COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

BACHELOR´S FINAL PROJECT

USING NEURAL NETWORKS TO FORECAST THE ELECTRIC DEMAND IN IBERIA

Author: Guillermo Varas Yuste

Director: Rosendo Castañón Naseiro

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título Using Neural Networks to Forecast the Electric Demand in Iberia en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2021/22 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Guillermo Varas Yuste Fecha: 30/08/2022

GUILLERMO VARAS YUSTE
Firmado digitalmente por GUILLERMO VARAS YUSTE
Nombre de reconocimiento (DN): cn=GUILLERMO VARAS YUSTE, o, ou, email=gvyuste@gmail.com, c=ES
Fecha: 2022.09.13 16:01:10 +02'00'

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Rosendo Castañón Naseiro
Firmado digitalmente por Rosendo Castañón Naseiro
Fecha: 2022.09.13 16:41:21 +02'00'

Fdo.: Rosendo Castañón Naseiro Fecha: 30/08/2022

COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

BACHELOR´S FINAL PROJECT

USING NEURAL NETWORKS TO FORECAST THE ELECTRIC DEMAND IN IBERIA

Author: Guillermo Varas Yuste

Director: Rosendo Castañón Naseiro

Madrid

ABSTRACT

The electricity network sector is undoubtedly one of the most relevant and powerful giants to be known of in the modern world. Moving billions of euros every year, becomes of great interest to the general public and more specifically to companies such as the investment type due to its increasing short-term volatility. Therefore, finding a reliable way to forecast the demand prices considering a large variety of factors would be very interesting both academic and economically. This is where neural networks come into place, being an extensively used prediction method over the last few decades in a wide range of fields, namely artificial intelligence, social trends detection, data management, machine learning and even medicine.

A variety of input factors should be considered when applying the neural networks methodology, the main ones being temperature, climatology and historic demand.

The two main items that will need deeper research are to be the peninsular electric network operation and its intrinsic relationship with each one of the individual factors that will be used to perform the calculations inside the neural network, and of course, the operation method of the neural network, being the latter the main focus of the study as not so much on how the electric system works.

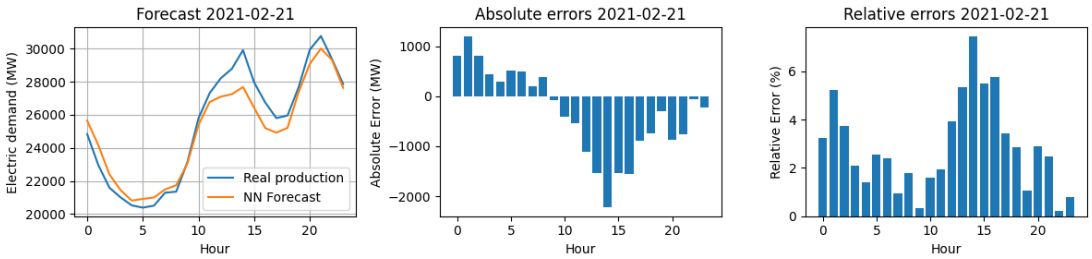
Data both from Red Eléctrica Española S.A. and National Oceanic Atmospheric Administration (USA) has been collected and used to train a pair of three-layered feedforward neural networks using the backpropagation algorithm, modifying their parameters to fulfill the objective of the project, which is a semi-accurate prediction on the peninsular electric demand.

After studying and learning about the neural networks concept, we have successfully been able to create said the two neural networks. One of them receiving 4 inputs, mean temperature, type of day, month and season and the other network receiving a total of 27 different inputs, that correspond to month, season, type of day and 24-hourly mean temperatures of the desired day for prediction.

These both networks behavior is determined by three different customizable parameters which are the number of neurons in the hidden layer, a concept more deeply explained later in the paper, the number of total iterations the networks do and finally the learning factor, a parameter also explained later in detail.

Combining al these factors, both the training inputs (temperature and demand from 2020) and the network configuration we have successfully achieved peninsular electric demand predictions with an overall mean absolute percentual error of just under 4%, that as explained more in detail in the introduction, is more than enough to ensure the efficacy and usefulness of the forecasted data.

An output example of the neural network prediction, being compared with real demand can be seen in the following image.



We have studied how changing the different configuration parameters affect the total overall output errors and also studied the output difference between the two distinct neural networks and their precision depending on the day we have wanted to forecast. Critically analyzing output data and finding out why on specific cases the predictions where not as precise as ideally wanted.

Finally, new possible ideas have been brought into the table to further improve the overall results on hypothetical future projects, like different input data consideration such as regional temperatures, difference between workdays or national legislation due to the COVID-19 pandemic effect, which has been highly impactful on industries not only nationally but across the globe

RESUMEN

El sector de redes eléctricas es sin duda uno de los mayores y más relevantes gigantes industriales conocidos en el mundo contemporáneo. Moviendo cantidades billonarias de capital todos los años, resulta de gran interés al público general y más específicamente a compañías inversoras debido a su volatilidad a corto plazo. Siendo así muy interesante tanto académica como económicamente encontrar una forma fehaciente de predecir la demanda eléctrica tomando y considerando múltiples factores. Es aquí cuando toma lugar el uso de redes neuronales, un método ampliamente utilizado durante las últimas décadas en variedad de campos de investigación como la inteligencia artificial, la detección de patrones sociales, manejo de datos, *machine learning* o incluso medicina.

Una gran variedad de factures de entrada deben ser considerados cuando se aplica el método de las redes neuronales a nuestro problema, siendo unos de ellos, datos como la temperatura, climatología o la demanda eléctrica histórica del país.

Los dos principales núcleos de estudio en este proyecto han sido el funcionamiento y operación del sistema eléctrico español y su relación con los factores de entrada a tener en cuenta sobre la variación de la demanda eléctrica y por supuesto y principalmente, el funcionamiento y la operación de una red neuronal artificial multicapa, siendo este el objeto de estudio mayoritario del trabajo.

Datos tanto de la Red Eléctrica Española S.A. como de la NOAA (National Oceanic Atmospheric Administration (USA)) del 2020 han sido recolectados, tratados y usados para entrenar no una, sino dos redes neuronales formadas por tres capas que usan los algoritmos de *feedforward* y *backpropagation* para lograr cumplir el objetivo principal del trabajo, lograr una predicción semi-precisa de la demanda eléctrica peninsular.

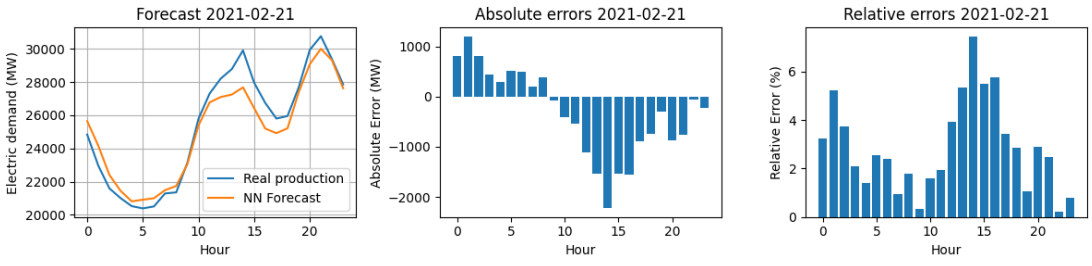
Después de estudiar y aprender sobre el concepto de las redes neuronales, hemos sido capaces de crear ambas redes neuronales de forma exitosa.

Una de ellas recibe 4 entradas, siendo estas la temperatura media del día a predecir, el tipo de día que es según su laboralidad, el mes y la estación del año en que se encuentra, mientras que la otra red neuronal es capaz de recibir 27 entradas distintas, siendo estas el mes, estación y el tipo de día según su laboralidad y adicionalmente 24 datos de temperatura de ese día, correspondientes a las temperaturas medias horarias a lo largo del mismo.

El comportamiento y entrenamiento de las redes viene dado por la configuración que apliquemos a las mismas, siendo esto un conjunto de tres parámetros personalizables. Estos son el número de neuronas en la capa oculta, número de iteraciones máximas y factor de aprendizaje, todos conceptos explicados en mayor detalle dentro de la memoria.

Combinando todos estos factores, tanto los datos de aprendizaje (temperatura y demanda del año 2020) y la propia configuración de la red hemos conseguido efectuar de forma exitosa predicciones de demanda eléctrica con un valor de error medio absoluto porcentual de entorno al 4%, que, como explicado en la introducción es más que suficiente para considerar las predicciones como útiles a nivel estadístico y económico.

Un ejemplo de salida de la predicción de un día del año 2021 se puede ver en la siguiente imagen.



Se ha estudiado el efecto de la modificación de los diferentes parámetros en los errores de las predicciones de salida y realizado una comparativa entre los diferentes resultados ofrecidos por las dos redes neuronales confeccionadas. Analizando de forma crítica los resultados y encontrando la razón por la cual las predicciones no han sido tan precisas como deseado en ciertas fechas específicas.

Finalmente, se ha reflexionado sobre posibles formas de mejorar o añadir entradas a la red en hipotéticos futuros proyectos, como por ejemplo la separación de temperaturas en distintos territorios según su temperatura media anual, discernir entre los mismos días laborales según su proximidad a fines de semana o días festivos, o la gran implicación que ha tenido la pandemia global del COVID-19 en el año de entrenamiento de la red junto con los cambios de legislación debido a la misma.

INDEX

MEMORY DEVELOPMENT

1. Introduction

1.1. Overview on the Spanish electric system

Even though the deep comprehension of the electric Spanish system is not the main objective of the paper, a shallow introductory knowledge is much needed in order to apply the neural network method, using its context and characteristics to create the best possible outcome. The Spanish electric network is composed of three distinct networks on its own, the Peninsular, Balear and Canary, the latter one being independent of the other two due to the geographical distance. The Peninsular network is connected to neighbor countries as Portugal, France and Morocco and it is linked to the Balear network through an underground submarine connection known as Project Rómulo being able to transfer up to 400 MW of electrical power. Since this paper will focus on the Peninsular network, it is important to acknowledge its most important features, summarized on the following table of contents.

Total Power Installed	115608 MW
Electric Lines Installed	44687 km
Transformation Capacity	93871 MVA
Absolute Maximum Power	45450 MW
Maximum power in 2021	37171 MW
Renewable energy percentage	46.7 %

Table 1: *Peninsular Network Characteristics*

The Red Eléctrica Española S.A. (REE) [1], where all the electrical data for this project has been collected, is the exclusive operator and manager of the electric transportation network, including both big Spanish island groups, Balear and Canary. One, if not the biggest challenge they face is maintaining the balance between electric generation and demand. Following, we expose the principal motives of this statement.

1.1.1. The production and demand equilibrium

When transporting electricity using power lines to deliver it all over the country it is crucial that the voltage value stays the way we want, and that is constant, on whatever value we are using, high or low voltage, but another critical value that must stay constant is the electrical frequency, the latter highly related to the balance between electric generation and demand. Frequency dependent machines are everywhere, from clocks and timers to industrial motors. A slight chance or imbalance on the input frequency would cause problems and malfunctions on these devices, leading into a catastrophic economic loss overall. To further understand the importance if this concept, we may present a classical generation and consumption example between hydroelectric turbines generating power using the potential energy from a waterfall and industrial charges consuming the generated power.

In an equilibrium state, the same amount of energy produced by the turbines is consumed by the charges, but if the said charges were to consume a larger amount of energy without changing the produced quantity, the remaining amount of energy necessary would be taken from somewhere else, in this particular case, most probably from the kinetic energy in the electric machines rotor, thus producing a greater resistance and slowing the turning speed down, consequently resulting in a lower frequency value to the originally desired one. To make this example even more illustrative, on a regular production scale (10000 MW) if the demand variation where to be upped by a slight 1%, the changes in frequency would constitute on a 1.5 Hz/min fall, which on a European 50 Hz network would be highly impactful, economically speaking.

The reverse situation may also be possible, as climate dependent energy sources are not able to constantly output the required amount of power. As seen in the previous table, in 2022, renewable energy sources constitute a total amount of 46.7 % of Spain’s energetic production, the two main methods being solar and aeolic. And because energy storage is still a very upcoming but underdeveloped technology, energy produced in these both ways is instantly utilized. Therefore, being highly dependent on climatologic conditions to output a large part of the total Spanish energy production. This concept can be better perceived in the following graphs, gotten from the REE official webpage.

Figure 1: *2022-08-26 Aeolic generation*

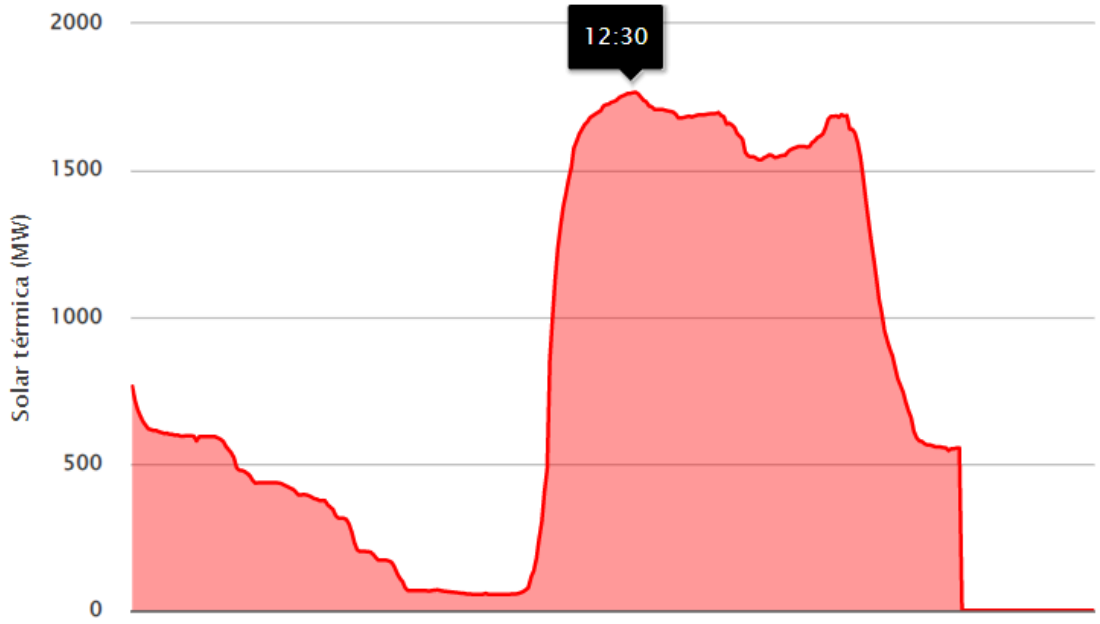


Figure 2: *2022-08-26 Solar generation*

On both graphs we observe a 24-hour period of energy production, both from the 25th and 26th of August 2022.The difference being the green representing aeolic and the red, solar energy. It is clearly visible that the two principal sources of renewable energy operate in a particular way, peaking in particular periods in time during the day, thus not being able to produce a constant amount of energy all day long. This is one of the main reasons why being able to predict demand is a very interesting capability.

Although his imbalance possibility is at first scary, there is no need to succumb to panic, because the Spanish electric network is equipped with different firewalls to prevent the frequency from changing downstream.

In brief summary these services are regulation and interruptibility. Regulation being separated into three different parts, primary, secondary and tertiary. Primary regulation is a mandatory service, which artificially modifies the rotatory speed of the generators in order to maintain electrical frequency. Secondary regulation is no longer a mandatory service but a voluntary paid one, consists of having a power reserve capable of responding in less than 20 seconds and maintaining energetic balance for a minimum of 15 minutes, the cost varies on availability and the amount of megawatts required. Finally, tertiary regulation is also mandatory and serves the purpose of restituting secondary regulation by modifying the generator programs. It must respond on a limit time of 20 minutes and last for at least 2 hours.

The second service provided to ensure power stability is the interruptibily service, which au contraire to the previously explained services, modifies demand, lowering it so the production amount is capable of meeting the demand needs. The power provider demands big industrial companies attached to the system to lower their energy demand, for this such industries receive double remuneration.

Even though energy balance regulation processes are able to respond between minutes to hours, hourly energy prices are set one day before by OMIE (electric market operator), consulting various commercial companies to set up the next day’s price program during that day’s market operation horizon.

This whole process must be done before 2 o'clock in the afternoon on the day before the electric market opens again. Then REE will set calculate and set up previously mentioned regulation prices in case they were needed.

Taking all these factors into consideration, we arrive at the conclusion that making short term demand predictions is both necessary and highly valuable, not only for economic purposes, but also for environmental ones, as a better energetic prediction would cut on renewable energy energetic waste taking further advantage of energy peaks.

Furthermore, a question strikes. How precise would our prediction have to be to be impactful. The answer may seem obvious, as precise as possible of course, but as seen in [2], a numeric study taken in the state of Arizona (USA) using the Monte Carlo methods, an energetic forecast with a precision of 5% is enough to deliver significant results, being that predictions with lower error percentage would almost not affect significantly the overall benefits, as long as demand peaks are well adjusted.

1.1.2. Decisive factors of electric demand

To study and model an electric demand forecast calculation, the first step is to determine what inputs are going to be used to achieve the goal. These will be the most determining factors that will be considered on our neural network model. The first factor that comes to mind is the economic wealth of the country of study. This is measured by the GDP (gross domestic product). It is an important factor as with a higher GDP the electric demand tends to also become higher.

On the order end, this pattern has been recently changing since the year 2015 at that is the main reason REE does not include GDP comparison graphs in their webpage since. Other consideration would be that the goal of this project is to make short term demand predictions, so since the GDP varies very little on a short time spectrum, it will not be taken into account as a determining factor.

The next factor that will be considered to bring or not into the inputs table will be the type of day, divided in weekdays, Saturdays or Sundays (this is mainly because most of the population works on weekdays and some on Saturdays, but a very small portion actually work on Sundays). In this specific case we can simply compare data from a workday and a weekend day to see that this factor is decisive. These next figures have been taken from the REE real-time datasets [3].

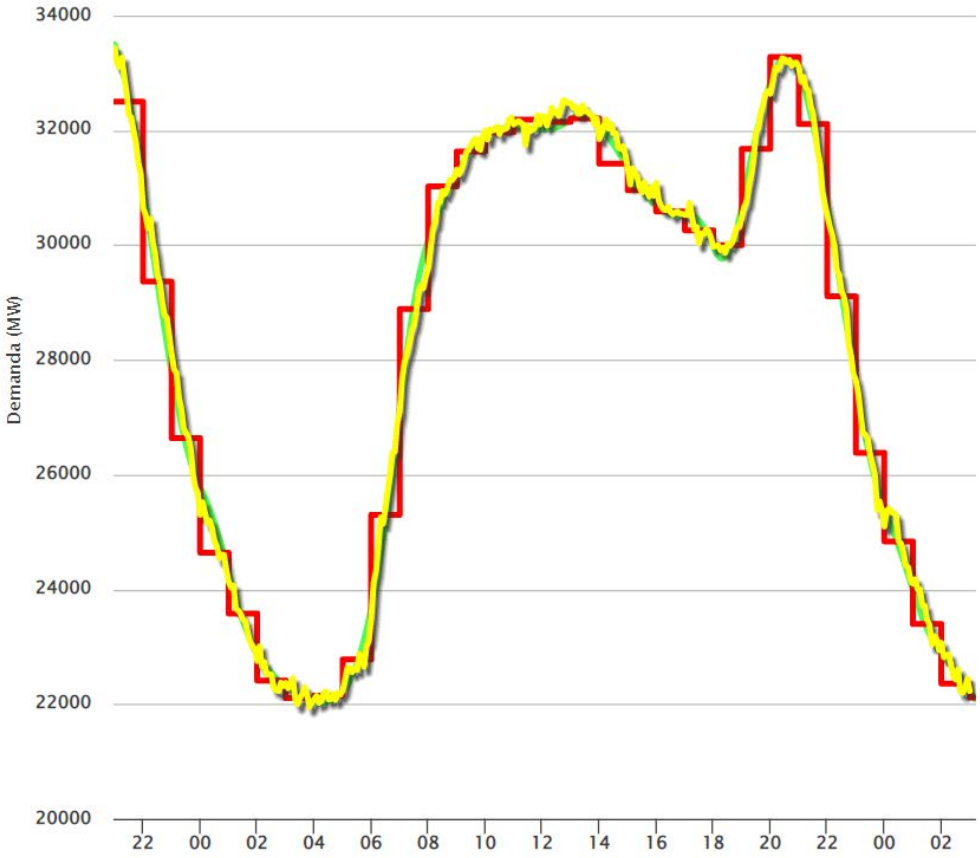


***Figure 3**: Electric demand 2022-03-15 (Tuesday)*

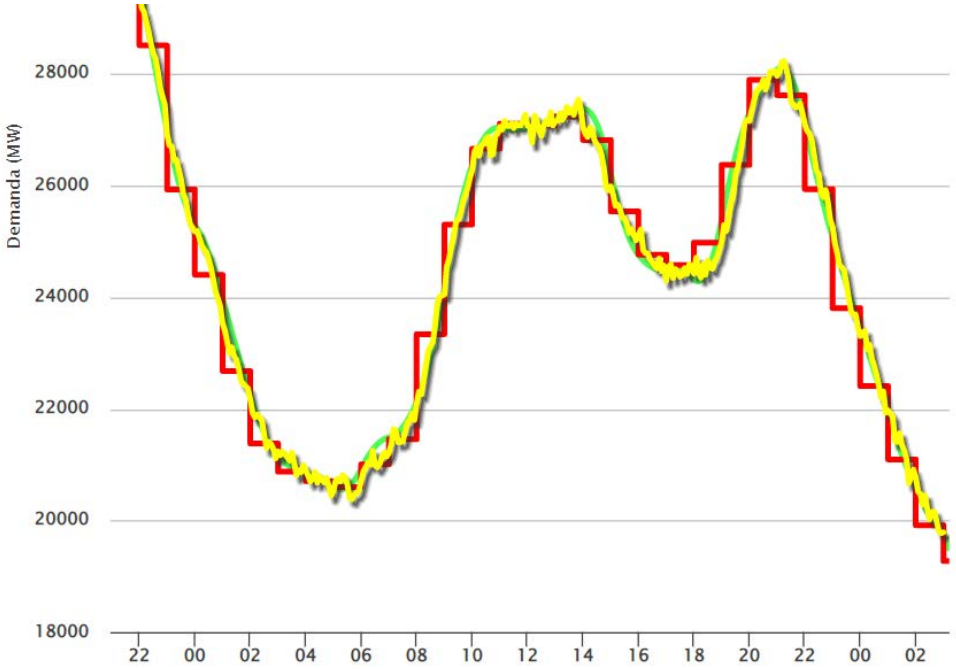


***Figure 4**: Electric demand 2022-03-20 (Saturday)*

As observed in both graphs there is a notable difference between both day's maximum and average energetic demand, being one on a workday and other on a Saturday.

The final factor that will be considered is temperature. This is probably what most people think when it comes to energetic consumption variables, and it is in fact a huge factor. To prove this, we just need to compare the demand graphs on weekdays with similar characteristics other than the temperature, for this we have chosen two different Wednesdays on opposite seasons, winter and summer. Figures 5 and have also been collected from the REE website, real-time demand datasets.

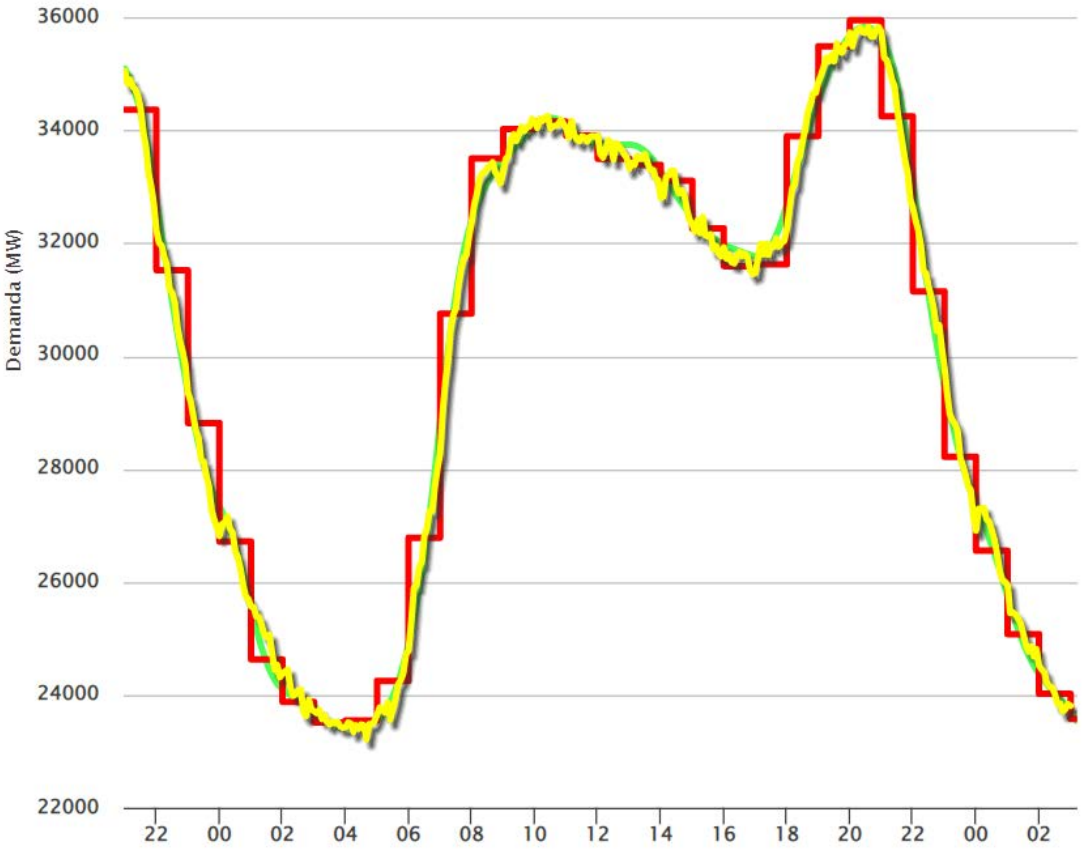


***Figure 5**: Electric demand 2022-01-13 (Wednesday)*

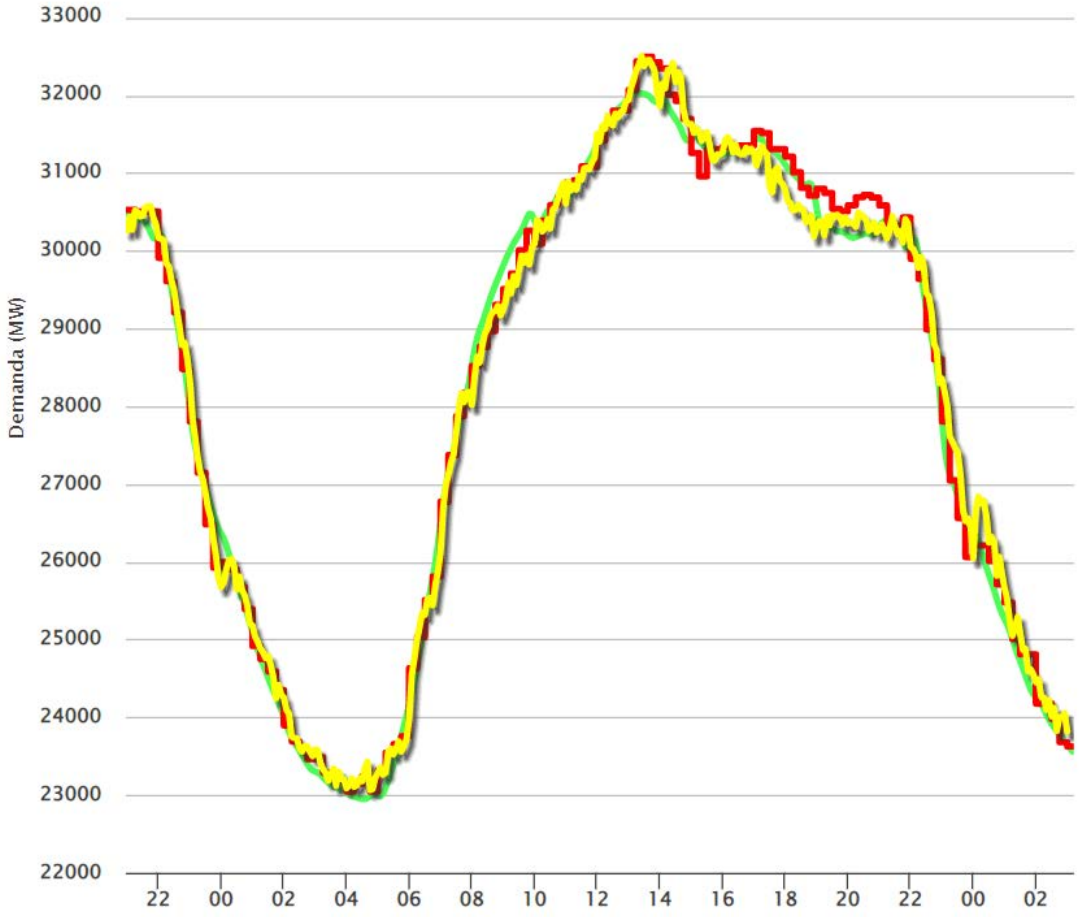


***Figure 6**: Electric demand 2022-6-09 (Wednesday)*

We observe a notorious change in between seasons with different temperatures, specifically in these days the mean temperature in Spain was 2.7ºC for January 13th (Figure 5) and 21.9ºC for June 9th (Figure 6). At first 21.9ºC may not seem like a very high temperature for a summer day, but we need to consider this is the mean temperature of all the Spanish Peninsula, considering both day and night and also colder regions. To have a better standpoint, the hottest day of the year 2022 has been August the 13rd with the highest mean temperature of 27ºC. All this temperature data has been taken from the National Centers for Environmental Information, on the NOAA (National Oceanic and Atmospheric Administration) subsection [4], later down the paper will be explained how the data has been treated to collect the important and interesting data.

To further emphasize the temperature factor consideration, the study from [5] has also been taken into account, getting from it the following figure, which shows the direct relation between energy consumption and temperature.

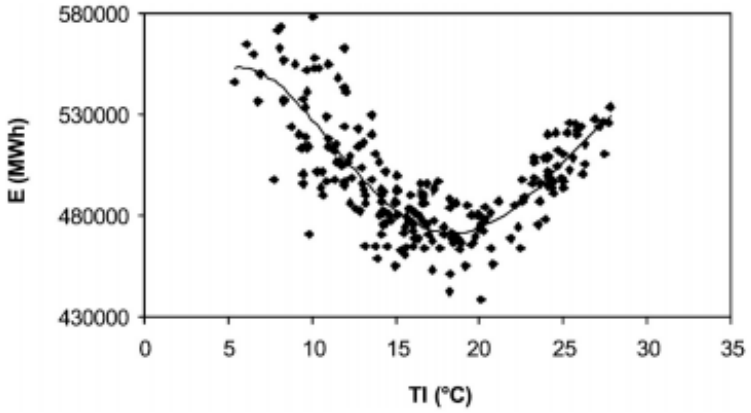


***Figure 7**: Relation between energy consumption and temperature in Spain*

1.2. Neural Networks

1.2.1. The neuron, biologic instrument

The neural part of our program model comes exactly from the biological meaning. The neuron is the cell capable of transmitting electrical impulses in our brain with the purpose to make us understand, develop, learn and function. there is an approximate number of 86 billion neurons in the brain on the average human body. In figure number 8 we can observe the basic structure of a neuron.

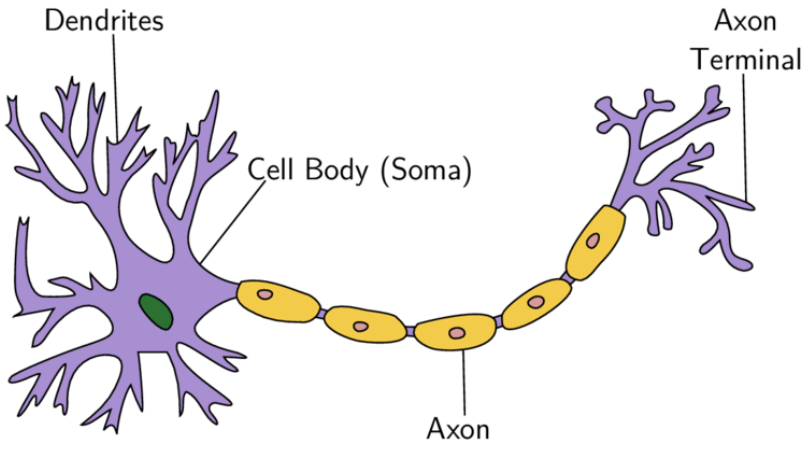


***Figure 8**: Biologic neuron representation*

The neuron is constituted of a main part called body or soma; in which interior the nucleus of the cell is found. From the soma, is born the axon, an elongated part of the cell that branches out similarly to a tree into hundreds of terminals, which help connect the neuron to the rest of the biological neural network. Closest to the body, the neuron has several inputs called dendrites, which are used to get signals from other neighbor neurons.

In a simplified way, the way a neuron works is getting electric signals from other neurons from the dendrites which can be activators or inhibitors, in a way that several impulses can be cancelled partial or totally. The reception of these impulses triggers the raise or lower the membranes voltage, which in doss mode is usually about -70mV. When such impulses accumulate a threshold voltage of about -55mV, the neuron activates sending an electric pulse of its own through the axon, this pulse is known as action voltage, and it is transmitted through the axon terminals to the following neurons from the network. Summarizing, we could say the neuron adds up the pulses received from the dendrites and if the total voltage reaches an established limit, the neuron activates sending a signal to other neurons.

This same principle is applied when creating an artificial neural network. In figure 9 an artificial neuron from a perceptron network has been illustrated. This specific type of networks were the first studied ones by Rosenblatt in 1958 [6] and are made from only two layers, and input layer that receives output data and an output layer that sends data to the exterior of the network (Figure 10).

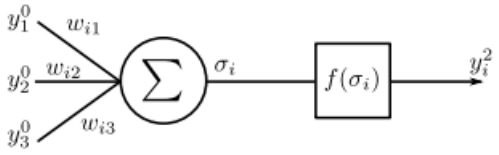


***Figure 9**: Artificial neuron*

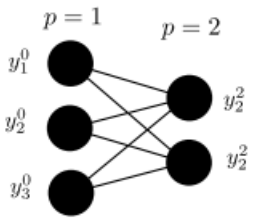


***Figure 10**: Two-layered perceptron type neural network*

The artificial neuron is divided into several parts, which are:

- **Inputs**, y_j^0: input values that the neuron *i* receives from neuron *j*, similarly to how dendrites work.
- **Weights**, ω_{ij}: represent the connection strength between neuron *i* and neuron *j*, indicating how important is the input.
- **Sum**, σ_i: a weighted sum considering the weights ω_{ij} from m input neurons (see eq. (2)).
- **Activation function**, $f(\sigma_i)$*:* in a perceptron type network the activation function is a classic step function, with θ as the threshold value. When σ is above this threshold value, the function outputs a 1, on the opposite case, it outputs a 0.
- **Outputs**: signal or signals outputted by the neuron, the equivalent to the axon biological output signal.

Following the previous explanation, it is clearly seen how the artificial neuron works, getting input values, adding them considering each of their weights and comparing it to a threshold value to decide if it produces an output or not.

The problem with this first method is that the output can only be a discrete value (0 or 1) and in our neural networks we want to be able to compute continuous values. This is where the sigmoid function comes into play as a way to change values between 0 and 1 continuously.

$$f(x) = \frac{1}{1 + e^{-x}} \quad (1)$$

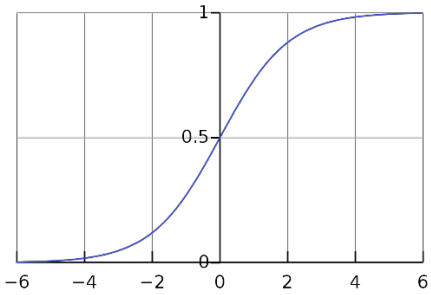


***Figure 11**: Sigmoid function graph*

The hyperbolic tangent function is sometimes also used, when it is convenient to change between the values -1 and 1 instead of and 1, this being useful when working with negative output values.

1.2.2. Multilayer Feedforward Networks

Taking a perceptron type network and expanding it creates a multilayer feedforward network. This network will have at least an input layer and an output layer, with whatever number of neurons each, depending on our specific problem. Apart from the input and output layer we may add how many additional layers we consider necessary, which will be part of an internal structure called hidden layer. In a feedforward network,

neurons of each layer can only be connected to those neurons from the previous layer or the next one, no connection between neurons of the same layer exist, as well as there are not any connections between a neuron and itself (self-recurrence). On the following figure we observe a multilayer feedforward neural network formed by *N* layers marked by the letter *p*. From $p = 2$ to $p = N - 1$ is what are considered the hidden layers.

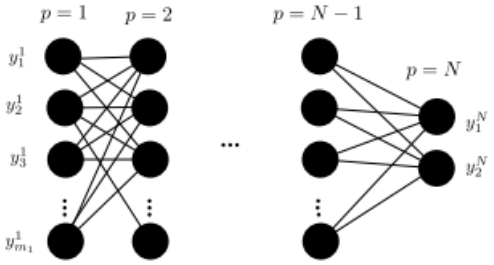


***Figure 12**: N layered feedforward neural network*

Following figure 12, we can determine any output of any neuron *i* from layer *p*. This will be the weighted sum of all previously connected neurons, expressed as:

$$\sigma_i^p = \sum_{j=1}^{m_p - 1} \omega_{ij}^p \, y_j^{p-1} \tag{2}$$

$$y_i^p = f(\sigma_i^p) \tag{3}$$

Successively applying this expression to each neuron on each layer, we propagate the inputs signal until the output layer, achieving what is called feedforward propagation.

Let’s apply these concepts to a simple three-layered feedforward neural network to further understand the behavior of the network.

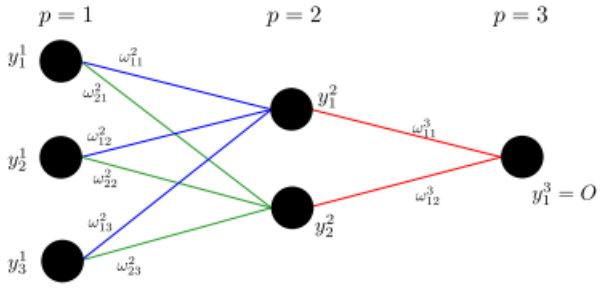


***Figure 13**: Three-layered feedforward network*

Applying the previous expression to neuron i = 1 from p = 2 we obtain the second layer's output:

$$\sigma_1^2 = \sum_{j=1}^{3} \omega_{1j}^2 y_j^1 \Rightarrow y_1^2 = f(\sigma_1^2) \tag{4}$$

$$\sigma_2^2 = \sum_{j=1}^{3} \omega_{2j}^2 y_j^1 \Rightarrow y_2^2 = f(\sigma_2^2) \tag{5}$$

Reapplying again the previous expression but with the newly calculated outputs we finally arrive the neural network's final output:

$$\sigma_1^3 = \sum_{j=1}^{2} \omega_{1j}^3 y_j^2 \;\Rightarrow\; y_1^3 = f\,(\sigma_1^3) \tag{6}$$

We have seen how signals propagate through the system, but without the correct weights, the output of the network will be completely random. To achieve the desired outputs, we need to find the adequate weights for each individual connection.

This could be done by hand if we had a very reduced number of neurons, but since that will almost never be the case, we need to figure out a way for the neural network to 'learn' and change its own weights slowly finding the appropriate ones.

1.2.3. Backpropagation algorithm

When we input a series of values in a neural network, we end up with a number of values as outputs, these can be expressed as an array $y^N = (y_1^N, y_2^N, \ldots, y_m^N)$. Suppose we know the desired output values beforehand, values we will call pattern and be expressed also as an array $Y = (Y_1, Y_2, \ldots, Y_m)$. With both these vectors we are able to compute the error between the current output and the desired one using the error function $E(\omega)$.

$$E(\omega) = \frac{1}{2}\sum_{i=1}^{m_N}(Y_i - y_i^N)^2 \quad (7)$$

Where w represents the whole group of network weights, which will be represented as a matrix. The main goal here is to find and use the weights with the minimum error, arriving at them with the gradient descent method.

If we represent the error function $E(w)$ as a surface (Figure 14) we find the minim value when traveling the surface in the direction of the descending gradient, finally finding the error function minim which will be used to correctly modify our weights.

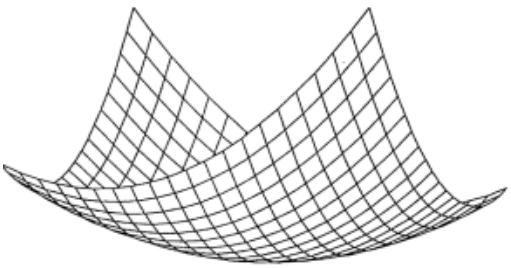


***Figure 14**: Surface given by representing the linear error function*

The exact amount the weights need to be shifted is given by the following expression:

$$\Delta\omega_{ij}^{p} \;=\; -\eta\,\frac{\partial E(\omega)}{\partial \omega_{ij}^{p}}$$

(8)

Where η is called *self-training rate*, a factor that determines how much we descend on the surface in each of the process's steps. For this technique we start with the weights connecting the output layer and the one immediately before, being those, layer N and layer $N-1$.

Applying the previous expression, we arrive at:

$$\Delta\omega_{ij}^{N} \;=\; -\eta\,\frac{\partial E(\omega)}{\partial \omega_{ij}^{N}} \;=\; -\eta\,\frac{\partial E(\omega)}{\partial \sigma_{i}^{N}}\frac{\partial \sigma_{i}^{N}}{\partial \omega_{ij}^{N}}$$

(9)

Which can be separate in two different parts for clarity purposes, ending up with this expression:

$$\frac{\partial \sigma_{i}^{N}}{\partial \omega_{ij}^{N}} \;=\; \frac{\partial}{\partial \omega_{ij}^{N}}\left(\sum_{k=1}^{m_{p-1}} \omega_{ik}^{N}\, y_{k}^{N-1}\right) \;=\; \sum_{k=1}^{m_{p-1}} \delta_{j}^{k}\, y_{k}^{N-1} \;=\; y_{j}^{N-1}$$

(10)

And this other one:

$$\frac{\partial E(\omega)}{\partial \sigma_{i}^{N}} = \frac{\partial E(\omega)}{\partial y_{j}^{N}}\frac{\partial y_{j}^{N}}{\partial \sigma_{i}^{N}} = \frac{\partial}{\partial y_{j}^{N}}\left[\frac{1}{2}\sum_{i=1}^{m_{N}}(Y_{i}-y_{i}^{N})^{2}\right]\frac{\partial}{\partial \sigma_{i}^{N}} f(\sigma_{i}^{N})$$

(11)

Combining the two previous equations we obtain:

$$\Delta\omega_{ij}^{N} = \eta(Y_{i}-y_{i}^{N}) f'(\sigma_{i}^{N}) y_{j}^{N-1} = \eta\delta_{i}^{N} y_{j}^{N-1}$$

(12)

From where we determine *N* layer's *delta* as:

$$\delta_i^N = (Y_i - y_i^N) f'(\sigma_i^N)$$

(13)

The activation function's derivative appears in the *delta* expression, and this is the reason a continuous and derivable function as the sigmoid or the hyperbolic tangent is needed. This last expression can be equally obtained for any desired layer of the network applying the exact same steps but for *p* instead of *N*. The only problem we find is that $E(\omega)$ is dependent on y_j^N, thus needing to apply the method in the correct order. Calculating *delta* for each neuron and layer depending on the next layers error ($p + 1$) until the *N* layer as follows:

$$\delta_i^N = f'(\sigma_i^N) \sum_{j=1}^{m_{p+1}} \delta_j^{p+1} \sum_{n=1}^{m_p} \omega_{jn}^{p+1} \delta_i^n = f'(\sigma_i^p) \sum_{j=1}^{m_{p+1}} \delta_j^{p+1} \omega_{ji}^{p+1}$$

(12)

Once we have found every $\Delta\omega_{ij}^p$, we will be able to shift all weights between neurons, lowering the output error. Repeating this process over an over we finally arrive to the error function's minimum, getting the desired output and having trained the network for future different inputs.

This is the whole *backpropagation* process summarized step by step:

1. We assign random weights and get the current outputs for those specific weights and the initial inputs.
2. All output layer neuron *deltas* are calculated
3. The previous layers *deltas* are calculated, going back from the $N - 1$ layer until arriving at $p = 1$
4. Weight shifting corrections are calculated for the whole network.
5. Weights are actualized by adding the just calculated weight shifting values.

6. Repeat process until the output error is the minim or de desired one (finding the exact minimum could take too many iterations, consuming time and resources).

2. Objectives

The main goal of this project is designing an electric demand short term forecast model using the technology explained in the previous introductory sections. Only the peninsular data will be considered. And as said before, the input factors will be type of day, season and temperature. Since the objective of this paper is not to predict temperature but electric demand, the temperature data that will be used to predict the demand will be taken from real data, while on a real-world application we would have to be given a short-term approximate weather forecast of the designated day which electric demand wants to be predicted.

To further experiment with different networks, two feedforward backpropagation-trained neural networks have been designed. Both with three layers: inputs, outputs and a hidden layer formed by a customizable number of neurons. The difference between the two networks will be the number of inputs that enter the network. In the first one, this number will be 4, on the other network, 27 inputs will be implemented. In the four-input network, the entrance factors will be day type (discerning between working days, Saturdays and, Sundays and festivities, which will be in the same group), month, season (Spring, Summer, Autumn or Winter), and mean temperature of the desired day. On the other hand, the second network will have the same first three inputs, that is day type, month and season, but 24 additional inputs referring to the mean temperature of each hour of the day (hopefully achieving more precision, as more factors are considered). Both networks have been represented in the following figures.

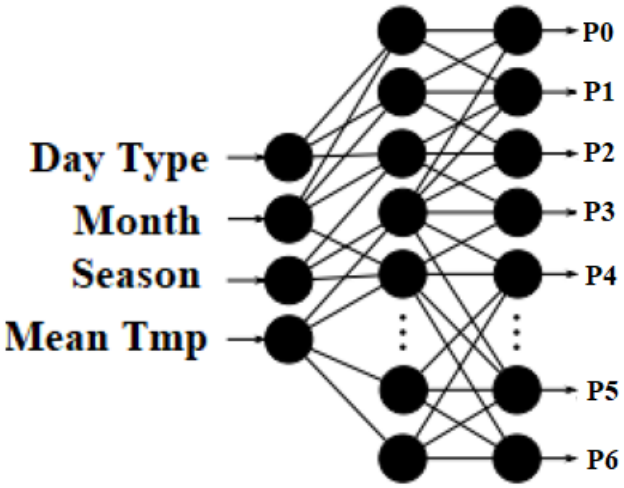


***Figure 15**: 4 Input neural network*

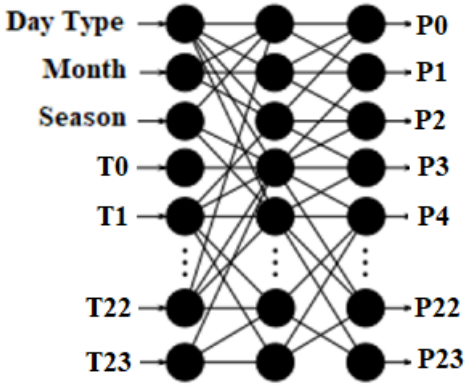


***Figure 16**: 27 Input neural network*

For both network's training we have used real data from the year 2021, both electric demand and temperatures. While for the prediction real temperature from 2022 has been used, as explained before since the main goal of the project is not to predict temperature, but electric demand. Also, demand data from 2022 will be used to check and validate our predictions. The energy datasets have been gathered from REE public information publisher, accessing by the API (application program interface) through the ESIOS python app [7] made by Santiago Peñate Vera, special thanks to him, who has been

very accessible and helpful with a specific problem that appeared along the way. Specifically, if anyone trying to run the program encounters an encoding error, the solution we found was to change the *client.py* source code from the *http* python internal library, from *latin-1* encoding to *utf-8* encoding.

After training the network and checking predictions, we have experimented with different types of configurations to examine the behavior of the network under different inputs, number of neurons in the hidden layer, number of iterations and learning factor η. To compare different results, several parameters have been considered, such as MAE (mean absolute error), relative errors, MAPE (mean absolute percentual error) and standard deviation.

3. Methodology

The process that has been implemented to do the project can be separated into three different phases. These being pre-network, network, and post-network or result understanding. For the pre-network phase, there are two main sub-parts, data gathering and treatment. In the network phase we separate between training the neural networks and making forecast calculations. Finally, the last phase is just analyzing the data using the statistic parameters mentioned earlier.

The methodology used in this project has been inspired by [8], [9] and [10], but optimizing running time from the python modules, actualizing modern usage of data analysis for current temperature and demand data and minimizing output error for post-pandemic predictions.

3.1. Data gathering and treatment

With the treated data gathered from 2021 we have trained the network, while with the data from 2022 we have checked and compared the forecast results.

In the gathering and treatment process we have ended up with three different data tables:

- **Annual Summary Table:** in this table, named ‘Table_2020’ and ‘Table_2021’ in the code we have a data summary of each day of the year, including the date (yyyy-mm-dd), the mean temperature, the type of day (0 for workdays, 5 for Saturdays and 0 for Sundays and festivities), and the season (1 for Spring, 2 for Summer, 3 for Autumn and 4 for Winter).

Date	Mean_Temperature	Day_Type	Season
2020-01-01	5,5	10	4
2020-01-02	5,7	0	4
2020-01-03	6,2	0	4
2020-01-04	6,4	5	4
2020-01-05	6,2	10	4
2020-01-06	5,5	10	4
2020-01-07	5,9	0	4
2020-01-08	7,1	0	4
…	…	…	…
2020-12-24	8,5	0	4
2020-12-25	6	10	4
2020-12-26	4,1	5	4
2020-12-27	4,5	10	4
2020-12-28	7,2	0	4
2020-12-29	5,6	0	4
2020-12-30	4,8	0	4
2020-12-31	5,3	0	4

***Table 2**: Annual summary table - 2020 (compressed)*

- **Annual Temperature Table:** in this table, named ‘Temperratures-2020’ and ‘Temperatures-2021’ in the code, we have the hourly mean temperature data for each day of the year. As mentioned before this data has been gathered from the National Oceanic and Atmospheric Administration (NOAA) [4] Filtering in the webpage by country and then selecting stations from up to 40 different pages of data for each of the 12 moths of the year we obtain individual cvs files with up to 120 000 rows of hourly temperature data. From these raw files and using the python Temperature data management code shown in the annex, we are able to clean the files and create new monthly cvs files with just the data we find useful.

This being the columns **Name** (‘Weather station’, SP), **Date** (‘yyyy-mm-ddTHH:MM:SS’), **Report_Type** (‘FM-15’ for 30-minute measurements or ‘FM-12’ for hourly ones), and **TMP** (t, dc) t being the temperature in degrees Celsius multiplied by 10 and dc being a factor that indicates if the value is correct with a 1.

hour	2020-01-01	2020-01-02	2020-01-03	2020-01-04	…	2020-12-28	2020-12-29	2020-12-30	2020-12-31
0	3.9	3.3	4.6	4.9	…	6.6	5.3	3.5	3.5
1	3.5	3.1	4.8	4.5	…	6.3	5.0	3.3	3.0
2	3.2	2.8	4.2	4.5	…	6.4	4.9	3.2	3.0
3	2.7	2.2	3.8	4.5	…	6.2	4.7	3.0	2.9
4	2.7	2.6	4.0	3.6	…	6.4	4.5	2.7	2.6
5	2.5	2.3	3.9	4.0	…	6.3	4.4	2.6	2.3
6	2.4	2.1	3.8	3.9	…	6.3	4.1	2.4	2.3
7	2.1	2.1	3.6	3.6	…	6.2	3.9	2.1	2.3
8	2.7	2.3	3.8	3.2	…	5.4	3.4	1.8	2.1
9	3.1	3.4	4.7	5.2	…	6.9	4.9	3.7	4.3
10	5.5	5.6	6.2	6.7	…	7.6	6.3	5.4	5.9
11	7.7	7.4	7.6	8.4	…	8.5	7.4	6.6	7.3
12	9.0	9.9	8.7	9.5	…	9.0	8.3	7.5	8.4
13	10.1	10.2	9.7	10.3	…	9.1	8.5	8.0	9.0
14	10.6	10.9	10.1	10.9	…	9.6	8.5	8.5	9.1
15	11.3	11.1	10.2	10.9	…	9.8	8.6	8.7	9.1
16	9.9	10.4	9.5	10.3	…	8.9	7.6	8.0	8.1
17	8.8	8.8	8.5	8.9	…	8.1	6.6	6.9	7.2
18	7.1	7.5	7.5	7.7	…	7.4	5.8	5.9	6.6
19	5.9	6.5	6.8	6.8	…	7.1	5.2	5.3	6.1
20	5.1	5.9	5.9	6.2	…	6.5	4.7	4.7	5.8
21	4.3	5.4	6.0	5.7	…	6.4	4.4	4.5	5.5
22	4.2	5.1	5.4	5.4	…	6.0	4.0	4.0	5.3
23	3.9	4.8	5.0	5.0	…	5.7	3.7	3.7	5.0

***Table 3**: Annual Temperature Table - 2020 (compressed)*

- **Annual Electric Demand Table:** in this table, named 'Production-2020' and 'Production-2021' in the code, we have the same data structure as in the Annual Temperature table, but exchanging mean temperatures with electric demand. As said before, this data has been downloaded using the ESIOS online app provided by Santiago Peñate Vera on his GitHub account. It has also been treated via python coding until arriving to table 4. Pruning the original table was necessary because from the ESIOS app we would get the demand values for every 10-minute period.

hour	2020-01-01	2020-01-02	2020-01-03	2020-01-04	…	2020-12-28	2020-12-29	2020-12-30	2020-12-31
00:00:00+00:00	23039	23441	27094	27483	…	25499	26341	26524	27143
01:00:00+00:00	22695	21245	24668	25378	…	22989	23856	23990	24115
02:00:00+00:00	21567	19731	22988	23362	…	21325	22065	22144	22024
03:00:00+00:00	20284	18892	22070	22072	…	20662	21115	21276	20961
04:00:00+00:00	19265	18726	21331	21903	…	20091	20863	21082	20404
05:00:00+00:00	18742	18986	21737	21757	…	20440	21135	21221	20510
06:00:00+00:00	18679	20434	22843	21752	…	21449	21959	22241	21238
07:00:00+00:00	19095	23671	25736	22904	…	24254	24593	24840	23081
08:00:00+00:00	19444	27406	29084	24571	…	27256	27796	28149	25434
09:00:00+00:00	19281	30101	31625	26506	…	29944	30481	30624	27577
10:00:00+00:00	20831	32677	33909	29051	…	32337	32600	32622	30271
11:00:00+00:00	22587	33477	34717	30281	…	33177	33325	33372	31178
12:00:00+00:00	23562	33704	34573	30198	…	33412	33033	33040	30996
13:00:00+00:00	24049	33427	34045	30002	…	33471	33032	32710	30231
14:00:00+00:00	24531	33401	33972	30024	…	33814	33333	32860	30057
15:00:00+00:00	23711	32188	32557	29429	…	32157	32135	31507	29147
16:00:00+00:00	22593	31337	31869	28382	…	31267	31402	30978	28583
17:00:00+00:00	22544	31241	31190	27866	…	31140	31075	30647	28400
18:00:00+00:00	23983	32191	32281	28641	…	32416	32279	32017	29939
19:00:00+00:00	26187	33589	33305	30489	…	33594	33606	33605	31994
20:00:00+00:00	27061	3373	33647	30968	…	33887	33936	34041	31574
21:00:00+00:00	27870	34132	33565	31507	…	34012	34172	34134	30863
22:00:00+00:00	27458	32507	32142	30733	…	32031	32572	32730	28528
23:00:00+00:00	26039	30313	30127	29102	…	29500	29814	30177	26538

Table 4: *Annual demand table - 2020 (compressed)*

3.2. Neural network operation

As mentioned before, both networks that we will be training and using are built with three neuron layers: input layer, hidden layer and output layer. The hidden layer is formed by an arbitrary *N* number of neurons. This, together with the number of iterations (that is the number of times the learning process is repeated until finding adequate weights) and the learning factor η are the parameters we, as builders of the network have control over. The configuration array is composed by these customizable parameters, in that order. For example, if we were to choose a configuration with 25 hidden layer neurons, 2000 iterations and a learning factor of 0.1, our configuration array would look like this: *conf = [25, 2000, 0.1]*.

There are two main functions in the neural network script. *Training(conf, typ)* and *ForecastCalculation(date_str, conf, typ).* The parameter conf is the configuration array we just have talked bout, while *typ* would indicate the type of network we wish to train or use for the calculations, the 4-input one (*typ* = 1) or the 27-input one (*typ* = 0). Once the training is complete, the achieved weights matrices are saved in a text document that can be used afterwards in case we want to use once again a past configuration without the need to train again the network.

Let's settle an example of what lines of code we would need to input through the python console or simply write in an alternative script in which we have previously imported the neural network script, again, shown in the annex.

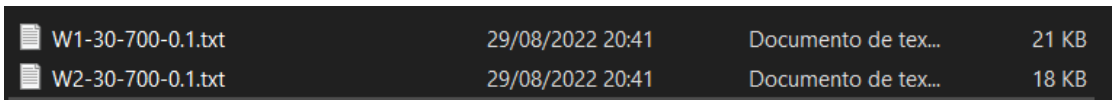
```
from neural_network import *

conf1 = [30, 700, 0.1]

inputs_27 = 0
inputs_4 = 1

Training(conf1, inputs_27)
```

In this particular example, we are using a secondary script to facilitate the writing and reading of the different instructions. First, we import the neural network python file with all the necessary code, we then set up a configuration array with the values shown: 30 number of neurons in the hidden layer, 7000 maximum iterations and a learning factor of 0.1. In addition, we pass the input number 0 to determine that we will be using the 27-input neural network. We will later experiment with different parameters to compare results.

After running this script, the console output shows:

```
##########################################
Training network with configuration: [30, 700, 0.1]
Network Imputs: 27
Training completed
Execution time: 31.78s
##########################################
[Finished in 38.126s]
```

This meaning the network has successfully been trained. Now we can check our project folder to find two different text files containing the weight matrices that our program has achieved with this particular configuration.



Now, in order to make a prediction for any particular day in this case of the year 2021, we just need to call the function *ForecastCalculation()* as follows:

```
10  ForecastCalculation('2021-02-21', conf1, inputs_27)
```

Just making sure the configuration and network type are the same ones as we introduced in the training function. Otherwise, the *ForecastCalculation()* function will train a new network with the newly introduced configuration. Running this new command will output the following:

```
##########################################
Inputs to network: 27
Forecast Date: 2021-02-21
Day: 10.0
Season: 4.0
Neurons in hidden layer: 30
Iterations: 700
Learning factor: 0.1
##########################################
Mean absolute error (MAE): 1185 MW
Maximum absolute error: 3114.0 MW
Minimum absolute error: 20.0 MW
Standard deviation: 1137.19MW
Mean absolute percentual error (MAPE): 4.36 %
```

Showing us the specific data from the day we have chosen as well as some statistic parameters we have chosen to value the precision of our forecast. In this case, we obtain a Mean Absolute Percentual Error (MAPE) of 4.36%, which is under the 5% error that was mentioned in the introduction. The program will also plot three graphs, to help visualizing the prediction:

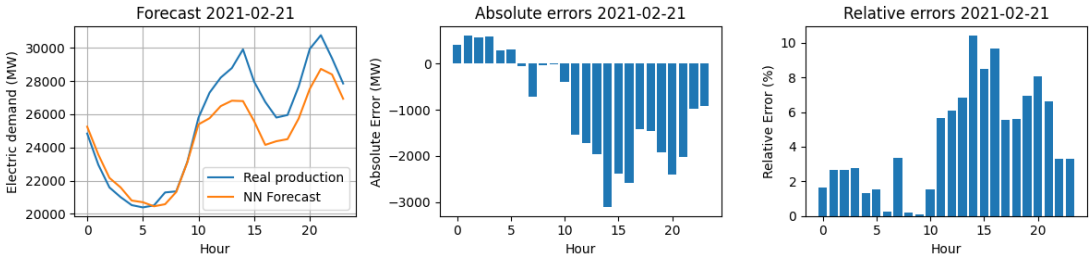


***Figure 17**: Electric demand prediction 2021-02-21 (27-input network)*

In addition, we also get a cvs file with a detailed table with the prediction data for the specified day.

Importing them to excel is a great way to clearly see the data:

Date	Power (GW)
21/02/2021 0:00	252.510
21/02/2021 1:00	235.740
21/02/2021 2:00	221.670
21/02/2021 3:00	215.970
21/02/2021 4:00	208.040
21/02/2021 5:00	207.030
21/02/2021 6:00	204.580
21/02/2021 7:00	205.830
21/02/2021 8:00	213.190
21/02/2021 9:00	231.130
21/02/2021 10:00	253.910
21/02/2021 11:00	257.620
21/02/2021 12:00	264.900
21/02/2021 13:00	268.150
21/02/2021 14:00	267.920
21/02/2021 15:00	255.820
21/02/2021 16:00	241.560
21/02/2021 17:00	243.760
21/02/2021 18:00	245.010
21/02/2021 19:00	257.370
21/02/2021 20:00	275.270
21/02/2021 21:00	287.300
21/02/2021 22:00	283.960
21/02/2021 23:00	269.320

Table 5: Electric demand prediction hourly data 2021-02-21 (27-input network)

3.3. Results analysis indicators

To evaluate the operation of the network comparing real demand data from 2021 and the predicted one, we will be using various indicators. When comparing hourly data, the absolute error will be used, which is the direct difference between the real electric demand and the value predicted by the neural network:

$$\varepsilon_h = P_{V,h} - P_{NN,h} \tag{13}$$

The relative hourly error will also be used, it is defined as:

$$\varepsilon_{r,h} = \left|\frac{\varepsilon_h}{P_V}\right| \cdot 100\% \tag{14}$$

To evaluate the general daily resemblance between the prediction and reality, the parameter that we will be using will be the Mean Absolute Error (MAE):

$$E_h = \frac{1}{24}\sum_{h=0}^{23}|\varepsilon_h| \tag{15}$$

Very useful in demand prediction studies is also the Mean Absolute Percentual Error (MAPE), making the previous indicator easier to understand and compare:

$$E_r = \frac{1}{24}\sum_{h=0}^{23}\frac{|\varepsilon_h|}{P_V} \cdot 100\% \tag{16}$$

Finally, we will also be using the absolute error's standard deviation indicator:

$$\sigma = \sqrt{\frac{1}{24}\sum_{h=0}^{23}(\varepsilon_h - E_h)^2} \tag{17}$$

4. Results analysis

We will be checking and comparing predictions made by both neural networks for several days of the year 2021 (remember that the network has been trained using data from 2020). Since it would be almost impossible and very time and space-consuming showing predictions for every day of the year, we will be choosing a specific list of days that we believe help represent almost every situation (different seasons, different types of day, different months…). For workdays, we have chosen several days, always close to the 15th day of the month and Wednesdays, to ensure the minimum influence from neighbor weekends, same goes for the Saturdays and Sundays chosen, close to the center mark of the month and as far aways as possible from festivity days. The list of days which predictions have been calculated are the following:

- Workdays: 2021-01-13, 2021-03-17, 2021-08-18, 2021-10-20
- Saturdays: 2021-07-17, 2021-12-18
- Sundays: 2021-02-14, 2021-05-16
- Festivities: 2021-11-01

4.1. 4-Input Neural Network results

We will initially use a configuration with 20 neurons in the hidden layer, 1500 iterations and a learning factor η of 0.1. We will later vary these parameters to observe how the network reacts to such changes.

In the following table we can appreciate all the errors from the days list predictions:

Date	Day Type	MAE (MW)	Eh,max (MW)	Eh,min (MW)	σ (MW)	MAPE (%)
2021-01-13	Workday	2188	3073	1169	563.6	6.54
2021-03-17	Workday	448	1220	51	392.4	1.57
2021-08-18	Workday	1105	2044	87	1128.2	3.89
2021-10-20	Workday	535	1076	74	626.2	1.99
2021-07-17	Saturday	799	1462	50	688.3	2.94
2021-12-18	Saturday	1118	2097	235	1266.2	4.17
2021-02-14	Sunday	326	880	14	414.3	1.33
2021-05-16	Sunday	1471	2374	538	486	6.5
2021-11-01	Festivity	813	1503	176	419.9	3.55

***Table 6**: 4-Input neural network error list for the chosen day group*

The prediction graphs can be seen as follows, divided into workdays and non-working days.

Workdays:

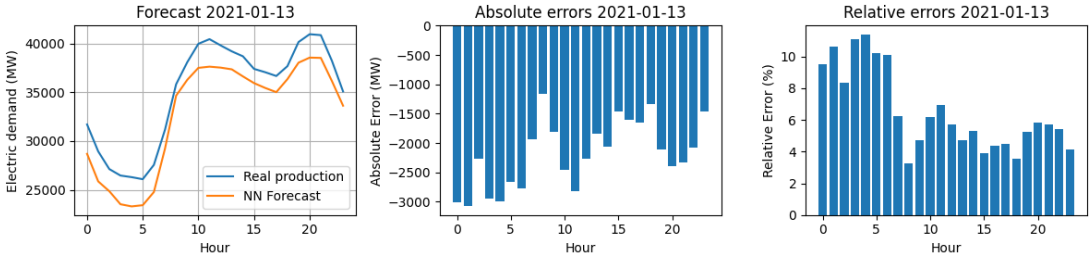


***Figure 18**: 4-Input network prediction for 2021-01-13*

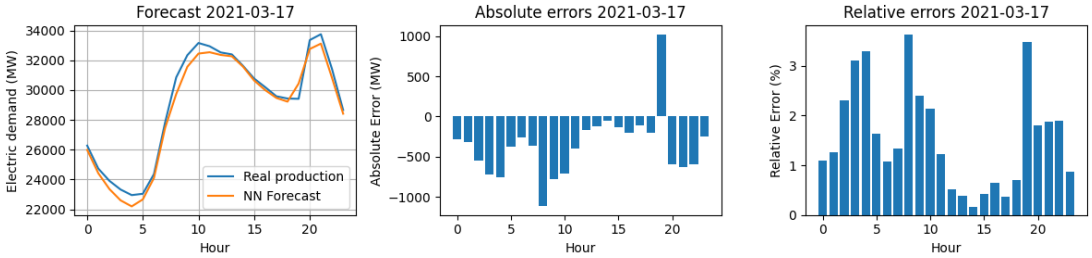


***Figure 19**: 4-Input network prediction for 2021-03-17*

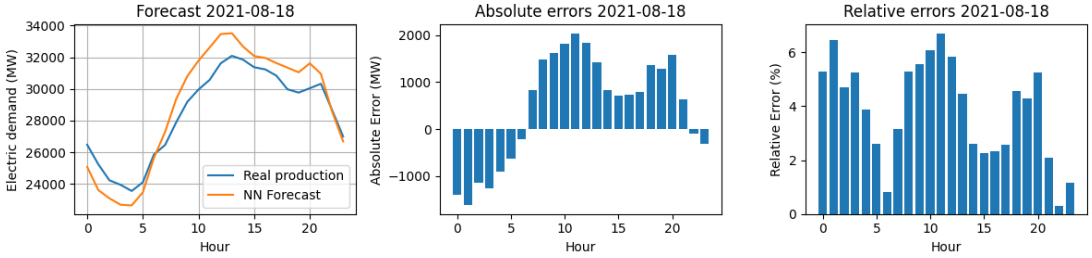


***Figure 20**: 4-Input network prediction for 2021-08-18*

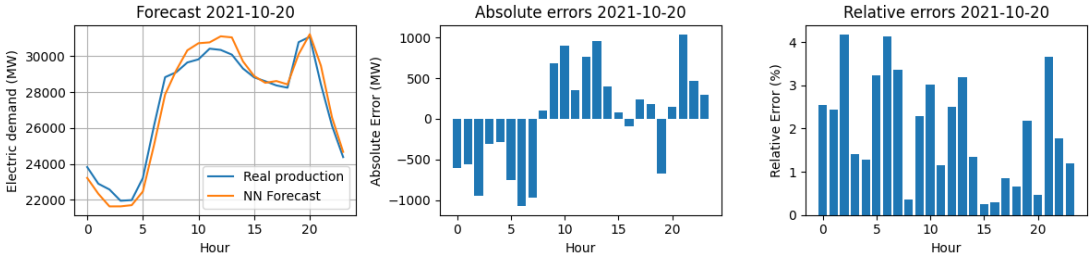


***Figure 21**: 4-Input network prediction for 2021-10-20*

Non-working days:

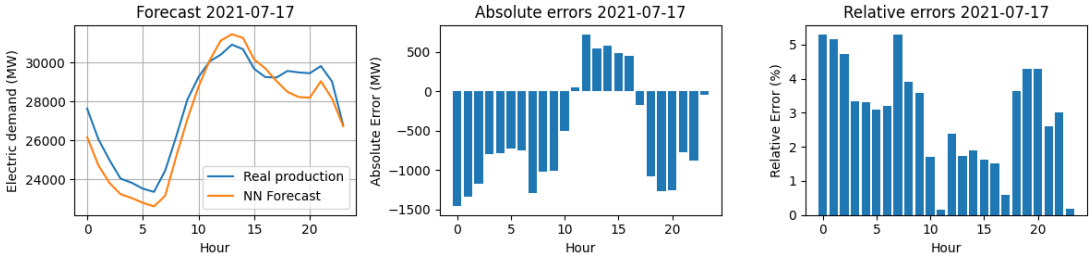


***Figure 22**: 4-Input network prediction for 2021-07-17*

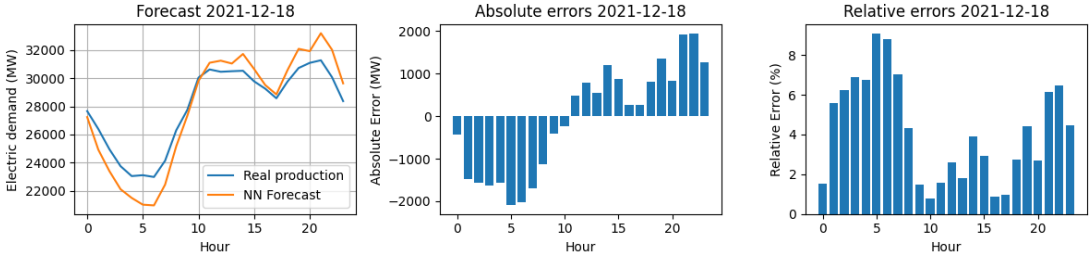


***Figure 23**: 4-Input network prediction for 2021-12-18*

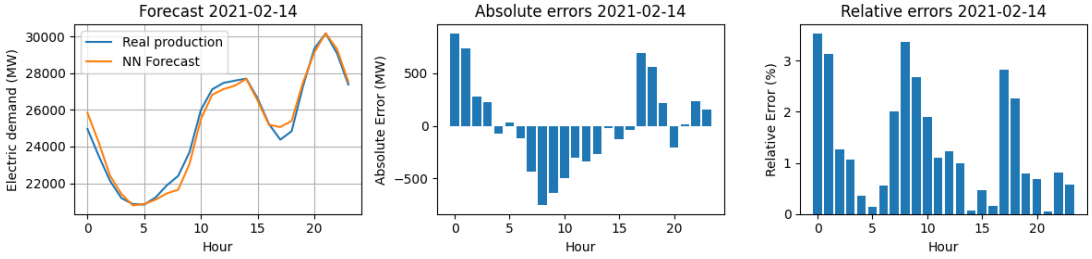


***Figure 24**: 4-Input network prediction for 2021-02-14*

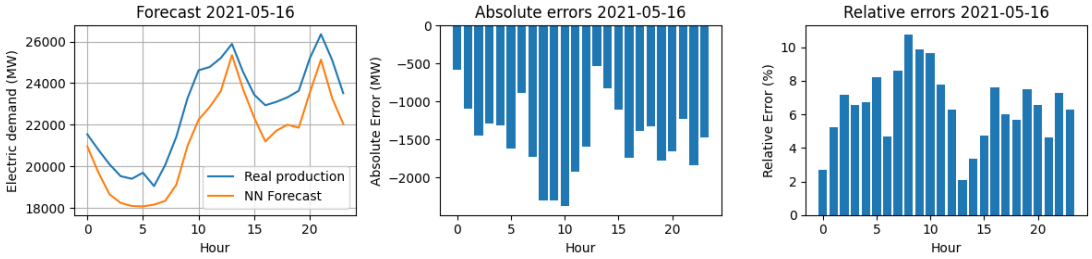


***Figure 25**: 4-Input network prediction for 2021-05-16*

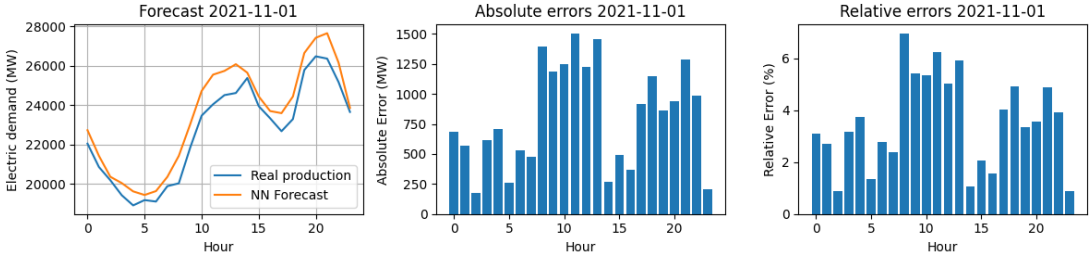


***Figure 26**: 4-Input network prediction for 2021-11-01*

Most predictions are very accurate, with a mean absolute percentual error between 1 to 6 % Now we will proceed to change some configuration parameters to see if we can obtain even better predictions for both days with MAPE superior to 6%.

After numerous changes and variations, we have arrived at the optimal error values the neural network can accomplish for these two days. In both cases the optimal number of iterations has been 2000, and the optimal number of neurons has resulted to be the one we already had, 20.

The main difference has been given by the training factor, which for the date 2021-01-13 has been $\eta = 0.3$ and for 2021-05-16, as been $\eta = 0.2$.

Optimal calculations of both days are as follow:

2021-01-13

```
##########################################
Inputs to network: 4
Forecast Date: 2021-01-13
Day: 0.0
Season: 4.0
Neurons in hidden layer: 20
Iterations: 2000
Learning factor: 0.3
##########################################
Mean absolute error (MAE): 2106 MW
Maximum absolute error: 3148.0 MW
Minimum absolute error: 664.0 MW
Standard deviation: 680.84MW
Mean absolute percentual error (MAPE): 6.24 %
```

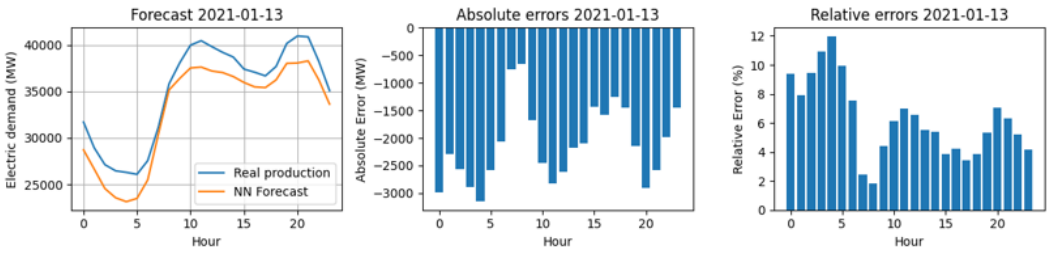


***Figure 27**: Optimal forecast for date 2021-01-13 on the 4-input neural network*

2021-05-16

```
##########################################
Inputs to network: 4
Forecast Date: 2021-05-16
Day: 10.0
Season: 1.0
Neurons in hidden layer: 20
Iterations: 2000
Learning factor: 0.2
##########################################
Mean absolute error (MAE): 1330 MW
Maximum absolute error: 2254.0 MW
Minimum absolute error: 354.0 MW
Standard deviation: 506.53MW
Mean absolute percentual error (MAPE): 5.92 %
```

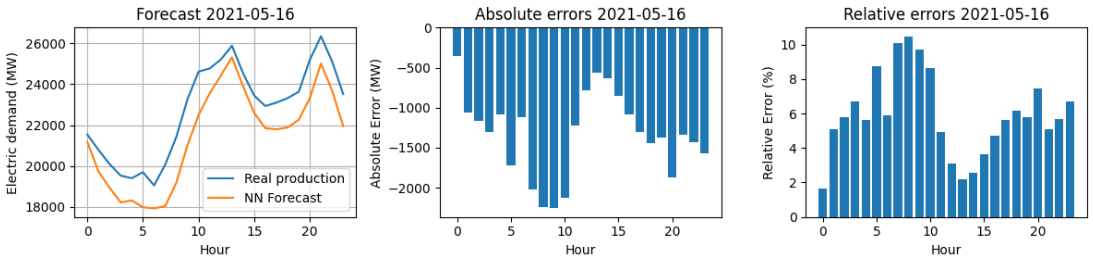


***Figure 28**: Optimal forecast for date 2021-05-16 on the 4-input neural network*

We observe a slight upgrade compared to the previous results, in the last case being able to lower the MAPE below 6%. All other possible combinations of number of neurons, number of iterations or learning factor, lead us to slightly worse results, up to an error of 8% for some extreme simulations. This goes to show that a higher number of neurons or learning factor does not always mean a better performance.

4.2. 27-Input Neural Network results

Now we proceed to test the second neural network with the same initial configuration and days list from the previous section, to be able to directly compare results and see if a 27-input network makes better predictions compared to the 4-input one.

Again, for a 20-neuron hidden layer, with 1500 iterations and a learning factor of 0.1, the 27-input network error results are:

Date	Day Type	MAE (MW)	Eh,max (MW)	Eh,min (MW)	σ (MW)	MAPE (%)
2021-01-13	Workday	3080	3779	2296	396.9	8,93
2021-03-17	Workday	532	15	13	589,5	1,84
2021-08-18	Workday	1445	2734	73	1279	4,94
2021-10-20	Workday	825	1581	71	624,5	2,96
2021-07-17	Saturday	546	1054	2	620,7	1,93
2021-12-18	Saturday	737	1874	38	855,7	2,81
2021-02-14	Sunday	499	1444	1	480,8	2,04
2021-05-16	Sunday	1097	2281	65	513.3	4,79
2021-11-01	Festivity	665	1369	66	635,2	3,01

These are the prediction graphs associated with the training and predictions just mentioned, divided into workdays and non-working days.

Workdays:

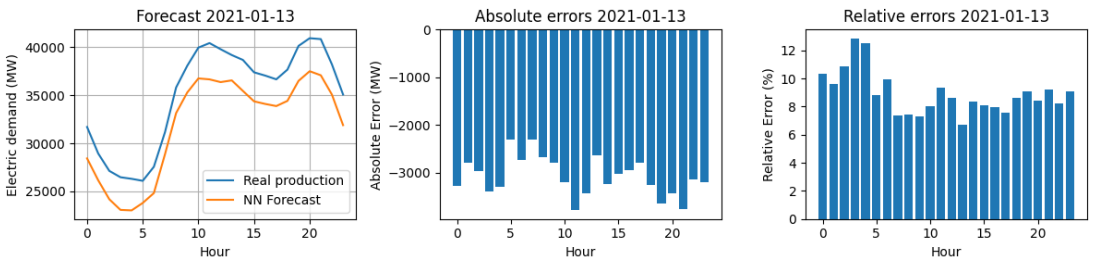


***Figure 29**: 27-Input network prediction for 2021-01-13*

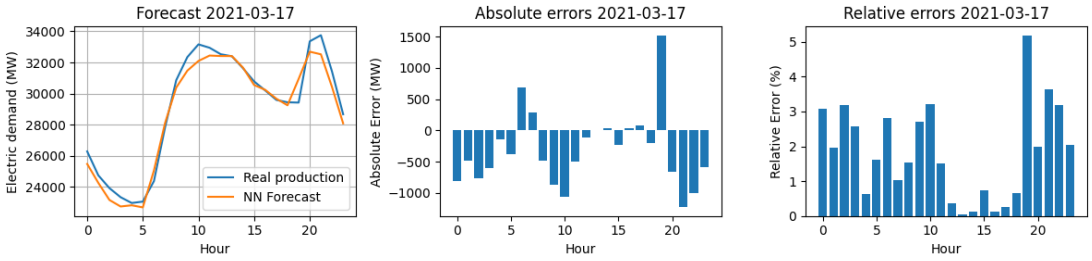


***Figure 30**: 27-Input network prediction for 2021-03-17*

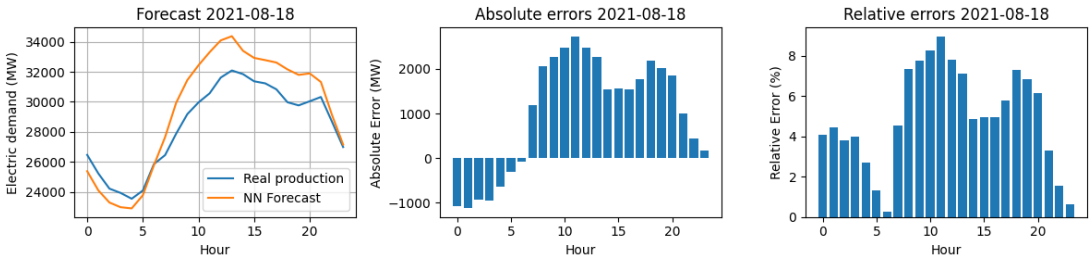


***Figure 31**: 27-Input network prediction for 2021-08-18*

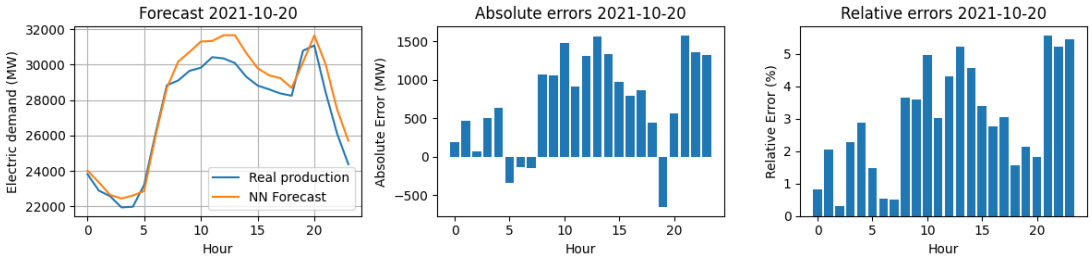


***Figure 32**: 27-Input network prediction for 2021-10-20*

Non-working days:

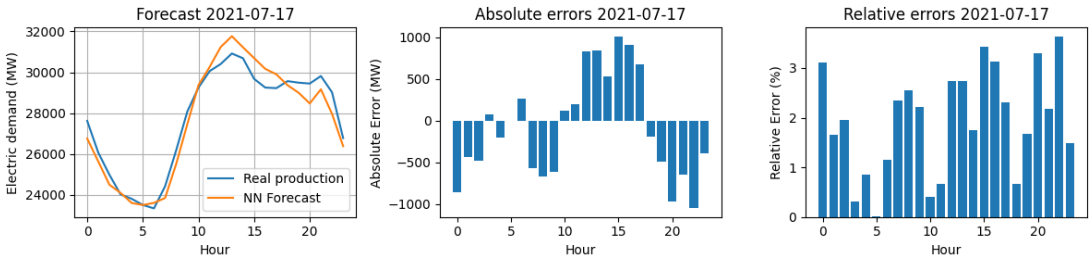


***Figure 33**: 27-Input network prediction for 2021-07-17*

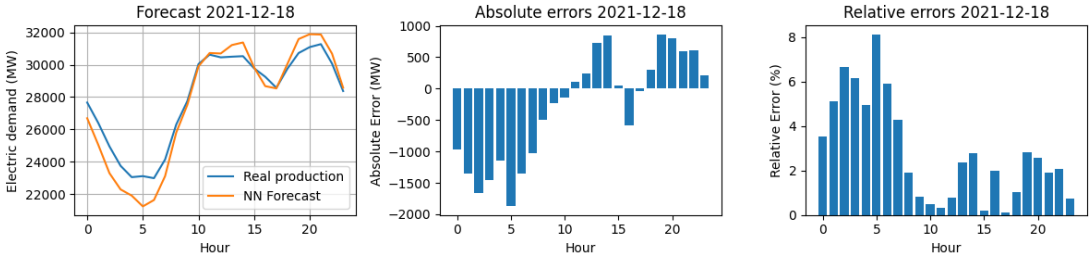


***Figure 34**: 27-Input network prediction for 2021-12-18*

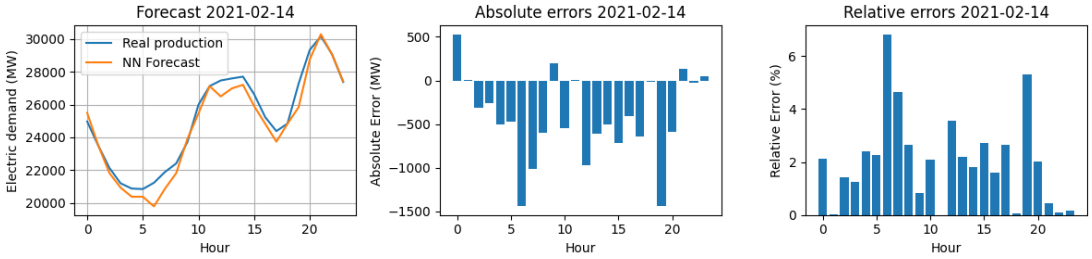


***Figure 35**: 27-Input network prediction for 2021-02-14*

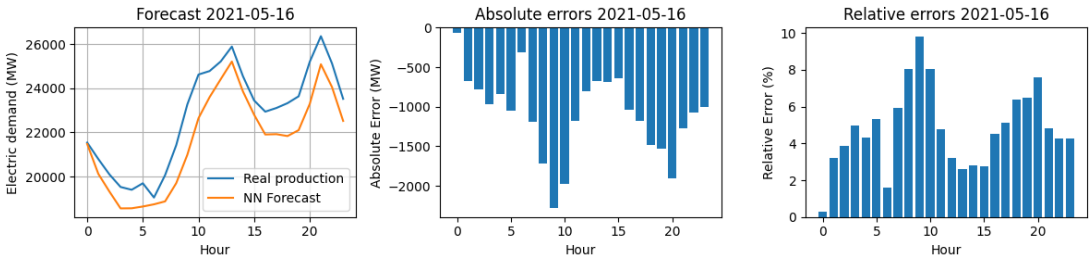


***Figure 36**: 27-Input network prediction for 2021-05-16*

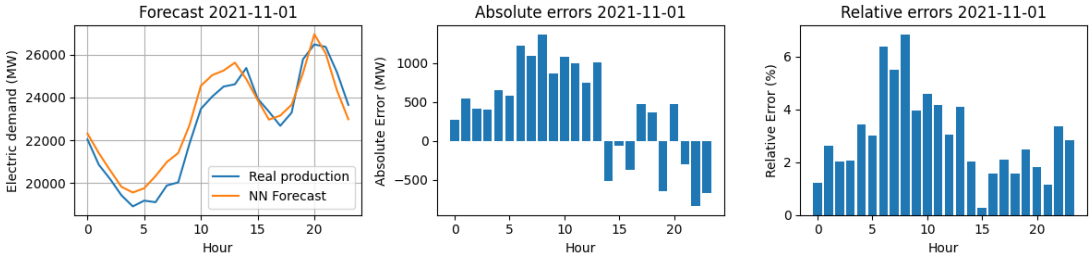


***Figure 37**: 27-Input network prediction for 2021-11-01*

Something interesting happens when comparing these new results with the previous 4-input network ones, we generally get better results, almost all of them are below the 5% MAPE threshold, with one exception, January the 13th. In some other days, as October 21st even though we are way below 5% MAPE, we are getting a slightly worse prediction when comparing mean absolute percentual errors. This is due to a higher variation between hourly temperature on comparison with a more stable mean temperature. Even then, we can confirm, that as expected, in general the 27-input neural network operates under a lower error threshold, thus being more competent.

Now, same as in the previous section, we will modify the initial configuration to find the optimal parameter values for the worst prediction of the group, in this case, January the 13th.

Iterating through a wide range of different combination of configuration parameters, we have arrived at the conclusion that the best configuration to get the minimum error possible on January the 13th 2021, according to our 27-input neural network is N = 10 (hidden layer neurons), 1000 iterations and a learning factor of 0.1 getting the following results.

```
########################################
Inputs to network: 27
Forecast Date: 2021-01-13
Day: 0.0
Season: 4.0
Neurons in hidden layer: 10
Iterations: 1000
Learning factor: 0.1
########################################
Mean absolute error (MAE): 3048 MW
Maximum absolute error: 4338.0 MW
Minimum absolute error: 2156.0 MW
Standard deviation: 599.83MW
Mean absolute percentual error (MAPE): 8.74 %
```

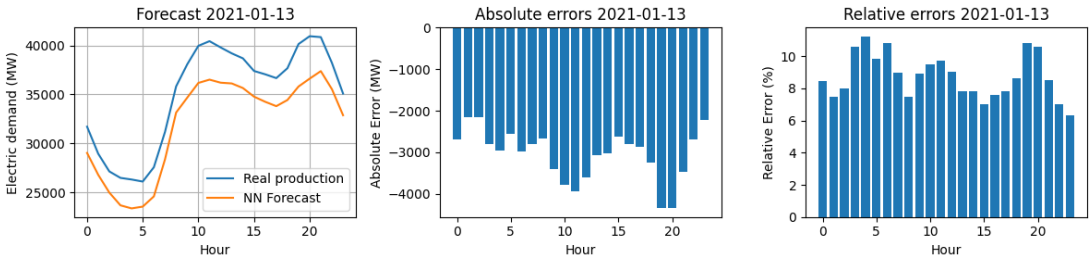


***Figure 38**: Optimal forecast for date 2021-01-13 on the 27-input neural network*

This result is still over the 5% MAPE, and a worse prediction than the 4-input neural network. This is due to the high volatility of hourly temperatures on this exact date. It is also interesting noting that in this case, the best result has appeared after converging 27 inputs into 10 neurons, which at first may seem contradictory, but if there is an overload of input variations, condensing them into a lower number of nodes has appeared to be the best choice.

On both neural network results, we haven't noticed a specific generally known distribution for the error functions. What we have noticed is that error peaks always match up the demand peaks of real data, this gives us a hint that an additional possible input tracking other factors may have helped both networks learn more about demand peaks. We discuss later what these possible new input additions could have been.

5. General discussion

Firstly, note that except for a couple of cases, the neural network has been able to accomplish very detailed and precise electric demand predictions, adjusting accordingly depending on the type of day, season, temperature and date. In general, from results we are able to determine that we have gotten better results from the 27-input neural network with overall better MAPE values, which is what should be expected as more input values are considered. We have also observed how different network configurations affect the overall output when iterating trying to find the optimal value for the specific dates that were over the error margin. Not always maximizing parameters has been useful, as like seen in previous examples, the optimal number of iterations and/or hidden layer neurons is sometimes on the lower scale. A pattern has emerged when looking through different configurations, as it seemed the higher iterations benefited colder weather liked results. On the same page, a higher learning factor value has often not help with the error management, as on most cases a learning factor of 0.1 to 0.3 has been the best option. Centering our focus now on the number of neurons in the hidden layer, we have discovered that usually, the sweet spot stands around 10 to 25 neurons, usually at $N = 20$.

The execution time is an interesting factor to take into account. As the number of iterations have gone up, the time to run the program has been substantially increased. Not presenting a problem in this particular project since the number of layers, neurons and iterations have not been very large compared to a hypothetical industrial program, with these values on a much higher scale, which in that case time optimization could be consider a valuable asset.

Lastly, we believe the years of study, both the network training year (2020) and the predicted year (2021) have been fairly irregular to one another and to the previous years due to the global COVID-19 pandemic and its strong impact on both the economy and industries. Quarantine regulations have highly affected the electricity market, affecting both demand and prices.

Regulations have very quickly changed between both years, so it is understandable that not having considered them as an input to the network (would have been a total overcomplication of the issue), has sacrificed predictions accuracy as they have not been as accurate as they would have been considering this political and infrastructure considerations.

5.1. Improvement proposals

A few ideas come into mind when thinking on how to improve the network. In this study we have not considered the slight difference between each individual working day, as Mondays or Fridays tend to have a clear influence from weekends electrical demand. We have also worked with the mean temperature of the whole peninsula, while maybe treating the landscape as different regions depending on their own temperature as different network inputs may have affected positively the output of the experiment. Finally, as mentioned before, using COVID-19 local regulations as inputs would have definitely improved the forecast calculations, although we believe this would have been an arduous challenge.

6. Conclusion

The main goal of the study was to learn about both electricity system and neural networks, getting to create our own neural network capable of predicting electric demand on a short-term horizon. It has been proved that making this network was completely plausible even with a fairly, at first glance, simple structure of three layers, obtaining great results considering the number of factors computed. We have learnt that the network configuration is key to obtaining valuable predictions, and that changing each individual parameter of the configuration leads to widely different results depending on the moment in time the prediction is made, considering season, month, temperature and type of day.

We have observed that were predictions are most distant from real values is in electric demand peaks, learning different lessons on how every condition affects the final output and keeping them in mind for future projects or studies.

Finally, we have compared and analyzed results leading to ways of improving the already precise methodology by considering new altering factors or different ways of treating and using the inputs we have already used.

BIBLIOGRAPHY

[1] Red Eléctrica Española, https://www.ree.es/es

[2] Kanaweera, D. K., Karady, G. G., Farmer, R. G., "Economic impact analysis of load forecasting", IEEE Trans. PWRS, vol 12 núm. 3 (1997), 1388-1392.

[3] Red Eléctrica Española – Seguimiento de la demanda eléctrica, https://demanda.ree.es/visiona/seleccionar-fecha

[4] National Oceanic and Atmospheric Administration (NOAA), Dataset global-hourly download, https://www.ncei.noaa.gov/access/search/data-search/global-hourly?pageNum=22&startDate=2021-06-01T00:00:00&endDate=2021-06-30T23:59:59&dataTypes=TMP&bbox=44.151,-13.306,34.549,4.740

[5] Pardo, A., Meneu, V., Valor, E. "Temperature and seasonality influences on Spanish electricity load. Energy Economics", Energy Economics, 24 (1), pp. 55-70.

[6] Rosenblatt, F., "The perceptron: A probabilistic model for information storage and organization in the brain", Psychological Review 56, 386-408, 1958.

[7] Santiago Peñate Vera, ESIOS Python data download API program, https://github.com/SanPen/ESIOS

[8] Carlos Mallo González, "Predicción de la demanda eléctrica horaria mediante redes neuronales artificiales", Universidad de Oviedo, 2004

[9] Abraham Rubio Ortega, "Previsión de la demanda eléctrica peninsular mediante redes neuronales", Universidad Nacional de Educación a Distancia, 2018

[10] David Díaz Vico, José Dorronsoro, "Deep Neural Networks for Wind and Solar Energy Prediction", UAM, 2022

ANNEX – CODE

Demand data treatment code

```
import pandas as pd
import numpy as np
from datetime import timedelta
from ESIOS import *
import urllib.request

"""
      Production_Summary_Table()
      Function: generates a cvs file with hourly production data from all years days
      Inputs: REE data from the ldpREEAnnual() function
      Outputs: cvs summary table and dataframe with all production data
"""


def Production_Summary_Table(year):

    df = pd.read_csv('D:/ICAI/Cuarto/TFG/Data/Annual-Production-'+str(year)+'.csv',
names=['f', 'production'], skiprows=1)
    df = df.join(df['f'].str.split(' ', expand=True).rename(columns={0: 'date', 1:
'hour'}))
    df = df.drop(columns=['f'])
    df = df[['date', 'hour', 'production']]

    days_list = df['date'].unique()
    hours_list = df['hour'].unique()
    df.set_index(['date'], inplace=True)

    Summary_Table = pd.DataFrame(index=hours_list[::6], columns=days_list)  # [::6] is
needed because from ESIOS we receive data every 10 minutes

    for n in days_list:
        df_tmp = df.loc[n]
        df_tmp = np.rint(df_tmp.loc[:, 'production'])
        Summary_Table.loc[:, n] = df_tmp.values[::6]  # Sames as above

    Summary_Table.index.name = 'hour'
    Summary_Table.to_csv('D:/ICAI/Cuarto/TFG/Datasets/Production-'+str(year)+'.csv')

    return Summary_Table


"""
      ldpAnnualREE()
      Function: downloads yearly data from the ESIOS online application
      Inputs: year
      Outputs: csv file and dataframe with yearly production data with the REE format
"""


def ldpAnnualREE(year):

    token = 'token is private and has been purposely removed to avoid problems'

    indicators_ = [1293]  # Production indicator code
    esios = ESIOS(token)
    names = esios.get_names(indicators_)

    start_ = '01-01-'+str(year)+'T00:00:00'
    end_ = '31-12-'+str(year)+'T23:59:59'
    df_list, names = esios.get_multiple_series(indicators_, start_, end_)
    df_merged = esios.merge_series(df_list, names)  # merge the DataFrames into a
single one
```

```
    df = df_merged[names]
    df.index = df.index+timedelta(hours=1)
    df.columns = ['Production(MW)']

    df.to_csv('D:/ICAI/Cuarto/TFG/Data/Annual-Production-'+str(year)+'.csv')

    return df
```

Temperature data treatment code

```
import calendar
from datetime import timedelta
from datetime import datetime
import pandas as pd
import numpy as np
from pandas.io.parsers.readers import read_csv
"""
      Here are included all the necessary programms for the temperature data analysis
          - Ready_Temperatures(): cleans all the NOAA files and creates clean ones and
a table with all the years temperatures
          - Annual_Summary_Table(): generates a summary table with all the data from
each day (month, mean_t, day_type, etc.)
"""


"""
      Ready_Temperatures():
      Function: cleans the csv files from NOAA, stores them and generates a summary
table
      Input: csv file from NOAA
      Output: summary_df: dataframe with the annual summary
      Summary_Table()
          - File: summary_df
          - File: temperatures_df, format used in Calc_Mean_Temp(df, m)
"""


def Ready_Temperatures(year):

    # List useful columns from NOAAs files
    col_list = ['NAME', 'DATE', 'REPORT_TYPE', 'TMP']

    # List of out-of-peninsula locations
    P_insular = ['CEUTA', 'IBIZA', 'MELILLA', 'MENORCA', 'PALMA DE MALLORCA CMT',
'PALMA DE MALLORCA']

    summary_df = Annual_Summary_Table(year)
    summary_df.set_index('Date', inplace=True)
    for m in range(1, 13, 1):
        full_df =
pd.read_csv('D:/ICAI/Cuarto/TFG/Temperature/'+str(year)+'/'+str(year)+'-
'+str(m).zfill(2)+'.csv', usecols=col_list)
        df = full_df[full_df['REPORT_TYPE'] == 'FM-12']
        df = df.reset_index(drop=True)
        df = df.join(df['NAME'].str.split(',', expand=True).rename(columns={0:
'Location', 1: 'Country'}))
        df = df.join(df['TMP'].str.split(',', expand=True).rename(columns={0: 'Temp',
1: 'Quality'}))

        # Formating date and time
        df['DATE'] = pd.to_datetime(df['DATE'])
        df = df.drop(columns=['Country', 'Quality', 'TMP', 'NAME', 'REPORT_TYPE'])

        # We obtein each hours temperature
        df['Temp'] = (pd.to_numeric(df['Temp']))/10

        # We substitute the measure error code (999.9) with a nan
        df.loc[df['Temp'] == 999.9, 'Temp'] = np.nan

        for i in range(len(P_insular)):
            df = df[df['Location'] != P_insular[i]]

        # We save the clean data
        df.to_csv('D:/ICAI/Cuarto/TFG/Temperature/'+str(year)+'/Clean/'+str(year)+'-
'+str(m).zfill(2)+'-clean.csv')

        # We calculate each days mean temperature
        df_temperatures = Calc_Mean_Temp(df, m, year)
```

```
        # We fill the summary table
        for c in range(df_temperatures.shape[1]):
            N_Column = df_temperatures.columns[c]
            mean = df_temperatures.iloc[:, c].mean()
            summary_df.loc[N_Column, 'Mean_Temperature'] = np.around(mean, 1)

        summary_df.to_csv('D:/ICAI/Cuarto/TFG/Datasets/Table_'+str(year)+'.csv')
    return summary_df


"""
      Calc_Mean_Temp(df, m):
      Function: calculates the mean temperature of each day
      Inputs:
              - df: dataframe, with monthly data from month 'm'
              - m: month
      Outputs:
              - df_temperatures: dataframe with hourly temperatures of each month with
the following format:
                                    hour | date | date | date ... (hour: 0,1,2,...,23)
(year-month-day)
              - file: file the the previous dataframe
"""


def Calc_Mean_Temp(df, m, year):

    cols_list = []
    df_temperatures = np.zeros((24, calendar.monthrange(year, m)[1]), dtype=float)

    for d in range(calendar.monthrange(year, m)[1]):
        for h in range(24):
            date_hour = datetime(year, m, d+1, h, 0, 0)
            df_tmp = df[df['DATE'] == date_hour.strftime('%Y-%m-%d %H:00:00')]
            mean_h = df_tmp['Temp'].mean()
            if (np.isnan(mean_h)):
                mean_h = 0
            df_temperatures[h, d] = np.around(mean_h, 1)

        cols_list.append(date_hour.strftime('%Y-%m-%d'))
    df_temperatures = pd.DataFrame(df_temperatures)
    df_temperatures.columns = cols_list

df_temperatures.to_csv('D:/ICAI/Cuarto/TFG/Temperature/'+str(year)+'/Clean/'+str(year)
+'-'+str(m).zfill(2)+'-Temperatures.csv')

    return df_temperatures


"""
      Annual_Summary_Table():
      Function: makes a summary yearly table with the following format:
                  Date | Mean_Temperature | Day_Type | Season
                  - Date: year-month-day
                  - Mean_Temperature: mean temperature of the specific day
                  - Day_Type: working day (0), Saturday (5), Sunday (6)
                  - Season
      Output: summary table to add the temperatures
"""


def Annual_Summary_Table(year):

    cols_list = ['Date', 'Mean_Temperature', 'Day_Type', 'Season']
    summary_df = pd.DataFrame(np.zeros((365, 4)))

    # National festivity days list
    # festivity_list = ['2020-01-01', '2020-01-06', '2020-04-10','2020-05-01', '2020-
08-15','2020-10-12', '2020-12-08', '2020-12-25']
    festivity_list = ['2021-01-01', '2021-01-06', '2021-02-10', '2021-05-01', '2020-
10-12', '2021-11-01', '2021-12-06', '2021-12-08', '2021-12-25']

    date = datetime(year, 1, 1)
    for d in range(0, 365, 1):
        date_str = date.strftime('%Y-%m-%d')
        summary_df.iloc[[d], [0]] = date_str
```

```
        day_type = datetime.weekday(date)
        if (day_type < 5):
            day_type = 0
        elif (day_type == 5):
            day_type = 5
        elif (day_type == 6):
            day_type = 10
        summary_df.iloc[[d], [2]] = day_type

        if date_str in festivity_list:
            summary_df.iloc[[d], [2]] = 10  # We will treat festivities as Sundays
        summary_df.iloc[[d], [3]] = season(date)
        date = date + timedelta(days=1)

    summary_df.columns = cols_list
    return summary_df


"""
      temperature_data(date_str):
      Function: gets the temperature data for the input date, used to train the
network
      Inputs: Date
      Outputs: table with the temperature date of the chosen day
"""


def temperature_data(date_str):
    temp_data = np.zeros((24), dtype=float)
    date_dt = datetime.strptime(date_str, '%Y-%m-%d')
    year = date_dt.year
    path_file =
'D:/ICAI/Cuarto/TFG/Temperature'+str(year)+'/Clean/'+date_dt.strftime('%Y-%m')+'-
Temperatures.csv'
    df = pd.read_csv(path_file)
    temp_data = df[date_str].values.T

    return temp_data


"""
      Temp_Summary_Table()
      Function: generates a csv file with the hourly temperature data of every day of
the year
      Inputs: NOAA clean files
      Outputs: Summary_table.csv
"""


def Temp_Summary_Table(year):
    df =
pd.read_csv('D:/ICAI/Cuarto/TFG/Temperature/'+str(year)+'/Clean/'+str(year)+'-01-
Temperatures.csv', index_col=0)

    for m in range(2, 13, 1):
        df_tmp =
pd.read_csv('D:/ICAI/Cuarto/TFG/Temperature/'+str(year)+'/Clean/'+str(year)+'-
'+str(m).zfill(2)+'-Temperatures.csv', index_col=0)
        df = df.join(df_tmp)

    # Now we have to eliminate the nan:
        # For complete day missing, we copy the day before
        # For a missing hour, we copy the same hour from the day before

    for c in range(df.shape[1]):
        if (all(np.isnan(df.iloc[:, c])) or (all(df.iloc[:, c]) == 0)):
            df.iloc[:, c] = df.iloc[:, c-1]
        elif (any(np.isnan(df.iloc[:, c])) or (any(df.iloc[:, c] == 0))):
            for f in range(df.shape[0]):
                if (np.isnan(df.iloc[f, c])):
                    df.iloc[f, c] = df.iloc[f, c-1]

    df.index.name = 'hour'
    df.to_csv('D:/ICAI/Cuarto/TFG/Datasets/Temperatures-'+str(year)+'.csv')
```

```
    return df


"""
      season(date):
      Function: determines what is the current season depending on the date
      Input: Date (year-month-day)
      Output: Season (spring (1), summer (2), autumn (3), winter (4))
"""


def season(date):
    day = date.timetuple().tm_yday

    if day in range(80, 172):
        return 1  # Spring
    elif day in range(172, 264):
        return 2  # Summer
    elif day in range(264, 355):
        return 3  # Autumn
    else:
        return 4  # Winter
```

Neural network code

```
import numpy as np
import pandas as pd
import time
import random
import os
from datetime import datetime
from datetime import timedelta
import matplotlib.pyplot as plt


class Neural_Network (object):
    def __init__(self, N_Inputs, conf, weightload, typ):
        self.N_Input = N_Inputs
        self.N_Hidden_Layer = conf[0]
        self.N_Output = 24
        self.eta = conf[2]

        # Initialize seed to compare results
        np.random.seed(20)

        if (weightload):
            self.LoadWeights(conf, typ)
        else:
            # Declare weight matrixes randomly
            self.W1 = np.array(np.random.randn(self.N_Hidden_Layer, self.N_Input),
ndmin=2)
            self.W2 = np.array(np.random.randn(self.N_Output, self.N_Hidden_Layer),
ndmin=2)

    def sigmoid(self, x):  # Activation function
        return 1/(1+np.exp(-x))

    def primesigmoid(self, x):  # First derivative of sigmoid function
        return x*(1-x)

    def FF(self, input):  # Feedfordward

        input = np.array(input, ndmin=2).T
        # First layer:
        output_array = np.dot(self.W1, input)
        output_array = self.sigmoid(output_array)
        # Second layer:
        output_array = np.dot(self.W2, output_array)
        output_array = self.sigmoid(output_array)

        return output_array

    def SaveWeights(self, conf, typ):
        if (typ == 0):
            np.savetxt('D:/ICAI/Cuarto/TFG/Data/Weights/W1-'+str(conf[0])+'-
'+str(conf[1])+'-'+str(conf[2])+'.txt', self.W1, delimiter=',')
            np.savetxt('D:/ICAI/Cuarto/TFG/Data/Weights/W2-'+str(conf[0])+'-
'+str(conf[1])+'-'+str(conf[2])+'.txt', self.W2, delimiter=',')
        else:
            np.savetxt('D:/ICAI/Cuarto/TFG/Data/Weights/W1-TM-'+str(conf[0])+'-
'+str(conf[1])+'-'+str(conf[2])+'.txt', self.W1, delimiter=',')
            np.savetxt('D:/ICAI/Cuarto/TFG/Data/Weights/W2-TM-'+str(conf[0])+'-
'+str(conf[1])+'-'+str(conf[2])+'.txt', self.W2, delimiter=',')

    def LoadWeights(self, conf, typ):
        # If typ is 0 is hourly network
        # If typ is 1 is daily network
        if (typ == 0):  # 24 temperatures
            self.W1 = np.loadtxt('D:/ICAI/Cuarto/TFG/Data/Weights/W1-'+str(conf[0])+'-
'+str(conf[1])+'-'+str(conf[2])+'.txt', delimiter=',')
            self.W2 = np.loadtxt('D:/ICAI/Cuarto/TFG/Data/Weights/W2-'+str(conf[0])+'-
'+str(conf[1])+'-'+str(conf[2])+'.txt', delimiter=',')
        else:  # 1 temperature
```

```
            self.W1 = np.loadtxt('D:/ICAI/Cuarto/TFG/Data/Weights/W1-TM-
'+str(conf[0])+'-'+str(conf[1])+'-'+str(conf[2])+'.txt', delimiter=',')
            self.W2 = np.loadtxt('D:/ICAI/Cuarto/TFG/Data/Weights/W2-TM-
'+str(conf[0])+'-'+str(conf[1])+'-'+str(conf[2])+'.txt', delimiter=',')

    def training(self, input, output):

        Input_array = np.array(input, ndmin=2).T

        Goal_array = np.array(output, ndmin=2).T

        Output_array_1 = np.dot(self.W1, Input_array)
        Output_array_layer1 = self.sigmoid(Output_array_1)

        Output_array_2 = np.dot(self.W2, Output_array_layer1)
        Output_array_RN = self.sigmoid(Output_array_2)

        Output_errors = Goal_array - Output_array_RN

        # Correction of output weights

        delta = Output_errors * self.primesigmoid(Output_array_RN)
        delta = self.eta * np.dot(delta, Output_array_layer1.T)
        self.W2 = self.W2 + delta

        # Correction of input weights

        Hidden_layer_error = np.dot(self.W2.T, Output_errors)
        delta = Hidden_layer_error * self.primesigmoid(Output_array_layer1)
        self.W1 = self.W1 + (self.eta * np.dot(delta, Input_array.T))


"""
      Training24T (conf)
      Function: Trains the network with the 'conf' configuration with the 24
temperature input network
      Inputs: Network configuration [N_Hidden_Layer, Max_iter, eta]
              - N_Hidden_Layer: Number of neurons in the hidden layer
              - Max_iter: number of iterations for training
              - eta: learning factor
      Outputs:
              - Two text files with training weights (RN)
"""


def Training(conf, typ):
    Start_time = time.perf_counter()

    # N_Inputs = 27

    if (typ == 0):
        N_Inputs = 27
    else:
        N_Inputs = 4
    inputs = np.zeros((N_Inputs), dtype=float)
    max_iter = conf[1]
    columns = ['Date', 'Mean_Temperature', 'Day_Type', 'Season']

    # Initializing neural network

    NN = Neural_Network(N_Inputs, conf, False, typ)

    # Clean and filtered data tables

    production_table = pd.read_csv('D:/ICAI/Cuarto/TFG/Datasets/Production-2020.csv',
index_col=0)
    temperature_table = pd.read_csv('D:/ICAI/Cuarto/TFG/Datasets/Temperatures-
2020.csv', index_col=0)
    summary_table = pd.read_csv('D:/ICAI/Cuarto/TFG/Datasets/Table_2020.csv',
usecols=columns)

    max_t = np.max(summary_table.loc[:, 'Mean_Temperature'])

    min_t = np.min(summary_table.loc[:, 'Mean_Temperature'])

    max_t_27 = temperature_table.max().max()
```

```
    min_t_27 = temperature_table.min().min()

    ndays = np.arange(temperature_table.shape[1])
    random.seed(10)
    random.shuffle(ndays)

    print('##########################################')
    print('Training network with configuration: ' + str(conf))
    if (typ == 0):
        print('Network Imputs: 27')
    else:
        print('Network Imputs: 4')

    errors = []

    for i in range(max_iter):
        for n in ndays:
            date_str_temp = summary_table.loc[n, 'Date']

            date_dt_temp = datetime.strptime(date_str_temp, '%Y-%m-%d')
            inputs[0] = summary_table.loc[n, 'Day_Type']  # Type of day
            inputs[1] = date_dt_temp.month  # Month
            inputs[2] = summary_table.loc[n, 'Season']  # Season

            if (typ == 0):
                inputs[3:] = temperature_table.loc[:, date_str_temp].values  #
Temperatures
                inputs = normalize_inputs(inputs, min_t_27, max_t_27)

            else:
                inputs[3] = summary_table.loc[n, 'Mean_Temperature']
                inputs = normalize_inputsMeanT(inputs, min_t, max_t)

            pattern = production_table.loc[:, date_str_temp].values

            pattern = normalize_power(pattern)

            NN.training(inputs, pattern)

    NN.SaveWeights(conf, typ)  # We save the weights with the specific configuration
    print('Training completed')
    print('Execution time: ' + str(round(time.perf_counter() - Start_time, 2)) + 's')
    print('##########################################')

    return errors


"""
      ForecastCalculation24T(date_str, conf)
      Function: calculates the forecast for day 'date_str' with the 'conf'
configuration
      Inputs:
              - date_str: date on 'year-month-day' format
              - conf: see previous function
      Outputs:
              - Graph with the forecast, absolute and percentual errors per hour
              - .csv file with the forecast result
"""


def ForecastCalculation(date_str, conf, typ):
    Start_time = time.perf_counter()

    path_root = "D:/ICAI/Cuarto/TFG/Output Data/"
    if (typ == 0):
        N_Inputs = 27
    else:
        N_Inputs = 4
    inputs = np.zeros((N_Inputs), dtype=float)
    date_dt = datetime.strptime(date_str, '%Y-%m-%d')
    year = date_dt.year
    columns = ['Date', 'Mean_Temperature', 'Day_Type', 'Season']

    # Initializing neural network

    NN = Neural_Network(N_Inputs, conf, True, typ)
```

```
    production_table = pd.read_csv('D:/ICAI/Cuarto/TFG/Datasets/Production-' +
str(year) + '.csv', index_col=0)
    temperature_table = pd.read_csv('D:/ICAI/Cuarto/TFG/Datasets/Temperatures-' +
str(year) + '.csv', index_col=0)
    summary_table = pd.read_csv('D:/ICAI/Cuarto/TFG/Datasets/Table_' + str(year) +
'.csv', usecols=columns, index_col=0)

    max_t = np.max(summary_table.loc[:, 'Mean_Temperature'])
    min_t = np.min(summary_table.loc[:, 'Mean_Temperature'])

    inputs[0] = summary_table.loc[date_str, 'Day_Type']
    inputs[1] = date_dt.month
    inputs[2] = summary_table.loc[date_str, 'Season']

    max_t_27 = np.max(np.max(temperature_table))
    min_t_27 = np.min(np.min(temperature_table))

    print('############################################')
    if (typ == 0):
        print('Inputs to network: 27')
    else:
        print('Inputs to network: 4')

    print('Forecast Date: ' + date_str)
    print('Day: ' + str(inputs[0]))
    print('Season: ' + str(inputs[2]))
    print('Neurons in hidden layer: ' + str(conf[0]))
    print('Iterations: ' + str(conf[1]))
    print('Learning factor: ' + str(conf[2]))
    print('############################################')

    if (typ == 0):
        inputs[3:] = temperature_table.loc[:, date_str].values  # Temperatures
        inputs = normalize_inputs(inputs, min_t_27, max_t_27)
    else:
        inputs[3] = summary_table.loc[date_str, 'Mean_Temperature']
        inputs = normalize_inputsMeanT(inputs, min_t, max_t)

    real_production = production_table.loc[:, date_str].values

    # Calculating the forecast
    forecast = NN.FF(inputs)[:, 0]
    forecast = np.rint(renormalize_power(forecast))

    # Calculating errors and showing results
    Err_Array = Error_Calc(forecast, real_production)
    plot(forecast, real_production, date_str, path_root, Err_Array, conf, typ)

    # Preparing a Data Frame with the forecast data

    df = pd.DataFrame(forecast.T)
    index_list = [date_dt + timedelta(hours=x) for x in range(24)]
    df.index = index_list
    df.index.name = 'Date'
    df.columns = ['Power (MW)']

    # Saving the output data

    df.to_csv(path_root + 'Forecast-NN-' + date_str + '.csv')
    print('Execution time: ' + str(round(time.perf_counter() - Start_time, 2)) + 's')
    return [Err_Array, forecast]


"""
      test(date_str, conf, typ, file_name)
      Function: Lets tets the operation point of the network changing parameters as
eta, max_iter and N_Hidden_Layer
      Inputs:
              - Date
              - Conf: net configuration on the same format as the neural netowork, the
values must be in array format, for example:
                  conf = [['2020-02-15', '2020-03-15'],[[30],[1000, 3000, 5000], [0.1,
0.2]]]
              -file_name: name of the file, default will be test.csv
```

```
      Outputs:
                - File with the forecast results for all possible combinations, first 24
rows are the forecast while the latter ones are the errors
                    MAE, E_Max, E_Min, MAPE, Std_Dev

"""


def test(date_str, conf, typ, file_name=0):

    Start_time_iter = time.perf_counter()
    path_out = 'D:/ICAI/Cuarto/TFG/Output Data/'
    path_weights = 'D:/ICAI/Cuarto/TFG/Data/Weights/'

    # Net configuration

    N_Hidden_Layer, max_iter, eta = conf

    n_columns = []
    results_table = pd.DataFrame(np.zeros((29,
len(max_iter)*len(N_Hidden_Layer)*len(eta)), dtype=float))
    results_table.rename(index={24: 'MAE', 25: 'E_Max', 26: 'E_Min', 27: 'Std_Dev',
28: 'MAPE'}, inplace=True)

    c = 0  # With this c we control the column in the summary table

    for k in date_str:  # Date
        for e in eta:     # Learning factor
            for i in max_iter:  # Number of iterations
                for n in N_Hidden_Layer:
                    if (typ == 0):
                        file_W1 = path_weights + 'W1-' + str(n) + '-' + str(i) + '-' +
str(e) + '.txt'
                        file_W2 = path_weights + 'W2-' + str(n) + '-' + str(i) + '-' +
str(e) + '.txt'

                    else:
                        file_W1 = path_weights + 'W1-TM-' + str(n) + '-' + str(i) + '-
' + str(e) + '.txt'
                        file_W2 = path_weights + 'W2-TM-' + str(n) + '-' + str(i) + '-
' + str(e) + '.txt'

                    if((os.path.isfile(file_W1)) and (os.path.isfile(file_W2))):
                        forecast = ForecastCalculation(k, [n, i, e], typ)
                    else:
                        Training([n, i, e], typ)
                        forecast = ForecastCalculation(k, [n, i, e], typ)

                    abs_error, rel_error, std_error = forecast[0]
                    forecast = np.around(forecast[1], 0)

                    # Preparing column names
                    n_columns.append(k+','+str(n)+','+str(i)+','+str(e))

                    # Error calculation

                    results_table.loc['MAE', c] = round(np.mean(abs(abs_error)), 2)
                    results_table.loc['E_Max', c] = round(np.max(abs(abs_error)), 2)
                    results_table.loc['E_Min', c] = round(np.min(abs(abs_error)), 2)
                    results_table.loc['MAPE', c] = round(np.mean(rel_error), 2)
                    results_table.loc['Std_Dev', c] = round(std_error, 2)

                for f in range(24):
                    results_table.iloc[f, c] = forecast[f]
                c += 1

    # Changing the columns names (Date, Neurons, Eta, Iterations) and saving the file

    results_table.columns = n_columns
    results_table.index.name = 'Hour'
    if (file_name == 0):
        results_table.to_csv(path_out+'Test.csv')
    else:
        results_table.to_csv(path_out+str(file_name)+'.csv')
```

```
    print('\t Test completed in ' + str(round(time.perf_counter()-Start_time_iter,
2))+'s')
    return results_table


"""
      Error_Calc(forecast, real_production)
      Funtion: calculates the error on the prediction
      Inputs:
              - Forecast obtained by the neural network
              - Real production of the electrical network
      Outputs:
              - Error array
                  * Absolute errors per hour
                  * Relative errors per hour
                  * Standard deviation
              - Prompted output
                  * Mean absolute error (MAE)
                  * Maximum and minimum absolute error
                  * Standard deviation
                  * Mean percentual absolute error (MAPE)
"""


def Error_Calc(forecast, real_production):

    abs_error = forecast-real_production
    rel_error = (np.abs(abs_error)/real_production)*100
    std_error = np.std(abs_error)

    print('Mean absolute error (MAE): '+str(round(np.mean(abs(abs_error))))+' MW')
    print('Maximum absolute error: '+str(round(np.max(abs(abs_error)), 2))+' MW')
    print('Minimum absolute error: '+str(round(np.min(abs(abs_error)), 2))+' MW')
    print('Standard deviation: '+str(round(std_error, 2))+'MW')
    print('Mean absolute percentual error (MAPE): '+str(round(np.mean(rel_error),
2))+' %')

    return [abs_error, rel_error, std_error]


"""
      plot(forecast, real_production, date_str, path, Array_Error)
      Function: plots the forecast and error graphs
      Inputs:
              - Neural network forecast
              - Real production of the electrical network to compare the results
              - date_str
              - path
              - Array_Error
      Outputs:
              - Forecast vs Real graph
              - Absolute error per hour graph
              - Relative error per hour graph
"""


def plot(forecast, real_production, date_str, path, Array_Error, conf, typ):

    hours = np.arange(0, 24)
    abs_error, rel_error, std_error = Array_Error

    fig = plt.figure(figsize=(12, 3))

    ax = fig.add_subplot(131)
    ax.set(ylabel='Electric demand (MW)', xlabel='Hour', title='Forecast ' +
str(date_str))
    ax.plot(hours, real_production)
    ax.plot(hours, forecast)
    ax.grid(visible=True, which='major', axis='both')
    ax.legend(('Real production', 'NN Forecast'), loc='best')

    # Plotting the absolute errors per hour
    bx = fig.add_subplot(132)
    bx.set(ylabel='Absolute Error (MW)', xlabel='Hour', title='Absolute errors
'+str(date_str))
    bx.bar(hours, abs_error)
```

```
    # Plotting the realative errors per hour
    cx = fig.add_subplot(133)
    cx.set(ylabel='Relative Error (%)', xlabel='Hour', title='Relative errors
'+str(date_str))
    cx.bar(hours, rel_error)

    fig.tight_layout()
    plt.savefig(path+'graph_forecast'+str(date_str)+'-'+str(conf[0])+'-
'+str(conf[1])+'-'+str(conf[2])+'-red-'+str(typ)+'.pdf', format='pdf')
    plt.show()


"""
      normalize_power(pattern)
      Function: normalizes the pattern to be able to compare with the output of the
network
      Inputs: non normalized pattern
      Outputs: normalized pattern between 0 and 1
"""


def normalize_power(pattern):
    p_max = 45000
    p_min = 15000
    return (pattern-p_min)/(p_max-p_min)


"""
      renormalize_power(output)
      Function: renormalizes network outputs
      Inputs: normalized pattern between 0 and 1
      Outputs: non normalized pattern
"""


def renormalize_power(output):
    p_max = 45000
    p_min = 15000
    return (output*(p_max-p_min)) + p_min


"""
      normalize_inputs(inputs)
      Function: normalizes inputs before getting them into the network
      Inputs: non normalized inputs for the neural network
      Outputs: normalized inputs
"""


def normalize_inputs(inputs, min_t_27, max_t_27):
    inputs[0] = (inputs[0])/10  # Normalize the day
    inputs[1] = (inputs[1]-1)/11  # Normalize the month
    inputs[2] = (inputs[2]-1)/4  # Normalize the season
    inputs[3:] = (inputs[3:27]-min_t_27)/(max_t_27-min_t_27)  # Normalize the inputs
    return inputs


"""
      normalize_inputsMeanT(inputs)
      Function: normalizes inputs before getting them into the network
      Inputs: non normalized inputs for the neural network
      Outputs: normalized inputs (0 to 1)
"""


def normalize_inputsMeanT(inputs, min_t, max_t):
    inputs[0] = (inputs[0])/10  # Normalize the day
    inputs[1] = (inputs[1]-1)/11  # Normalize the month
    inputs[2] = (inputs[2]-1)/4  # Normalize the season
    inputs[3:] = (inputs[3]-min_t)/(max_t-min_t)  # Normalize the inputs
    return inputs
```

ANNEX: Project Alignment with the United Nations Sustainable Development Goals (SDG)

This project is mostly aligned with the sustainable development goal of the millennium number seven, as being able to predict electric demand and energy consumption is an enormous step towards achieving environmental sustainability, as optimal energy usage is far more likely when knowing beforehand the demand peaks and general forecast. Renewable energetic sources can be much more optimally used, as like explained in detail previously in the paper, these kind of energy sources spike on certain times of the day or the year.

According to the more in-depth list from the United Nations sustainable development goals, this project clearly aligns with the goals number seven and eight, for the same reasons mentioned above. Number eight specifically thanks to the economic upper hand that knowing the electric demand and acting in consequence brings to the table. Greater resources can be spent on clean energy sources if the ultimate usage to the fullest extent can be given to these utterly vital assets to the planet.

LIST OF FIGURES

LIST OF TABLES