\{x_i(k) \,|\, k \in \{1, \dots, \ell\}\}.$$

In this situation, the time series $\mathbf{x}_i(k)$ is normalized as

$$\widetilde{\mathbf{x}}_i(k) = \frac{\mathbf{x}_i(k) - \mathbf{x}_i^{\min}}{\mathbf{x}_i^{\max} - \mathbf{x}_i^{\min}} \tag{3.2.2}$$

Note that now the normalized time series has the interval $[0, 1]$ as range.

The use of normalization techniques is crucial in ANNs. Clear examples of neural networks using normalization are the LSTM. The underline intuitive reason for this is the fact that, in general, LSTM cells contain sigmoid or hyperbolic tangents activation functions (see Fig. 2.1) inside their multiple gates; see Section 2.3 for further details. Note that ranges of these activation functions are the intervals $(0, 1)$ and $(-1, 1)$, respectively. Furthermore, the interval $(0, 1)$ is clearly contained in the part of the corresponding function domains where the non-saturation behaviour occurs. On the other side, focusing on the particular problem treated here, HVAC power consumption data set original ranges, both of the features and targets, are very large compared with the range of the activation function (Fig. 2.1). Therefore, normalization is mandatory.

3.2.2 Data sampling

As described in Section 3.5, when making future predictions of multiple time steps, the original 1 minute period of the data sampling becomes unfeasible for the resources available. This is because with this original data time resolution, a one day (short term) forecast will require about 1440 neurons in the output layer of the first multi step model proposed in Section 3.5.2, and about 1440 input delays in the second multi-step architecture.

To overtake this difficulty, and hence in order to preserve the information of the data during the downsampling process, a data sampling technique is required. This process will consist essentially in two steps. First an increase of the time separation between samples, in the discrete time variable domain, is performed. Second, a suitable value for the time series is proposed. In this Thesis, an increase of 15 minutes is stated. The corresponding value of the new time series will be the mean of the values of $\mathbf{x}_i$ in the enlargement period, so that the information within the 15 minutes is not lost. More precisely, the details are as follows. First of all, let us say that we will consider the original discrete time variable k (see Eq. 3.2.1), which takes values from 1 to ℓ minutes, and a new time variable, namely k^*, which takes values from T to

nT minutes, where nT is the greater multiple of T being smaller than ℓ; in our case $T = 15$ minutes. So, $k = Tk^*$. Then, the new time series is defined as

$$\mathbf{z}_i(k^*) = \frac{1}{T} \sum_{k=k^*T}^{k=(k^*+1)T} \mathbf{x}_i(k) \tag{3.2.3}$$

The selected sampling period T in the multi step architectures was set to 15 minutes. This resolution allows to reduce the number of samples in the prediction of a day, from 1440 to 96.

3.3 Data arrangement

In this section, a subdivision of the input data set into subsets that are used in the performance and supervision of the machine learning process of the ANN is tackled. Let us assume that the input data set $\mathcal{A}_{\text{data}}$ has already been preprocessed by means of the techniques described in Section 3.2.

In the learning process two main parts can be distinguished: the training and the certification, or test, on how the network has been trained. Based on this, the data set $\mathcal{A}_{\text{data}}$ is decomposed into three subsets

$$\mathcal{A}_{\text{data}} = (\mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{val}}) \cup \mathcal{A}_{\text{test}},$$

where the first two subsets are devoted to the training and dynamical validation of the training, while the third subset is used to test the final result of the training process. In addition, due to the sequential nature of our problem, it is crucial that the mentioned subsets contain ordered data to preserve the temporal behaviour of the time series.

The training subset $\mathcal{A}_{\text{train}}$ is used to train the artificial neural network. Thus, this data set is provided to the back-propagation algorithm (see Section 2.2) to adjust the weights of the model in order to minimize the featured error. Usually, this first partition element has the largest proportion of data, among the three subsets, so that process has enough examples to train accurately.

During the training process, overfitting may occur. In this case, the trained system tends to learn and memorize only the samples provided during the training process. This leads to a poor performance when new examples are assessed and, in this situation, the ANN does not generalize properly. The aim of the subset $\mathcal{A}_{\text{val}}$ is mainly to prevent this phenomenon. The main idea is to assess the samples in the validation group during training to verify how the system behaves when non training examples are provided. The validation subset does not interact at all with the back-propagation algorithm, it only brings the designer an insight of the overfitting of the network so that hyperparameters can be wisely adjusted.

Once the training process has concluded, and the weights have been set to their final values, the system has to be tested. For this purpose, the test subset $\mathcal{A}_{\text{test}}$ is used. This subset contains data elements that has not been utilized during the training process of the ANN. In order to test the trained network, the elements in

$\mathcal{A}_{\text{test}}$ are provided to the RNN, and the results are compared with the corresponding known outputs.

In order to train and test the system, the size of the described subsets were fixed to 75% for the training subset $\mathcal{A}_{\text{train}}$, 5% for the validation subset $\mathcal{A}_{\text{val}}$, and 20% for testing the system. In Fig. 3.4 one may see a description of the whole data arrangement process.

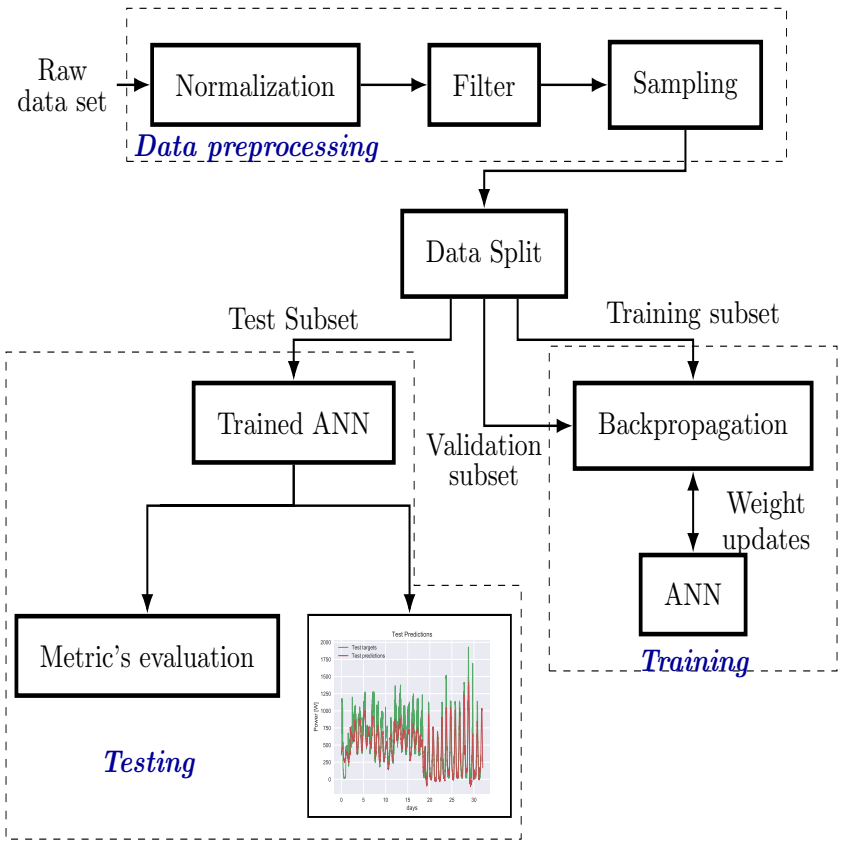


Figure 3.4: Diagram of the design process.

3.4 Neural architecture

This section is devoted to introduce the neural network used to forecast the consumption of the HVAC system. All the subsequence models described throughout this document are extensions, or modifications, of the architecture proposed here. Furthermore, all of them consists of recurrent neural networks as their main processing units.

In addition, initially the ANN was considered to forecast the consumption at only one step ahead. Nevertheless, due to the practical limitations of this model, an evolved architecture is also designed. This novel architecture exploits the single step forecast ANN to enable the prediction of multiple future time steps of the power

consumption. Section 3.5 extends the main model proposed here to implement the different prediction structures.

The consumption of HVAC systems, as well as the input variables, are highly influenced by multiple periodic behaviours. On one hand, outdoor temperature, solar irradiance and humidity have a notorious periodic variation depending on both the hour of the day and the month of the year. Thus, it can be seen as a daily periodicity with lower frequency variations attached. On the other hand, the human behaviour enforces temporal dependencies in other input variables such as the indoor reference set point and the actual indoor temperature. All the periodic input mentioned above make the output power consumption to behave periodically as well. Therefore, in order to take into account these strong temporal dependencies of the time series to be forecasted, a RNN is highly recommended.

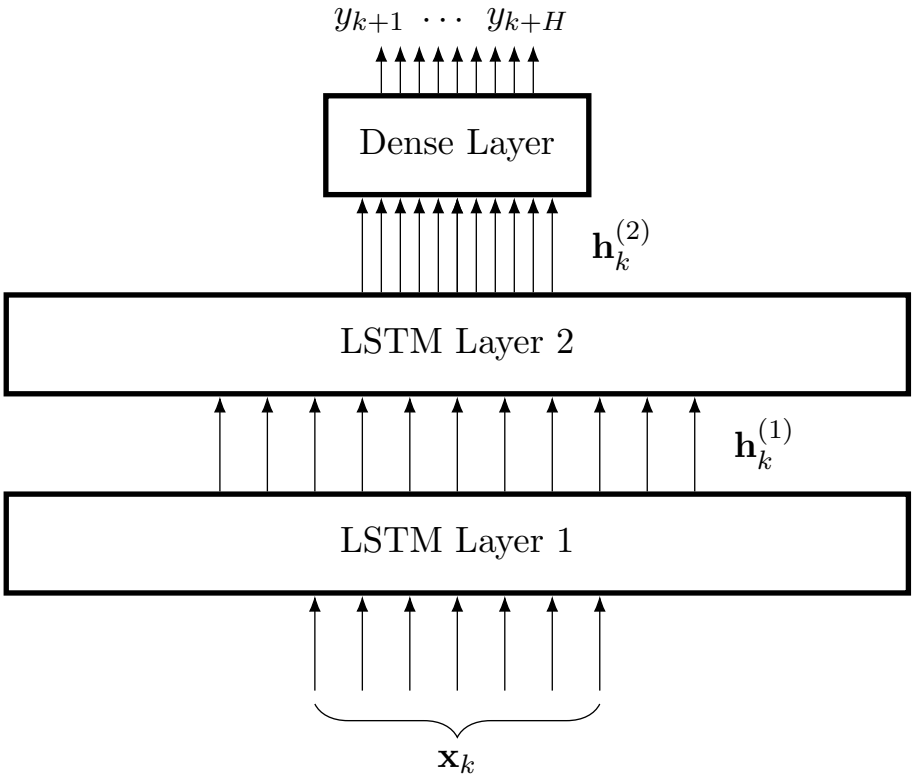


Figure 3.5: Main structure of the proposed model. LSTM layers are building blocks containing LSTM neural networks and dense layer represents a single layer perceptron.

Fig. 3.5 shows the main RNN used throughout this document. This generic model will be adapted and optimized along the remaining sections. Let H be the number of predictions of the network with $H = 1$ or $H = 96$ that corresponds to 1 day prediction after applying the data preprocessing techniques described above. It is formed by two LSTM layers (described in Section 2.3), with N_1 LSTM units in layer 1 and N_2 in layer 2 (the selection of a suitable value of N_1 and N_2 is studied in chapter 4), followed by a perceptron layer to output the corresponding prediction. The internal layers 1 and 2 consist of the LSTM computational graph described in Section 2.3 (see Fig.

2.4).

To be more precise, let $\mathbf{X}$ denote the matrix

$$\mathbf{X} = (\mathbf{x}_1(k), \ldots, \mathbf{x}_7(k)) = \begin{bmatrix} x_1(1) & \cdots & x_7(1) \\ \vdots & & \vdots \\ x_1(\ell) & \cdots & x_7(\ell) \end{bmatrix} \tag{3.4.1}$$

where $\mathbf{x}_i(k)$ is as in Eq. 3.2.1. Furthermore, let be denoted $\mathbf{x}_k$ as the k-th row of $\mathbf{X}$, that is

$$\mathbf{x}_k = \begin{bmatrix} x_1(k), \cdots, x_7(k) \end{bmatrix} \tag{3.4.2}$$

The first layer of the network receives $\mathbf{x}_k$ as input to generate $\mathbf{h}_k^{(1)}$ as output (see Eq. 2.3.1 and Fig. 2.4). $\mathbf{h}_k^{(1)}$ is a vector which length equals the number of units in the layer (N_1). This new vector $\mathbf{h}_k^{(1)}$ is then taken as input of the second layer. The second layer, using again the computational graph in Fig. 2.4, generates a new vector, $\mathbf{h}_k^{(2)}$, which length is the number N_2 of units of the second LSTM layer. Finally, the non-recurrent single layer gets $\mathbf{h}_k^{(2)}$ to derive the prediction values $y_{k+1}, \ldots, y_{k+H}$.

In the next section, we will see different modifications of the main model to obtain the predictions.

3.5 Prediction techniques

The main model, described in Section 3.4, predicts H future steps. When $H = 1$, the prediction problem is addressed by means of an artificial neural network that provides the next step in the time series. Experiments in Section 4.4 show that this approach has high performance results. However, under a realistic point of view, forecasting step by step, with short temporal sampling resolution, does not provide enough information. In order to overtake this disadvantage, different modified versions of the neural architecture for $H > 1$ are proposed. The main differences between the aforementioned prediction methods are the system's output shape, the sampling resolution, and the data inputs and targets arrangement used to feed the RNN during the training process.

More precisely, the next subsections describe how to extend the main model proposed in Section 3.4 to deal with the prediction problem; in Subsection 3.5.1 we approach the single step prediction, and in Subsection 3.5.2 the multi step architectures are presented. Both models are designed using unfolding techniques of the RNN so that a deeper understanding of the temporal behaviour is achieved.

3.5.1 Single step ahead prediction architecture

In this subsection, $H = 1$ is considered, i.e. the RNN predicts only one step ahead. In addition, let τ be the unfolding index used to unroll the recurrent part of the network (see Subsection 2.2). Thus, the network predicts the power consumption at time step $k + 1$ considering the τ previous instants. The number τ of past time steps depends on the sampling resolution of the data set. That is, τ is taken as

$$\tau = \frac{1440}{\text{resolution}} \tag{3.5.1}$$

where 1440 is the number of time steps in one day within the 1 minute original resolution of the data set. Therefore, a single step ahead predictor may be represented as (see Eq. 3.4.2):

$$\mathbf{y}_{k+1} = \hat{f}(\mathbf{x}_k, \mathbf{x}_{k-1}, \ldots, \mathbf{x}_{k-\tau}) \tag{3.5.2}$$

where $\hat{f}$ denotes the execution of the designed RNN.

Fig. 3.6 represents the unfolding of the model explained before. On one hand, the vertical arrows depict the data flow between layers at a fixed time step. On the other hand, the horizontal arrows mean the recursion among time steps. This recurrence is represented with the delivery of the hidden state and cell state to the next time step. Every time step can be understood as a new instance of the model with the corresponding hidden state ($\mathbf{h}_k^{(j)}$), cell state ($\mathbf{c}_k^{(j)}$), and input at that precise instant. The size of the unfold is equal to the number of considered past time steps.

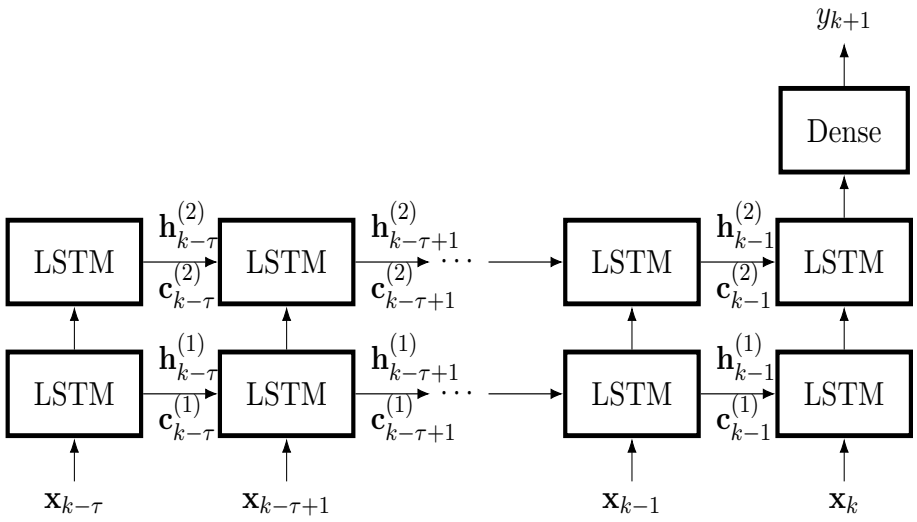


Figure 3.6: Unrolling of the single step prediction recurrent neural model in Fig. 3.5.

As a final comment, let us remark that, in the single prediction model, a new input is provided to the architecture at every time step. However, the single prediction is only made at the last time step (corresponding to the previous instant of the forecast). This is an important issue and one of the main differences with the multi step forecast treated in the Subsection 3.5.2.

3.5.2 Multi step ahead prediction architectures

The main aim of the ANN of this Thesis is to make short term multi step predictions of the consumption. More precisely, the model will take the previous day in order to forecast the next day. The following formula describes the main behaviour of multi step prediction:

$$[y_{k+1}, y_{k+2}, \ldots, y_{k+H}] = \hat{f}(\mathbf{x}(k), \mathbf{x}(k-1), \ldots, \mathbf{x}(k-\tau)) \tag{3.5.3}$$

This means that, in contrast with Eq. 3.5.2, the system's output is a vector with the predictions of the next day.

There are several techniques to achieve the multi step predictions. In this document we propose three different methods.

First multi step prediction approach: MSPM-1

The first approach, refered to as Multi Step Prediction Model 1 (MSPM-1), is to forecast the entire following day at once. All the predictions will be made at the last time step by providing the output perceptron layer with enough neurons. This model will require H neurons in the output layer, where H is the number of future predictions to be forecasted. Thus each neuron in this non-recurrent layer is responsible of the forecasting of a single time step prediction. The day ahead prediction is formed by the union of all neuron's output.

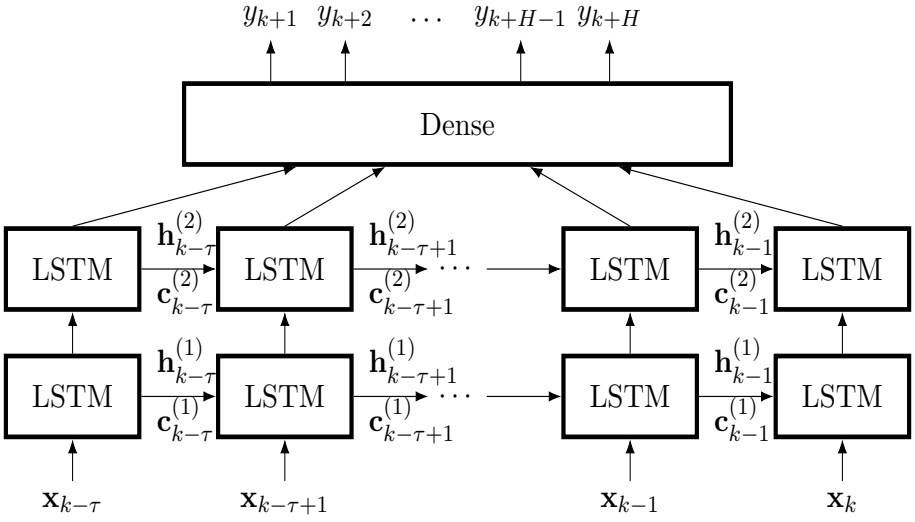


Figure 3.7: Unrolling of the MSPM-1

Fig. 3.7 shows the unrolling of the MSPM-1. This structure is highly similar to the one displayed in Fig. 3.6. The main difference is the manner in which the model makes the predictions. In both figures the predictions are made at the last time step, once all the input sequence has been provided. In the former, the input of the perceptron layer is only the hidden state of the second LSTM layer at the last time step, fixing all the sequence information in a single vector. However, in MSPM-1, the perceptron layer is fed with the hidden states of all the past time steps providing the forecast of the next day on the same instant. A last remark on this first multiple step prediction approach is that it does not impose the length τ of the input sequence, and the length H of the output sequence, or number of time steps to forecast, to be equal.

Second multi step prediction approach: MSPM-2

The second model, proposed makes the short term predictions with a noticeable different method. This approach does not wait for the whole input sequence to be provided in order to make the predictions. It outputs the estimations of the consumed power in a continuous data flow at the same time the new inputs are fed.

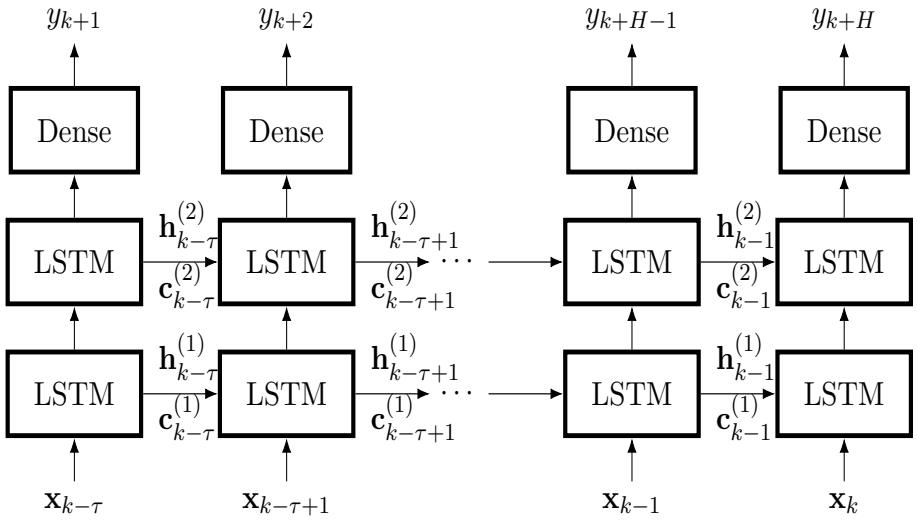


Figure 3.8: Unrolling of the MSPM-2

Fig. 3.8 illustrates the MSPM-2. In this architecture the output perceptron layer consists of a single neuron. The contrast with the former structures resides in the fact that the model makes a prediction per time step starting when the first input individual at time $t - \tau$ is provided. At the last instant, when the last element of the input sequence is fed, all the predictions of the next day are made. In addition, the same perceptron layer (with the same weights) is applied at every time step being provided with the corresponding outputs of the LSTM layer.

This second approach has some limitations that will be mostly overtaken in the last proposed multiple step prediction method. The first constraint of this structure is, as mention above, that the input sequence length (τ) and the forecasted sequence (H) must be the same. This is caused by the continuous flow of the output data. In addition, because predictions are made at the same time that inputs are provided, the outputs only take into account the input at previous instants. This causes a worse performance in early time steps due to the lack of enough input elements of the sequence to process. This issue leads to the second constraint of the MSPM-2.

Third multi step prediction approach: MSPM-3

Finally, in order to solve the problem described above, MSPM-3 is proposed. This new approach is based on the artificial neural network called encoder–decoder

neural network proposed in Cho et al. (2014b), as well as on the sequence–to–sequence neural network described in Sutskever et al. (2014). The architecture contains two basic recurrent layers: the encoder or reader and the decoder or writer. The encoder is responsible of transforming an input sequence of data into a fixed-length vector representing the whole sequence, namely the context($\mathbf{c}$). In contrast, the decoder receives the context produced by the encoder and converts it into the decoded sequence, which is indeed the output sequence.

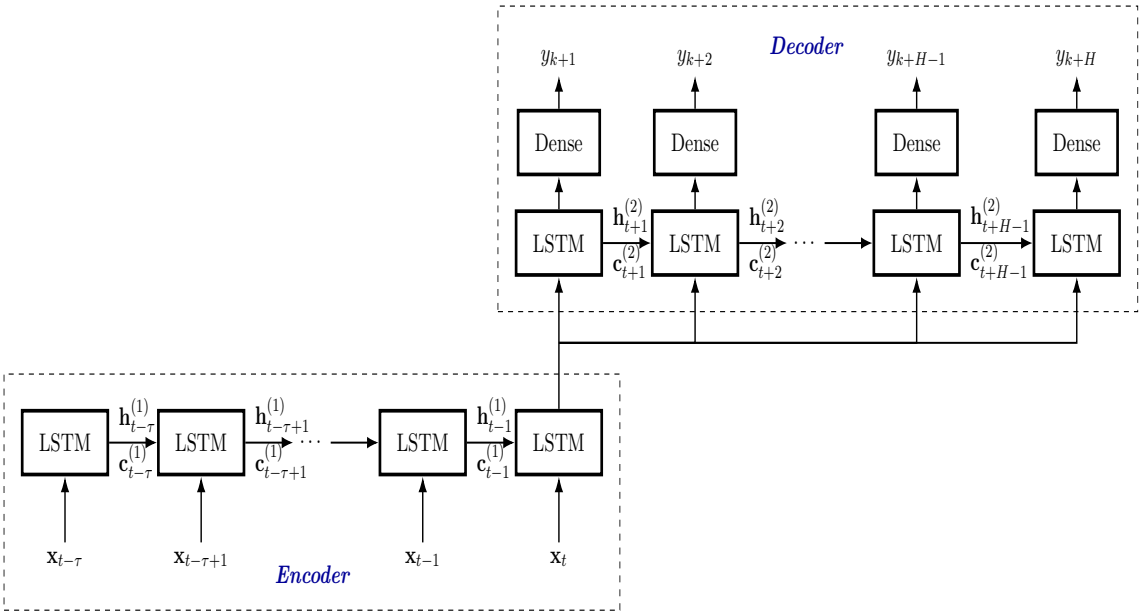


Figure 3.9: Unrolling of the MSPM-3.

Fig. 3.9 represents the architecture of the unfolded encoder–decoder neural network. It can be noticed that in the proposed MSPM-3 both encoder and decoder are LSTM layers. The feedforward dense layer adapts the decoded sequence to generate the actual prediction. In addition, the MSPM-3 can be understood as a hybrid architecture of the single step prediction model and MSPM-2.

The encoder–decoder architecture is mainly applied to the field of machine translation, such as in Cho et al. (2014a), where the encoder receives a sequence of words in a language $\mathcal{A}$ and the decoder returns the translated sequence from language $\mathcal{A}$ to a second language. However, in order to solve multi step ahead time series prediction problem, the model has to be adapted and some modifications need to be made. Firstly, for the data set treated in this project, the context vector does not provide enough resolution to maintain the relations between the input features and the power predictions. Aiming to solve this issue, in this Thesis, the feeding of the input sequence to the decoder as well (jointly with the context) is considered. This modification was proposed in Rahman et al. (2018) when applied to an energy prediction problem.

Chapter 4

Implementation and Verification

This chapter is devoted to describe the different steps followed in the implementation of the models designed in Chapter 3. In addition, the assessment of the architectures is developed, comparing the performance provided by each of them.

Firstly, the utilized implementation tools and programming environment is introduced. Moreover, the reader can consult the corresponding section in order to acquire some basic insights of the operation and characteristics of the libraries used.

Afterwards, several metrics, aiming to measure the performance of the models, are stated. Those metrics are used in the last section of the chapter to verify the proper behaviour of the final systems. Before the models are finally assessed, an empirical procedure to select the most suitable parameters of the MSPMs is carried out. This process will be applied to the number of layers, the number of units, the learning rate parameters, and the learning optimizer to be used.

To conclude, the final models with the previously selected parameters are tested in the frame of visualizing its time series predictions and computing the metrics of those output forecasts. The MSPMs' output values are compared concluding the most suitable ones in a real application.

4.1 Implementation environment and tools

In this section we introduce the tools used during the implementation and verification of the RNNs described in Chapter 3. The models were mainly implemented and assessed in the python framework called Keras. Although Keras was mostly used, a lower level Python's framework called Tensorflow, was studied and analysed as a complementary implementation tool. These libraries enable the possibility to run their programs on Graphical Processing Units (GPU) with the CUDA parallel computing. Indeed, all the code developed during this research was run on a GPU, providing a huge enhancement of the execution time. More precisely, both Keras and TensorFlow are open source Python's machine learning frameworks for the building, training and testing of ANNs.

On one hand, TensorFlow uses the programming paradigm of computational graphs. A computational graph is the set of nodes, inputs and outputs and data flow

that describe the behaviour of an algorithm. The nodes represent the mathematical operations to be computed, inputs are the input values of the algorithm or expression, and outputs are the result of applying the corresponding computations. TensorFlow's computational graphs are created in an static way (in contrast with the dynamic computational graphs) where the nodes and data flow are generated with generic inputs called placeholders. Then, the user can execute the computational graph by creating a session and running the graph with an assignation of the defined input placeholders. This approach enables the running of the same graph with different inputs in the same session.

On the other hand, Keras is a higher level library that can run on top of the lower lever frameworks TensorFlow, Theano or CNTK (being TensorFlow the one used in this Thesis). Keras enables a programming environment with a friendly, modular and scalable creation of ANNs. The Keras's models can be generated in two different approaches: the sequential or the functional models. The two of them facilitate an alternative manner to build the systems before they are compiled and trained. The sequential one stands out for the simplicity of the models. In contrast, the functional approach can be used for the creation of more complex ANNs unreachable with the sequential model. Both possibilities are briefly analysed below:

- Sequential models. They provide to the designer an intuitive implementation tool of the ANNs. The models are created by adding layers sequentially to the structure and compiling it when the architecture is completed. Listing 4.1 shows the implementation of a multilayer perceptron with 2 layers, 10 inputs and 4 outputs. One can observe that the compilation method also establishes the loss metric (see Eq. 2.2.3) and the backpropagation optimizer to be used (optimizers will be described in Section 4.3.2). The fit method is utilized to perform the training of the ANN with the specified training parameters.

Listing 4.1: Sequential model example of a MLP with Keras

```
model.Sequential()
model.add(Dense(8, input_shape=(10,)))
model.add(Activation('tanh'))
model.add(Dense(4))
model.add(Activation('tanh'))
model.compile(loss = 'mse', optimizer = 'adam')

model.fit(X_train, Y_train, nb_epoch=50, batch_size=16)
```

- Functional models. They enable the creation of more complex ANNs. Some examples of features allowed by the functional model are for instance the multi input and multi output neural networks, parallel layers that can be merged and shared layers. In contrast with the sequential model each layer receives as argument the output of the previous layers (the input is considered a layer in this approach). Finally the inputs and outputs of the system that will be used in the machine learning algorithm must be specified. The implementation of a MLP neural network can be checked in Listing 4.2.

Listing 4.2: Functional model example of a MLP with Keras

```
mlp_input = Input(shape = (10,))
layer1 = Dense(8, activation='tanh')(mlp_input)
layer2 = Dense(4, activation='tanh')(layer1)
model = Model(inputs=[mlp_input], outputs=[layer2])
model.compile(loss = 'mse', optimizer = 'adam')

model.fit(X_train, Y_train, nb_epoch=50, batch_size=16)
```

4.2 Metrics

In order to evaluate the performance of the model after the training process is applied, several metrics should be selected. However, the suitable metrics vary depending on the application. Due to the fact that time series prediction is the aim of this research, metrics based on the numerical error and correlation computation are applied. Thus, this section describes the metrics used to assess the fitness of the model in Section 4.4.

The first metric considered is the mean square error (MSE). Equation 4.2.1 shows the formula for computing this metric,

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{4.2.1}$$

where y and $\hat{y}$ denote the original and forecasted time series respectively, and N stands for the number of samples used for the computation. In addition, the MSE metric is also used as the error metric in the backpropagation algorithm in order to update the weights.

The next error metric is the root mean square error (RMSE) which is represented by Equation (4.2.2).

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2} \tag{4.2.2}$$

The former metrics are displayed in the same units as the time series to be evaluated. In order to measure the performance independently of the time series units, the NRMSE metric is proposed. The NRMSE is described in Eq. 4.2.3, where $y_{\max}$ is the maximum value of the target time series along the temporal axis (k).

$$\text{NRMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\hat{y}_i - y_i}{y_{\max}}\right)^2} \tag{4.2.3}$$

Finally, a metric that aims to compute the correlation between the two time series is proposed. This task is performed by the Pearson correlation coefficient ($\rho_{y,\hat{y}}$) between the target and predicted time series. Eq. 4.2.4 shows the computation of $\rho_{y,\hat{y}}$.

$$\rho_{y,\hat{y}} = \frac{E[(y - \mu_y)(\hat{y} - \mu_{\hat{y}})]}{\sigma_y \sigma_{\hat{y}}} \tag{4.2.4}$$

In this formula, $E[(y-\mu_y)(\hat{y}-\mu_{\hat{y}})]$ is the covariance between y and $\hat{y}$ and σ_y, $\sigma_{\hat{y}}$ are the standard deviations of y and $\hat{y}$ respectively.

4.3 Parameter optimization

This section deals with the justification, by means of an empirical analysis and comparison, of the selection of some of the most relevant parameters involved in the design of the ANN and the machine learning algorithm. The strategy to achieve this goal is as follows: for different values of the parameter under analysis, one iterates, builds and trains the corresponding model as well as assesses its performance. One may consider a more automated mechanism to successfully perform the parameter tuning, for instance by means of another ANN. Nevertheless, this is left as future research.

The parameters that are analyzed in this section are the number of LSTM layers, the number of units in those layers in the frame of the neural structure. In addition, the learning optimizer and the learning rate α (see Equation 2.2.1) are analysed.

As a comparing tool, a boxplot representation is used. For this purpose, the parameter to be optimized is depicted along the x axis, while the distribution of the error vector

$$\mathbf{e} = (|y_1 - \hat{y}_1|, \ldots, |y_\ell - \hat{y}_\ell|) \tag{4.3.1}$$

is represented in the y axis.

4.3.1 Number of LSTM layers and units

The first sort of parameters involved in the tuning are those related with the structure of the model. More precisely, the number of layers and the number of units per layer in the MSPMs. These two variables define the structure and computational capacity of the architecture. Selecting an insufficient number of layers or units leads to a poor performance. However, a large number of those parameters could produce an overfitted trained system. In addition, the more layers and units the model has the larger the execution time is.

We start analyzing the number of layers. For this purpose, the remaining parameters have been fixed to 50 epochs, adam optimizer (treated further in this section) and $\alpha = 0.001$. In addition, it should be noticed that MSPM-3 needs a minimum of 2 LSTM layers, and hence for one layer the method is not considered.

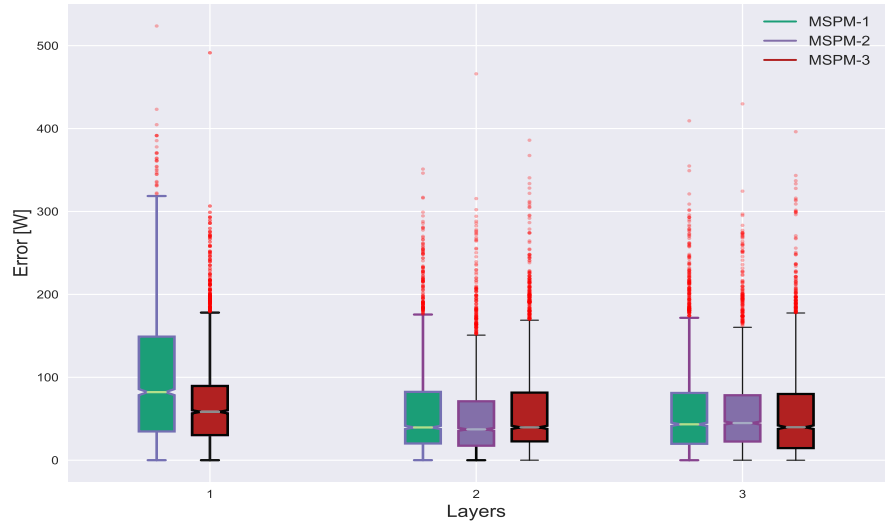


Figure 4.1: Comparison of models: number of layers. Each boxplot comprises observations ranging from the first to the third quartile. The median is indicated by a horizontal bar, dividing the box into the upper and lower parts. The whiskers extend to the farthest data points that are within 1.5 times the interquartile range. Outliers are shown with a plus symbol.

Fig. 4.1 compares the impact that the number of layers produces when using the different MSPMs. From the empirical results shown in the figure, one observes that all treated models' behaviour with 2 and 3 layers is similar, and therefore we conclude that, for simplicity, 2 layers is the best selection.

In order to analyze the number of units per layer, the models were built with two LSTM layers, and the same training parameters as in the previous study of the number of layers. Note that in the MSPM-3, the number of units is equal for all layers, since the the encoder and decoder hidden states and cell states must have the same length. Therefore, in the analysis performed here the same number of units for all the layers is considered. The results of the experiment appears in Fig. 4.2.

One can observe, in Fig. 4.2, that for MSPM-1 at least 20 units are needed in order to provide the RNN with enough computational power. In addition, for large number of units, the performance starts to decrease. This issue is probably caused by the overfitting phenomenon. The best choice of the number of units for MSPM-1 seems to be 32 units, giving the smallest error among the depicted experiments. In the case of MSPM-2 and MSPM-3 a general improvement in performance can be observed, being superior in the case of MSPM-3. It is also noticeable that these two models saturate their performance instead of decreasing it for large number of units. The best performance among the represented experiments of MSPM-2 is 50 units. Finally, the number of units for MSPM-3 will be fixed to 25 despite of the similar performance for higher values. This criterion provides the less expensive model in terms of computational power and GPU memory consumption.

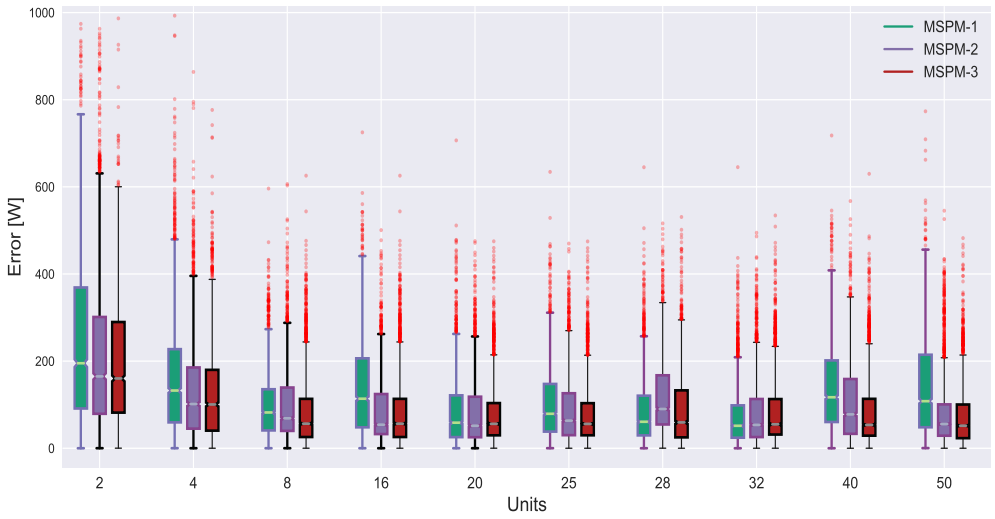


Figure 4.2: Comparison of models: number of units per layer

4.3.2 Learning algorithm optimizer and learning rate

Learning algorithm optimizers are variations of the original backpropagation algorithm (see Chapter 2). In this Thesis, the considered optimizers are the Stochastic gradient descent (SGD), the RMSprop optimizer and the Adam optimizer.

The SGD optimizer is a basic form of the backpropagation algorithm in which a gradient descent is performed with every sample of the data set. Thus, SGD optimizer can provide an online training of the ANN by updating the weights with every training example. This algorithm has many limitations and is exceeded by other optimizers in many applications. Some of the disadvantages of this optimizer are its noticeable fluctuations and the excessive training time to reach the minimum.

The RMSprop optimizer Hinton (2012) proposes a variation of the mini–batch backpropagation method with an adaptive learning rate. The basic idea is the modification of Equation 2.2.1 as follows, where for simplicity the gradient is denoted as $\mathcal{G}_\theta(r) = \nabla_{\mathbf{\Theta}}\mathcal{L}(\mathbf{\Theta}(r))$,

$$\left.\begin{array}{rcl} \mathbf{v}(r) & = & \beta \cdot \mathbf{v}(r-1) + (1-\beta)\cdot(\mathcal{G}_\theta(r) \odot \mathcal{G}_\theta(r)) \\ \mathbf{\Theta}(r+1) & = & \mathbf{\Theta}(r) + \dfrac{\alpha}{\sqrt{\mathbf{v}(r)+\epsilon}}\mathcal{G}_\theta(r) \end{array}\right\}. \tag{4.3.2}$$

Usually β is taken as 0.9 as stated by the author.

Finally, the last optimizer taken into account is the Adam optimizer proposed in Kingma and Ba (2014). Adam algorithm also consists in a modification of Equation 2.2.1 with an adaptive learning rate. It introduces the following adaptive α:

$$\mathbf{\Theta}(r+1) = \mathbf{\Theta}(r) - \alpha\frac{\hat{\mathbf{m}}(r)}{\sqrt{\hat{\mathbf{v}}(r)}+\epsilon} \tag{4.3.3}$$

where $\hat{\mathbf{m}}(r)$ and $\hat{\mathbf{v}}(r)$ are the bias-corrected first and second moment estimates. $\hat{\mathbf{m}}(r)$

and $\hat{\mathbf{v}}(r)$ are computed as

$$\left.\begin{array}{rcl} \hat{\mathbf{m}}(r) & = & \dfrac{\mathbf{m}(r)}{1-\beta_1^r} \\ \hat{\mathbf{v}}(r) & = & \dfrac{\mathbf{v}(r)}{1-\beta_2^r} \\ \mathbf{m}(r) & = & \beta_1 \cdot \mathbf{m}(r-1) + (1-\beta_1) \cdot \mathcal{G}_\theta(r) \\ \mathbf{v}(r) & = & \beta_2 \cdot \mathbf{v}(r-1) + (1-\beta_2) \cdot (\mathcal{G}_\theta(r) \odot \mathcal{G}_\theta(r)) \end{array}\right\} \tag{4.3.4}$$

The decaying value of $\hat{\mathbf{m}}(r)$ and $\hat{\mathbf{v}}(r)$ with the increase of the iteration index r can be observed. In addition, the values of β_1 and β_2 were fixed to 0.9 and 0.999 respectively as proposed in the paper.

Fig. 4.3 depicts the boxplot graphs of the error between the different models and the mentioned optimizers. The performance of the models is quite dependent on the optimizer used. The SGD optimizer behaves poorly for MSPM-1 and MSPM-2. However, this optimizer fits remarkably better with MSPM-3. It can be observed how the Adam method provides the best performance for all the models. Thus it has been chosen as optimizer for the architectures.

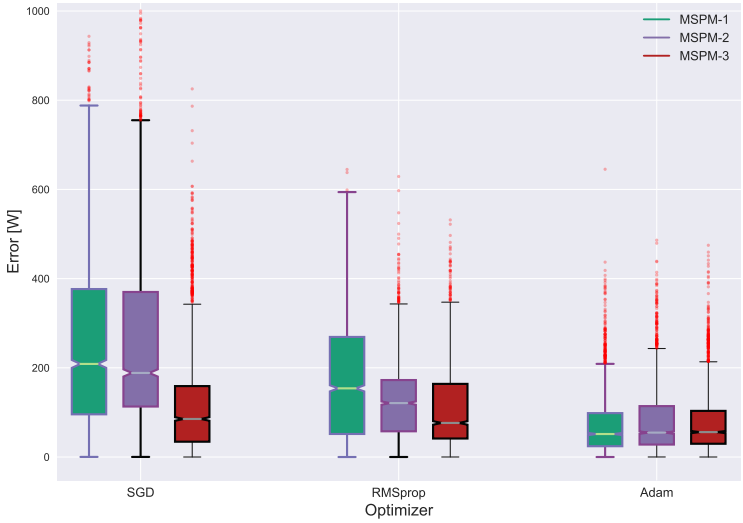


Figure 4.3: Comparison of learning algorithm optimizers

The last part of this section deals with the analysis of the learning rate α. As explained in Chapter 2, the learning rate establishes the proportion of the gradient's module that is utilized to perform the gradient descent. Thus, a low learning rate could produce a slow learning with small steps in the gradient descent, and hence a longer execution time would be needed. In contrast, if a large value of α is fixed, then the algorithm would probably diverge from the solution. However, one can not totally depend on this knowledge because of the fact that the treated problem has a great impact on the behaviour of the system. For this reason, an empirical verification is needed.

Fig. 4.4 represents the comparison between the models with different learning rates variations. It is appreciated how low learning rates produce a poor performance

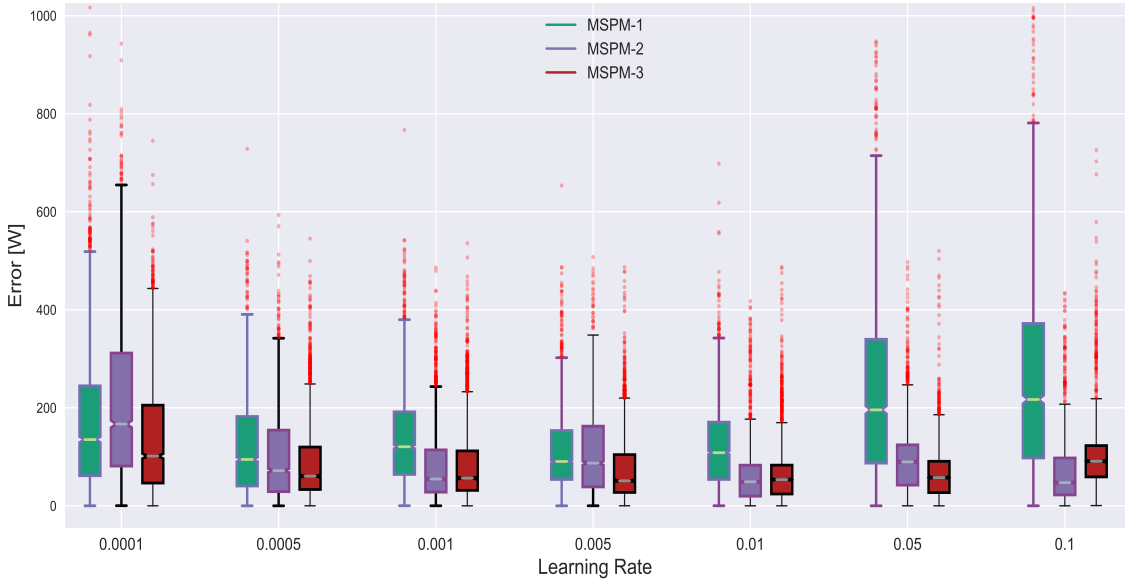


Figure 4.4: Comparison of the learning rate.

in the models. However the results are improved with the increment of α, highlighting the values of α=0.005 for MSPM-1, α=0.01 in the case of MSPM-2 and MSPM-3 (with most of the individuals with remarkably low error). For large values, the model's errors become unfeasible with very large errors. In the case of MSPM-1 this decline can be observed in the figure and for MSPM-2 and MSPM-3 it starts to occur for values of α around 0.5. However it was not represented due to the fact that the boxes would be mess the figure's visualization.

To sum up the selections carried out in this Section, Table 4.1 collects the chosen parameters and optimizer for the MSPMs.

Table 4.1: Summary of the selected parameters for the MSMPs.

Model	Layers	Units	Optimizer	α
MSPM-1	2	32	Adam	0.005
MSPM-2	2	50	Adam	0.01
MSPM-3	2	25	Adam	0.01

4.4 Results

For each of the models, several figures will be displayed to illustrate the behaviour of the model's predictions. In addition, the correlation between the actual and the

predicted time series will be also shown for each of the architectures. The evaluation of the metrics, described in Section 4.2, applied to all the models is arranged in Table 4.2. In this table, the values of the MSE metric are displayed in their normalized version. This is because this metric is mainly applied after the normalization preprocessing is done. Moreover it is the metric used as the loss function because it needs the data between 0 and 1. In addition to the MSE, the RMSE, NRMSE and Pearson correlation coefficient are exposed. These metrics are applied to the different models and under the training and test datasets.

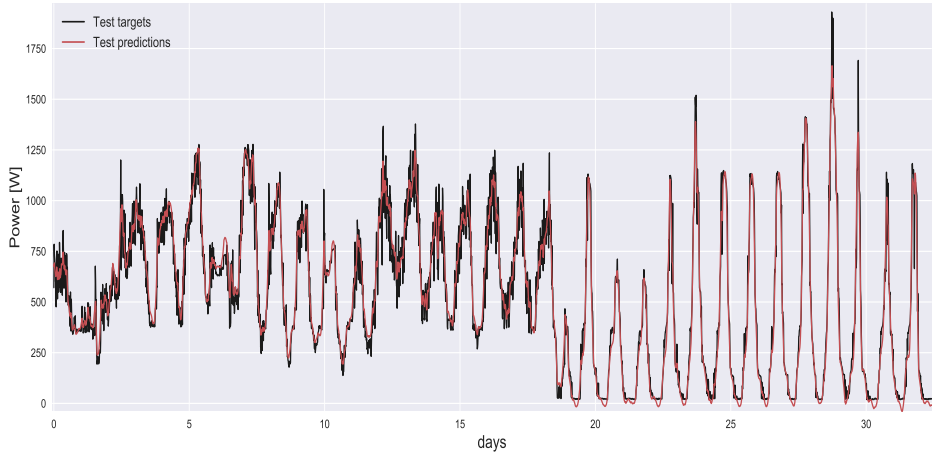


Figure 4.5: Assessment the single step ahead predictor when test data set is provided.

Firstly, the single step ahead predictor is assessed. Fig. 4.5 graphically shows the behaviour of this model's output values when the test subset is provided: the black time series represents the targets or measured power consumption, and the red time series depicts the model's predictions. In this figure, the strong fitting of the prediction with the real time series can be appreciated, resulting to be a high performance solution. The result of applying the metrics to this model is shown in Table 4.2. It can be observed that the NRMSE values are very low in the training as well as in test, indeed they are the lowest. Furthermore the Pearson correlation coefficient is almost, again for both cases, maximal.

As introduced in Section 3.5, the architecture designed to forecast the next time step proves to have a high performance. However, this model should just be understood as a first approach of the problem because of the lack of practical use in real time forecasting. This leads us to the multiple step forecast architectures. The plotting of the temporal response of these models is displayed below. Moreover these models have been built and trained with the parameters and optimizer selected in Section 4.3 and displayed in Table 4.1.

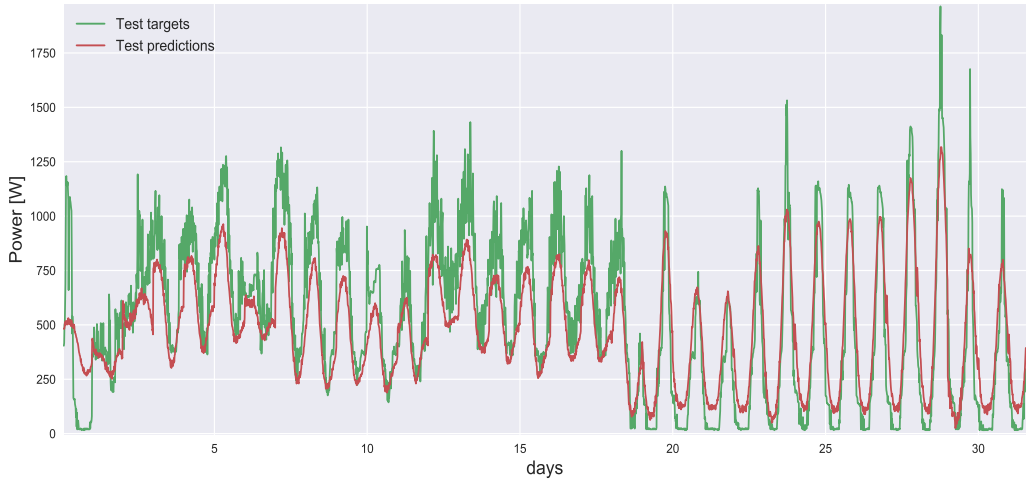


Figure 4.6: Assessment of MSPM-1 when test data set is provided

Fig. 4.6 introduces the response of MSPM-1 when the test data set is provided. One can highlight the decrease in performance of this model by observing Table 4.2. More precisely, although the errors and the correlation coefficients of MSPM-1 are good, in comparison to the other considered models, they are in a second level of optimality. Indeed, NRMSE is around 0.054 and 0.075 respectively, and $\rho_{y,\hat{y}}$ is around 0.9, which is closed to the upper bound.

However, it should be noticed that the MSPMs forecast the whole next day of power consumption, which leads into a much more complex task to solve. Thus, with perspective, this model excels in making the forecasts. Although this first approach to forecast multiple steps successfully accomplishes its prediction task, the remaining models remarkably overpass its performance.

Figure 4.7: Assessment of MSPM-2 when test data set is provided

In addition to the MSPM-1, the output time series of the predictions made by the MSPM-2 are depicted in Fig. 4.7. One can easily appreciate the enhancement in terms of fitting between the time series. Although its output seems to have more attached noise than the previous model, it is able to adapt to higher frequency variations of the target time series.

Finally, Fig. 4.8 represents the test predictions made by the MSPM-3 and the test targets.

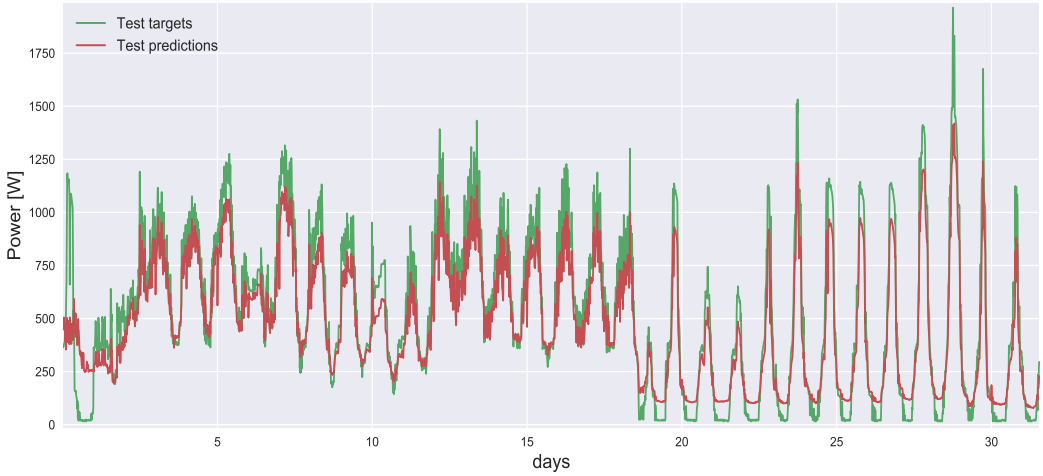


Figure 4.8: Assessment of MSPM-3 (Encoder–decoder) when test data set is provided

In addition, the resulting metrics of this MSPM-2 and MSPM-3 can be consulted in the corresponding entries of Table 4.2. These models obtain lower values of the errors compared with MSMP-1. However the single step ahead predictor reaches better metrics. As a comparison between MSPM-2 and MSPM-3, the latter presents a slightly improvement compared with MSPM-2. However, because the difference would be minimum, in a real application it would be preferable to implement MSPM-2 because of its shorter processing time of a sequence (see Chapter 3).

	Training				Test			
Model	MSE	RMSE[W]	NRMSE	$\rho_{y,\hat{y}}$	MSE	RMSE[W]	NRMSE	$\rho_{y,\hat{y}}$
OSPM	$3.61 \cdot 10^{-4}$	33.25	0.019	0.99	$8.41 \cdot 10^{-4}$	50.75	0.029	0.98
MSPM-1	$2.9 \cdot 10^{-3}$	94.5	0.0539	0.92	$5.56 \cdot 10^{-3}$	131	0.0747	0.89
MSPM-2	$2.26 \cdot 10^{-3}$	84	0.048	0.94	$2.7 \cdot 10^{-3}$	91	0.051	0.97
MSPM-3	$2.12 \cdot 10^{-3}$	78.75	0.045	0.94	$2.51 \cdot 10^{-3}$	85.75	0.049	0.97

Table 4.2: Collection of all the model's metrics for training and test data set.

To conclude with the verification of the models, a comparison between the aforementioned output time series is described. Fig. 4.9 depicts the correlation between the predicted and the real time series for all the treated models. At the top of the figures it can be observed the power consumption distribution of the target time series and on the right side the distribution of the predicted power consumption. Moreover, the figures show the value of $\rho_{y,\hat{y}}$ that equals the ones stated in Table 4.2. It can be pointed out the high correlation between the model's predictions and the targets. Although the OSPM showed a higher performance above in this section, it does not noticeably stand out from the MSPM-2 and 3 in terms of correlation. Furthermore, there is a high linearity of the depicted graphs.

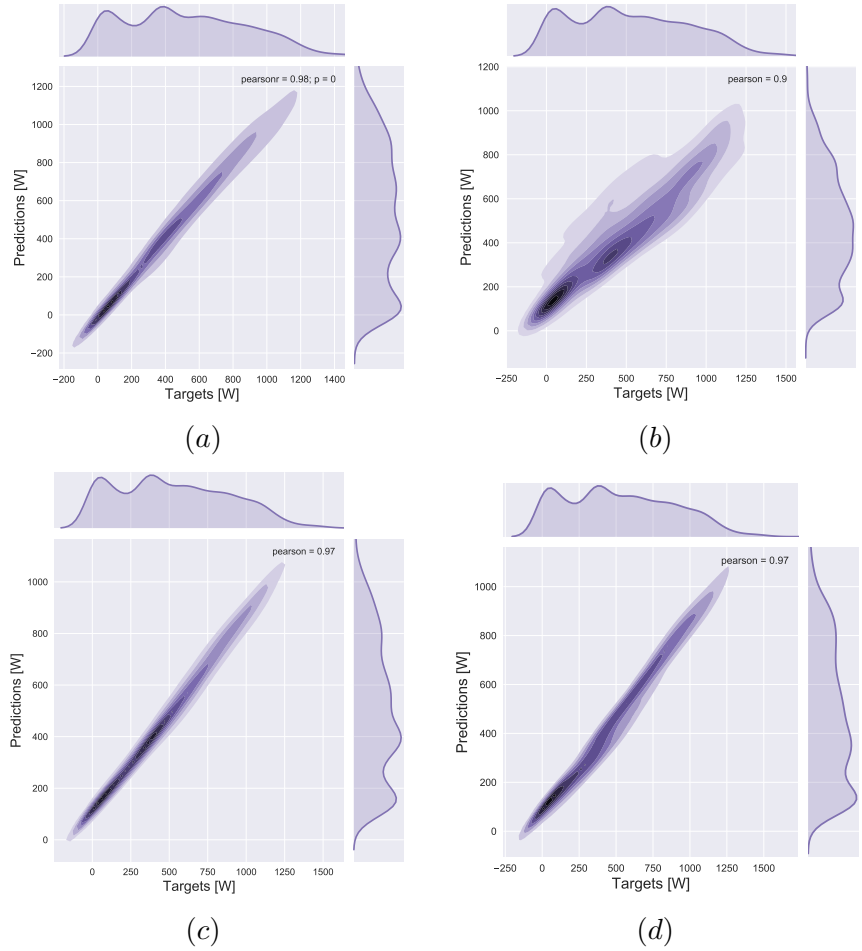
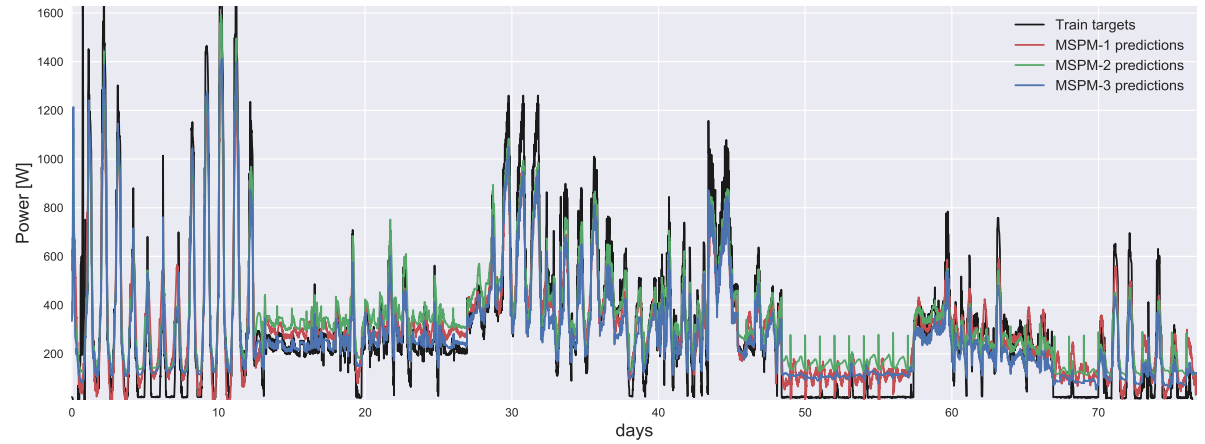
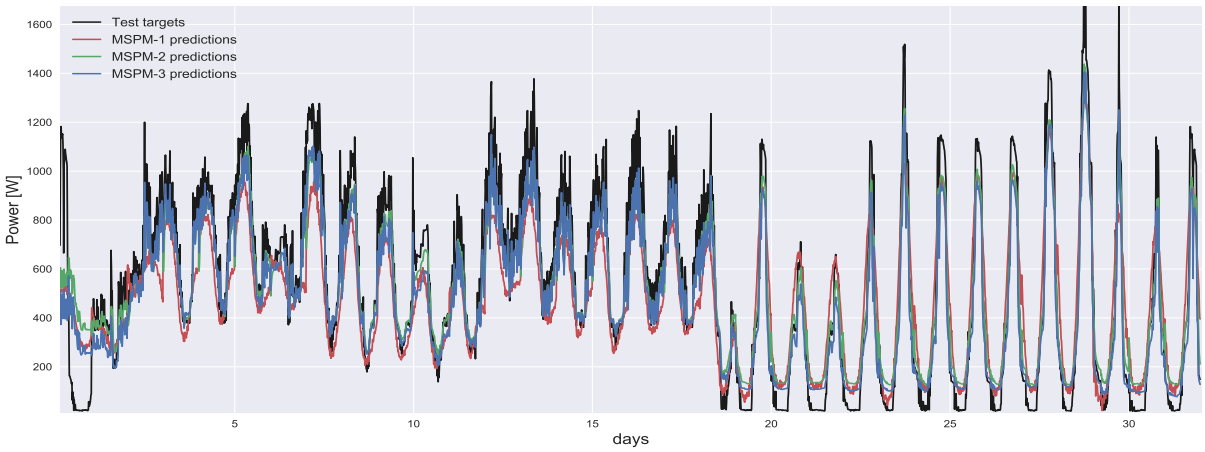


Figure 4.9: Representation of the correlation between the model's predictions and the target time series. (a) OSPM's predictions. (b) MSPM-1's predictions. (c) MSPM-2's predictions. (d) MSPM-3's predictions.

Fig. 4.10 a represents the output predictions of all the multi step models and Fig. 4.10 b the target time series for training and testing data sets respectively in a single graph.



(a)



(b)

Figure 4.10: (a) Comparison of the predictions made by the models when the training data set is provided. (b) Comparison of the predictions made by the models when the test data set is provided.

It can be observed how MSPM-2 and MSPM-3 provides, in general, a similar output in terms of fitting during the test experiment. However, when the train subset is utilized, the MSPM-2 decreased the performance in some intervals. In addition, the models significantly reduce their efficiency when a sequence of zero values are provided.

As a final comparison, it can be observed in Fig. 4.11 the predictions of the models paying attention to a 4 day window to deeply appreciate the fitting of the time series. As an observation to point out, the time series of the figure shows some discontinuities along their predictions. This issue was intentionally produced to reinforce the fact that the models predict the whole next day (depicted within the mentioned discontinuities).

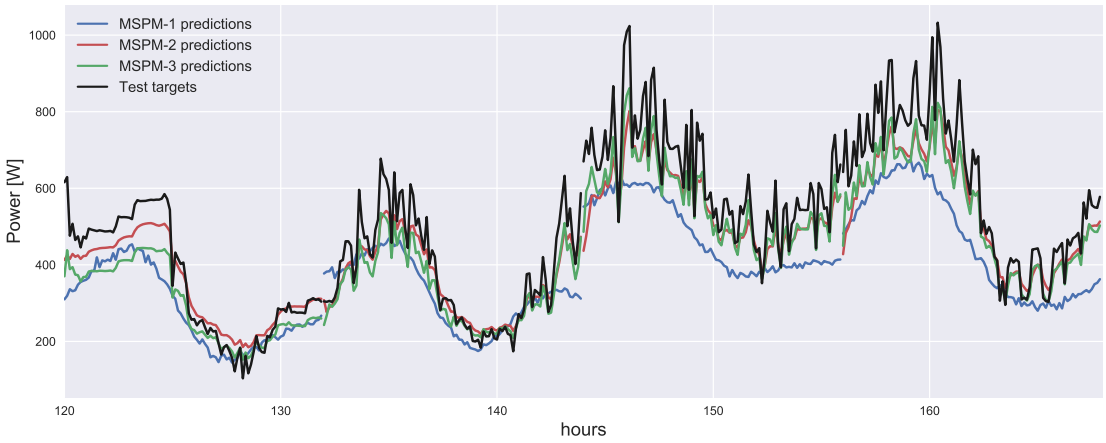


Figure 4.11: Predictions of the models in a window of 4 days.

The designed and implemented models have a high performance in terms of a low error and a remarkably proficient fitting of the time series to be predicted. MSPM-2 and MSPM-3 exceeded the results of MSPM-1 with a appreciable difference. However, the general fitness of these two models is greatly similar. Thus, MSPM-2 could be desirable due to the lower processing time of the sequence.

Chapter 5

Conclusion and future lines:

This chapter is devoted to summarize the most relevant conclusions and results of this Thesis, as well as to propose some of the main open problems left as future research lines.

5.1 Conclusions

The forecast of the power consumed by an HVAC system located in a self sufficient solar house was the addressed problem. The house, called MagicBox and located at the Escuela Técnica Superior de Ingenieros de Telecomunicación (ETSIT) of the Universidad Politécnica de Madrid (UPM), is equipped with a monitoring system to acquire the data. The main goal was to predict the next day (short term forecasting) of the power time series given the previous day as an input sequence. To accomplish this task three prediction models were proposed. These models are RNNs based on LSTM layers to capture the sequential nature of the time series. The designed RNNs were implemented with the ANN Python's library Keras.

More precisely, for this purpose, a main neural architecture was presented that yields, as generalizations, one single step ahead prediction architecture (that can be seen as a first step towards the solution of the problem), as well as three multi step ahead predictor architectures which main features are

- The first multi step model (MSPM-1) is designed to forecast the entire following day at once. Its main characteristic is that all the predictions are made at the last time step by providing the output perceptron layer with enough neurons.

- The second proposed multi step model (MSPM-2) does not wait for the whole input sequence to be provided in order to make the predictions. It outputs the estimations of the consumed power in a continuous data flow at the same time the new inputs are fed.

- The last developed multi step model (MSPM-3) is based on the artificial neural network called encoder–decoder, and it can be considered a hybrid architecture of the single step prediction model and the previous model.

After an empirical tuning of the model's parameters, the architectures were assessed in the frame of computing their performance metrics and visualizing their

prediction's values. The model proved to provide excellent results, fitting the targeted real measures in a remarkably accurate way. Some of the featured metric obtained from the most effective model, among the designed ones, were:

- A correlation between the target time series and the predictions of 0.97. This correlation was computed by the Pearson correlation coefficient.
- A Normalized Root Mean Squared Error of 0.049. This metric allows the comparing with other system's due to its normalized value.

As a very final conclusion, the designed and implemented models have a high performance in terms of a low error and a remarkably proficient fitting of the time series to be predicted. To be more precise, models MSPM-2 and MSPM-3 exceeded the results of MSPM-1 with an appreciable difference. However, the general fitness of these two models is greatly similar. Thus, MSPM-2 could be chosen because of the lower processing time of the sequence.

5.2 Future research lines

The scientific context, where the treated problem in this Thesis is framed, is an active research field. A proof of this claim is that, during the development of this work, different new challenging questions have appear. In the following, some of them are briefly described.

- To refine the designed predictors, some novel ideas could be applied to the models. Some options of these modifications could be the integration of a Convolutional Neural Network jointly with the current recurrent architecture, the use of LSTM layer's variations (i.e. the Peephole LSTM layer). In addition, a deeper learning could be performed by means of more powerful computational processing units.
- Another question is the integration of the developed predictor into a distributed DSM system to control as a deferable load. This would allow the operation scheduling in order to flatten the aggregated demand response.
- In addition to the previous line, other deferable consuming machines could be studied to enhance the overall distributed system.
- Finally, a more automated procedure to perform the parameter tuning of Section 4.3 can be analysed. Some ideas to overcome this task are the design of another ANN or a genetic algorithm.

Bibliography

Asghar, K. M., Nadeem, J., Anzar, M., Ali, K. Z., and Nabil, A. (2015). A generic demand side management model for smart grid. *International Journal of Energy Research*, 39(7):954–964.

Beccali, M., Cellura, M., Brano, V. L., and Marvuglia, A. (2008). Short-term prediction of household electricity consumption: Assessing weather sensitivity in a mediterranean area. *Renewable and Sustainable Energy Reviews*, 12(8):2040 – 2065.

Caamaño-martín, E., Egido, M. A., Neila, J., Bedoya, C., Santos, A. G., Jiménez, F. J., and Magdalena, L. (2005). Spanish participation in the "solar decathlon 2005" competition: New proposals for zero-energy houses. In *Proceedings of the 20th European Photovoltaic Solar Energy Conference*, pages 2587–2590, 6-10 June 2005, Barcelona, Spain.

Calvo-Fernández, M., E. Vega, J., A. Egido, M., and Caamaño-Martín, E. (2005). Spanish participation in the "solar decathlon 2005": Design and simulation of the photovoltaic system. pages 1958–1963.

Castillo-Cagigal, M., Caamaño-Martín, E., Matallanas, E., Masa-Bote, D., Gutiérrez, A., Monasterio-Huelin, F., and Jiménez-Leube, J. (2011a). Pv self-consumption optimization with storage and active dsm for the residential sector. *Solar Energy*, 85(9):2338 – 2348.

Castillo-Cagigal, M., Gutiérrez, A., Monasterio-Huelin, F., Caamaño-Martín, E., Masa, D., and Jiménez-Leube, J. (2011b). A semi-distributed electric demand-side management system with pv generation for self-consumption enhancement. *Energy Conversion and Management*, 52(7):2659 – 2666.

Castillo-Cagigal, M., Matallanas, E., Gutiérrez, l., Monasterio-Huelin, F., Caamaño-Martín, E., Masa-Bote, D., and Jiménez-Leube, J. (2011c). Heterogeneous collaborative sensor network for electrical management of an automated house with pv energy. *Sensors*, 11(12):11544–11559.

Ceperic, E., Ceperic, V., and Baric, A. (2013). A strategy for short-term load forecasting by support vector regression machines. *IEEE Transactions on Power Systems*, 28(4):4356–4364.

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv:1409.1259*.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Demuth, H. B., Beale, M. H., De Jess, O., and Hagan, M. T. (2014). *Neural Network Design*. Martin Hagan, USA, 2nd edition.

Deng, J. and Jirutitijaroen, P. (2010). Short-term load forecasting using time series analysis: A case study for singapore. In *2010 IEEE Conference on Cybernetics and Intelligent Systems*, pages 231–236.

Gajowniczek, K. and Zabkowski, T. (2014). Short term electricity forecasting using individual smart meter data. *Procedia Computer Science*, 35:589 – 597.

Giles, C. L., Lawrence, S., and Tsoi, A. C. (2001). Noisy time series prediction using recurrent neural networks and grammatical inference. *Machine Learning*, 44(1):161–183.

González, P. A. and Zamarreño, J. M. (2005). Prediction of hourly energy consumption in buildings based on a feedback artificial neural network. *Energy and Buildings*, 37(6):595 – 601.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Graves, A., Mohamed, A., and Hinton, G. E. (2013). Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778.

Haykin, S. S. (2009). *Neural networks and learning machines*. Pearson Education, Upper Saddle River, NJ, third edition.

Hinton, G. (2012). Neural networks for machine learning. *Coursera, video lectures*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Hou, Z. and Lian, Z. (2009). An application of support vector machines in cooling load prediction. In *2009 International Workshop on Intelligent Systems and Applications*, Wuhan, China.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lusis, P., Khalilpour, K. R., Andrew, L., and Liebman, A. (2017). Short-term residential load forecasting: Impact of calendar effects and forecast granularity. *Applied Energy*, 205:654 – 669.

Park, D. C., El-Sharkawi, M. A., Marks, R. J., Atlas, L. E., and Damborg, M. J. (1991). Electric load forecasting using an artificial neural network. *IEEE Transactions on Power Systems*, 6(2):442–449.

Rahman, A., Srikumar, V., and Smith, A. D. (2018). Predicting electricity consumption for commercial and residential buildings using deep recurrent neural networks. *Applied Energy*, 212:372 – 385.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA.

Solano, J., Olivieri, L., and Caamaño-Martín, E. (2017). Assessing the potential of pv hybrid systems to cover hvac loads in a grid-connected residential building through intelligent control. *Applied Energy*, 206:249 – 266.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

Ullah, M. N., Javaid, N., Khan, I., Mahmood, A., and Farooq, M. U. (2013). Residential energy consumption controlling techniques to enable autonomous demand side management in future smart grid communications. *2013 Eighth International Conference on Broadband and Wireless Computing, Communication and Applications*, pages 545–550.

Appendix A

Impact

This appendix is devoted to describe the impact of this project in different aspects. Social, economical and environmental points of view are considered.

- Social impact: this project will have a direct impact on people who are interested in the integration of intelligent systems in their properties. These systems will enhance the energy management of their housings with an economical improvement.
- Economical impact: this project will have an impact in the economic field with the incorporation of the predictor in energy management techniques. These techniques' aim is to flatten the energy demand curve, resulting in a more efficient use of the the energy under an economical point of view.
- Environmental impact: this project will have a direct environmental impact because the designed predictor is intended to be integrated in energy management systems that control that energy consumption is performed efficiently. In addition, these systems promote the use of renewable energies, such as the photovoltaic energy production.

Appendix B

Budget

This project has been developed during 6 months in the laboratory of robotics and control of the Escuela Técnica Superior de Ingenieros de Telecomunicación using some of its resources. An approximate budget is estimated taking into account human resources, software, technical equipment and some laboratory material used during the project.

- Costs derived from human resources: This item should consider the salary of all the people involved in the project: project manager (engineer) and the engineering student, author of this Thesis as shown on Table B.1.

	Cost per hour (€)	**Working hours**	**Total cost (€)**
Project manager	22	75	1650
Engineering student	10	1250	12500
TOTAL			14150

Table B.1: Costs derived from human resources

- Costs derived from software and technical equipment: For this Thesis, the software and technical equipment listed on Table B.2 has been used. The total costs are computed by the product of the depreciation cost per month and the time of use.

	Lifetime (years)	**Units**	**Cost (€)**	**Depreciation (€/month)**	**Time used (months)**	**Total cost (€)**
GTX 1080 Ti	5	1	1000	16.66	6	99.96
Personal computer	4	1	1000	20.83	6	124.98
MATLAB License	1	1	2000	166.66	6	999.96
TOTAL						1224.9

Table B.2: Costs derived from software and technical equipment