Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial
Especialidad Vehículos Aeroespaciales

Application of Machine Learning techniques for the prediction of solar radiation. (Aplicación de técnicas de Machine Learning para la predicción a corto plazo de la radiación solar.)

Autor: Pablo Egea Hervás
Tutor: Carlos Vivas Venegas

Dep. Ingeniería de sistemas y automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Proyecto Fin de Grado
Grado en Ingeniería Aeroespacial
Especialidad de Vehículos Aeroespaciales

Application of Machine Learning techniques for the prediction of solar radiation

Autor:
Pablo Egea Hervás

Tutor:
Carlos Vivas Venegas
Profesor DC

Dep. de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Proyecto Fin de Carrera: Application of Machine Learning techniques for the prediction of solar radiation

Autor: Pablo Egea Hervás

Tutor: Carlos Vivas Venegas

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

"Knowing your own ignorance is the first step to enlightenment"- Patrick Rothfuss, The Wise man's Fear

Acknowledgments

Well, this was a tough year. Time for endings and new beginnings. This work is the last milestone of the path I started 5 years ago when I came to Seville to study aerospace engineering. It is also the first milestone of a new path that I do not know yet. In this five years, a lot have happen: friendship that come and go, others that stay; countless hours of studying in our dear library; countless exams that made your brain melt. But I believe everything was worth it and that I will remember this period of my life very fondly in the future. This work goes to all the people that have supported me trough and through during this long and, at the same time, short five years. I want to thank my family and my friends, the new ones and the ones that have always been there, and of course I want to thank the supervisor of this work because without his patience this work would have never been possible.

Abstract

Generation of electricity from the sun is an important renewable energy source, and the number of both, solar thermal energy systems and photovoltaic system, are proliferating around the world. However, the integration of large amounts of solar production into the electricity grid poses technical challenges due to the fluctuating characteristics of available solar energy sources. Solar energy output is not easily predictable in advance and varies based on both weather conditions and site specific conditions. Such variability of solar energy resources at ground level thus raises concerns regarding how to manage and integrate output from the solar energy systems to the power grid.

Given the issues above, there is increasing interest in more precise modeling and forecasting of solar power. Irradiance is a measurement of solar power and usually measures the power per unit area. Most works consider the solar irradiance forecasting at a site, which is essentially the same problem as forecasting solar power. The ability to forecast solar irradiation will enable power grid operators to be able to ensure the quality and control of solar electricity supplies and will allow them to better accommodate highly variable electricity generation in their scheduling, dispatching, and regulation of power.

In particular, the possibility to forecast solar irradiance can became fundamental in making power dispatch plans, and also a useful reference for improving the control algorithms. Today it is widely acknowledged by power producers, utility companies and independent system operators that it is only through advanced forecasting, communications and control that these distributed resources can collectively provide a firm, dispatchable generation capacity to the electricity market.

Different solar irradiance forecast time horizons are usually employed: Some of them forecast up to 24 h or even more in what it is usually termed as mid-to-long term forecast. These forecast methods are tightly linked to weather prediction methodologies and provide averages of solar irradiation on daily or weakly basis. Whilst useful for mid-term planning of energy production, such techniques do not meet the demands of modern electricity markets. In contrast to other commodities, electricity is characterized by some specific features that are difficult to deal with such as non-storability, the simultaneousness of electricity production, transmission and consumption, and potential network congestions, among others. All these issues lead to the inconvenience of having to program electricity generation in advance. For this reason, in the electricity liberalized markets, the day-ahead market is commonly referred to as the spot market.

Nevertheless, after the closure of the day-ahead market, participants are able to sell or buy the surplus or deficit electricity in more real time (Intraday and Balancing markets). Thus, during intraday trading, participants continue to finetune their positions in the light of new information about their own production,consumption and also the overall system position. For renewable energy, solar in particular, to be able to compete in intraday market, it is compulsory to improve the short-term predictability of solar production a different time scales. For example the Spanish electricity market considers intraday energy auctions as short as four hours ahead, that is, the utility commits to produce a given amount of energy four hours ahead of the real dispatch. Other markets operate in even more stringent basis: The German European Energy Exchange (EEX) operates a continuous trading system where power in the intraday market can be traded until 1.25 hours before delivery, and the Australian electricity market uses 5-min dispatch price and a 30-min trading price. Thus, accurate short-term forecast is essential for energy market participation, both due to forward contracting and the need for a predictable, stable and smooth supply. Accurately forecasting direct normal irradiance or global horizontal irradiance in the seconds-to-minutes time-frame (usually termed as Nowcasting) ultimately enables finely-tuned dynamic operational schedules that can reduce fuel costs, increase network stability or maximize system lifetimes.

The aim of this project is to apply machine learning techniques to predict solar radiation for applications in Solar energy plants. The objective is to be able to perform nowcasting (5 min to 1 hour ahead), in the solar radiation using recent past registry of available meteorological data (solar irradiance, ambient temperature, moisture, wind velocity, barometric pressure, etc). Solar irradiance forecast methods include time series [14, 15], wavelet analysis and fuzzy logic [17, 16], satellite data and sky images [18, 19] and statistical learning methods such as artificial neural network (ANN) [20, 21].

This work will focuss on the use of ANN to predict solar radiation. The software `MATLAB` and its *Toolbox* for *Artificial Neural Networks* will be used. This document can be divided into three main parts: the first part establishes the theoretical background of artificial neural networks focussing in multilayer perceptrons as the main tool to be used in this work; the second part explains how the raw meteorological data has been processed (normalized, classified and filtered) to be used for ANN training purposes; and the third and final part presents a series of experimental evaluations to analyze the performance of the technique proposed.

The experiments carried out in this work have the objective of evaluating different sets of inputs and ANN configurations to analyze the best performing structure for nowcasting. The available inputs, are subsets of the following channels of information: global radiation, diffuse radiation, relative humidity, atmospheric pressure, temperature and the position of the sun represented by its zenith angle.

All the data used for training the multilayer perceptrons is provided by the GTER, in particular from its radiological station in the roof of the laboratories building of the Escuela Técnica Superior de Ingeniería (Superior School of Engineering) of the University of Seville, located in Seville in Cartuja's island.

With this data and the usage of the software `MATLAB` and its *Toolbox* for *Artificial Neural Networks* several neural networks will be created and trained as well as tested. The experiments that will be carried out in this work deal with different sets of inputs from the available measurement , which are global radiation, diffuse radiation, relative humidity, atmospheric pressure, temperature and the position of the sun represented by its zenith angle. Another characteristic to take in account is the structure of the multilayer perceptron: number of neurons and number of layers. Different structures are also tested in this work to achieve the optimal structure that provides best results.

After the experiments were carried out the main conclusions were that, with the data available, only clear days or clear parts of one day can be predicted accurately, while cloudy days with irregular change in the solar radiation could not be predicted with good results. In the final chapter of this work the results of the experiments are discussed. Suggestions of future lines of work to improve this one are included as well in the final chapter.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem

In a world such as the one we live in today, the finite nature of fossil fuels as a primary energy source, and the environmental impact of their use, makes necessary to devise alternative ways of energy production. This new ways to obtain energy are based in the idea of using the already existing energy on Earth i.e. wind force, tides, geothermal energy and solar energy. This sources of energy are environmentally friendly and renewable. among the different types of renewable energies, this project will be focused in solar energy which comes from the radiation of the sun that reaches the earth and it aims at transforming the solar radiation into conventional thermal or electrical energy for human use. Two main technological approaches are available for such a purpose: Solar thermal energy, where the solar radiation is used to heat some working fluid as a means of ultimately producing electricity and/or heating/cooling; and Solar Photovoltaics, where solar radiation is directly transformed into electric energy.

Generation of electricity from the sun is an important renewable energy source, and the number of both, solar thermal energy systems and photovoltaic system, are proliferating around the world. However, the integration of large amounts of solar production into the electricity grid poses technical challenges due to the fluctuating characteristics of available solar energy sources. Solar energy output is not easily predictable in advance and varies based on both weather conditions and site specific conditions. Such variability of solar energy resources at ground level thus raises concerns regarding how to manage and integrate output from the solar energy systems to the power grid.

Given the issues above, there is increasing interest in more precise modeling and forecasting of solar power. Irradiance is a measurement of solar power and usually measures the power per unit area. Most works consider the solar irradiance forecasting at a site, which is essentially the same problem as forecasting solar power. The ability to forecast solar irradiation will enable power grid operators to be able to ensure the quality and control of solar electricity supplies and will allow them to better accommodate highly variable electricity generation in their scheduling,

dispatching, and regulation of power.

In particular, the possibility to forecast solar irradiance can became fundamental in making power dispatch plans, and also a useful reference for improving the control algorithms. Today it is widely acknowledged by power producers, utility companies and independent system operators that it is only through advanced forecasting, communications and control that these distributed resources can collectively provide a firm, dispatchable generation capacity to the electricity market.

Different solar irradiance forecast time horizons are usually employed: Some of them forecast up to 24 h or even more in what it is usually termed as mid-to-long term forecast. These forecast methods are tightly linked to weather prediction methodologies and provide averages of solar irradiation on daily or weakly basis. Whilst useful for mid-term planning of energy production, such techniques do not meet the demands of modern electricity markets. In contrast to other commodities, electricity is characterized by some specific features that are difficult to deal with such as non-storability, the simultaneousness of electricity production, transmission and consumption, and potential network congestions, among others. All these issues lead to the inconvenience of having to program electricity generation in advance. For this reason, in the electricity liberalized markets, the day-ahead market is commonly referred to as the spot market.

Nevertheless, after the closure of the day-ahead market, participants are able to sell or buy the surplus or deficit electricity in more real time (Intraday and Balancing markets). Thus, during intraday trading, participants continue to finetune their positions in the light of new information about their own production,consumption and also the overall system position. For renewable energy, solar in particular, to be able to compete in intraday market, it is compulsory to improve the short-term predictability of solar production a different time scales. For example the Spanish electricity market considers intraday energy auctions as short as four hours ahead, that is, the utility commits to produce a given amount of energy four hours ahead of the real dispatch. Other markets operate in even more stringent basis: The German European Energy Exchange (EEX) operates a continuous trading system where power in the intraday market can be traded until 1.25 hours before delivery, and the Australian electricity market uses 5-min dispatch price and a 30-min trading price. Thus, accurate short-term forecast is essential for energy market participation, both due to forward contracting and the need for a predictable, stable and smooth supply. Accurately forecasting direct normal irradiance or global horizontal irradiance in the seconds-to-minutes time-frame (usually termed as Nowcasting) ultimately enables finely-tuned dynamic operational schedules that can reduce fuel costs, increase network stability or maximize system lifetimes.

Since the radiation is so variable, so is the energy produced by solar plants in instantaneous terms. But that does not mean that it is impossible to predict the energy that will be produced by these plants; as in terms of mean values it is fairly well predicted by theoretical models or another means, such as the one that will be used in this work: *Artificial Neural Networks*. Solar irradiance forecast methods include time series [14, 15], wavelet analysis and fuzzy logic [17, 16], satellite data

and sky images [18, 19] and statistical learning methods such as artificial neural network (ANN) [20, 21]. ANN have proved a better approximation of the reality than theoretical models. According to [6]:"The ANN models are found to predict solar radiation more accurately thanÅngström model, conventional, linear, non-linear and fuzzy logic models". ANN have become a great tool to use for pattern recognition, functions interpolation, nonlinear regression and more problems. ANN are discussed in more detail in the chapter 2 of this work.

As it has been noted, there are many works that try to predict solar radiation, but in a wide window of time such as a day, a week or a month (*forecasting*). The motivation of this work is to use ANN to perform *nowcasting* of solar radiation at a given location. This knowledge is important because in order to control the energy supply, it is necessary to know how much energy is being produced at each station. As the solar radiation is variable, it is not easy to accurately predict the *instantaneous* amount of energy produce by solar energy plants because there are no reliable ways to predict solar radiation in short term i.e. an hour. If this is known it would be easier and more efficient to control the energy supply and it will avoid waste of energy that might happen with a sudden peak in the solar radiation. But even though there are many works that focus on forecasting or in the prediction of solar radiation in different locations, there are few works that deal with the nowcasting of solar radiation.

1.2 Previous works

In this field there are several works that use machine learning approaches to obtain a prediction of the solar radiation, but they are usually aimed to forecasting, which is the prediction in a large window of time such as a day, a week or a month depending on the variable to predict. To solve this problem many works use ANN, as they have proved that it provides a better performance than theoretical predictions (see [6]). This is a tool that fits very well this field of study because it deals with the prediction of the weather, which is a really complex problem where the cause and effect relation is not always clear. ANN allow to obtain a relation between some inputs and the outputs that might no be seen analytically, or to obtain a relation when there is a lack of an equation to relate those variables. This is why ANN are the tool chosen to deal with this kind of problem as can be seen in the following works.

The work in [3] is similar to this work in the way that they also used `MATLAB` to implement the ANN, although an older version that is now obsolete regarding the tools to work with *Artificial Neural Networks*. They use maximum temperature, mean wind speed, sunshine hours, mean relative humidity ans solar radiation to predict and generate a weather model for Al Ain City predicting the mean monthly global solar radiation.

Article [6] consists of a summary of several works that use *Artificial Neural Networks* to predict solar radiation. There are several works where different training algorithms and different sets of inputs were used. For example: in [8] an ANN

model is developed to estimate the solar radiation using combinations of latitude, longitude, altitude, months, average temperature, average cloudiness, average wind velocity and sunshine duration. [9] use multilayer feedforward network to predict several types of solar radiation using different variables such as wind direction, wind speed, ambient temperature, relative humidity, cloudiness and water vapor. In [10] combinations of day, maximum air temperature, mean air temperature and relative humidity are used to estimate diffuse solar radiation and it shows that using relative humidity and daily mean temperature result in a better performance of the ANN model.

In [5] they use innovative machine learning techniques to predict the daily global solar radiation. It is innovative because it implements an algorithm to find the optimal size of the neural network.

As can be seen in this work descriptions, most of them try to predict daily radiation or monthly radiation and they use average variables. That is why this work is different.

1.3 Contributions of This work

In this work, as in many of the works exposed in the past section, ANN will be used to predict solar radiation. But in this case, the prediction will aim to nowcasting instead of forescasting. The aim of this project is to apply machine learning techniques,, ANNs in particular, to predict solar radiation for applications in Solar energy plants. The objective is to be able to perform nowcasting (5 min to 1 hour ahead), in the solar radiation using recent past registry of available meteorological data (solar irradiance, ambient temperature, moisture, wind velocity, barometric pressure, etc). For this to work is necessary to have a large data base of meteorological and radiological variables. For this there is data available from a radiological station in the roof of the *Escuela Superior de Ingeniería* that provides three years data every five seconds, which is a fair amount of data to work with.

Given the huge amount of data, this work is divided in two different blocks: the first one being the arrangement of the data to be use in the training of the ANN (chapter 3) and the second being the experiments (chapter 4).

There is also a chapter dedicated to the theoretical background of machine learning (chapter 2). But why machine learning? Since there are only approximate models to predict the behavior of solar radiation and they usually need lots of parameters (some of them not easily measurable), machine learning techniques should provide more accurate models with more manageable and measurable parameters. In machine learning, the machine, which in this case is a computer, "learns" from previous data the relations between some inputs and the desired outputs (targets); relation that can be used later to predict the output only with the inputs. This process is known as the learning process and this , in particular, is supervised learning which is a particular type of learning procedure (different learning procedures are explained in more detail in chapter 2). Thus, once the algorithm has learned this

relations it is capable of using data that has not be presented before as inputs and of returning an output that will be the prediction according to past behavior. Here is assumed that the evolution of the solar radiation is not something random but it rather follows some kind of law that cannot be put into equations. In order to obtain a good performance, the important thing is to select the proper inputs and the optimal size of the multilayer perceptron. This process can be seen as a way to obtain a nonlinear regression of the data points in the historic. Machine learning techniques are used precisely to obtain nonlinear regressions among other things, such as pattern classification and association to name a few.

In order to implement this machine learning techniques, the software `MATLAB` will be used as it implements a toolbox design to work with artificial neural networks. This toolbox provides the algorithms to create and train a neural network that can be customize to fulfill the needs of the user.

To finish this introduction, it will be good to summarize the structure of this work which will be as follows:

- An introduction to the theory of machine learning and *Artificial Neural Networks*.
- The data that will be used and how it is processed.
- The experiments carried out.
- Results discussion.
- Conclusions.

Chapter 2

Machine Learning

As explained in the introduction, the aim of this project is to perform nowcasting predictions of the solar radiation at a given location. To do the predictions, machine learning methods will be used. In particular Multilayer Perceptrons, which are a type of *Artificial Neural Network*, will be used as the predictor. First, there is an explanation of what is Machine Learning.

According to [1] " Machine learning is programming computers to optimize a performance criterion using example data or past experience." As it is said too in [1] this is needed when is not possible to use a computer script to solve the problem, either because there is not enough information about the process to obtain the solution or because the process is not known. In other words, the algorithm is unknown. When the algorithm is unknown all there is, is data. When there is data, there are different samples. The inputs and the outputs are known so the machine learning techniques draw a function out of the samples to relate the inputs and the outputs in the optimal form the data can provide. So, one could say machine learning techniques allow a machine (a computer) to learn by means of examples the way these examples are connected. Thus, obtaining a relationship between the examples. The major application of machine learning are learning associations, classification and regression. Regression is the application that meet the requirements of this project, in particular non-linear regression.

This chapter centers in one particular technique: *Artificial Neural Networks* and makes a special emphasis in multilayer perceptrons (MLP), since they are the type of ANN that will be used in this work.

2.1 Artificial Neural Networks

First of all, what is a *Neural Network*? According to [2] " A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.

2. Interneuron connection strengths, known as synaptic wights, are used to store the acquired knowledge."

In this work, *neural networks* are used, more precisely *multilayer perceptrons*. A *multilayer perceptron* is only a particular architecture of an *artificial neural network*.

It is important to notice why they are called Artificial Neural Network. They are called like this because the main idea behind this tool is based in the human brain and how it processes information. It is known the brain is formed by *neurons*, which can be seen as simple processor units that are, according to [2], five or six order of magnitude slower than modern silicon logic gates. But the brain makes it up having a massive number of neurons and even a greater number of connections between them, which left as a result an incredibly efficient "machine". What neurons do, extremely simplified, is receive information in the form of electrical pulses, transform it and then passing it through its connection to other neurons that would perform the same process.

2.1.1 Neuron model

According to [2] "A neuron is an information-processing unit that is fundamental to the operation of a neural network." A schematic of a neuron model can be seen in figure 2.1 and it is formed by three basic elements:

- Synapses or connection links characterized by a weight of its own. A signal x_j at the input of synapse j connected to a neuron k is multiplied by the synaptic weight w_{kj}. They can be positive or negative [2].

- An adder for summing up the input, weighted by the synaptic weights [2].

- An activation function for limiting the amplitude of the output neuron. The normalize amplitude range of the output of a neuron is written as the closed unit interval $[0, 1]$ or alternatively $[-1, 1]$ [2].

The bias include in the model has the effect of increasing or lowering the net input of the activation function, depending on whether it is positive or negative [2].

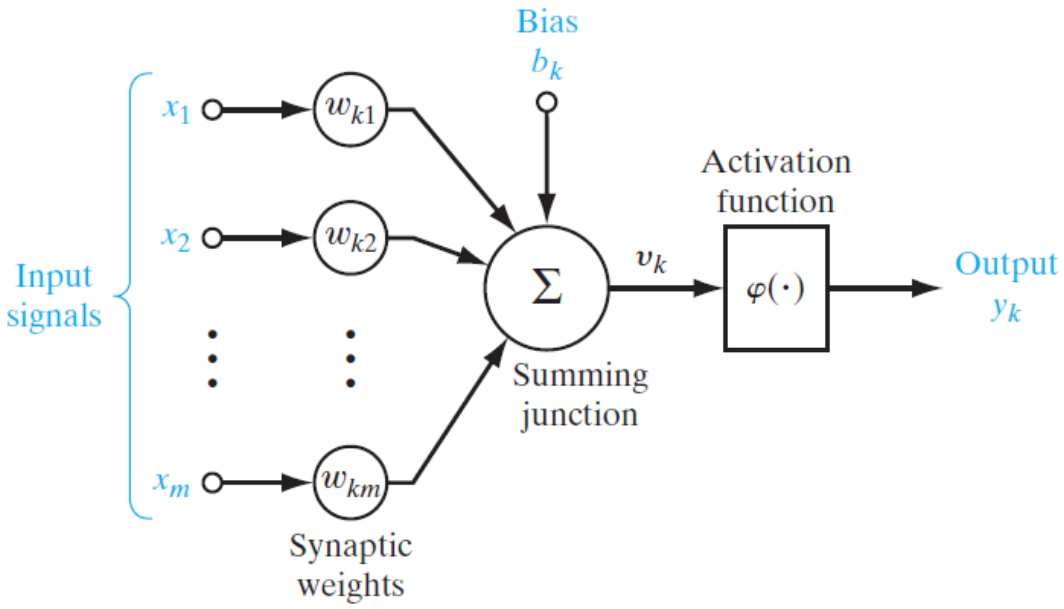


Figure 2.1: Neuron Model taken from [2]

Mathematically, according to [2], this neuron can be describe with the next equations.

$$y_k = \phi(u_k + b_k) \tag{2.1}$$

where

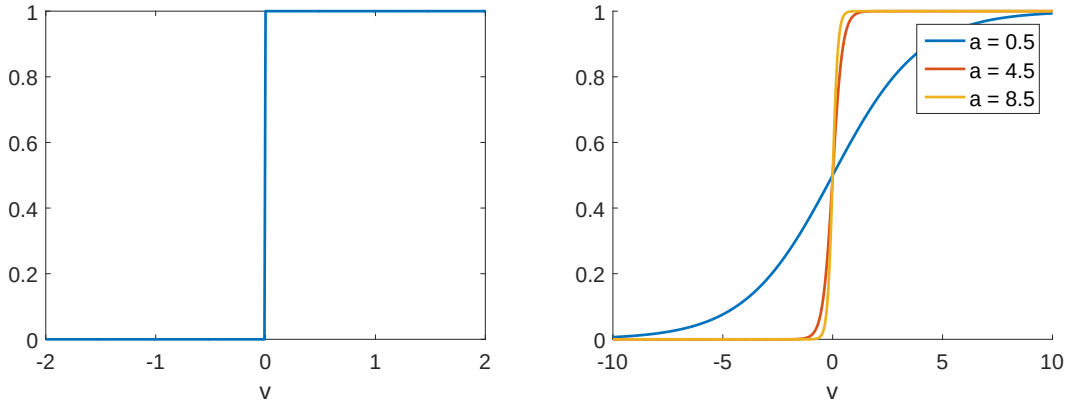
$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.2}$$

This equations say that the output of the neuron is the result of using the activation function on the weighted inputs plus the bias. The bias can be put as another input that has its value fixed and equal to +1 with a synaptic wight equal to the value of the bias.

2.1.2 Activation Functions

The two basic activation functions are:

- Threshold function: it returns 1 if the argument is positive or 0 if the argument



(a) Threshold function (b) Sigmoid function

Figure 2.2: Activation functions

is negative. As shown in equation 2.3. Its graphic can be seen in figure 2.2a.

$$\phi(v) = \begin{cases} 1 & if \quad v \geq 0 \\ \\ 0 & if \quad v < 0 \end{cases} \tag{2.3}$$

- Sigmoid function: it is "S" shaped and it is the most common form of activation function used in neural networks. It is a strictly increasing function that exhibits a balance between linear and nonlinear behavior. An example is the logistic function:

$$\phi(v) = \frac{1}{1 + exp(-av)} \tag{2.4}$$

where a is the slope parameter. Varying a, sigmoid functions with different slopes can be obtained (Figure 2.2b).

This two equations result in a range from 0 to +1. Sometimes it is desirable to have a range from -1 to +1. To obtain this, the threshold function is rewritten as the commonly known *signum function*:

$$\phi(v) = \begin{cases} 1 & if \quad v > 0 \\ \\ 0 & if \quad v = 0 \\ \\ -1 & if \quad v < 0 \end{cases} \tag{2.5}$$

whereas the hyperbolic tangent function can be used as the corresponding form for the sigmoid function [2].

$$\phi(v) = tanh(v) \tag{2.6}$$

2.1.3 Architectures

The architecture of a network is the same as its structure. Defining the number of neurons and how many connections between the neurons are lay out. There are three fundamentally different classes of network architecture [2].

2.1.3.1 Single-Layer feedforward networks

This is the simplest form of a layered neural network where the input layer of source nodes connects directly with an output layer. It is called feedforward because the transmission of information is strictly from the input to the output with no feedback of any kind. An example can be seen in figure 2.3.

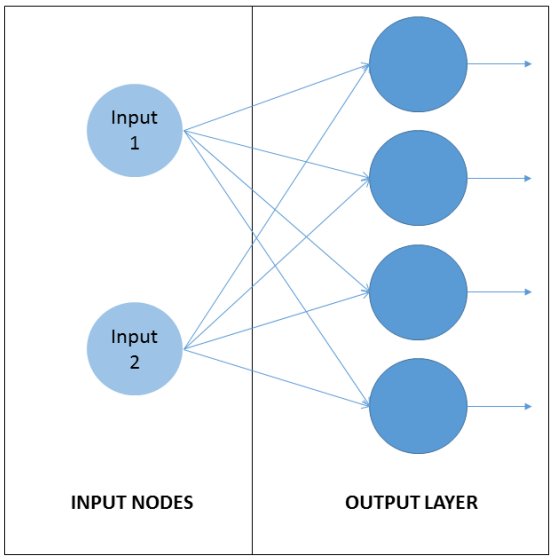


Figure 2.3: Single-Layer feedforward network

2.1.3.2 Multilayer Feedforward Networks

In this type of feedforward network the information goes, as well, one way only. The difference is that there is one or more layer between the input and the output layer. An example of this is a multilayer perceptron. A MLP can be seen in the next section in figure 2.4. These kind of networks can be either fully or partially connected. In fully connected networks every neuron is connected to each neuron in the next layer (the multilayer perceptron is one of this kind). In partially connected networks some of this connections are missing [2].

2.1.3.3 Recurrent Networks

This kind of structures have at least one feedback loop and, thus, they are no longer feedforward networks. Dynamics networks use this type of structure in order to be able to recognize time series. Due to the complexity of this type of networks this kind of structure is left out of this work. But it is worth naming them since they are one of the major kind of ANN.

2.1.4 Multilayer Perceptron

According to [1] "A multilayer perceptron is an artificial neural network structure and is a nonparametric estimator that can be used for classification and regression. In this case, the multilayer perceptron is used for its capability of regression". This is the type of structure that will be used in this work. It can be defined too as a multilayer feedforward network fully connected.

A general picture of the structure of a *multilayer perceptron* can be seen in figure 2.4. Here, the different layers of the perceptron are shown. To know: input layer, hidden layer(s) and the output layer. It can be seen also, that every layer is formed by several units, these units are "the simple processing units" that [2] discussed in its definition of a neural network. Now there is a parallel distributed processor, which is the *multilayer perceptron* and the "simple processing units" which are known as *neurons* for its resemblance with the human brain as explained in the previous sections. In the definition, it is said that the network can acquired knowledge through a learning process, known also as training process. It is also said, that the interneuron connection strengths, which can be seen in the schematics 2.4, are the feature that store the knowledge of the network. Thus, it is clear to see that the learning process seeks to set the right *synaptic weights* for the network to

store the knowledge needed. To set these weights in the right way the most basic algorithm used is the backpropagation algorithm.

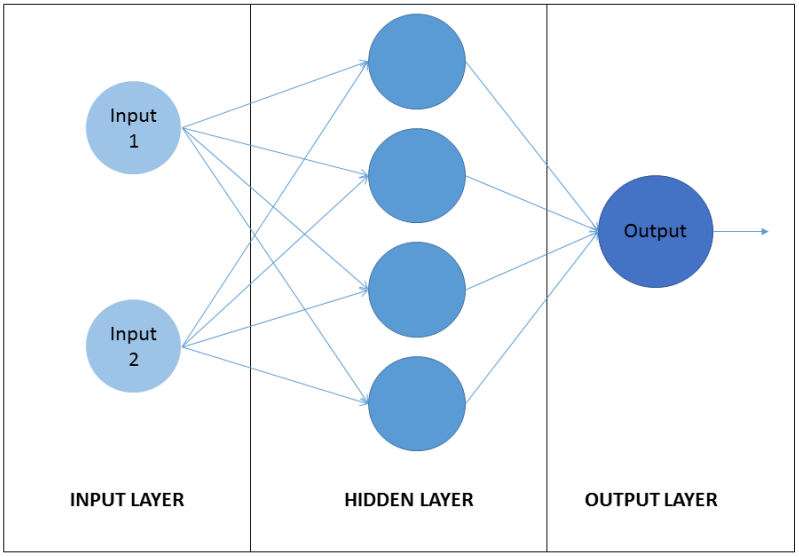


Figure 2.4: Architecture of a multilayer perceptron

In the following lines it is discussed how the neurons belonging to the hidden layer behave. Note that the neurons in the hidden layer(s) are the ones that process the inputs. Generally there would be i inputs which correspond to the number of neurons in the input layer (they only act as buffers for the input signals) and a number j of neurons in the hidden layer. As can be seen in 2.4, each neuron in the input layer is connected to each neuron in the hidden layer, so the connections ij are defined. For every connection there is a synaptic weight w_{ij}. Thus, given the inputs signals $x_i(i = 1, 2, ...n)$ coming from the input layer, each neuron j in the hidden layer sums up its input signal x_i after weighting them with the synaptic weights of the connections w_ij and computes its output y_j as a function of the sum.

$$y_j = f(\sum_{i=1}^{n} w_{ji}x_i) \tag{2.7}$$

For this architecture to be a MLP, it is said in [2] that the activation function of each neuron must be nonlinear and differentiable. Usually, these functions are a simple threshold function, a sigmoidal function, a hyperbolic tangent or a radial basis function, as stated in [3]. Some of these activation functions have been already

discussed in the previous section.

This process occurs in a similar manner in the output layer.

Before starting to discuss the backpropagation algorithm is necessary to establish the two types of signal that travel trough the network (as explained in [2].

- *The function signals*: the signal that comes from the inputs and travels neuron by neuron until it reaches the output, processed in each neuron the way explained before in this section.

- *The error signals*: the signal that originates in the output and propagates backward. It is called error signal because its computation involves an error dependent function in one form or another.

Because of these two types of signals, each hidden or output neuron is design to perform two computations: the computation of the inputs as explained before and the computation of an estimate gradient vector that represents the evolution of the error.

Having this as a background, the backpropagation algorithm and the training method can be, and will be, explained in the next section.

2.1.5 Backpropagation algorithm

Note that from here, when the word training is used it refers to supervised training, unsupervised training is not dealt with in this work.

Defining a training set of data of N samples where $\mathbf{x}(n)$ are the inputs and $\mathbf{d}(n)$ are the target outputs for $n = 1, ..., N$. Given a neuron in the output layer, j, its output produced by the input $\mathbf{x}(n)$ is denoted by $y_j(n)$. Thus, the error signal produced at the output of the neuron j of the output layer is $e_j(n) = d_j(n) - y_j(n)$. The *instantaneous error energy* of the neuron j is defined as 2.8.

$$\mathscr{E}_j(n) = \frac{1}{2} e_j^2(n) \tag{2.8}$$

Summing, the error energy of all the neurons of the output layer is:

$$\mathscr{E} = \frac{1}{2} \sum_{j=1}^{C} e_j^2(n) \tag{2.9}$$

Where C is the number of neurons in the output layer. The error energy averaged

over the training sample, given there is N samples is:

$$\mathscr{E}_a v = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j=1}^{C} e_j^2(n) \tag{2.10}$$

These expressions of the error are function of the synaptic weights. The aim of the training procedure is to minimize this error by updating the synaptic weights. There is two ways of doing this, defining two types of training: online learning and batch learning:

- **On-line learning**: in this kind of training the weights are updated after each sample n of the training set is presented to the network. This kind of training is popular for solving pattern-classification problems. The advantages are that is simple to implement and it is effective in solving largescale and difficult pattern recognition problems.

- **Batch learning**: This type of training updates the weights after every sample of the training set have been presented to the network, when every sample have been presented it is said that an epoch have past. The samples are randomize after each epoch. The advantages are that it provides an accurate estimation of the gradient vector, which guarantees a convergence of the process in the local minimum. Another perk is that it allows parallelization of the process, so it is usually quicker than on-line learning. The downsize is that it is very demanding in terms of storage requirements. This type of learning is effective for nonlinear regression problems. This fact is why this is the type of learning used in this work.

The updating of the weights Δw_{ij} is defined as:

$$\Delta w_{ij}(n) = \eta \delta_j(n) y_j(n) \tag{2.11}$$

Where η is the learning-rate parameter of the bakcpropagation algorithm and δ_j is the local gradient, defined as:

$$\delta_j(n) = \frac{\partial \mathscr{E}(n)}{\partial v_j(n)} = \frac{\partial \mathscr{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \phi_j'(v_j(n)) \tag{2.12}$$

Where $y_j(n)$ is the output of the neuron, ϕ_j is the activation function of the neuron and $v_j(n) = \sum_{i=0}^{m} w_{ij} y_i(n)$ and m is the number of inputs. This is how the

weights of the neuron j with the sample n are updated. In batch learning this is done once every sample have been presented and using $\mathscr{E}_ave$ instead of the instantaneous error energy .

The calculation of the error and , thus, the calculation of the gradient vector is different if the neuron is in the output layer or in the hidden layer. If the neuron is in the output layer, the calculation of the error is straightforward: $e_j(n) = d_j(n) - y_j(n)$. On the other hand, if the neuron is hidden it depends on the next layer, so it is a bit more complicated. The expression of the local gradient vector of a hidden neuron results in:

$$\delta_j(n) = \phi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{2.13}$$

Where the index k are the neurons in the next layer. To clarify: if it is assumed that, in the schematics 2.4, the hidden layer is the one being calculated, the neurons of this hidden layer are neurons j and the the neurons of the output layer will be the neurons k. All of this calculations can be consulted in depth in [2].

2.1.6 Levenberg-Marquardt method

The *Levenberg-Marquardt* method is the training method that will be used in this work because of its good implementation in `MATLAB`. As said in `MATLAB` documentation: This algorithm appears to be the fastest method for training moderate-sized feedforward neural networks (up to several hundred weights). It also has an efficient implementation in `MATLAB` software because the solution of the matrix equation is a built-in function. So, the attributes of this training method become even more pronounced in a `MATLAB` environment. According to [2] this method is a compromise between two methods:

- Newton's method: which converges rapidly near a local or global minimum, but may also diverge [2].
- Gradient descent: which is assured of convergence through a proper selection of the step-size parameter, but converges slowly [2].

Merging both methods allows it to obtain the best of each method and reduce the weaknesses. The mathematical development can be consulted in [2] (chapter 4, section 16).

Chapter 3

Data Processing

Once the theoretical background has been described and keeping in mind that the method that will be used in this work is the *multilayer perceptron* to create a model for nowcasting of solar radiation, the next thing to know is how it is going to be trained. As discussed in the previous chapter a method of supervised training will be applied using the software `MATLAB` and, for this training, the most important thing is the data that will be feed to the network to train it. This chapter deals with such data and how it is process in order to use it to train the network in different scenarios that will be develop in chapter 4.

3.1 Data

The popularity of artificial neural networks (ANN) is increasing since its capacity to model very complex problems. Improving training efficacy of ANN based algorithm is a topic of relevant interest. The performance of Multi-layer Perceptrons (MLP) trained with Back Propagation Artificial Neural Network (BP-ANN) method is highly influenced by the size of the datasets and the data-preprocessing techniques used. In the context of this work, the data fed to the network; essentially meteorological data, requires filtering and adaptation to guarantee a good forecasting performance of the network.

3.2 Data available

The data available for this work is obtained from a meteorological station of GTER (Grupo de trabajo de energías renovables) at the laboratories building of Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla (figure 3.1), located in

the city of Seville, in southern Spain. The station registers a variety of meteorological data with a sampling period of 5 seconds:

(a) Seville

(b) Station location

Figure 3.1: Locations

- Diffuse Radiation: Diffuse radiation measured with a band in W/m^2
- Global Radiation: Horizontal global irradiation in W/m^2
- Global Radiation at 27 South: global irradiation at 27 south in W/m^2
- Global Radiation at 27 from cell: global irradiation at 27 south extracted from cell in W/m^2
- Pyranometer: Measurement of the pyranometer in W/m^2
- Horizontal from cell: Horizontal radiation extracted from cell W/m^2
- $Hb0_NIP$: Direct radiation measured with a normal-incidence pyrheliometer in W/m^2
- $Hb0_CHP1$: Direct beam radiation with a pyrheliometer in W/m^2
- Diffuse Radiation from balls: Diffuse radiation measured with balls in W W/m^2
- Temperature: Temperature in C
- Wind Velocity: Velocity of the wind in m/s
- Wind Direction: Direction of the wind in degrees.

- Pressure: Pressure at the location in $mBar$.
- Relative Humidity: measurement of the relative humidity in the environment in percents.

The system registers up to sixteen different meteorological magnitudes with a sampling time o five seconds. The information is stored in conventional text files, where every row in the file represents the magnitudes collected for a given time instant. The first column in the files indicates the time when the data was collected in the format hh:mm:ss. A sample of one of this files can be observed in figure 3.2.

The system automatically creates a new file for every natural day of the year with the format meteo_YYYY_NNN.txt, where YYYY represents the year, and NNN represent the day of the year (from 1 to 365 or 366 in leap years). For instance, the file meteo_2012_004.txt stores the data corresponding to 4th January 2012, from 00:00:00 to 23:59:55 that day.

For the purpose of network training, a segment of data covering three full years (from 2012 to 2014) was selected. Sixteen variables measured every day, every five seconds for three years is a huge amount of data and that needs proper processing and filtering.

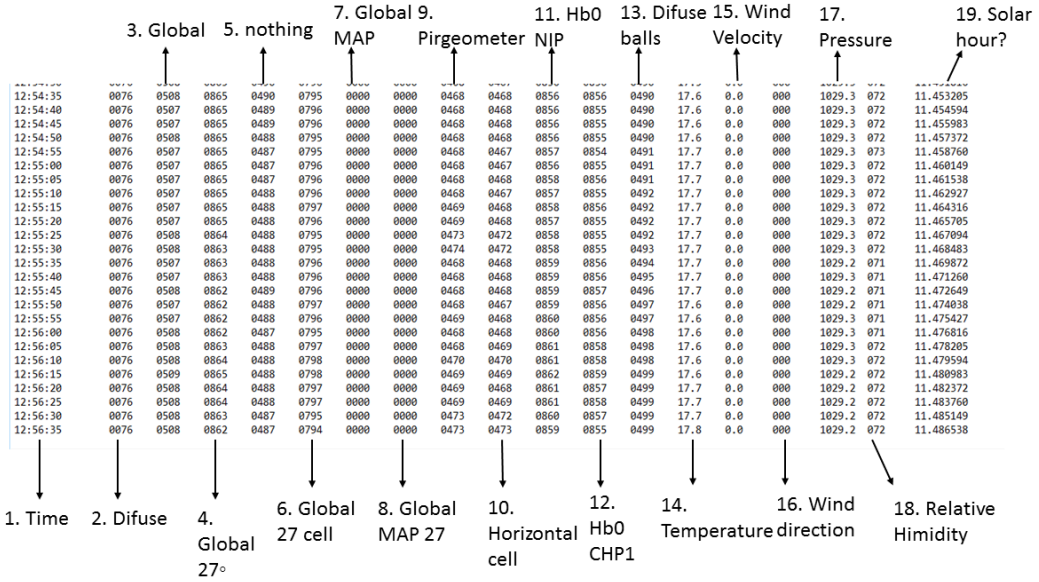


Figure 3.2: Example of the format in `txt`

As the collection of data relies on sensitive equipment prone to measurement errors and biases, the information collected is supervised on daily basis to verify its

integrity, or ammend/correct whenever possible. This is information in provided in separate files (journals of register) by GTER, where possible issues arising with the sensors are recorded for every days. One of this Journal files are provided for every year of recording, stored in MS excel files, and with as many rows as the number of days registered within the year, and a number of columns that represent the different sensors available for data collection. The general convention in the files is that every cell contains value 1 if the variable is reliably recorded, and value 0 if any issue affecting that particular sensor at that specific day makes the information no reliable. These files provide crucial information to appropriately filter the data and construct the processed data files to be fed for ANN training. An excerpt of one of these Journal files can be seen in figure 3.3.

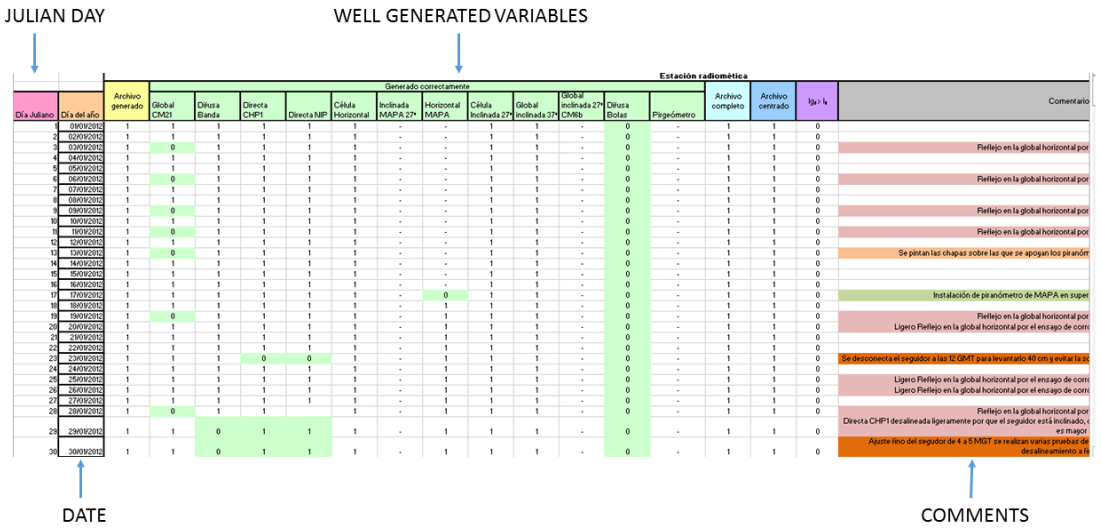


Figure 3.3: Example of the format of the journal

As stated before the the `Neuron Network Toolbox` of `MATLAB` was used for the training and simulation of the networks. It was therefore necessary to adapt the format of raw data above described to a more `MATLAB` friendly format.

It is worth mentioning that ANN training is an intensive computational task both in terms of the size of the data sets employed and the time required for such a training. It is thus of paramount importance to design an efficient and flexible data structure within the `MATLAB` environment to store the data necessary for manipulation of the `Neuron Network Toolbox`. After different considerations for efficiency and flexibility, it was decided to store the data in a cell array of size 3×366 for

every full year of raw data available; that is, every cell corresponding to one day. Each cell contains a structure array with four fields storing all the information that might be of interest, namely:

- *time*: a matrix of 17280 rows and 3 columns. Since the time is put on the form hh:mm:ss, every hour is stored in a row with three columns; the first being the hours, the second the minutes and the third the seconds.

- *values*: a matrix of 17280 rows and 18 columns. This is the data put directly from the txt files into a matrix. Here, all of the variables provided by the radiological station are collected.

- *ok*: a vector of ones and zeros drawn from the journal of the station. Each number corresponds to one usable variable and indicates if the data is reliable (one) or not (zero). It has been taken from converting the `.xls` file of the journal to one `.cvs` file and then reading it with `MATLAB`.

- *zenith*: a vector of 17280 components. It stores the zenith angle of the sun at every hour. This angle of the sun is related to the location, the time of the day and the time of the year. It is calculated using the functions *sunPosition* from Vincent Roy (Copyright (c) 2004).

This structure provides the necessary flexibility to incorporate new data if necessary (as new fields to the structure), or add/remove full-day entries of the register in case specific days are deemed as unreliable or not convenient for training purposes. It allows too a quick and simple localization of data. Each day can be choose easily and the different variables are easy to track.

3.3 Data processing

As it has been mentioned, the technical instruments employed to measure and register the raw meteorological data, are sensitive pieces of hardware prone to faults and disadjustments, that need to be properly treated and processed. As for the raw data files is concerned, three different types of faults have been detected:

- Isolated faulty measurement (Outliers): Some specific sensor may provide, random measurements at specific time instants, which exhibit uncorrelated behaviour with respect to temporal series where it is embedded. Electric

surges or communication failures are the most common causes of this kind of errors. This outlier measurements are easily detected and filtered as they present values significantly different than that of surrounding measurement. In this case the faulty measurement is replaced by a linear interpolation between the previous and next reliable values.

- Faulty data segment: In this case, the sensor operates in faulty mode for time period, providing unreliable information. The fault can be detected as abrupt jumps in the measurements not consistent with the continuous nature of the magnitudes recorded. This gaps of information can be sometimes reconstructed from interpolation of known data, depending on the duration of the fault.

- Missing data segment: In this case, the sensor stops to operate for a period of time, and no measurement or default error value is stored in the record. This error usually arises due to sensor power failures, maintenace operations, etc.

It is clear to see that data with this kind of faults in not the ideal scenario to train an ANN because this faults do not correspond with the reality of the problem being solved. This faults create inconsistencies in the relationships between input data and target data making it harder for the network to *learn* this relation correctly. Most of this faults are contained in the journal of the station and are easily traceable and corrected. The method employed to solve this faults in the raw data have been tracking down the outliers, the faulty data segments and the missing data segment and remove them. These removed data have been replace by new data obtained by the interpolation of already existing data. This way it is assured that the days follows some kind of law in the evolution of its parameters and make it easier for the network to detect it and *learn* it.

One last thing to know is that the days have been also sorted into four categories according to their global radiation distribution through the day:

- type 1: clear days. The distribution is similar to a bell.

- type 2: clear mornings. The first half of the day is clear and the afternoon is a noisy distribution or have abrupt changes.

- type 3: clear afternoons. The first half of the day seems noisy or with abrupt change.

- type 4: not clear at all. There are peaks and noise throughout the day.

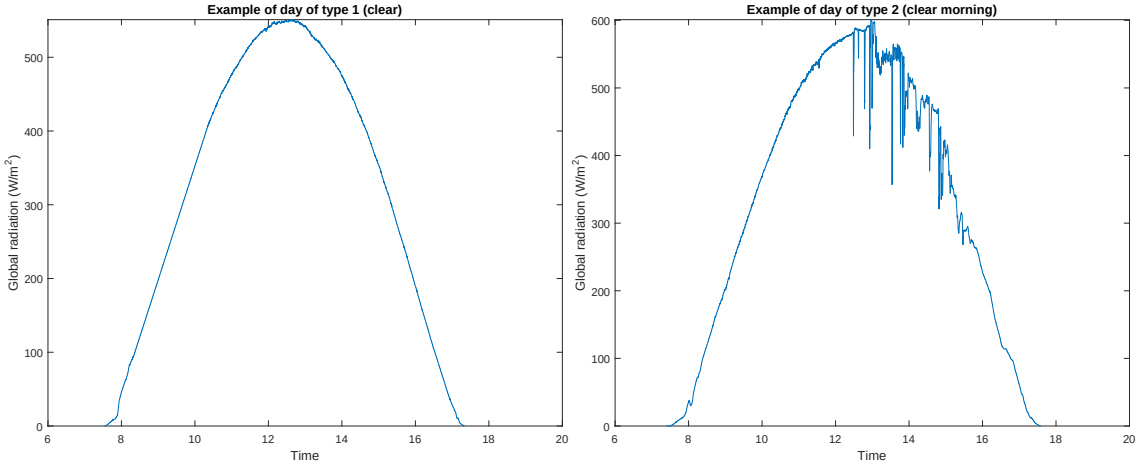
This process has been performed manually, visualizing the shape of the solar radiation distribution. The reason to divide the days in four different types was because in early experiments it was discovered that with the data available the prediction of cloudy parts of the day was not accurate enough while the clear parts were fairly well predicted. It was thought that separating cloudy days from clear days would allow the network to perform better of only clear days were presented. This have a downside, if only clear days are presented the network would only be able to predict accurately if the sky is clear. In order to reduce this disadvantage days were separated in the four types described above. It is necessary for the network to be able to predict solar radiation whatever the weather, but as the weather is very unpredictable this is a big issue concerning the accurate prediction of solar radiation.

It will be seen in the next section that with the days separated this way more accurate predictions can be achieved but in the cloudy parts of the day this prediction will not be reliable. The types of days that will be used for training will be days of type 1, 2 and 3 because even though type 2 and 3 are not perfectly clear they follow a similar pattern to the pattern of a clear day. Days of type 4 are days that do not appear to have any kind of pattern and the data available for the training does not appear to have a clear relation with the clearness of the sky. The only variable available that have something to do with clearness is the diffuse radiation but to predict diffuse radiation it is necessary to have some other variable, for example the presence of clouds in the sky, and this kind of variables are not available in this work. In this work there is proposal for future works regarding this problem. In figure3.4 a visual representation of the 4 different types of day is shown.

In the next section (section 3.4) all the process from the `txt` to the cell array is explained in more detail. But this is not all, once we have the data stored, it is necessary to prepare it to be used in the training of the network.

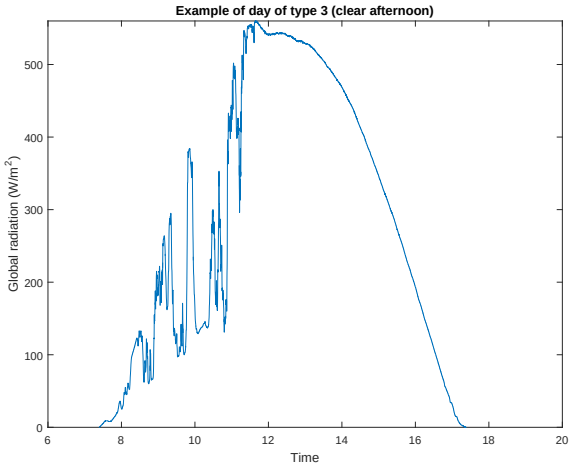
3.3.1 Experiment parameters

The next step is to process this information to obtain the best performance of the networks. This will be done trying different combinations of inputs, the use of past values and smoothing the data with moving means to soften the peaks that appear in the raw data. Days that cannot provide a certain variable are left out depending on the necessary variables. The different experiments will be saved as structures

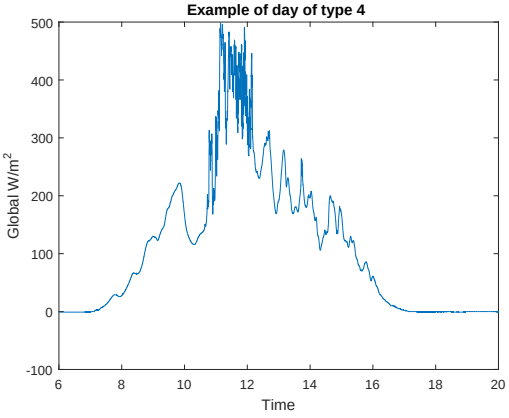


(a) Type 1

(b) Type 2



(c) Type 3



(d) Type 4

Figure 3.4: Different types of days

with different fields that correspond with the experiment parameters, such as the inputs, the average of the moving mean, the window of the prediction or the days to use in the experiment.

This line of work were chosen because it provides all the important information of the experiment in one look and it provides an easy way to work with different experiments. This structure is created by the function experimentDef.mat that ask the user for every parameter asking first if the experiment is new or the purpose is to modify an already existing one. This way, the definition of the experiment parameters is very flexible and accessible and the information is always quickly available. The fields of the Experiments are listed below:

- variables used for inputs (*inputs*)

- average Period for the moving mean for the inputs ($averagePer$)
- average period for the moving mean for the Target ($averagePerTarget$)
- size of the prediction window ($windowSize$)
- years used in the experiment ($years$)
- days used in the experiment ($days$)
- category of the days that will be used in the experiment ($dayType$)
- Past values to add to training as inputs in seconds ($pastValues$)
- name of the experiment ($name$)

Once the principal parameters of the experiments are defined and saved the data is processed using this parameters. There are two stages of processing: the first one exclude the variables that will not be used in the experiment, the data is soften and the values of the night time are left out; the second one put the inputs and the targets into a matrix structure and add the past values if defined in the experiment, getting the data ready to be fed to the network by the MATLABs function `train`.

The first stage of processing is performed by the function `dataPreparationStruct` whose only input is the structure *experiment* and it returns the processed inputs ($dataInput$) and target ($dataTarget$), the days ($days2use$) and the variables (var) used for the training of the nets with the parameters stated in the experiment. Also, it returns two time vectors to plot later the results ($timeInputFinal$ and $timeTargetFinal$). In $dataInput$ the data that will be used as input is saved in a cell array of size (3×1) where every cell contains the data of one year, the same with $dataTarget$, $timeInputFinal$, $timeTargetFinal$ and $days2use$. This is done this way to be able to locate the data and being always aware of what data it is being used in the training. When everything is done and the data is processed all of this variables are saved in a file named `dataSetEXPERIMENTNAME` where EXPERIMENTNAME is the name of the experiment being processed, as the name of the experiment is included in the experiment structure this is a simple way to have everything located. Naming everything based on the name of the experiment helps to not get lost within all the experiments and being aware at any time of what one is doing.

The second stage of the processing takes the data returned by `dataPreparationStruct` and put it in a matrix structure using the function `prepareInTaForTraining`. This

function have some extra parameters as inputs such as if the day of the year is wanted as an input and if the prediction is for the same day or for a day ahead. This function returns the data ready for training. It divides the data into training data and test data. The training data is the 90% of the data contained in $dataInput$ (and $dataTarget$)and the testing data is the 10% that is left.

Finally, with the function `trainANNFinal`, a net is trained using the chosen experiment parameters and the trained net is returned, along with its training parameters and the inputs and the target in form of a matrix. After the training the error of the prediction will be measured with the MSE and the RMSE two times: the first using the training data and the second using the test data to see if the network is able to perform good in data that have not been presented to the network before.

All the functions made to perform this work are listed and explained in the annex and are discussed more thoroughly in the next sections.

3.4 Processing Procedure

3.4.1 From text to MATLAB

This section explains the process followed to go from `.txt` files to the cell array that stores the data in `MATLAB`. This is done to be able to work with the data in MATLAB easily. It is clear that the first thing to do is read the text file and save them with a format compatible with `MATLAB`. In order to achieve this, the script `data_processing_TFG` is used. The objective of this program is to read the text files from its path (they have to be arranged in a certain way) and save it in a cell array with a structure of two fields in each cell: time and values. They are two of the four fields exposed in the previous section. So far, there is a file named `data.mat` which is a cell array containing the time and the variables exactly as they appear in the text files but now they can be easily manipulated with `MATLAB`.

Here is explained how this function works. The first thing to do is to have all the text files separated in three different folders, one per year. The name of the folder should be the year they correspond to. The next thing to note is that he text files have a structured name that helps to track them down and use the name of the file to assign the day and the year to the file. It has been said in previous sections that the data will be saved in a cell array of size 3×366 in order to have a cell for each day. The name of the files follow this structure: `meteo_YEAR_DAY`. This program reads all the files containing the word `meteo` in a chosen folder and goes one by one obtaining

the year and the day from the name of the file and creating the corresponding cell in the cell array `data.mat`. This program takes the first column, which contains the time in the form of $hh : mm : ss$ it separate the three variables and record them as a matrix with three columns in the field time of the cell (`dataY,D.time`). It then takes the rest of the columns and records them in the filed values of the cell (`dataY,D.values`). To see what variables are included in this file see 3.1. Note that the only thing being done here is save all the data available in MATLAB without modifying any of it. It is important to have easy access to all the data even though some of this data will be manipulated or erased later.

As it has been said only two of the four final fields are added in this stage of the process. The next step is to add the field *ok*, indispensable for the further filtering of the data.

As the next step, the field *ok* is added to the cell array. This information is drawn from the journal of the radiostation, which is a MS Excel file. The structure of the journal and what it contains is explained more in depth in section 3.1. As a summary one could say that this journal provides information of the data that is reliable day by day. Using the function `lectura_csv` the values of the field *ok* are drawn from the journal. This function needs to have the sheets of the journal, where each sheet corresponds to a year, separated and saved in a .csv format, since it is easier to work with this type of files in MATLAB. As it is explained in section 3.1 each sheet of the journal contains the information of one year. Inside one sheet (year) every row represents a day and every column represent a different variable (only radiological variables). In every cell of the sheet there is a one or a zero. If there is a one the data is reliable that day and if there is a zero it is not. This information will be vital to the filtering of the data further in the process, particularly in the preparation of the data for training (section 3.5). Only radiological measurements are included in this journal. Weather measurements such as temperature or humidity is not included. This program reads all this information and save it in the corresponding cell dataY,D in a new field called *ok*. This field is a vector of ones and zeros with 17 components. Schematics of the work flow so far can be seen in figure 3.5.

At this point all the data provided by the GTER is included in the cell array `data.mat` in three different fields: time, values and ok. The next thing to do is check if the data is correct or if there are some flaws.

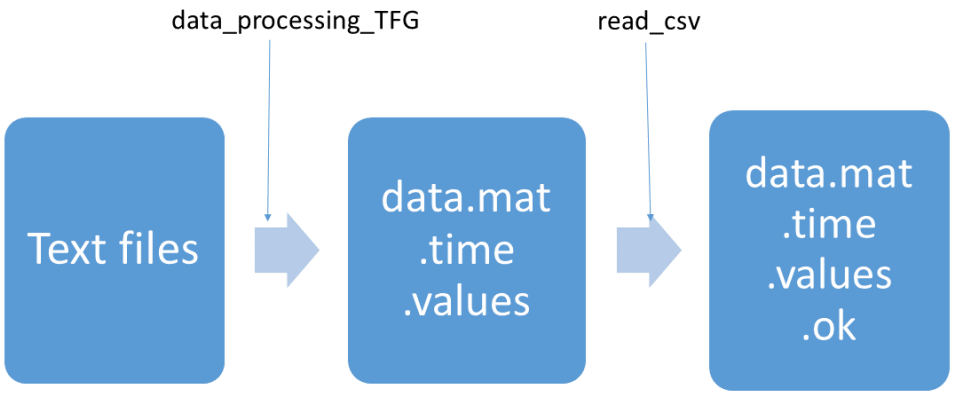


Figure 3.5: Work flow from text files to cell array with three fields

3.4.2 Data validation and normalization

At this point the data is available for its usage in `MATLAB` or so it seems. It is worth to mention that working with this data one realize that there are some days that are faulty or have missing data and need a little adjustment. This was discovered because some days did not have the number of rows they were supposed to have. As said above, the values of the magnitudes are measured and saved every five seconds. So, in a day of 24 hours that leaves 17280 entries for every magnitude and there was some days that did not have this exact number. Digging in the data it was discovered that repetition of a time in some days occurred. The days affected by this are random as well as the times that are repeated. This was a milestone in this work because having days with different structures can be a difficult thing to deal with, so it was decided that the first thing to do was to adjust the times that were wrong along with the information associated with this times in order to have the same structure (17280 rows, one every five seconds) in every day.

This faulty days appeared in tow different forms. There were days that had more than 17280 rows and there were days that had 17280 rows as they should but they did not contain every time right as there was times that did not appear or times that were repeated. The first form was detected pretty straightforward as it easy to track down matrices with more rows than they should, so the first thing that was done was

to point out the exceeding rows and delete them from the file. The functions used are `Test_Data` that detects the days that are wrong showing a message through the command window. The results were recorded manually in three different vectors, one per year.`Which_hour` which detects the hour that is repeated and then removes it from the cell, schematics of the process can be seen in figure 3.6 a).

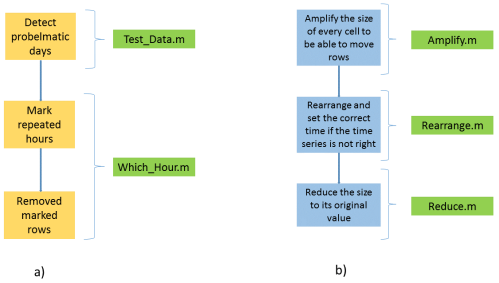


Figure 3.6: More rows solution

The second form of faulty days was detected later when working with the data. There were errors that made no sense and it was discovered looking into the data that some days did not have the right time series. There are days that have some hours missing but they have 17280 entries as they should have. Looking again in detail into the data is detected that, is some cases the clock goes backwards at some points i.e. it would past from 12:00:00 to 11:20:05, a time that have already been saved. To tackle this problem and try to avoid similar undetected problems that might come up later the process to follow will be check that the time is right in every day and rearrange those that are wrong. The process schematics is shown in figure 3.6 b).

As this was the second form of faulty days it was decided to use a program to check all the days and their time series. The program reads every day and check if one row and the next one are five seconds apart. If they are not the next row is moved one row down and a new row is created and the right time is assigned. In order to be able to manipulate rows, every cell was previously amplified to have room to move the rows. When every time series is correct in the cell all the faulty parts are located below the 17280 so the last thing to do is to remove the exceeding rows and return the size of the cell to 17280 rows.

This way it is assured that all days follow the same time series and makes it easier to work with them and to manipulate the data. The only problem that there is left is that there is missing information in the new rows created.So, the last

thing to do is fill up this gaps in the data. To do this, the data is interpolated using the existing data as interpolation nodes. The function that does this is the function `data_interpolation` that uses the `MATLAB` function `interp1` to interpolate the data. It fills up variable by variable using all the known information of one day, so interpolations are done day by day, variable by variable.

To end this part of the processing, the last thing to do is add the last field to the cell array, which is the field *zenith*. The zenith angle is a variable that determines the position of the sun in the sky. So, this variable is related to the location, the time of day and the time of the year. For this it will be used as a very important input in the experiments of section **??**. The zenith is calculated using a MATLAB function developed by Vincent Roy ([7]) which contains the solar equations to obtain several parameters but in this work only the zenith will be used. The function `sun_position` needs inputs such as the location and the time. The time input is a structure with the fields year, month, day, hour, minute, second and UTC. As there is information of the time available for every sample, this inputs are already included in the file `data.mat` so it is very easy to obtain the zenith angle with this equations. The location is also required in the form of a structure with the fields of longitude, latitude and altitude. The location of the station is in the coordinates: $37.38°N$, $5.99°W$ and in $8.26m$ altitude (information taken from [11]).

To finish this section a summary of the process is shown. Read the data from `.txt`, add the field ok from `.csv`, treat the data and add the field zenith. Finally a file `data_fixed_filled` is obtained, which contains a cell array where every cell is a different day determined by the year and the number of the day. Every cell consist of a structure array of 4 fields: time, values, ok and zenith. The field time indicates the time of every row in the form of a vector $[hhmmss]$. The field values is a matrix of 18 columns where every column is a different variable and every row is a different hour of the day. The field ok is a vector that contains 1 and 0 indicating the goodness of the different variables that day. The field zenith associates the zenith angle of the sun for the different times of the day. This file is the one that will be the departure point of the data that will be filtered in order to use it in the training process.

3.5 Preparing the data for training

This section explains the procedure to prepare the data to train the ANN, departing from the file `data_fixed_filled`. The process is summarize as follows: first, the experiment parameters are defined; later the data is process, chosen the desired variables and applying moving means to soften it; finally it is rearranged in a matrix structure and it is used to train the network.

3.5.1 Setting the experiment parameters

The first thing to note is that it is very important to define the parameters of the experiment that will be carried out in order to process the data accordingly. To define these parameters that are the ones that determined the way the data will be processed and prepared the function `experimentDef.mat` is used. The parameters of the experiments are:

- *inputs*: in string format, it determine the variables to use in the prediction separated by one blank space. For example *inputs 'Zenith Humidity Global'* will use the zenith, the relative humidity and the global radiation as inputs.

- *averagePer*: This parameter indicates the period (in seconds) of the moving means to use to soften the data and avoid peaks.

- *averagePerTarget*: In order to have more training data, the target is also calculated with moving averages. This parameter indicates the period (in seconds) of this moving averages. For example if we want to predict the hourly mean global radiation we will set this parameter to 3600 seconds.

- *windowSize*: This parameter indicates the window of the prediction in seconds. For example if we want to predict the global radiation in two hours from the current time instant this parameter will be 7200.

- *years*: A vector of one, two or three components representing the years to be used to extract the data.

- *days*: A vector of two components, being the first one the first day of the first year and the second one the last day to use of the last year, all days in between are used.

- *pastValues*: to indicate if we want to use values of the inputs in $t - pastValues$ as more inputs. It is introduced in seconds. For example, if we want to use the data in $t - 5min$ and the data in t, past Values is set in 300 seconds.

- *dayType*: a vector indicating the day types to use in the experiment. See section 3.1 for more information bout day types.

All these parameters can be defined using the function `experimentDef` that will ask the user to define the parameters and will save it as a structure named *Experiment* saved in a file with the name of chosen by the user. Having the parameters recorded in one file is very helpful to keep track of the experiments that are being carried out or will be carried out. Loading the file of the experiment and typing `Experiment` in the command window will provide all the parameters at once. It is an easy and quick way to have a look at the experiment under review. Once the parameters of the experiments are defined the nest thing is to process the data so that experiment could be carried out.

3.5.2 Processing for training

With the function *dataPreparationStructTime* the data process according to the experiments parameters. The only input of this function is the structure *Experiment*. This function apply moving means to the inputs and target variables defined in the experiment with the defined periods. The application of moving means is used because, as the raw data is taken instant by instant from the sensors and this sensors are not perfect, there are some discontinuities that do not reflect the reality of the evolution of the variables. The use of moving means makes the variables much more soft and more similar to reality. Besides this, it also helps to reduce the effect of outliners points that might be present due to some malfunction of the sensors.

This function takes the raw data form the file `data_fixed_filled` and take only the days that have the field ok equal to one. This function removes the data that was collected at night as it does not affect solar radiation. The outputs are:

- *dataInput*: a cell array with three components, one per year where the inputs selected previously are stored in a matrix where every column is a variable, being the last one the day of the year.

- *timeInput*: cell array with three cells where every cell is a vector of times that divide that makes the hour match the variables in *dataInput* in order to further analysis.

- *dataTarget*: a cell array with three cells, one per year. Every cell consists of a vector with the target variable, matching the cell *dataInput*. The first row of the targets is the target for the first row of inputs and so on.
- *timeTarget*: the same as *timeInput* but for the target.
- *days2use*: cell array of three cells, one per year indicating the days that are contained in *dataInput* and *dataTarget*.
- *var*: a string vector that contains the variables used.
- *pastValues*: the value of this parameter, because is used in further processing.

Once we have the data soften and with the days of interest separated by years, it is necessary to put it all in a single matrix in order to use the `MATLAB` *train* to train the net (the NNTrain Toolbox is explained in **??**). In order to do this the last processing function is perform in the cell arrays *dataInput* and *dataTarget*. This last function is the function *perpareInTaForTrainig* that returns the Input and Target for the training and the vector of time to plot the results versus time. The inputs of the function are *dataInput*, *dataTarget*, *timeInput*, *timeTarget* , *pastValues*, *days2use* and two flags: *dayOfYear* which is set to 1 if the day of the year is desired as an input and *dayAfter* which is set to 1 if the target is defined in the next day ($t + 24hours$). This function is the one that add all the necessary parameters that have not been used yet, such as pastValues or the day of the year, and prepare the matrix structure to be ready to be fed to the network without any more processing. It was decided to divide the final processing in two stages (*dataPreparationStructTime* and *perpareInTaForTrainig*) because this way is easier to follow the process followed by the data that will be used for training. Besides, the problems that this functions solved are different. *dataPreparationStructTime* deals with the location and processing of the data that will be used for training and *perpareInTaForTrainig* deals with the rearrangement of the already processed data to be fed directly to the net.

Finally with the Input and Target matrices, the function *trainANNInTa* can be used to train a network. Changing the script in *trainANNInTa* allow to customize the network using the `MATLAB` functions of the *Neural Networks toolbox* (see **??**).

The function *TrainApp* nets can be created from the data sets of dataInput and dataTarget because it implement the functions *prepateInTaForTraining* and *trainANNInTa*, asking the user for the necessary parameters.

Having explained the processing procedure and having stated the terms of the preparation of the data for the experiments it is time to move to the experiments that have been performed and the conclusions that can be drawn out of these experiments, the experiments discussion is preceded by a description of MATLAB's ANN toolbox and the variables that will be used for the experiments.

Chapter 4

Experiments

Given all the theoretical background and the preparation of the data it is time to deal with the experiments to discover how a *Multilayer Perceptron* behaves with the data available. In the next sections several experiments will be discussed. Different inputs configurations, different scopes and different types of day will be tested to draw some conclusion. For the next experiments only days of type 1, 2 or 3 or a combination of these types will be used. The table 4.1 shows the number of days that can be used for training per year and per day type.

Year	**Type** 1	**Type 2**	**Type 3**	**Type 4**
2012	89	21	23	233
2013	94	23	21	227
2014	32	14	10	309
Total	215	54	58	759

Table 4.1: Number of days

4.1 Variables

As stated in chapter 3 the variables provided by the radiological station are different types of global and diffuse radiation, plus some meteorological data. As there are several types of global and diffuse radiation but they are in essence the same variable only the total global and total diffuse radiation will be used for the experiments. Regarding the meteorological variables, the ones that seem well recorded are the temperature, the relative humidity and the atmospheric pressure. Even though the wind speed and wind direction are valuable variables according to some of the works discussed in the introduction, they are unfortunately not useful as they appear always with a value of zero in the measurement of the station. So in the next experiments the variables to use will be: zenith angle (Z), temperature(T), relative humidity(H), pressure(P), day of the year (DOY), diffuse radiation(D), global radiation (G).

4.2 Neural Network Toolbox

As said previously on this work the tool that will be used to create and train the networks is MATLAB's neural network toolbox (NNTool). This toolbox allow the user to use simple functions to create and train different neural networks and control the training parameters and the structures of the networks. In this work, networks created with the function `feedforwardnet.m` will be used. This function creates a feedforward network with the desired structure. The only input is the a vector with as much components as hidden layer desired, where every component is the number of neurons of the layer. Figure 4.1 shows a network created with this function with an input [10 5].

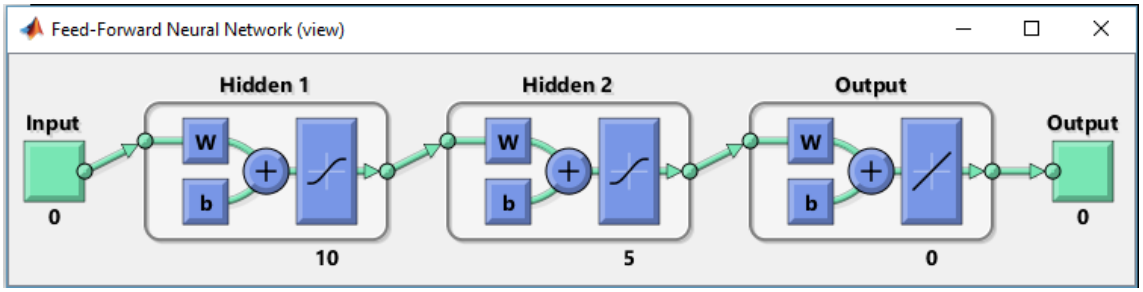


Figure 4.1: Network created with `feedforwardnet.m`

Once the network is created, the training parameters can be change varying he

different fields of the network. Some of this fields are:

- net.inputs1.processFcns and net.outputs2.processFcn: this field determine the process function that will be applied to the inputs in order to normalize it before feeding it to the net. In this work `mapminmax` will be used. mapminmax processes input and target data by mapping it from its original range to the range [-1 1] [13].

- net.divideFcn: it defines the function used to divide the training data. The default setting is `dividerand`, which divide the data randomly. This is the setting employed in this work.

- net.divideMode: This property defines the target data dimensions which to divide up when the data division function net.divideFcn is called [13].

- net.divideParam: .trainRatio, .valRatio, .testRatio. This parameter define the percentages of the training data used for actual training, validation and testing. In this case it will be 70%, 20% and 10% respectively.

- net.trainFcn: it defines the training algorithm to be used by the function `train`. In this work the Levenberg-Marquadt algorithm will be adopted. For this, the `trainFcn` is set as `trainlm`.

- net.performFcn: this defines the parameter to be employed to measure the performance of the net. Due to the limitations of the Levenberg-Marquadt method, only the mean square error (MSE) or the sum square error (SSE) can be chosen. Since mse is one of the most popular method to asses the performance of a neural network, it will be adopted for this work too. Rooted mean squared error (RMSE) can be obtained easily from the MSE ($RMSE = \sqrt{MSE}$). RMSE has the same units as the data, so is easier to interpret.

There are much more features that can be customize and they can be consulted in the MATLAB help documentation ([13]).

Now that there is a network structure and the training parameters are set there is one middle step before starting the training. Using the function `configure` with the network, the inputs and the outputs of the training as inputs the function will define the input layer and the output layer preparing the network for best performance when trained.

Finally, the network can be trained. The function `train` has the network, the inputs and the targets meant to the training as inputs and it returns the trained network and the training parameters. The inputs and targets are normalize before training according with the function defined in the parameter net.inputs1.ProcessFcns and net.outputs2.processFcns. The data is randomize too to avoid overfitting. During the training, a GUI window pop up, summarizing the state of the training (figure 4.2)

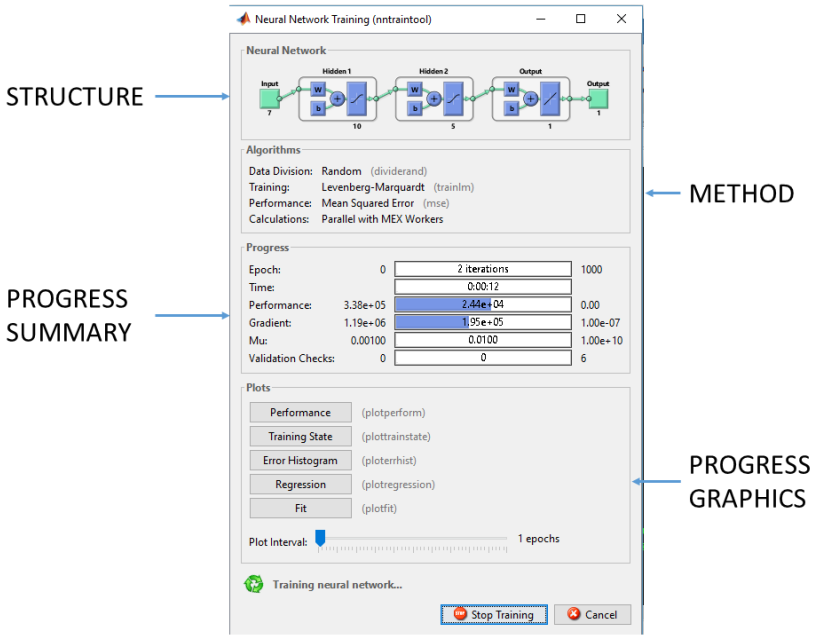


Figure 4.2: GUI of the training tool

4.3 Experiments

In this section different experiments will be carried out and different aspects will be compared. The first experiments serve to see the effect and the behavior of the network with the different types of days. Later some experiments will be carried out to study the effect of the different sets of variables, different time windows will be put to test too and finally different architectures will be tested in order to find the structure of the net that meets a compromise between performance and computation time.

	Inputs	hiddenLayerSize	dayType	pastValues	Error Training		Error Test	
					MSE($(W/m^2)^2$)	RMSE(W/m^2)	MSE($(W/m^2)^2$)	RMSE(W/m^2)
Ex1	ZHG+DOY	10	1,2,3	Yes	304.68	18.56	2.19e+3	46.75
Ex2	ZHG+DOY	10	1	Yes	25.69	5.069	34.28	5.85
Ex3	ZHG+DOY	10	2	Yes	671.81	25.92	582.50	24.13
Ex4	ZHG+DOY	10	3	Yes	807.57	28.42	2.57e+03	50.73

Table 4.2: Experiments to compare day types

4.3.1 Different types of days

In this first section of experiments, the different types of days will be compared. There are 4 experiments that will consist of the next set of inputs: the zenith, the relative humidity and the global radiation in t and in $t - 5min$ the window of the prediction will be 1 hour and the value to predict will be the mean global radiation in that hour.For this first experiment, all days of type 1, 2 and 3 will be used. This leaves the next parameters in the experiment:

- Inputs: *'Zenith Humidity Global'* + the day of the year
- averagePer: 300
- averagePerTarget: 3600
- years: [1 2 3]
- days: [1 365]
- pastValues: 300
- dayType: [1 2 3] for experiment1, days of type 1, 2, and 3 for experiment 2, 3, and 4 respectively

The net has the following architecture: multilayer perceptron with 3 layers (input layer, hidden layer and output layer). The architecture can be seen in figure 4.3. There are 7 inputs, 10 hidden neurons and 1 output. The results can be consulted in table 4.2. The performance of the network is shown twice in the table; the first time (column Error Training) is the performance with data that have already been fed to the net and the second time it appears it indicates the performance that the network have not seen before (column Error Test). The training data is the 90% and the test data the 10% that there is left.

Some clear conclusions can be drawn from table 4.2: the network predicts much better days of type 1 (clear days) than days with some noise, obtaining a *RMSE* of

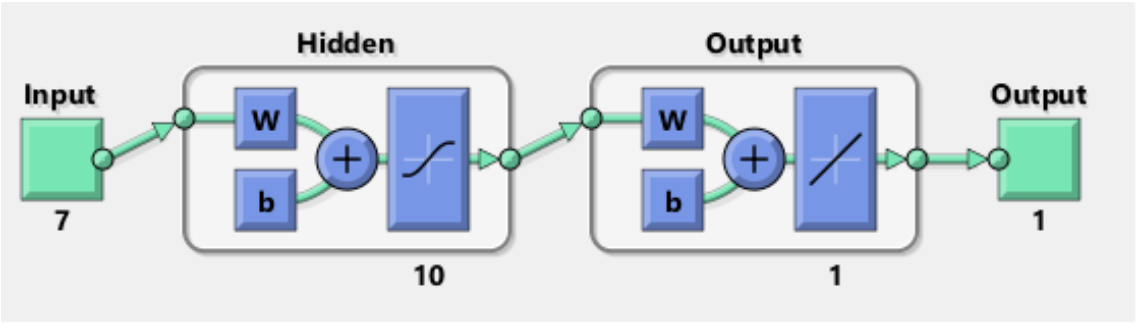


Figure 4.3: Architecture of the net for Experiment 1

5.85 W/m^2 with only clear days against a *RMSE* of 24.13 W/m^2 and 50.73 W/m^2 for days of type 2 and 3. It can be observed too that, usually, the performance applying the network to the test data is poorer than the performance with the training data. This is because the network has seen before the training data, in fact it has *learnt* to predict from that data, whereas the test data is new to the network. The case of Ex3 where the performance with the test data is better is rare.

Now, let's have a look onto some graphics that show random points that have been predicted from the test data and analyze them. In figure 4.4 50 random pairs of prediction and target are plot for each experiment. In the first graphic, experiment 1 is represented but the graphic does not lead to think the error would be high, as the prediction seem to fit in most of the points, but there are some points that are far apart between the prediction and the target. This points probably belong to a cloudy part of a day, where the network does not perform very well. In opposition, in the second graphic a much better fit between prediction and target is achieved with a *RMSE* of 5.85 W/m^2. Experiments 3 and 4 can be seen in figure 4.4 in the third and fourth graphic, they are similar to experiment 1.

Finally in this section let's see how it would be the prediction against the targets of a day. Figure 4.5 shows how the predictions are good in clear parts of the days and not as good in noisy parts of the graphics.

Now that has been stated that the network predicts better clear days, the next thing to do is to try different sets of inputs and see the effect in the performance of the networks.

4.3.2 Different sets of inputs

In this section different sets of inputs are tested. In table 4.3 there is a summary of the experiments.

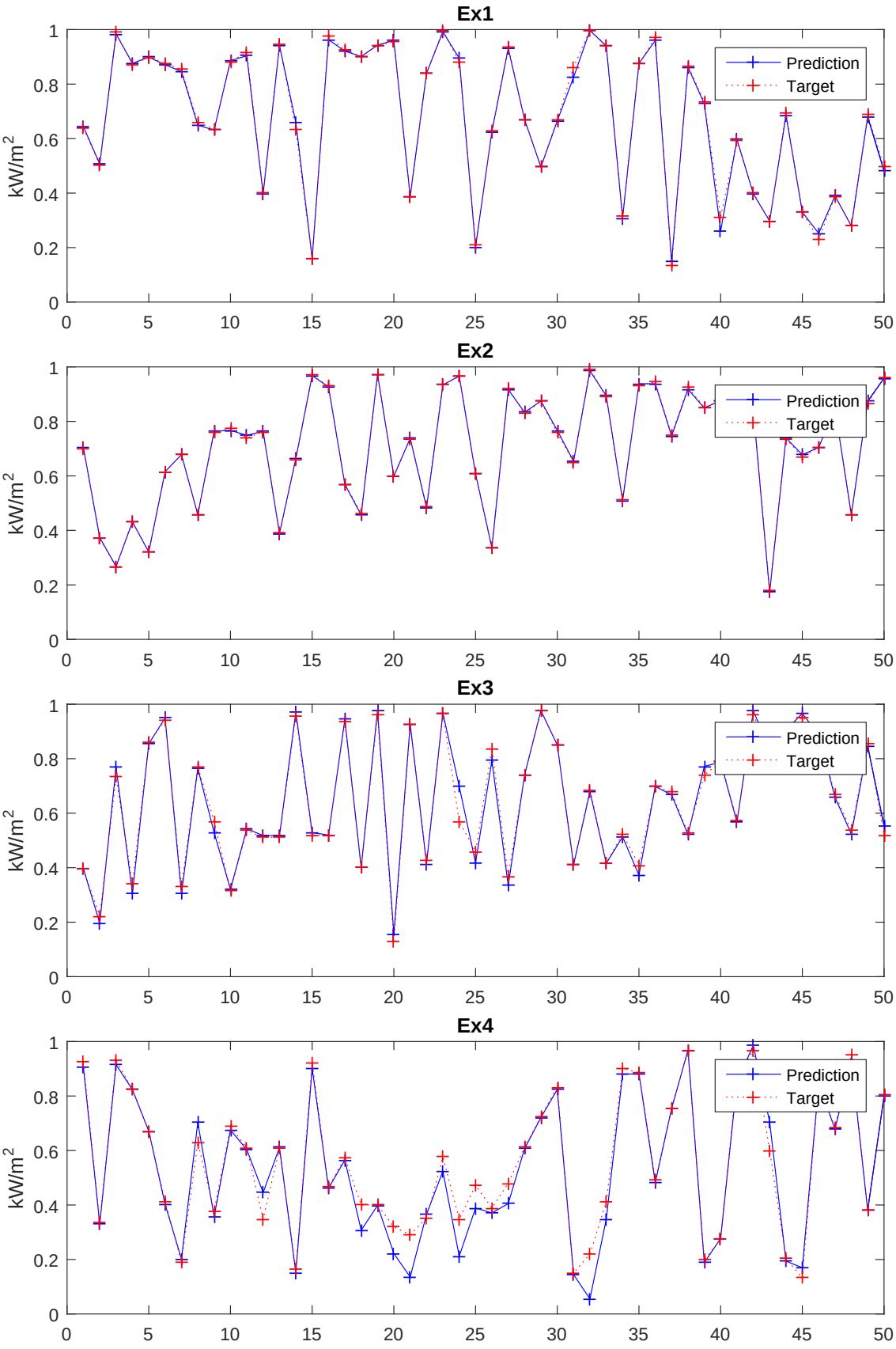
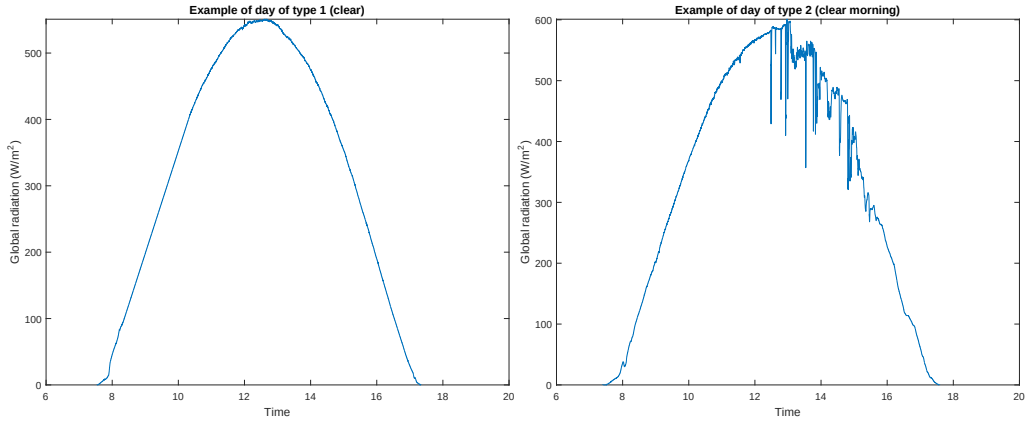
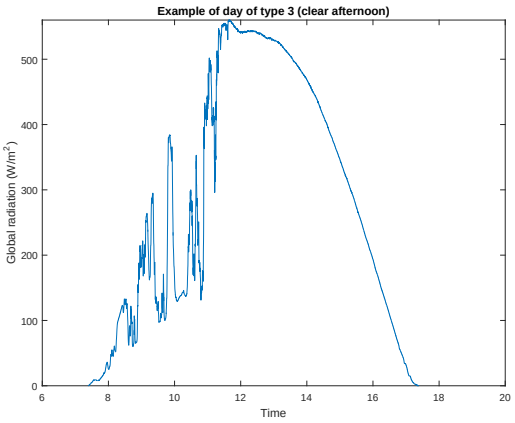


Figure 4.4: Comparison between prediction and target (Experiments 1, 2, 3 and 4)



(a) Type 1 (b) Type 2



(c) Type 3

Figure 4.5: Comparison between types

	Inputs	hiddenLayerSize	dayType	pastValues	Training Error		Test Error	
					MSE($(W/m^2)^2$)	RMSE(W/m^2)	MSE($(W/m^2)^2$)	RMSE(W/m^2)
Ex1_2_1	G+DOY	10	1	No	5.91e+03	76.87	6.19e+03	78.68
Ex1_2_2	G+DOY	10	1	Yes	189.60	13.66	191.30	13.83
Ex2_2	ZG+DOY	10	1	Yes	26.31	5.13	26.28	5.126
Ex3_2	ZHG+DOY	10	1	Yes	26.30	5.12	30.57	5.53
Ex4_2	ZHG	10	1	Yes	27.52	28.42	30.27	5.52
Ex5_2	ZHPG+DOY	10	1	Yes	25.07	5.00	34.72	5.89
Ex6_2	ZHPG+DOY	10	1,2,3	Yes	334.42	18.28	3.76e+05	613.19
Ex7_2	ZHDG+DOY	10	1	Yes	36.27	6.022	396.76	19.92
Ex8_2	ZHDG+DOY	10	1,2,3	Yes	399.83	19.99	4.79e+05	692.10
Ex9_2	ZHDPG+DOY	10	1	Yes	37.09	6.09	127.19	11.28
Ex10_2	ZHDPG+DOY	10	1,2,3	Yes	370.79	19.26	6.67e+03	81.6701
Ex11_2	HDPG+DOY	10	1,2,3	Yes	1.02e+03	31.93	2.21e+03	47.01
Ex12_2	ZHPG	10	1	Yes	27.25	5.22	38.25	6.19
Ex13_2	ZTHG + DOY	10	1	Yes	25.31	5.03	258.64	16.08

Table 4.3: Experiments with different sets of inputs

In this experiments the network use is the same and with the same parameters as in section 4.3.1. Since it is the same network the errors are of the same order of magnitude in the experiments with a similar sets of inputs.

First it is worth noting that the difference between Ex_1_2_1 and Ex_1_2_2 is exclusively the inclusion of past values in the inputs. It can be seen that this addition improves the behavior of the network. Continuing with Ex_2_2 it can be seen again that the inclusion of the Zenith as inputs for days of type 1 improves also the performance. Again, the addition of days of type 2 and 3 worsens the performance, nevertheless the *ultimate objective* of this work is to predict the solar radiation whatever type of day it is and that is why they are also tested. It is clear to see why the inclusion of the zenith angle and some past values improve the behavior as the zenith is directly connected with the position of the sun on which solar radiation depends and the past values gives some context information regarding the part of the day. With the relative humidity and the pressure the relation is not so direct but, clearly, they affect solar radiation, for the performance improves with their inclusion. Comparing Ex_5_2 with Ex_12_2 and comparing Ex_3_2 with Ex_4_2 it can be seen that the inclusion of the day of the year improves slightly the performance of the network. Looking at the table it can be seen too that the experiments that contain the diffuse radiation in the inputs tend to have a bigger difference between the error in the prediction of the training data and the prediction of the test data. This indicates that when diffuse radiation is used, the net becomes less general.

A visual representation of some of these experiments is shown in figure4.6. The experiments represented, again with 50 random pairs of prediction and target, are Ex_5_2, Ex_7_2, Ex_10_2 and Ex_13_2. Looking at the table is clear to see that Ex5_2 is one of the best in this section with a *RMSE* of $5.89W/m^2$ in the test data whereas Ex_10_2 is one of the worst with a *RMSE* of $81.67W/m^2$ nevertheless it can be observed that this experiment is not a bad predictor in many points but is terrible in a few, this might be caused because of the nature of the *RMSE*.

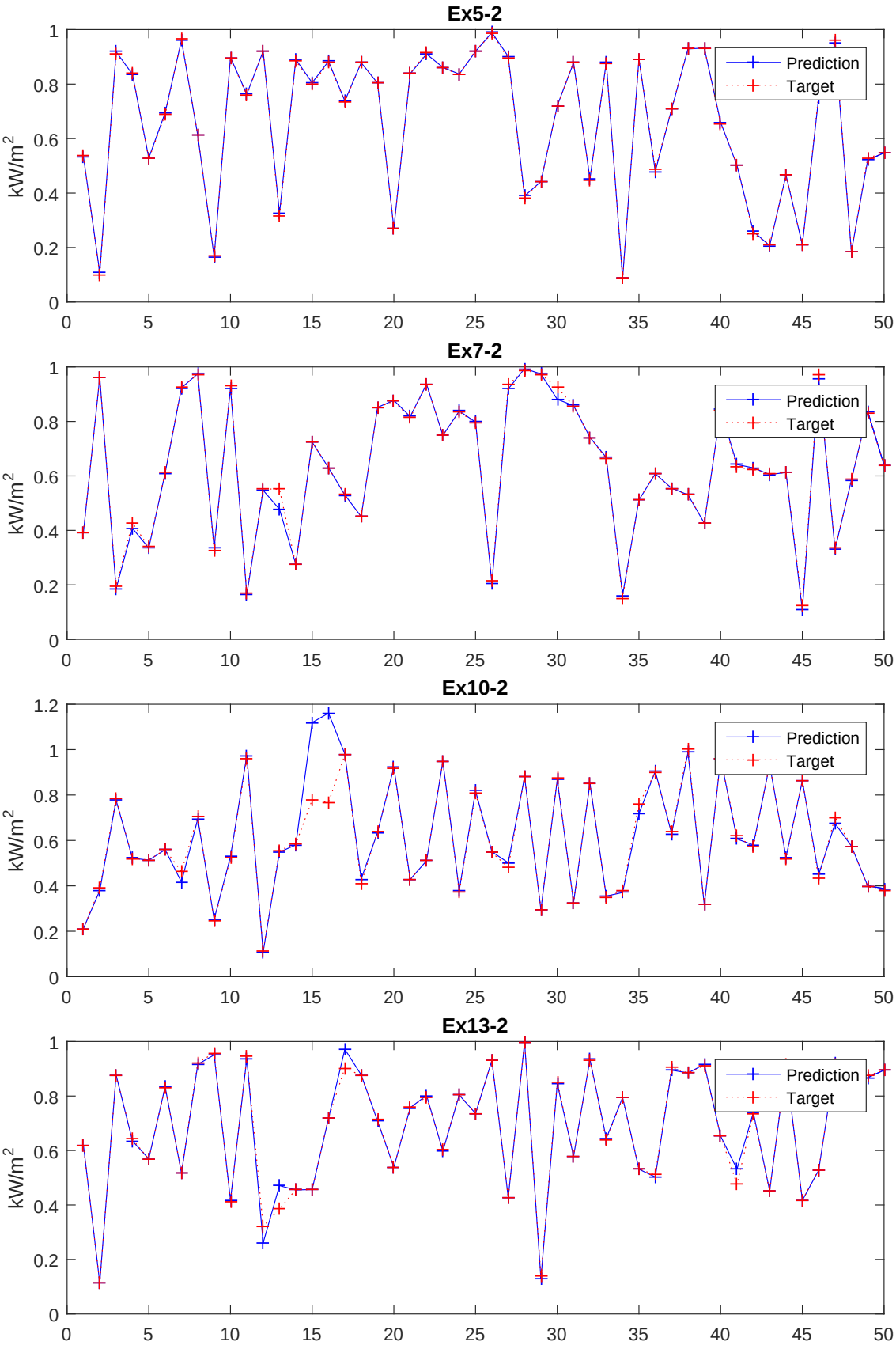


Figure 4.6: Comparison between prediction and target (Experiments 5_2, 7_2, 3_2 and 4_2)

	Inputs	hiddenLayerSize	dayType	pastValues	Training Error		Test Error	
					MSE($(W/m^2)^2$)	RMSE(W/m^2)	MSE($(W/m^2)^2$)	RMSE(W/m^2)
Ex1_3	ZHPG+DOY	20	1	Yes	21.78	4.67	56.10	7.49
Ex2_3	ZHPG+DOY	50	1	Yes	19.91	4.46	626.17	25.23
Ex3_3	ZHPG+DOY	80	1,2,3	Yes	191.91	13.85	4.98e+04	223.24
Ex4_3	ZHPG+DOY	100	1	Yes	15.78	3.97	1.08e+04	103.99
Ex5_3	ZHDG+DOY	10-5	1	Yes	20.41	4.518	35.6273	5.9689
Ex6_3	ZHPG+DOY	20-10	1,2,3	Yes	211.234	14.534	3.520e+03	59.331
Ex7_3	ZHPG+DOY	15-7-2	1	Yes	28.384	5.32	3.9865e+04	199.663
Ex8_3	ZHDG+DOY	40	1,2,3	Yes	282.671	16.813	969.6988	31.14
Ex9_3	ZHDG+DOY	20-10	1	Yes	14.93	3.87	803.5615	28.3

Table 4.4: Experiments with different structures

As a conclusion of this section, the experiments show that the diffuse radiation reduces the capacity of the network for generalization and that the variables that seem to be better for the prediction of global solar radiation are meteorological data such as relative humidity, pressure and temperature. So far, only different types of day and different inputs have been discussed not paying attention at the structure of the net which is a very important characteristic if not the most important in this kind of artificial neural networks. In the next section, different structures will be tested.

4.3.3 Different structures

In this section different structures for the MLP are tested. Table 4.4 summarizes the results of different structures with different number of neurons and different number of layers.

The first thing to note in this table (table 4.4)is that the errors obtained for the test data are much higher than the errors obtained for the training data. Comparing this with the experiments of the previous section, the errors obtained for the training data are, indeed, lower as they are supposed to be when the number of neurons is increased. On the other hand, the error obtained for the test data prove that these networks are not able of generalization. When the networks are fed with data that has not been presented previously, it is not capable of making a good prediction. This might be due to over fitting. When the network is trained with too much data that is very similar it tends to be less sensitive to change and, therefore is not able to predict accurately data that has not seen before. This networks have been fed with all the data available for training, which is a lot of data and ti may have led to the undesirable overfitting. It can be observed that the only experiment that reduces the error is Ex3_3 that have the same inputs as Ex6_2 but the network

	Inputs	hiddenLayerSize	dayType	pastValues	Training Error		Test Error	
					MSE($(W/m^2)^2$)	RMSE(W/m^2)	MSE($(W/m^2)^2$)	RMSE(W/m^2)
Ex1_2_3	ZHPG+DOY	20	1	Yes	22.74	4.7691	45.04	6.7118
Ex2_2_3	ZHPG+DOY	50	1	Yes	18.0512	4.2487	4.6103+03	67.89
Ex3_2_3	ZHPG+DOY	80	1,2,3	Yes	193.469	13.90	2.44e+04	156.287
Ex4_2_3	ZHPG+DOY	100	1	Yes	15.58	3.9478	8.91e+03	94.37
Ex5_2_3	ZHDG+DOY	10-5	1	Yes	20.7508	4.555	134.011	11.5763
Ex6_2_3	ZHPG+DOY	20-10	1,2,3	Yes	206.90	14.38	6.599e+03	81.2375
Ex7_2_3	ZHPG+DOY	15-7-2	1	Yes	24.954	4.99	2.849e+04	168.80
Ex8_2_3	ZHDG+DOY	40	1,2,3	Yes	231.76	15.22	8.17e+03	90.41

Table 4.5: Experiments with different structures and less data for training

used have more neurons. Here, the error is reduced both in the training and in the test compared against the same experiment with a network with less neurons. This experiment is different from the rest because it includes three types of days and it means that the training data is more diversified than in the experiments with only days of type 1 that are more alike between them. The lack of variety in the data in the experiments with days of type 1 may have had led to overfitting.

Here there is another table (table 4.5) where the networks are trained with only a half of the total data available for training. Using only half, some information may be missing but it may be enough to provide a fairly accurate prediction and avoid overffiting. From the table there is some things to conclude. It can be seen that this table compared with table 4.4 provider poorer results except in the networks with 80 neurons (Ex_2_3) and 100 neurons (Ex_4_3). So, for networks with a number of neuron above 80 there is overfitting if the whole set of training information is used for training the network. In the rest of networks, that contains less neurons and therefore, less interconnections the downsizing of the training data affects in a negative manner, making the networks achieve less accuracy.

In figures there is a visual representation of 50 random points from the train data. At each point there is a comparison between the target and the prediction. Figure represents the experiments that uses the whole set of training data and in figure ,the experiments that use the data downsized is represented. In this figures is clear to see the that in the networks with few neurons the performance worsens and that the networks with high number of neurons improve. As said when the tables were discussed.

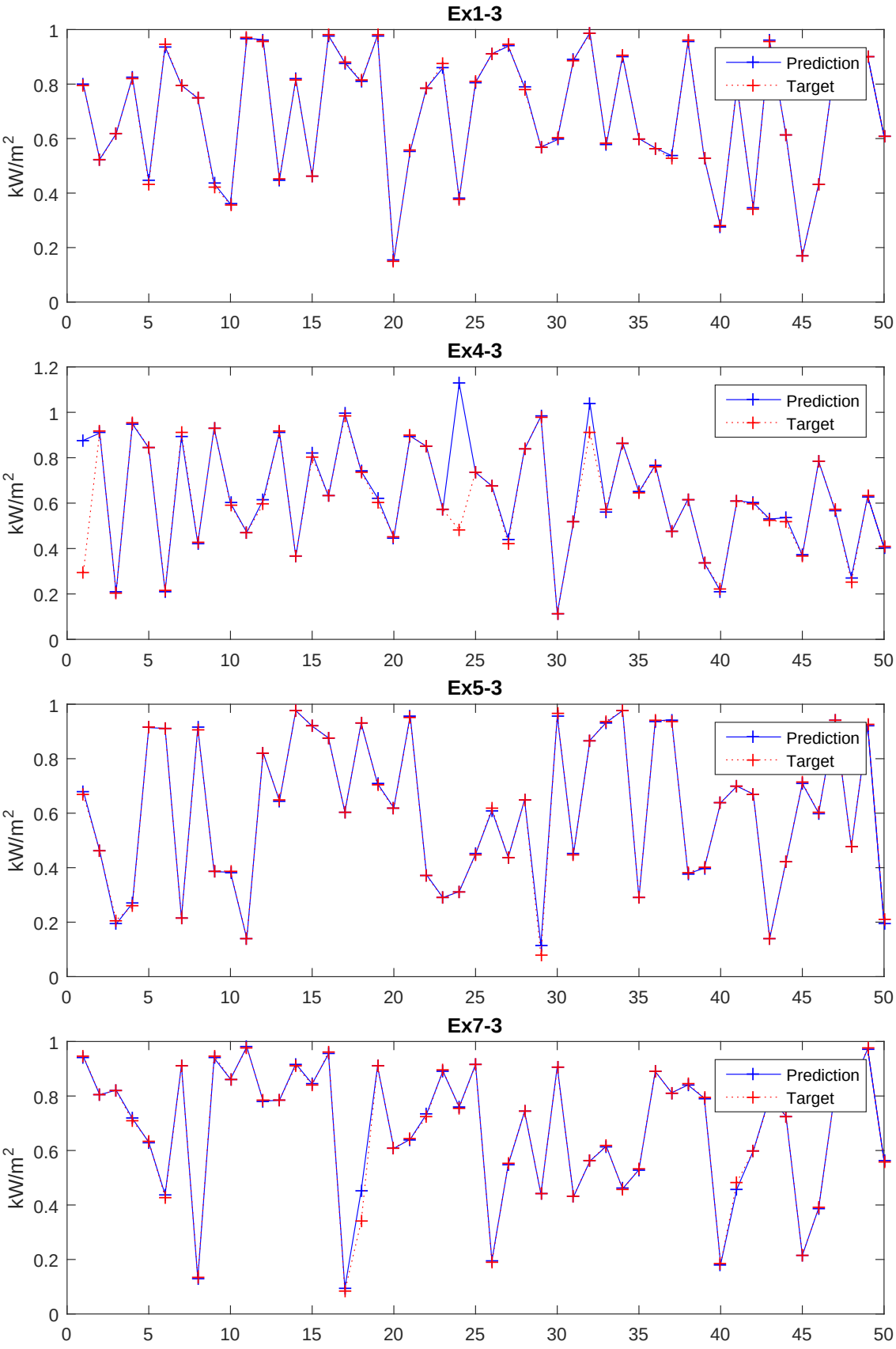


Figure 4.7: Comparison between prediction and target (Experiments 1_3, 4_3, 5_3 and 7_3)

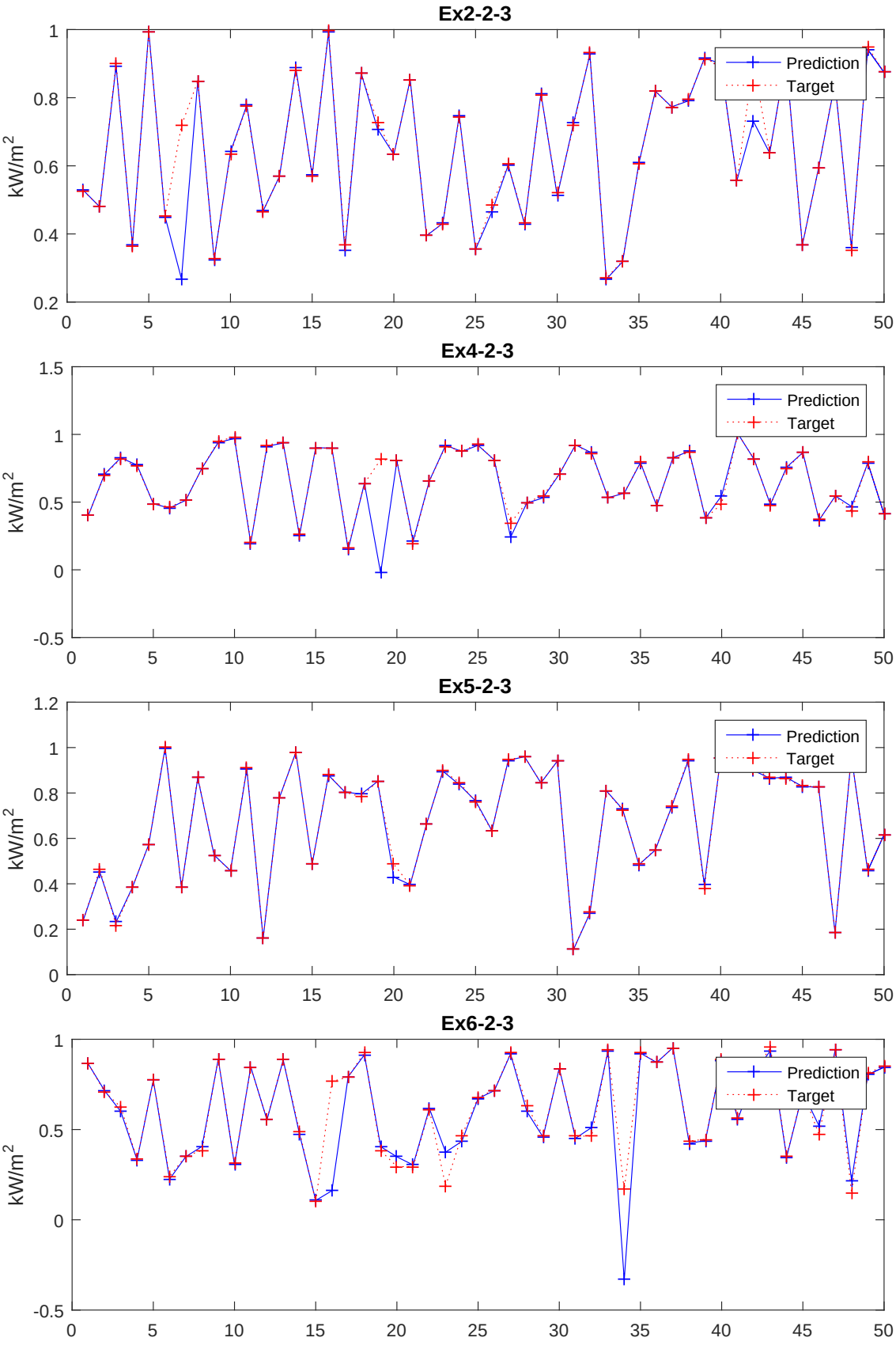


Figure 4.8: Comparison between prediction and target (Experiments 2_2_3, 4_2_3, 5_2_3 and 6_2_3)

Experiments	*windowSize*	*averageTargetPer*
Ex1_4	3600	3600
Ex2_4	3600	300
Ex3_4	3600	300
Ex4_4	900	900
Ex5_4	1800	1800
Ex6_4	86400	3600
Ex7_4	86400	3600

Table 4.6: Different window of prediction

	Inputs	**hiddenLayerSize**	**dayType**	**pastValues**	**Training Error**		**Test Error**	
					MSE($(W/m^2)^2$)	RMSE(W/m^2)	MSE($(W/m^2)^2$)	RMSE(W/m^2)
Ex1_4	ZHPDG+DOY	20-10	1,2,3,4	Yes	3.614e+03	60.1167	2.91e+04	170.5486
Ex2_4	ZHPG+DOY	50	1	Yes	57.76	7.6	1.807e+03	42.51
Ex3_4	ZHPDG+DOY	20-10	1,2,3	Yes	458.3365	24.4088	6.47e+03	80.4471
Ex4_4	ZHPG+DOY	50	1	Yes	14.66	3.8289	121.485	11.022
Ex5_4	ZHDPG+DOY	50	1	Yes	127.09	11.2734	2.968e+03	54.4854
Ex6_4	ZHPG+DOY	50	1	Yes	43.97	6.63	9.3876e+03	96.8896
Ex7_4	ZHPDG+DOY	20-10	1,2,3	Yes	226.59	15.05	1.177e+03	34.30

Table 4.7: Results of experiments with different windows

4.3.4 Different time windows to predict

In this section different windows of prediction and different mean values of the global radiation are calculated. The different windows to test are described in table 4.6. All the values of this table are in seconds. The period of the moving averages to soften the data is still $300s$. The experiments are described in more detail in table 4.7 where the error achieved is also included.

The different windows to test are: hourly mean radiation one hour from current instant t; the "instantaneous" global radiation one hour in the future; the global mean radiation every 15 minutes; the mean global radiation in half an hour and the hourly mean global radiation at the same time but one day in the future.

In table (4.7) the result of the different experiments with different windows are shown. It is worth noting that this experiments have been carried out with only half of the data available for training (as in table 4.5) because when trained with the whole set of data the experiments showed worse results in both, training and testing.

The choosing of nets of 50 neurons to predict days of type and nets of 20-10 neurons for the prediction of days of type 1,2 and 3 is motivated by the results of the previous section. Even though the performance with the training data of the

networks 20-10 with days of type 1 is lower than the performance of the networks of 50 neurons it is not so with the test data. The performance with the test data is the one that matters because it is data that have been never presented to the network.

In the table it can be seen that the best results are obtained again with only clear days, except in the prediction of the mean hourly radiation in the next day where the result is better with day types 1,2 and 3. The experiment Ex1_4 have been carried out to show that adding days of type 4 only worsens the prediction (the size of the window is the same as in the experiments of the previous section). The best performance is achieved in experiment Ex4_4 with a RMSE of $11.02 W/m^2$. This experiment corresponds with the prediction on $15 minutes$ of the mean global radiation of those $15 min$. It is clear that the best time window is the prediction of the hourly mean radiation one hour from the current moment (as it have been seen in the previous sections). The best performance of the whole bunch of experiment is achieved with experiment Ex3_3 with a structure 10-5, inputs ZHDG + DOY with a RMSE of 5.97.

In figure (4.9), experiments Ex1_4, Ex2_4, Ex4_4 and Ex7_4 are represented visually with 50 random pair of target vs prediction as in the previous sections. It can be seen that the only experiment that achieve a good prediction is Ex4_4 while the other do not seem to achieve an accurate predictions having too many points where the prediction diverges a lot from the target.

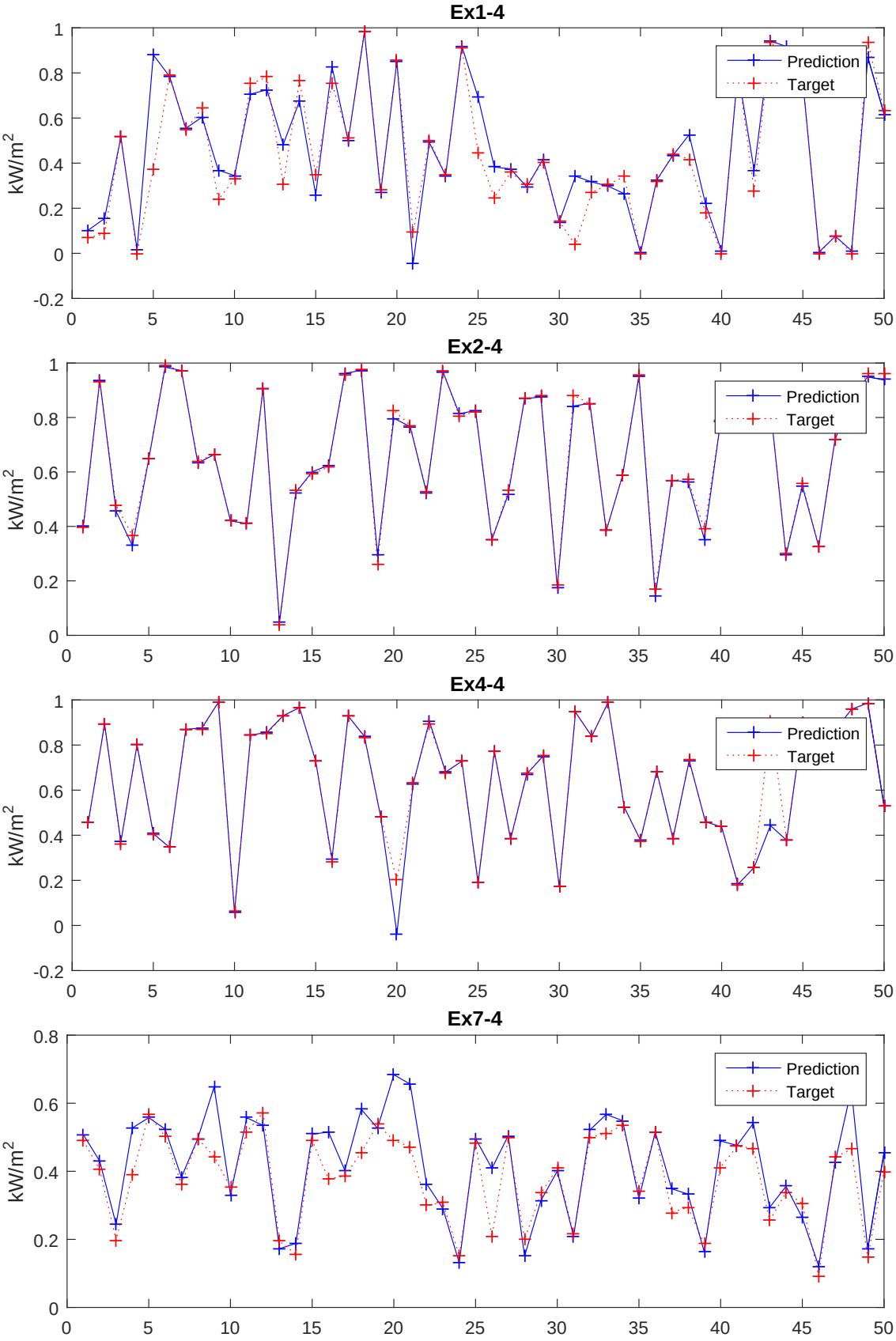


Figure 4.9: Comparison between prediction and target (Experiments 1_4, 2_4, 4_4 and 7_4)

Chapter 5

Conclusions

This last section consists of a summary of the project, the conclusions from the results and some suggestions for future lines of work.

5.1 Summary

The goal of this work is the nowcasting of the solar radiation using real-time data. In order to achieve that *Artificial Neural Networks* were used to make the predictions. In particular multilayer perceptrons. The software `MATLAB` has been used to process the data, to create, train and use the networks. The motivation to use this software is because it provides a toolbox to work with artificial neural networks. With the aim of the project in mind different nets with different inputs, architectures and windows of prediction were tested. The data available was radiological and meteorological data from the radiological station located in the roof of the ETSI's laboratories building in the city of Seville and it was provided by the GTER. Data was collected every five seconds during three years.

First, the data was rearranged in order to make it easier to work with in the MATLAB environment. Several programs were made to filter and customize the data to use in the training of the different networks.

In the experiments the days were separated in four categories depending on their clearness. Different sets of inputs and different structures were tested ,as well as different time windows for the predictions. In the next section the most important conclusions drawn from this experiments are discussed.

5.2 Conclusions from the experiments

Having carried out the experiments, the most important conclusions that can be drawn from this work are:

- with the meteorological data available accurate predictions were achieved in clear days (days of type 1). If the days used in the training and testing were cloudy at some point, the prediction error grew. This might be motivated by the lack of variables related directly with the clearness index that ,according to [12], is the most relevant input variable to the neural network in problems of solar radiation prediction along with relative air mass.

- the sets of inputs that provide better results are: the zenith, the pressure, the relative humidity, global radiation and the day of the year if only clear days are taken in consideration. But used with a combination of day types it does not generalize well. When the test data was presented the *RMSE* was too high. For a combination of different types of days the inclusion of the diffuse radiation proved to be better, improving the performance of the network when predicting the test data. This is caused because the diffuse radiation is indirectly related to the clearness index. When the diffuse radiation is used in the training of networks that use only clear days it worsens the performance. The conclusion is that, for the general prediction of solar radiation not depending on the type of day, the diffuse radiation is a powerful input but if the interest is to predict solar radiation only in clear days it is not a good choice.

- Selecting a precise number of neurons for the MLP revealed of paramount importance. it was found that in medium and large size networks if all the training data was used there was overfitting and the network was unable to generalize the results resulting in a very high error when the test data was presented. This fact prevents from using too much data for training stages, to avoid overfitting problems.

- If the days are clear, the MLP is able to perform an accurate prediction with an *RMSE* of $5.89W/m^2$ which is an error of $0.0058kW/m^2$. If the days are not entirely clear, the overall prediction is not as good () but looking only at the clear parts of a day the prediction is still accurate but the inaccuracy of the prediction in the cloudy parts makes the overall performance to worsen.

- The window of prediction that obtained the best performance was the prediction of the hourly mean radiation one hour from the current moment and taking only clear days in account. For days of days 1,2 and 3 the best is obtained prediction the hourly mean radiation one day (24 h) from the current time. So it is clear to conclude that the best variable to predict is the mean hourly radiation. The other window that have a performance is a window of 15 min where the mean radiation in those 15 min is predicted, but again it only takes in account clear days. This last window is nearer to the goal of this work.

5.3 Future work

There is a lot that can be done to improve the nowcasting done in this work. As have been said the clearness index and the air mass are very relevant when prediction solar radiation. Adding this variables and using them along with some meteorological variables should improve the performance of the predictors (networks). According to [12] it is important to contemplate the presence of clouds as it clearly affects the solar radiation that reaches the ground. For this work, this information was unavailable but it is key to obtain good results in the nowcasting. If it would be possible to predict when a cloud would pass above the zone of study, it would help a great deal in the prediction of cloudy part of days, which have been the weakness of this work. It would be interesting to try to predict this and it could be done, for example using image treatment along with artificial neural networks to predict the presence of clouds. Two futures lines of work are suggested in this section:

- The first is to use image processing and ANN to try to predict the presence of clouds above the zone of study. Or at least to feed data to the network that contains any information about the presence of clouds. It is suggested that a variable cloud, along with direction and velocity of the wind are added to the inputs. To obtain the variable cloud that would indicate if there is a cloud or not and how thick it is, image processing can be used. This would require a long term work since a historic of data is needed to train the ANN.

- Another line of work that is not included in this work is to use dynamic networks as the predictors instead of picturing the problem as a non linear regression it would be pictured as a problem of prediction of time series and dynamic networks are widely used in this field.

Appendices

Appendix A

Summary of functions.

In this annex a list with the description of the functions written ad-hoc for this work is included. They are listed in order of appearance.

A.1 Reading and processing

- data_processing_TFG: This code enables the user to read data from a txt file with a name with the following format meteo_year_day and to save it in a MATLAB file of type cell array that will be called `data.mat`. There are two fields in this cell array:the first one indicates the time of the day the data is taken from;the second field, named values, in which the values of the diferents variables will be saved in a vector. values=[Diffuse band (w/m^2), horizontal global, Global 27 $\circ$ S, nothing, global 27 $\circ$ S cell, global horizontal mapa, global 27 $\circ$ S mapa, Pyranometer, horizontal cell, Hb0 NIP, Hb0 CHP1, Diffuse balls, nothing, Temperature($\circ C$), wind velocity(m/s), wind direction($\circ$), pressure(mBar), relative humidity(%),nothing]; These are the variables as they appear in the txt files that have some columns that does not measure anything. That is why there are some components of values named nothing. For a MATLAB version after -7.3v is necessary to change the preferences to save the MAT-files with the 7.3v if the amount of data exceed 2GB (or is close to it because the cell arrays need more space in the memory). The format of the data is as follows: `datayearindex,day.field` where `yearindex` is 1 for 2012, 2 for 2013 and 3 for 2014 and `day` is the day of the year.

- lectura_csv: This script add a third field called `ok` to the file `data.mat`. This field is a vector of ones and zeros indicating if the variables are reliable (only for variables of radiation). Besides, from the component 13 to 17 it provides the following information data.ok=[...julian day, file generated, complete file, centered file, I_g0¿I_0]. This information is obtained from the journal of the radiological station. It is an Excel book where each sheet is a year. Every sheet needs to be saved apart with a `.csv` extension in order to be read by `MATLAB` and put into the vector `ok`.

- Test_Data: This function provides the days that have a structure different from the one they are supposed to have. In addition, it indicates days in which there was a blackout in the laboratory because they contain no reliable data or no data at all. `Test_Data(year,begin,ending,data,processed_years)` show a message in the screen if one of the days that are between `begin` (the day to start counting) and `ending` (the last day to test) of the `year` chosen (2012,2013 or 2014). The days that were found to have an error where put into a vector by hand. There is one vector for each year.

- Which_hour: This script is a program that contains the vector of bad days provided by the function Test_Data and fixes the days, putting them in a new file called Data_fixed (this program only fix days with more hours). It takes the days in the vector of bad days and check which are the hours that have more rows than the should have and erase them.

- Amplify: it amplifies the dimension of the cells of the file `data.mat` (This process is applied to every day in order to avoid future undetected problems). It duplicates the number of rows and create a new cell array called `data_amplify` with the amplified cells. This file is not saved.

- Rearrange: it uses the file `data_amplify` and reads every day from the hour $00:00:00$ to $23:59:55$ and check if every consecutive row have the right time. If not the faulty row is moved below and a new row is created with the right time and a vector values of zeros. Once all the days have been processed the faulty data would be located in the rows below the 17280th which should be the last one. It saves the new data file with the name `data_fixed2.mat`.

- Reduce: it takes the file `data_fixed2.mat` and, in every cell, it erase the rows below the 17280th. It creates a new file `dat_fixed.mat` with the filtered times and values and add the field `ok` taken it from the file `data.mat`. Finally, the file `data_fixed.mat` is saved.

- zenith: it calculates the zenith with the times of the file `data_fixed.mat`. To calculate the zenith angle the functions of [7] are used. This functions need the longitude, latitude and altitude as inputs. This information is taken from [11], which indicates that for the city of Seville the longitude is $5.9962 \circ W$ the latitude is $37.38 \circ N$ and the altitude is $8.26m$. This script add the field `.zenith` to the cell array `data_fixed.mat`.

- dataInterpolation: this script takes the file `data_fixed.mat` and fill in the gaps left by the program `rearrange` where the new rows had a vector *values* of zeros. It fills the gaps interpolating using the data of the adequate roes as interpolation nodes. To perform the interpolation it uses the `MATLAB` function `interp1`. It finally saves a file `data_fixed_filled.mat` with the complete data.

A.2 Preparing for training

- experimentDef: This script allows to create a structure for an experiment to be carried out by means of inputs through the command window. The fields of a Experiment are:

  - *.inputs*: Name of the variables to use as inputs separated by spaces. It is saved as a string vector.
  - *averagePer*: Period of the moving average to soften the inputs to reduce peaks. It is supposed to be in seconds.
  - *averagePerTarget*: Period of the moving average of the Target, in seconds. It indicates what value of the target to predict. If this parameter is to be set to 3600 the value to predict would be the hourly mean radiation. If this parameter is to be set to 1800 the value to predict would be the mean radiation in half an hour.
  - *windowSize*: window of the prediction in seconds.
  - *years*: vector of the years to use in the experiment 1 is 2012, 2 is 2013 and 3 is 2014. Example: [1 2 3] will take in account the years 2012,2013 and 2014 whereas [1 2] will only take in account 2012 and 2013.
  - *days*: vector of 2 components. The first indicating the first day to compute in the first year and the second the last day to compute of the last year. Example: years = [1 2 3] and days = [20 365], this will compute all days between day 20 of 2012 and day 365 of 2014.
  - *dayType*: vector from 1 to 3 components indicating the dayType(s) to compute. Type 1: clear days; type 2: clear morning; type 3: clear afternoon. Expample: dayType = 2 will only take in account days of type 2 whereas dayType = [1 2 3] will compute days of type 1, 2 and 3.
  - *pastValues*: This is a paremeter important for the training to give as an input, not only the data at a time t, but also values in $t - pastValues$. It goes in seconds. Example: pastValues = 300, then for training the inputs will be the inputs(t) + inputs(t-5min).
  - *name*: is the name of the file in which the experiment parameters will be saved.

- dataPreparationStructTime: The function dataPreparationStructTime process the data of an experiment with moving averages and prepares it to the final step before using the data for training. The only input of this function it the experiment structure created with the function experimentDef, defined above. The only requirement is that all the fields of experiment are filled in and that the file data_fixed_filled were in the MATLAB path. The outputs of this function are:

- dataInput: a cell array of size 3×1 where every cell contains the processed data of one year. The processed data is in a matrix structure where every row is a sample and every column is a variable. Here the night time and days whose field *ok* is 0 are left out.
- timeInput: a cell array of size 1×3 where every cell contains a vector with the time corresponding with each sample of the cell array dataInput.
- dataTarget: a cell array of size 3×1 where every cell contains the processed targets of one year. Every cell is a vector containing the target corresponding with each sample of dataInput.
- timeTarget: a cell array of size 1×3 where every cell contains a vector with the time corresponding with each sample of the cell array dataTarget.
- days2use: a cell array of size 3×1 where every cell contains the days that have been taken in consideration to be processed in one year.
- var: a cell array where every cell contains a string corresponding to one processed input.
- pastValues: it is the same parameter defined in experimentDef and it will be necessary in the following steps.

This function has two subfunctions:

- daysToIgnoreFuncMuchTypes: it returns the days that can be of use given the dayType of the experiment, the variables to be used and the field *ok* of the file data_fixed_filled. The variable usableDays contains the days usable if the target is aimed to the same day and the variable usableDay2Day contains the days that can be used if the target is aimed to one day in the future as they have to be consecutive days. Inputs: inputs, years, days, data_fixed_filled ,dayType. Outputs: daysToIgnore, var, vectorVar, usableDays, usableDay2Day
- Data_ave_mat: it returns the moving average of the chosen variable given the averagePer, the day of the year, the year, the time of dawn and the time of dusk along with the data file. It returns the moving averages along with the index of the first sample and the index of the last sample in the data file.Outputs: Data_ave, begin, ending. Inputs:variable, average_per, year, day, begin_hour, ending_hour, data ,processed_years.

- prepareInTaForTraining: [InputTrain, TargetTrain, InputTest, TargetTest, timeInputFinal, timeTargetFinal] this function rearrange the outputs of the function dataPreparationStructTime to merge the three cells in a matrix structure and adding the past values if desired. The variable dayOfYear is set to 1 if the day of the year is desired as an input and 0 if not. dayAfter is set to 1 if the prediction is for one day in the future or further and 0 if the prediction is aimed at the same day. This function returns:

  - InputTrain: the inputs in a matrix structure prepared to be fed to the network. It contains the 90% of the data contained in dataInput.
  - TargetTrain: the target in a matrix structure prepared to be used in the training procedure. It contains 90% of the data contained in dataTarget.
  - InputTest: the inputs in a matrix structure prepared to be fed to the network after training to test it. It contains 10% of the data contained in dataInput
  - TargetTest: the targets in a matrix structure prepared to be compared against the predictions obtained with the input of the matrix InputTest. It contains 10% of the data in dataTarget.
  - timeInputFinal: a vector of the times corresponding to each one of the samples in InputTrain and InputTest.
  - timeTargetFinal: a vector of the times corresponding to each one of the samples in TargetTrain and TargetTest.

- trainANNInTa: this function has as inputs the outputs of the function prepareInTaForTraining plus the structure of the network defined by a vector called hiddenLayerSize. The number of components define the number of hidden layers and the value of each component defines the number of neurons in each layer. This function returns the trained network and its training parameters.

- TrainApp: this is a program that have the functions prepareInTaForTraining and trainANNInTa implemented and asks for the parameters that are not defined in previous function through the screen of the command window. This parameters are dayOfYear, dayAfter and hiddenLayerSize. It asks the user if the network is to be saved and, if so, with what name. It can also resume the training of a network if a file checkPoint is available.

Resumen Trabajo de Fin de Grado: Aplicación de técnicas de Machine Learning para la predicción a corto plazo de radiación Solar

Pablo Egea Hervás

19 de septiembre de 2016

1. Resumen

El objetivo de este proyecto es utilizar técnicas de machine learning para realizar la predicción a corto plazo (*nowcasting*) de la radiación solar. En particular, la técnica usada serán redes neuronales artificiales (*Artificial Neural Networks*). La meta es conseguir predicciones precisas para un tiempo cercano usando información del instante presente. En este trabajo se realiza la prueba de diferentes arquitecturas de redes neuronales, así como diferentes ventanas de tiempo. Para ello, el GTER (Grupo de Trabajo de Energías Renovables) ha cedido información recogida por una estación radiológica situada en el tejado del edificio de los laboratorios de la Escuela Superior de Ingeniería de la Universidad de Sevilla, situado en Sevilla en la isla de la Cartuja. Se dispone así de varias variables radiológicas y meteorológicas: radiación global y difusa, presión atmosférica, humedad relativa y temperatura del aire. Usando la hora del día se calcula también la posición del sol como otra variable disponible.

Con estos datos y el uso del software MATLAB y su módulo (*toolbox*) para redes neuronales artificiales se crean y se entrenan diferentes redes para ser después puestas a prueba. El documento principal está dividido en tres partes: el entorno teórico en el que se trabaja, el procesado de los datos y finalmente los experimentos realizados y las conclusiones. Después de realizar diferentes experimentos con diferentes combinaciones de datos de entrada, diferentes arquitecturas de redes y diferentes ventanas de predicción se puede concluir que con los datos disponibles sólo es posible predecir de manera precisa días, o partes de un día, claros mientras que en los días nublados la radiación varía de tal manera que la red es incapaz de predecir su comportamiento.

2. Introducción

La razón por la que se busca la predicción precisa de la radiación solar a corto plazo tiene que ver con el uso que hacen las plantas solares de esa radiación. La energía que producen estas estaciones depende directamente de la radiación solar que llega al suelo en un instante. Actualmente es muy difícil predecir de manera precisa a corto plazo la radiación solar. Esto implica que es difícil conocer de antemano la cantidad de energía que producen estas plantas instantáneamente y esto sería de gran ayuda a la hora de evitar problemas de abastecimiento de la línea o de evitar excedentes de energía.

Las predicciones que se realizan normalmente de la radiación solar suelen ser en términos de valores medios y en tiempos de un día hasta tiempos de un mes o un año. Existen varios trabajos que eligen utilizar técnicas modernas como las redes neuronales artificiales en lugar de los modelos teóricos. Las redes neuronales artificiales han probado conseguir mejores resultados que métodos clásicos. De acuerdo con [6], se ha comprobado que las redes neuronales artificiales han conseguido una mejor aproximación de la realidad que el modelo de Ångstr$\acute{}$om, y modelos lineales,

no lineales convencionales. Las redes neuronales son una gran herramienta para interpolación de datos, reconocimiento de patrones y regresión no lineal. En el capítulo dos del documento se detalla la base teórica de esta herramienta.

Existen varios trabajos cuyo objetivo es predecir la radiación solar. La mayoría apunta a predicciones más a largo plazo como un día, una semana, un mes o un año. Hay también algunos trabajos que tienen el objetivo de predecir la radiación solar para distintas localizaciones. En este trabajo la meta es predecir la radiación solar en ventanas pequeñas de tiempo como media hora, una hora o dos horas. A continuación se nombran algunos de estos trabajos.

Como se ha dicho en este campo existen varios trabajos que usan técnicas de machine learning para la predicción de la radiación solar pero la mayoría están orientados a predicciones a largo plazo (*forecasting*). Muchos de estos trabajos usan, precisamente, redes neuronales artificiales. Estas redes permiten obtener una relación entre unos datos de entrada y sus correspondientes datos objetivo que pueden no tener una relación conocida de manera analítica. Al ser la radiación solar una de estas variables con una relación bastante compleja con el clima, las redes neuronales son una buena herramienta para lidiar con este tipo de problemas. En los siguientes trabajos se han usado redes neuronales u otra forma de machine learning para la predicción de la radiación solar.

En el trabajo [3] se usa también el software MATLAB para trabajar con redes neuronales, aunque en una versión del módulo ya obsoleta. Aquí se usan la temperatura máxima, la velocidad media del viento, las horas de luz solar, la humedad relativa media y la radiación solar para generar un modelo climático de Al Ain prediciendo la radiación solar.

En el artículo [6] se resumen varios trabajos que usan redes neuronales artificiales para predecir la radiación solar. Algunos de ellos son, [8], [9] y [10]. En [8] se desarrolla un modelo de red neuronal para estimar la radiación solar de una localización usando combinaciones de latitud, longitud, altitud, el mes, la temperatura media, la nubosidad media, la velocidad media del viento y las horas de luz solar. En [9] se unas redes de tipo *feedforward* para predecir diferentes tipos de radiación usando diferentes variables de entrada. En [10] se usan combinaciones de día, temperatura máxima del aire, temperatura media del aire y la humedad relativa para obtener la radiación difusa y se concluye que los mejores resultados se obtienen usando la humedad relativa y la temperatura media.

Uno de los problemas de las redes neuronales artificiales, es la optimización de la arquitectura de la red y de los datos de entrada. En [5] se usan innovadoras técnicas que permiten una optimización para la arquitectura de la red. En este trabajo ([5]) se pretende la radiación media diaria.

En este trabajo se usan también redes neuronales pero la predicción que se busca es a corto plazo. Para estas predicciones a corto plazo se necesitan datos meteorológicos y radiológicos recogidos cada poco tiempo. Estos datos los proporciona el GTER. Recogidos desde la estación radiológica situada en el tejado del edificio de los laboratorios de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla, estos datos están recogidos cada cinco segundos durante un período de tres años

(2012, 2013 y 2014). Lógicamente esto es un gran banco de datos que necesita de un procesado y un filtrado. Para ayudar al filtrado, el GTER también ha proporcionado un diario de la estación radiométrica que recoge las variables fiables de cada día, así como comentarios.

Para implementar las redes neuronales, como ya se ha comentado, se usa MATLAB por su módulo de redes neuronales artificiales que permite trabajar fácilmente con ellas. Como se ha dicho el proyecto consta de cuatro capítulos principales: entorno teórico, procesamiento y filtrado de los datos, experimentos realizados y conclusiones.

3. Machine Learning

En este capítulo se detalla la parte teórica de las redes neuronales y en particular de los perceptrones multicapa. [] define machine learning como la programación de ordenadores para optimizar un criterio de rendimiento usando datos de ejemplo o experiencia pasada. Se dice también que que este tipo de algoritmos se necesita cuando no es posible que un programa de ordenador, con ecuaciones implementadas, resuelva un problema; ya sea bien porque no se conoce el proceso o no existe información suficiente sobre él. Las redes neuronales se basan en datos, de los que recogen la forma en la que están relacionados, por medio del proceso de aprendizaje. Machine learning se utilizan para aprender asociaciones, clasificaciones o regresiones (lineales o no lineales). En este caso se usarán para obtener una regresión no lineal de los datos. En este trabajo se usan perceptrones multicapa, una determinada arquitectura de red neuronal.

Una red neuronal es un procesador distribuido masivamente paralelo distribuido formado por unidades de procesamiento simples que tienen una tendencia natural para conservar conocimiento basado en la experiencia y disponer de su uso. Está basado en el cerebro humano porque necesitan de un proceso de un proceso de aprendizaje que se basa en variar los pesos de las conexiones entre esos procesadores básicos (neuronas). Un perceptron multicapa es una red neuronal de más de dos capas (capa de entradas, capa(s) oculta(s) y capa de salida) que tiene conectividad completa. Todas las neuronas de una capa están conectadas a todas las neuronas de la capa siguiente. Básicamente lo que hace una neurona, es recibir un pulso eléctrico, transformarlo y mandarlo a la siguiente neurona. En el caso de las redes artificiales este pulso eléctrico son las variables de entrada modificadas por las neuronas anteriores.

En la figura se puede ver un modelo de neurona. Una neurona es, de acuerdo con [2]: una unidad de procesamiento de información que es fundamental para la operación de una red neuronal. Los elementos básicos de este modelo son: las conexiones , caracterizadas por un peso específico; un sumador que suma las entradas pesadas con los pesos sinápticos; una función de activación que limita la amplitud de la salida de una neurona. El sesgo (*bias*) incluido en el modelo incrementa o disminuye la entrada a la red de la función de activación, dependiendo de si es positivo

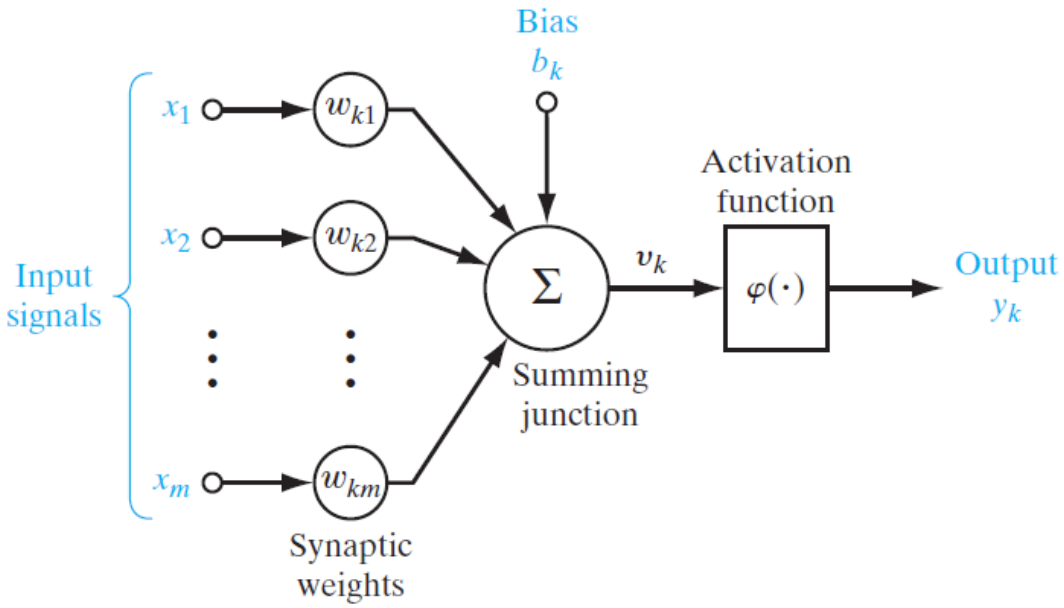


Figura 1: Modelo de neurona

o negativo. Lo que hace una neurona matemáticamente se muestra en las siguientes ecuaciones:

$$y_k = \phi(u_k + b_k) \tag{1}$$

donde

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2}$$

En resumen la señal de salida de una neurona es el resultado de aplicar la función de activación a la entrada ponderada con los pesos sinápticos más el sesgo.

Existen dos funciones de activación básicas: la función umbral y la función sigmoidal. La función umbral devuelve un 1 si el argumento es positivo y 0 si el argumento es negativo. La función sigmoidal tiene forma de "S" y es la mas comunmente usada en redes neuronales por carácter equilibrado entre comportamiento lineal y no lineal. Estas dos funciones tienen un rango entre 0 y +1. A veces es necesario tener un rango entre -1 y +1. Para ello se rescribe la función umbral como la función signo y la función sigmoidal se puede sustituir por la tangente hiperbólica.

Otra característica a tener muy en cuenta en las redes neuronales es su arquitectura. Existen tres arquitecturas básicas: Redes prealimentadas de una sola capa, redes prealimentadas multicapa y redes recurrentes (con retroalimentación). En las redes de una sola capa las entradas están directamente conectadas a la capa de salidas, son prealimentadas porque la información va estrictamente en un solo sentido (desde la entrada a la salida). En las redes multicapa hay, al menos, una capa oculta entre las entradas y las salidas. Pueden estar completamente interconectadas o

parcialmente interconectadas. Las redes recurrentes contienen, al menos, un bucle de retroalimentación. Las redes dinámicas son un tipo de redes recurrentes, se usan para solucionar problemas de series temporales entre otros.

En la figura 2 se puede apreciar la arquitectura de una red multicapa. En este trabajo se emplean perceptrones multicapa, que son un tipo de red neuronal con una arquitectura multicapa totalmente interconectada. Cada una de las conexiones indicadas en el esquema tiene asociado un peso sináptico. Se ha mencionado con

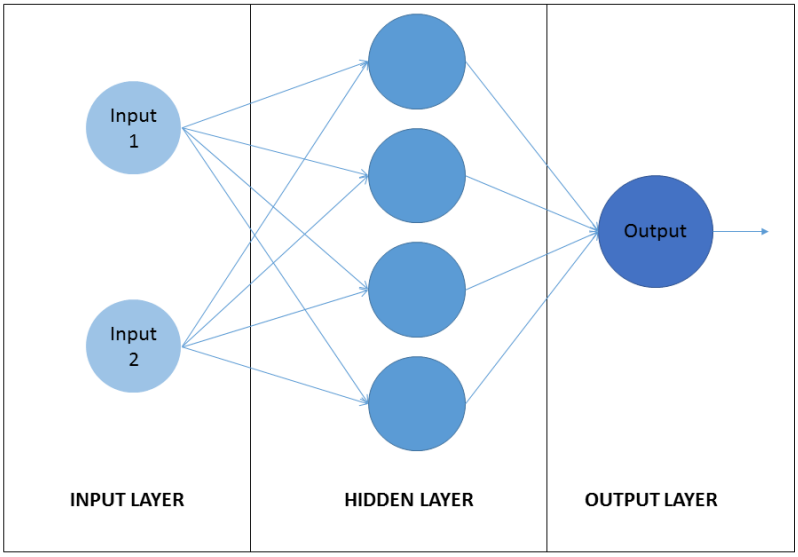


Figura 2: Arquitectura de un perceptron multicapa

anterioridad que las redes neuronales son capaces de *aprender*. Este proceso de aprendizaje se basa en, mediante ejemplos, ajustar estos pesos sinápticos para que cuando las entradas sean ponderadas con ellos la salida esté lo más cercana posible del valor objetivo. Para este proceso es necesario propagar una señal de error hacia atrás en la red (desde la salida hacia la entrada). Por lo tanto, en la red existen dos tipos de señales: las señales de función y la señal de error. El algoritmo de aprendizaje más básico se conoce como algoritmo de propagación hacia atrás, puesto que la señal viaja en el sentido contrario.

El algoritmo funciona como sigue: dado un conjunto de muestras N donde $x(n)$ son las entradas y $d(n)$ son los valores objetivo para $n = 1, ..., N$. Una neurona j en la capa de salida produce una salida $y_j(n)$. La señal de error es $e_j(n) = d_j(n) - y_j(n)$. La energía del error se define a su vez como $\mathscr{E}_j(n) = \frac{1}{2}e_j^2(n)$. Sumando para todas las neuronas en la capa de salida se obtiene la energía del error de todas las neuronas de la capa. Se puede usar también el valor medio de la energía del error para el conjunto de muestras N. El objetivo es actualizar los pesos sinápticos para reducir este error al mínimo posible. Dependiendo de cuando se haga esta actualización existen dos tipos de aprendizaje: aprendizaje en línea si se actualizan después de cada muestra o

aprendizaje por lotes si se actualizan después de que todas las muestras hayan sido presentadas a la red. En este trabajo se va a usar aprendizaje por lotes porque la implementación en MATLAB de este tipo de aprendizaje está mejor implementada y porque permite la paralelización del proceso. La desventaja es que requiere mucha memoria.

La actualización de los pesos sigue la siguiente ecuación: $\Delta w_{ij}(n) = \eta \delta_j(n) y_j(n)$ donde η es el parámetro de tasa de aprendizaje y δ_j es el gradiente de la energía de error. El gradiente de error depende de los pesos sinápticos en la forma: $\delta_j(n) = \phi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$ donde ϕ es la función de activación y δ_k es el gradiente de error de la capa siguiente (de donde procede la señal de error ya que va en sentido opuesto). Para más detalles ver [2].

El método de entrenamiento que se usa en este trabajo es el método Levenberg-Marquardt que aúna el método de Newton, que converge rápidamente pero es posible que diverja, y el método del gradiente descendente, que asegura convergencia pero converge lentamente. La unión de estos dos métodos permite obtener lo mejor de cada uno y reducir las desventajas. Este es el método usado porque es el método recomendado por MATLAB para este tipo de problemas. El desarrollo matemático de este método se puede consultar en [**?**] en el capítulo 4, sección 16.

4. Procesamiento de datos

Una vez se ha descrito la base teórica de cómo funciona una red neuronal y como *aprende* lo siguiente a saber es con qué variables va a ser entrenada. Para ello en esta sección se describen los datos disponibles y como se procesan y se filtran. Lo primero es conocer los datos de los que se dispone.

Como se indicó en la introducción los datos son proporcionados por el GTER tomados desde la estación radiológica en el tejado del edificio de los laboratorios de la ETSI en Sevilla. Proporciona las siguientes variables:

- Radiación difusa: en W/m^2
- Radiación global: en W/m^2
- Radiación global a $27°S$: en W/m^2
- Radiación global a $27°$ desde célula: en W/m^2
- Pirgeómetro: en W/m^2
- Horizontal desde célula: en W/m^2
- $H_{b0}NIP$: Radiación directa medida con la incidencia normal al pirheliómetro en W/m^2
- $H_{b0}CHP1$: Radiación directa medida con un pirheliómetro en W/m^2

- Radiación difusa bolas: en W/m^2
- Temperature: in $°C$
- Wind Velocity: in m/s.
- Wind direction: en grados.
- Pressure: en $mBar$
- Relative humidity: en %

Todos estos datos están recogidos en documentos .txt (uno por día). Además de estos datos, el GTER proporciona también un diario de la estación radiométrica que recoge que variables son fiables cada día. Para trabajar con estos datos en MATLAB fácilmente se guardarán en una estructura de celdas con una celda para cada día. Cada una de estas celdas es una estructura con cuatro campos: tiempo, valores, ok y cenit. En tiempo se recoge una matriz con tres columnas donde la primera indica la hora, la segunda los minutos y la tercera los segundos del momento en el que se han recogido los datos. En el campo valores se recoge una matriz cuyas columnas representan las distintas variables y las filas son las distintas muestras. El vector ok es un vector de unos y ceros que indican si esa variable es fiable o no ese día. Por último el campo cenit recoge el ángulo cenit del sol en el momento en que la muestra fue recogida. Se calcula con las funciones *sunPosition* de Vincent Roy (Copyright (c) 2004).

Es importante también mencionar que los días se han dividido en cuatro categorías dependiendo de su nivel de claridad: los días de tipo 1 son totalmente claros, los días de tipo 2 son claros por la mañana, los días de tipo 3 son claros por la tarde y los días de tipo 4 no son claros en absoluto.

A continuación se explica el proceso seguido para llegar desde los archivos .txt hasta el archivo .mat con la estructura de celdas.

4.1. Procedimento de procesado

Con el objetivo de hacer los datos manejables en MATLAB lo primero que se hará es pasar los datos desde los documentos text a MATLAB. Los datos se guardarán en una estructura de celdas, donde cada celda corresponderá a un día. El archivo se llamará data.mat, por ejemplo data1,2.mat corresponde al día 2 del año 1. El programa `data_processing_TFG.m` lee los datos de los archivos text y los guarda en una estructura de celdas con los campos *.time* y *.values*. Los archivos .txt están nombrados de cierta manera (meteo_año_día) por lo que es sencillo asignar el año y el día a la celda. Lo siguiente es añadir el campo *.ok*.

Una vez que se tiene la estructura de celdas con los campos tiempos y valores, el siguiente campo a añadir es el campo ok. El campo ok es un vector de ceros y unos, hay un vector para cada celda. Estos ceros y unos indican si las variables son usables o no. Estos datos están recogidos en el diario de la estación radiométrica

proporcionado por el GTER. Aquí solo están recogidas las variables de radiación, no así las variables meteorológicas. Para añadir este campo se usa el programa lectura_csv. Este programa coge las hojas del archivo MS Excel del diario (cada hoja corresponde a un año) guardadas previamente en formato csv para trabajar más fácilmente en MATLAB, las lee y asigna los valores pertinentes a cada vector ok. Estos datos servirán posteriormente para filtrar los datos con los que se va a trabajar.

Lo siguiente es arreglar los datos de las celdas para que cada día tenga la misma estructura pues se detectó que algunos datos no estaban bien y en algunos días había horas de más o horas repetidas. Los dos problemas principales encontrados fueron:

- Celdas con más filas de las debidas: como se ha mencionado anteriormente los datos son recogidos cada 5 segundos, por lo que en un día de 24 horas equivale a 17280 muestras. Cuando se comenzó a trabajar con los datos se detectó que había días que tenían mas filas de las debidas porque había horas que se repetían. La solución que se adoptó para este problema fue aislar esos días y borrar las filas que correspondías a horas repetidas. Para esto se usaron las funciones Test_Data y Which_hour. La primera detectaba los días con filas de más y la segunda procesaba esos días para borrar las filas de sobra.

- Celdas con el número de filas debido pero con horas faltantes: se detectó posteriormente que algunas celdas no seguían la serie temporal debida repitiendo algunas horas haciendo que hubiera horas que no existen en ese día. Cuando se detectó este otro problema se decidió procesar todos los días y comprobar que todos seguían la misma serie temporal. En los días con fallos estos fueron reescritos forzando a que se siguiera la serie temporal y borrando las horas que se repetían. Para ello se amplió cada celda, si la siguiente muestra no correspondía con la hora que debía tener se movía una fila hacia abajo y en la nueva fila vacía se reescribe el tiempo correcto y los valores se pusieron a cero. Siguiendo este proceso al final queda, en cada celda, una matriz con el doble de filas en la que las filas posteriores a la 17280 contiene los datos repetidos. Finalmente se borran las muestras sobrantes y cada celda contiene una estructure con los datos debidamente ordenados, aunque con datos faltantes. Para realizar este proceso se usaron, en orden, las funciones *Reduce*, *Rearrange* y *Amplify*. Para rellenar los datos faltantes se usa la función *data_interpolation* que interpola los datos que faltan cada día usando los datos existentes como nodos de interpolación.

Por último se añade el campo cenit calculando el ángulo cenit a partir de la hora de cada muestra y usando las funciones de la posición del sol (sun_position) de Vincent Roy.

A continuación se describe cómo se preparan estos datos base para el entrenamiento de redes neuronales.

4.2. Preparación para el entrenamiento.

Se ha pretendido hacer una serie de funciones que permitan el fácil manejo de los datos y la mayor determinación en los parámetros de los experimentos. Al realizarse varios experimentos es importante tener un sistema flexible que permita la fácil creación de diferentes experimentos con diferentes parámetros con su posterior aplicación a los datos. La función *exDef* pide los distintos parámetros por pantalla, permite crear un nuevo experimento o modificar uno existente y guarda los guarda en una estructura Experiment contenido en un archivo con el nombre dado por el usuario. Después la función *dataPreparationStructTime* alimentada por la estructura Experiment realiza el procesado de los datos base de acuerdo con los parámetros definidos por el experimento: realiza medias móviles para suavizar los datos sin procesar (debido a la estructura intrínseca de los sensores los datos presentan pequeños picos que no se corresponden con la realidad), se definen los pares entrada-objetivo, se desestiman los datos correspondientes a la noche y los que no proporcionan información fiable. El último paso es usar la functión *prepareInTaForTraining* para ponerlo todo en forma matricial para poder alimentarlo a la red neuronal.

5. Conclusiones

Una vez se han realizado varios experimentos con diferentes entradas, diferentes estructuras de la red y diferentes ventanas de predicción se ha llegado a las siguientes conclusiones:

- Con los meteorológicos disponibles se obtienen predicciones precisas para días claros (días de tipo 1). Si los días que se usan incluyen algo de nubosidad el error de la predicción crece. Esto puede venir motivado por la falta de datos meteorológicos directamente ligados con el índice de claridad, que de acuerdo con [12], es la variable más relevante para una red neuronal que trabaja con predicciones de radiación solar junto con la masa relativa de aire.

- El conjunto de variables que consigue mejores resultados es el conjunto del cenit, la presión atmosférica, la humedad relativa, la radiación global y el día del año si sólo se pretende predecir días claros. Usando días con algo de nubosidad esta combinación no consigue una buena generalización. El RMSE fue muy alto cuando se presentaron los datos de prueba. Para una combinación de días claros y algo nubosos (tipos 1,2 y 3) la inclusión de la radiación difusa probó ser mejor mejorando el rendimiento de la red con los datos de prueba. Esto está motivado por la relación entre la radiación difusa y la claridad. Sin embargo, cuando solo días claros son tenidos en cuenta, la inclusión de la radiación difusa empeora el rendimiento. La conclusión a la que se llega es: para la predicción general de la radiación solar (sin excluir días nubosos) la radiación difusa es una variable adecuada mientras que si solo se quiere usar la red para predecir días claros no es una buena opción.

- Con algunos números de neuronas escogidos para el MLP se desveló algo de gran importancia. Para las redes de un tamaño medio y grandes (80-100 neuronas) si se usan todos los datos disponibles para el entrenamiento la red no es capaz de generalizar (*overfitting*), siendo el error obtenido para los datos de prueba mucho mayor que para los datos de entrenamiento. Este hecho previene de usar toda la información disponible para evitar posibles problemas de generalización.

- Si los días son claros el MLP es capaz de obtener un buen rendimiento con un $RMSE$ de $5.89W/m^2$ que es un error de $0.0058kW/m^2$. Si los días no son del todo claros (tipos 2 y 3) la predicción general no es tan buena, pero si se centra la atención en las partes claras del día la predicción es bastante certera pero la incertidumbre en las partes nubosas hace que el rendimiento general disminuya.

- La ventana de predicción que obtiene el mejor resultado es la media en una hora para la hora siguiente al instante de tiempo presente, si solo se tienen en cuenta días claros. Para días no enteramente claros la mejor es la predicción de la media en una hora en la misma hora del día siguiente (24 horas). Queda claro que la variable a predecir para obtener mejores resultados es la media en una hora. La otra ventana que obtiene un rendimiento aceptable, aunque solo teniendo en cuenta días claros, es la ventana de 15 min calculando la radiación media en esos 15 min. Esta última es la ventana que más se ajusta al propósito de este trabajo.

Para continuar y mejorar este trabajo se proponen dos líneas de trabajo futuro:

- Usar procesado de imágenes por ordenador para conocer la presencia de nubes sobre la zona de estudio. Se propone esta medida para alimentar a la red alguna variable que esté directamente relacionada con la nubosidad. Esto junto con la velocidad y dirección del viento debería mejorar los resultados. Sería un trabajo a largo plazo ya que es necesario un histórico para entrenar las redes neuronales.

- Otra línea de trabajo no explorada en este trabajo es el uso de redes dinámicas viendo el problema como uno de predicción de series temporales en vez de una regresión no lineal. Este tipo de redes se usa en este tipo de problemas.

Bibliography

[1] AlpaydIn, Ethem,*Introduction to Machine Learning*, second edition, The MIT Press Cambridge, Massachusetts,London, England

[2] Haykin,Simon, *Neural Networks and learning machines*, third edition, McMaster University, Hamilton, Ontario, Canada

[3] Maitha H. Al Shamisi, Ali H. Assi and Hassan A. N. Hejase ¡¡Using MATLAB to Develop Artificial Neural Network Models for Predicting Global Solar Radiation in Al Ain City ? UAE¿¿

[4] U.Divya, Dr. Chitra Pasupathi, *Survey on Machine learning approaches for solar irradiation prediction*, IJESRT, October 2014.

[5] Salcedo-Sanz, Casanova-Mateo, Pastor-Sánchez, Sánchez-Girón, *Daily global radiation prediction based on a hybrid Coral Reefs Optimization - Extreme Learning Machine approach*

[6] Amit Kumar Yadav, S.S. Chandel, *Solar radiation prediction using Artificial Neural Networks techniques: a review*, Renewable and Sustainable Energy Reviews 33(2014) 772 - 781

[7] Copyright (c) 2004, Vincent Roy All rights reserved.

[8] (Koca A.,Oztop H.F.,Varol Y.,Koca G.O.) *Estimation of solar radiation using artificial neural networks with different input parameters for Mediterranean region of Anatolia in Turkey.* Expert Systems with Applications 2011;38:8756-62

[9] Elminir HK, Elsayed T.S. *Estimation of solar radiation components incident on Helwan site using neural networks.* Solar energy 2005;79:270-9

[10] Rehman S., Mohandes M. *Estimation of diffuse fraction of global solar radiation using artificial neural networks.* Energy Sources, partA2009;31:974-84

[11] *listado longitud latitud muncipios de Espaa a partir de datos del INE y Google maps www.bussinesintelligentfacil.info*

[12] Mellit, A., Kalogirou, S.A., *Artificial Intelligence techniques for photovoltaic applications: a review* 2008 Prog.Energy Combust. Sci. 34(5), 574-632

[13] *MATLAB (c) documentation R2015b* TheMathworks.Inc.

[14] P. Bacher, H. Madsen, H.A. Nielsen, Online short-term solar power forecasting, Solar Energy 83 (2009) 17721783

[15] Dazhi Yang, Panida Jirutitijaroen, Wilfred M. Walsh, Hourly solar irradiance time series forecasting using cloud cover index, Solar Energy 12 (2012) 35313543

[16] J.C. Cao, S.H. Cao, Study of forecasting solar irradiance using neural networks with preprocessing sample data by wavelet analysis, Energy 15 (2006) 34353445.

[17] M. Chaabene, M. Ben Ammar, Neuro-fuzzy dynamic model with Kalman filter to forecast irradiance and temperature for solar energy systems, Renewable Energy 7 (2008) 14351443

[18] A. Hammer, D. Heinemann, E. Lorenz, B. Lückehe, Short-term forecasting of solar radiation: a statistical approach using satellite data, Solar Energy 13 (1999) 139150.

[19] C.W. Chow, B. Urquhart, M. Lave, A. Dominguez, J. Kleissl, J. Shields, B. Washom, Intra-hour forecasting with a total sky imager at the UC San Diego solar energy test bed, Solar Energy 85 (2011) 28812893

[20] A. Mellit, A.M. Pavan, A 24-h forecast of solar irradiance using artificial neural network: application for performance prediction of a grid-connected PV plant at Trieste, Italy, Solar Energy 5 (2010) 807821.

[21] Fei Wang, Zengqiang Mi, Shi Su, Hongshan Zhao, Short-term solar irradiance forecasting model based on artificial neural network using statistical feature parameters, Energies 5 (2012) 13551370

Resumen Trabajo de Fin de Grado: Aplicación de técnicas de Machine Learning para la predicción a corto plazo de radiación Solar

Pablo Egea Hervás

19 de septiembre de 2016

1. Resumen

El objetivo de este proyecto es utilizar técnicas de machine learning para realizar la predicción a corto plazo (*nowcasting*) de la radiación solar. En particular, la técnica usada serán redes neuronales artificiales (*Artificial Neural Networks*). La meta es conseguir predicciones precisas para un tiempo cercano usando información del instante presente. En este trabajo se realiza la prueba de diferentes arquitecturas de redes neuronales, así como diferentes ventanas de tiempo. Para ello, el GTER (Grupo de Trabajo de Energías Renovables) ha cedido información recogida por una estación radiológica situada en el tejado del edificio de los laboratorios de la Escuela Superior de Ingeniería de la Universidad de Sevilla, situado en Sevilla en la isla de la Cartuja. Se dispone así de varias variables radiológicas y meteorológicas: radiación global y difusa, presión atmosférica, humedad relativa y temperatura del aire. Usando la hora del día se calcula también la posición del sol como otra variable disponible.

Con estos datos y el uso del software MATLAB y su módulo (*toolbox*) para redes neuronales artificiales se crean y se entrenan diferentes redes para ser después puestas a prueba. El documento principal está dividido en tres partes: el entorno teórico en el que se trabaja, el procesado de los datos y finalmente los experimentos realizados y las conclusiones. Después de realizar diferentes experimentos con diferentes combinaciones de datos de entrada, diferentes arquitecturas de redes y diferentes ventanas de predicción se puede concluir que con los datos disponibles sólo es posible predecir de manera precisa días, o partes de un día, claros mientras que en los días nublados la radiación varía de tal manera que la red es incapaz de predecir su comportamiento.

2. Introducción

La razón por la que se busca la predicción precisa de la radiación solar a corto plazo tiene que ver con el uso que hacen las plantas solares de esa radiación. La energía que producen estas estaciones depende directamente de la radiación solar que llega al suelo en un instante. Actualmente es muy difícil predecir de manera precisa a corto plazo la radiación solar. Esto implica que es difícil conocer de antemano la cantidad de energía que producen estas plantas instantáneamente y esto sería de gran ayuda a la hora de evitar problemas de abastecimiento de la línea o de evitar excedentes de energía.

Las predicciones que se realizan normalmente de la radiación solar suelen ser en términos de valores medios y en tiempos de un día hasta tiempos de un mes o un año. Existen varios trabajos que eligen utilizar técnicas modernas como las redes neuronales artificiales en lugar de los modelos teóricos. Las redes neuronales artificiales han probado conseguir mejores resultados que métodos clásicos. De acuerdo con [6], se ha comprobado que las redes neuronales artificiales han conseguido una mejor aproximación de la realidad que el modelo de Ångstr´om, y modelos lineales,

no lineales convencionales. Las redes neuronales son una gran herramienta para interpolación de datos, reconocimiento de patrones y regresión no lineal. En el capítulo dos del documento se detalla la base teórica de esta herramienta.

Existen varios trabajos cuyo objetivo es predecir la radiación solar. La mayoría apunta a predicciones más a largo plazo como un día, una semana, un mes o un año. Hay también algunos trabajos que tienen el objetivo de predecir la radiación solar para distintas localizaciones. En este trabajo la meta es predecir la radiación solar en ventanas pequeñas de tiempo como media hora, una hora o dos horas. A continuación se nombran algunos de estos trabajos.

Como se ha dicho en este campo existen varios trabajos que usan técnicas de machine learning para la predicción de la radiación solar pero la mayoría están orientados a predicciones a largo plazo (*forecasting*). Muchos de estos trabajos usan, precisamente, redes neuronales artificiales. Estas redes permiten obtener una relación entre unos datos de entrada y sus correspondientes datos objetivo que pueden no tener una relación conocida de manera analítica. Al ser la radiación solar una de estas variables con una relación bastante compleja con el clima, las redes neuronales son una buena herramienta para lidiar con este tipo de problemas. En los siguientes trabajos se han usado redes neuronales u otra forma de machine learning para la predicción de la radiación solar.

En el trabajo [3] se usa también el software MATLAB para trabajar con redes neuronales, aunque en una versión del módulo ya obsoleta. Aquí se usan la temperatura máxima, la velocidad media del viento, las horas de luz solar, la humedad relativa media y la radiación solar para generar un modelo climático de Al Ain prediciendo la radiación solar.

En el artículo [6] se resumen varios trabajos que usan redes neuronales artificiales para predecir la radiación solar. Algunos de ellos son, [8], [9] y [10]. En [8] se desarrolla un modelo de red neuronal para estimar la radiación solar de una localización usando combinaciones de latitud, longitud, altitud, el mes, la temperatura media, la nubosidad media, la velocidad media del viento y las horas de luz solar. En [9] se unas redes de tipo *feedforward* para predecir diferentes tipos de radiación usando diferentes variables de entrada. En [10] se usan combinaciones de día, temperatura máxima del aire, temperatura media del aire y la humedad relativa para obtener la radiación difusa y se concluye que los mejores resultados se obtienen usando la humedad relativa y la temperatura media.

Uno de los problemas de las redes neuronales artificiales, es la optimización de la arquitectura de la red y de los datos de entrada. En [5] se usan innovadoras técnicas que permiten una optimización para la arquitectura de la red. En este trabajo ([5]) se pretende la radiación media diaria.

En este trabajo se usan también redes neuronales pero la predicción que se busca es a corto plazo. Para estas predicciones a corto plazo se necesitan datos meteorológicos y radiológicos recogidos cada poco tiempo. Estos datos los proporciona el GTER. Recogidos desde la estación radiológica situada en el tejado del edificio de los laboratorios de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla, estos datos están recogidos cada cinco segundos durante un período de tres años

(2012, 2013 y 2014). Lógicamente esto es un gran banco de datos que necesita de un procesado y un filtrado. Para ayudar al filtrado, el GTER también ha proporcionado un diario de la estación radiométrica que recoge las variables fiables de cada día, así como comentarios.

Para implementar las redes neuronales, como ya se ha comentado, se usa MATLAB por su módulo de redes neuronales artificiales que permite trabajar fácilmente con ellas. Como se ha dicho el proyecto consta de cuatro capítulos principales: entorno teórico, procesamiento y filtrado de los datos, experimentos realizados y conclusiones.

3. Machine Learning

En este capítulo se detalla la parte teórica de las redes neuronales y en particular de los perceptrones multicapa. [] define machine learning como la programación de ordenadores para optimizar un criterio de rendimiento usando datos de ejemplo o experiencia pasada. Se dice también que que este tipo de algoritmos se necesita cuando no es posible que un programa de ordenador, con ecuaciones implementadas, resuelva un problema; ya sea bien porque no se conoce el proceso o no existe información suficiente sobre él. Las redes neuronales se basan en datos, de los que recogen la forma en la que están relacionados, por medio del proceso de aprendizaje. Machine learning se utilizan para aprender asociaciones, clasificaciones o regresiones (lineales o no lineales). En este caso se usarán para obtener una regresión no lineal de los datos. En este trabajo se usan perceptrones multicapa, una determinada arquitectura de red neuronal.

Una red neuronal es un procesador distribuido masivamente paralelo distribuido formado por unidades de procesamiento simples que tienen una tendencia natural para conservar conocimiento basado en la experiencia y disponer de su uso. Está basado en el cerebro humano porque necesitan de un proceso de un proceso de aprendizaje que se basa en variar los pesos de las conexiones entre esos procesadores básicos (neuronas). Un perceptron multicapa es una red neuronal de más de dos capas (capa de entradas, capa(s) oculta(s) y capa de salida) que tiene conectividad completa. Todas las neuronas de una capa están conectadas a todas las neuronas de la capa siguiente. Básicamente lo que hace una neurona, es recibir un pulso eléctrico, transformarlo y mandarlo a la siguiente neurona. En el caso de las redes artificiales este pulso eléctrico son las variables de entrada modificadas por las neuronas anteriores.

En la figura se puede ver un modelo de neurona. Una neurona es, de acuerdo con [2]: una unidad de procesamiento de información que es fundamental para la operación de una red neuronal. Los elementos básicos de este modelo son: las conexiones , caracterizadas por un peso específico; un sumador que suma las entradas pesadas con los pesos sinápticos; una función de activación que limita la amplitud de la salida de una neurona. El sesgo (*bias*) incluido en el modelo incrementa o disminuye la entrada a la red de la función de activación, dependiendo de si es positivo

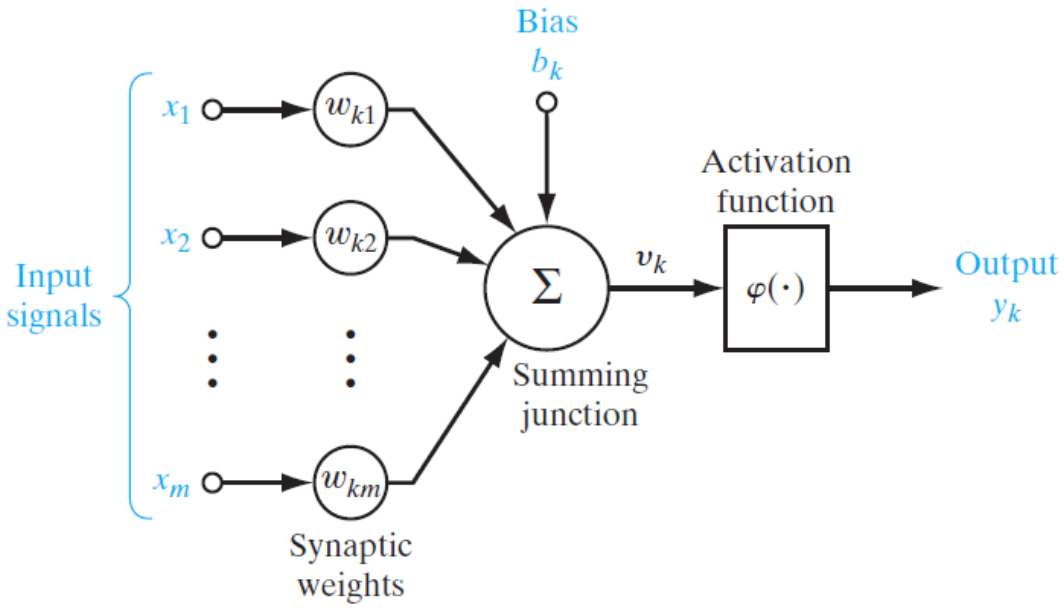


Figura 1: Modelo de neurona

o negativo. Lo que hace una neurona matemáticamente se muestra en las siguientes ecuaciones:

$$y_k = \phi(u_k + b_k) \tag{1}$$

donde

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2}$$

En resumen la señal de salida de una neurona es el resultado de aplicar la función de activación a la entrada ponderada con los pesos sinápticos más el sesgo.

Existen dos funciones de activación básicas: la función umbral y la función sigmoidal. La función umbral devuelve un 1 si el argumento es positivo y 0 si el argumento es negativo. La función sigmoidal tiene forma de "S" y es la mas comunmente usada en redes neuronales por carácter equilibrado entre comportamiento lineal y no lineal. Estas dos funciones tienen un rango entre 0 y +1. A veces es necesario tener un rango entre -1 y +1. Para ello se rescribe la función umbral como la función signo y la función sigmoidal se puede sustituir por la tangente hiperbólica.

Otra característica a tener muy en cuenta en las redes neuronales es su arquitectura. Existen tres arquitecturas básicas: Redes prealimentadas de una sola capa, redes prealimentadas multicapa y redes recurrentes (con retroalimentación). En las redes de una sola capa las entradas están directamente conectadas a la capa de salidas, son prealimentadas porque la información va estrictamente en un solo sentido (desde la entrada a la salida). En las redes multicapa hay, al menos, una capa oculta entre las entradas y las salidas. Pueden estar completamente interconectadas o

parcialmente interconectadas. Las redes recurrentes contienen, al menos, un bucle de retroalimentación. Las redes dinámicas son un tipo de redes recurrentes, se usan para solucionar problemas de series temporales entre otros.

En la figura 2 se puede apreciar la arquitectura de una red multicapa. En este trabajo se emplean perceptrones multicapa, que son un tipo de red neuronal con una arquitectura multicapa totalmente interconectada. Cada una de las conexiones indicadas en el esquema tiene asociado un peso sináptico. Se ha mencionado con

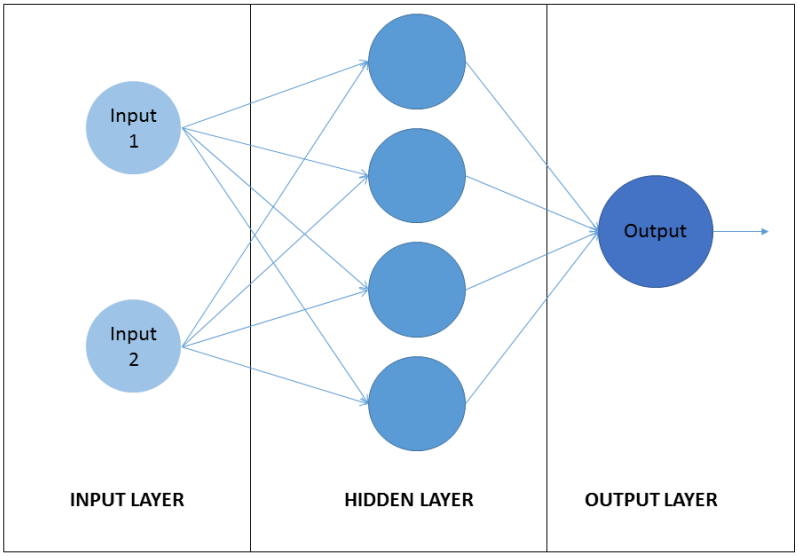


Figura 2: Arquitectura de un perceptron multicapa

anterioridad que las redes neuronales son capaces de *aprender*. Este proceso de aprendizaje se basa en, mediante ejemplos, ajustar estos pesos sinápticos para que cuando las entradas sean ponderadas con ellos la salida esté lo más cercana posible del valor objetivo. Para este proceso es necesario propagar una señal de error hacia atrás en la red (desde la salida hacia la entrada). Por lo tanto, en la red existen dos tipos de señales: las señales de función y la señal de error. El algoritmo de aprendizaje más básico se conoce como algoritmo de propagación hacia atrás, puesto que la señal viaja en el sentido contrario.

El algoritmo funciona como sigue: dado un conjunto de muestras N donde $x(n)$ son las entradas y $d(n)$ son los valores objetivo para $n = 1, ..., N$. Una neurona j en la capa de salida produce una salida $y_j(n)$. La señal de error es $e_j(n) = d_j(n) - y_j(n)$. La energía del error se define a su vez como $\mathscr{E}_j(n) = \frac{1}{2}e_j^2(n)$. Sumando para todas las neuronas en la capa de salida se obtiene la energía del error de todas las neuronas de la capa. Se puede usar también el valor medio de la energía del error para el conjunto de muestras N. El objetivo es actualizar los pesos sinápticos para reducir este error al mínimo posible. Dependiendo de cuando se haga esta actualización existen dos tipos de aprendizaje: aprendizaje en línea si se actualizan después de cada muestra o

aprendizaje por lotes si se actualizan después de que todas las muestras hayan sido presentadas a la red. En este trabajo se va a usar aprendizaje por lotes porque la implementación en MATLAB de este tipo de aprendizaje está mejor implementada y porque permite la paralelización del proceso. La desventaja es que requiere mucha memoria.

La actualización de los pesos sigue la siguiente ecuación: $\Delta w_{ij}(n) = \eta \delta_j(n) y_j(n)$ donde η es el parámetro de tasa de aprendizaje y δ_j es el gradiente de la energía de error. El gradiente de error depende de los pesos sinápticos en la forma: $\delta_j(n) = \phi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$ donde ϕ es la función de activación y δ_k es el gradiente de error de la capa siguiente (de donde procede la señal de error ya que va en sentido opuesto). Para más detalles ver [2].

El método de entrenamiento que se usa en este trabajo es el método Levenberg-Marquardt que aúna el método de Newton, que converge rápidamente pero es posible que diverja, y el método del gradiente descendente, que asegura convergencia pero converge lentamente. La unión de estos dos métodos permite obtener lo mejor de cada uno y reducir las desventajas. Este es el método usado porque es el método recomendado por MATLAB para este tipo de problemas. El desarrollo matemático de este método se puede consultar en [**?**] en el capítulo 4, sección 16.

4. Procesamiento de datos

Una vez se ha descrito la base teórica de cómo funciona una red neuronal y como *aprende* lo siguiente a saber es con qué variables va a ser entrenada. Para ello en esta sección se describen los datos disponibles y como se procesan y se filtran. Lo primero es conocer los datos de los que se dispone.

Como se indicó en la introducción los datos son proporcionados por el GTER tomados desde la estación radiológica en el tejado del edificio de los laboratorios de la ETSI en Sevilla. Proporciona las siguientes variables:

- Radiación difusa: en W/m^2
- Radiación global: en W/m^2
- Radiación global a $27°S$: en W/m^2
- Radiación global a $27°$ desde célula: en W/m^2
- Pirgeómetro: en W/m^2
- Horizontal desde célula: en W/m^2
- $H_{b0}NIP$: Radiación directa medida con la incidencia normal al pirheliómetro en W/m^2
- $H_{b0}CHP1$: Radiación directa medida con un pirheliómetro en W/m^2

- Radiación difusa bolas: en W/m^2
- Temperature: in $°C$
- Wind Velocity: in m/s.
- Wind direction: en grados.
- Pressure: en $mBar$
- Relative humidity: en %

Todos estos datos están recogidos en documentos .txt (uno por día). Además de estos datos, el GTER proporciona también un diario de la estación radiométrica que recoge que variables son fiables cada día. Para trabajar con estos datos en MATLAB fácilmente se guardarán en una estructura de celdas con una celda para cada día. Cada una de estas celdas es una estructura con cuatro campos: tiempo, valores, ok y cenit. En tiempo se recoge una matriz con tres columnas donde la primera indica la hora, la segunda los minutos y la tercera los segundos del momento en el que se han recogido los datos. En el campo valores se recoge una matriz cuyas columnas representan las distintas variables y las filas son las distintas muestras. El vector ok es un vector de unos y ceros que indican si esa variable es fiable o no ese día. Por último el campo cenit recoge el ángulo cenit del sol en el momento en que la muestra fue recogida. Se calcula con las funciones *sunPosition* de Vincent Roy (Copyright (c) 2004).

Es importante también mencionar que los días se han dividido en cuatro categorías dependiendo de su nivel de claridad: los días de tipo 1 son totalmente claros, los días de tipo 2 son claros por la mañana, los días de tipo 3 son claros por la tarde y los días de tipo 4 no son claros en absoluto.

A continuación se explica el proceso seguido para llegar desde los archivos .txt hasta el archivo .mat con la estructura de celdas.

4.1. Procedimento de procesado

Con el objetivo de hacer los datos manejables en MATLAB lo primero que se hará es pasar los datos desde los documentos text a MATLAB. Los datos se guardarán en una estructura de celdas, donde cada celda corresponderá a un día. El archivo se llamará data.mat, por ejemplo data1,2.mat corresponde al día 2 del año 1. El programa `data_processing_TFG.m` lee los datos de los archivos text y los guarda en una estructura de celdas con los campos *.time* y *.values*. Los archivos .txt están nombrados de cierta manera (meteo_año_día) por lo que es sencillo asignar el año y el día a la celda. Lo siguiente es añadir el campo *.ok*.

Una vez que se tiene la estructura de celdas con los campos tiempos y valores, el siguiente campo a añadir es el campo ok. El campo ok es un vector de ceros y unos, hay un vector para cada celda. Estos ceros y unos indican si las variables son usables o no. Estos datos están recogidos en el diario de la estación radiométrica

proporcionado por el GTER. Aquí solo están recogidas las variables de radiación, no así las variables meteorológicas. Para añadir este campo se usa el programa lectura_csv. Este programa coge las hojas del archivo MS Excel del diario (cada hoja corresponde a un año) guardadas previamente en formato csv para trabajar más fácilmente en MATLAB, las lee y asigna los valores pertinentes a cada vector ok. Estos datos servirán posteriormente para filtrar los datos con los que se va a trabajar.

Lo siguiente es arreglar los datos de las celdas para que cada día tenga la misma estructura pues se detectó que algunos datos no estaban bien y en algunos días había horas de más o horas repetidas. Los dos problemas principales encontrados fueron:

- Celdas con más filas de las debidas: como se ha mencionado anteriormente los datos son recogidos cada 5 segundos, por lo que en un día de 24 horas equivale a 17280 muestras. Cuando se comenzó a trabajar con los datos se detectó que había días que tenían mas filas de las debidas porque había horas que se repetían. La solución que se adoptó para este problema fue aislar esos días y borrar las filas que correspondías a horas repetidas. Para esto se usaron las funciones Test_Data y Which_hour. La primera detectaba los días con filas de más y la segunda procesaba esos días para borrar las filas de sobra.

- Celdas con el número de filas debido pero con horas faltantes: se detectó posteriormente que algunas celdas no seguían la serie temporal debida repitiendo algunas horas haciendo que hubiera horas que no existen en ese día. Cuando se detectó este otro problema se decidió procesar todos los días y comprobar que todos seguían la misma serie temporal. En los días con fallos estos fueron reescritos forzando a que se siguiera la serie temporal y borrando las horas que se repetían. Para ello se amplió cada celda, si la siguiente muestra no correspondía con la hora que debía tener se movía una fila hacia abajo y en la nueva fila vacía se reescribe el tiempo correcto y los valores se pusieron a cero. Siguiendo este proceso al final queda, en cada celda, una matriz con el doble de filas en la que las filas posteriores a la 17280 contiene los datos repetidos. Finalmente se borran las muestras sobrantes y cada celda contiene una estructure con los datos debidamente ordenados, aunque con datos faltantes. Para realizar este proceso se usaron, en orden, las funciones *Reduce*, *Rearrange* y *Amplify*. Para rellenar los datos faltantes se usa la función *data_interpolation* que interpola los datos que faltan cada día usando los datos existentes como nodos de interpolación.

Por último se añade el campo cenit calculando el ángulo cenit a partir de la hora de cada muestra y usando las funciones de la posición del sol (sun_position) de Vincent Roy.

A continuación se describe cómo se preparan estos datos base para el entrenamiento de redes neuronales.

4.2. Preparación para el entrenamiento.

Se ha pretendido hacer una serie de funciones que permitan el fácil manejo de los datos y la mayor determinación en los parámetros de los experimentos. Al realizarse varios experimentos es importante tener un sistema flexible que permita la fácil creación de diferentes experimentos con diferentes parámetros con su posterior aplicación a los datos. La función *exDef* pide los distintos parámetros por pantalla, permite crear un nuevo experimento o modificar uno existente y guarda los guarda en una estructura Experiment contenido en un archivo con el nombre dado por el usuario. Después la función *dataPreparationStructTime* alimentada por la estructura Experiment realiza el procesado de los datos base de acuerdo con los parámetros definidos por el experimento: realiza medias móviles para suavizar los datos sin procesar (debido a la estructura intrínseca de los sensores los datos presentan pequeños picos que no se corresponden con la realidad), se definen los pares entrada-objetivo, se desestiman los datos correspondientes a la noche y los que no proporcionan información fiable. El último paso es usar la functión *prepareInTaForTraining* para ponerlo todo en forma matricial para poder alimentarlo a la red neuronal.

5. Conclusiones

Una vez se han realizado varios experimentos con diferentes entradas, diferentes estructuras de la red y diferentes ventanas de predicción se ha llegado a las siguientes conclusiones:

- Con los meteorológicos disponibles se obtienen predicciones precisas para días claros (días de tipo 1). Si los días que se usan incluyen algo de nubosidad el error de la predicción crece. Esto puede venir motivado por la falta de datos meteorológicos directamente ligados con el índice de claridad, que de acuerdo con [12], es la variable más relevante para una red neuronal que trabaja con predicciones de radiación solar junto con la masa relativa de aire.

- El conjunto de variables que consigue mejores resultados es el conjunto del cenit, la presión atmosférica, la humedad relativa, la radiación global y el día del año si sólo se pretende predecir días claros. Usando días con algo de nubosidad esta combinación no consigue una buena generalización. El RMSE fue muy alto cuando se presentaron los datos de prueba. Para una combinación de días claros y algo nubosos (tipos 1,2 y 3) la inclusión de la radiación difusa probó ser mejor mejorando el rendimiento de la red con los datos de prueba. Esto está motivado por la relación entre la radiación difusa y la claridad. Sin embargo, cuando solo días claros son tenidos en cuenta, la inclusión de la radiación difusa empeora el rendimiento. La conclusión a la que se llega es: para la predicción general de la radiación solar (sin excluir días nubosos) la radiación difusa es una variable adecuada mientras que si solo se quiere usar la red para predecir días claros no es una buena opción.

- Con algunos números de neuronas escogidos para el MLP se desveló algo de gran importancia. Para las redes de un tamaño medio y grandes (80-100 neuronas) si se usan todos los datos disponibles para el entrenamiento la red no es capaz de generalizar (*overfitting*), siendo el error obtenido para los datos de prueba mucho mayor que para los datos de entrenamiento. Este hecho previene de usar toda la información disponible para evitar posibles problemas de generalización.

- Si los días son claros el MLP es capaz de obtener un buen rendimiento con un $RMSE$ de $5.89 W/m^2$ que es un error de $0.0058 kW/m^2$. Si los días no son del todo claros (tipos 2 y 3) la predicción general no es tan buena, pero si se centra la atención en las partes claras del día la predicción es bastante certera pero la incertidumbre en las partes nubosas hace que el rendimiento general disminuya.

- La ventana de predicción que obtiene el mejor resultado es la media en una hora para la hora siguiente al instante de tiempo presente, si solo se tienen en cuenta días claros. Para días no enteramente claros la mejor es la predicción de la media en una hora en la misma hora del día siguiente (24 horas). Queda claro que la variable a predecir para obtener mejores resultados es la media en una hora. La otra ventana que obtiene un rendimiento aceptable, aunque solo teniendo en cuenta días claros, es la ventana de 15 min calculando la radiación media en esos 15 min. Esta última es la ventana que más se ajusta al propósito de este trabajo.

Para continuar y mejorar este trabajo se proponen dos líneas de trabajo futuro:

- Usar procesado de imágenes por ordenador para conocer la presencia de nubes sobre la zona de estudio. Se propone esta medida para alimentar a la red alguna variable que esté directamente relacionada con la nubosidad. Esto junto con la velocidad y dirección del viento debería mejorar los resultados. Sería un trabajo a largo plazo ya que es necesario un histórico para entrenar las redes neuronales.

- Otra línea de trabajo no explorada en este trabajo es el uso de redes dinámicas viendo el problema como uno de predicción de series temporales en vez de una regresión no lineal. Este tipo de redes se usa en este tipo de problemas.