Master's Degree Project

Master's Degree in Industrial Engineering

ARTIFICIAL INTELLIGENCE APPLIED TO DEMAND FORECASTING

MAIN REPORT

Author: Xavier Vilanova Rubau
Director: Mònica Aragües / Antonio Emmanuel Saldaña

Date: June 2021

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Abstract

Long-Term Load Forecasting based on Artificial Intelligence (AI) methods are today seen as the main research interest for the optimal planning of smart energy grids. Numerous techniques have been applied by the researchers to forecast the future electrical energy demand, which can be broadly categorized as parametric (statistical) and non-parametric (artificial intelligence techniques). Due to the amount of data collected in the past years, non-parametric techniques are gaining a lot of attention. *Artificial Neural Networks (ANN)* compared with other intelligent techniques is able to map and memorize the non-linear relations between inputs and outputs variables.

This project focuses on long-term energy demand forecasting and for this purpose two *Statistical models (ARIMA, SARIMA)* and three *Artificial Neural Networks (2 RNN, 1 MLP)* have been developed. These models are implemented on real time electricity data from *5567 Households of UK Power Network*, during the period from November 2011 to February 2014 at 30-minute resolution intervals. Performance indicators have been computed and the advantages and disadvantages of each model have been analysed. The results, show that *Artificial Neural Networks* are found to be highly accurate (MAPE 7,20%) and have lower computational complexity than *Statistical Methods.*

CONTENTS

1. Glossary

ARIMA	Auto Regressive Integrated Moving Average
SARIMA	Seasonal Auto Regressive Integrated Moving Average
AR	Auto Regressive
MA	Moving Average
ARMA	Auto Regressive Moving Average
AIC	Aikaike's Information Criterion
BIC	Bayesian Information Criterion
RMSE	Root Mean Square Error
MSE	Mean Square Error
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
NN	Neural Network
LSTM	Long Short-Term Memory
ERU	Elman Recurrent Unit
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
MLP	Multiple Layer Perceptron
ReLU	Rectified Linear Unit
ACF	Auto Correlation Function
PACF	Partial Auto Correlation Function

2. Preface

2.1. Origin of the project and motivation

With the transition from the analog to the digital world, great changes have been brought with the use of Internet of Things, generating huge amounts of data in the energy system, which are mostly never used. The appropriate tracking, acquisition and processing of this data can drive into innovative tools and services.

The project arises from the need to know where it would be necessary to improve the electrical infrastructure both in maintenance and deployment. With the ability to make accurate long-term energy demand forecasts, we may be able to execute planning strategies that maintain and expand the distribution network in a cost-efficient way that also ensures a stable and safe operation in long term scenarios.

This thesis is intended to be applicable in small part to the *Big Data for Open Innovation Energy Marketplace (BD4OPEM H2020)* project, which proposes innovative solutions in big data and artificial intelligence applied to the energy sector to improve the management of electricity grids[21].

2.2. Previous requirements

For the correct understanding of this thesis its needed basic knowledge of statistics, data management and python3.

3. Introduction

Energy demand is constantly changing, current infrastructures must be updated and improved, and it is necessary to foresee where there will be significant changes and fluctuations in energy demand. The increase in technology over the last few years and the large amount of stored data open up new possibilities for intend to forecast this energy demand fluctuations.

Researchers during the last few years have developed different techniques (parametric and non-parametric) and applied to short – term energy demand forecasting (less than 1 year) with good results. However, less papers and studies are published referring to long – term demand forecasting (more than 1 year) so it is an unexplored field.

Basic statistical methods have been used for short-term energy demand forecasting and great results were obtained from *Artificial Neural Networks* (especially *Recurrent Neural Networks*). Nevertheless, the performance of *Statistical methods* and *Artificial Neural Networks* applied to long-term demand forecasting is a question mark in the science world and there is only recent research about its performance.

As input data of the project, it's been used a univariate time series data from UK 5567 Households of UK Power Network, during the period from November 2011 to February 2014 at 30-minute intervals.

3.1. Scope of the project

This study is part of the *Big Data for Open Innovation Energy Marketplace (BD4OPEM H2020)* project. It aims to understand the performance and behaviour differences of Statistical Methods versus Neural Networks applied to long – term forecasting. This master thesis is focused in developing simple *statistical models (ARIMA, SARIMA)* and simple structured *Artificial Neural Networks (RNN, MLP)* in order to compare them and determine which one performs better for univariate long – term energetic demand forecasting.

3.2. Project Goals

The main goal of this thesis is to test and compare different architectures of artificial intelligence techniques applied to long – term energetic demand forecasting. In order to compare the proposed approaches, some performance indicators are taken into account, such as, *RMSE, MSE, MSE and MAPE* to determine the accuracy and the execution time to

determine the computational efficiency. To do so, a dataset from UK 5567 Households of UK Power Network is analysed and 5 forecast of Aggregated Load Demand using 2 Statistical methods (*ARIMA, SARIMA)* and 3 Neural Networks (*SimpleRNN, LSTM, MLP)* in a winter scenario is created to compare the qualitative differences between models and study the performance.

.

4. State of the Art

4.1. PROBABILISTIC DEMAND FORECASTING – STATISTICAL METHODS

In this section different statistical methods subsequently used for long-term energy demand forecasting are presented. Each mathematical model will be described along with its properties and the methodology used for the estimation of its parameters.

The models studied are statistical models that use a part of the data to determine the internal constants of the model (*training*) by iteration and once the model is defined, a prediction is made, after all we can evaluate the quality of the model calculating the error between the real values and the forecasting. Fort the study has been chosen 2 reference models: *ARIMA* and *SARIMA* (adding the seasonality to the previous one).

4.2. Auto regressive Integrated Moving Average (ARIMA)

ARIMA (Auto regressive Integrated Moving Average) is a generalization of *ARMA (Auto Regressive Moving Average)* model in which one more parameter is added to control the stationarity of the data; it consists of a mathematical model developed at the end of the 20th century and systematized in 1976 by *Box & Jenkins*. *ARMA* is a dynamic model, that means that future estimates do not depend on independent variables, depend on past values of data. The model can be divided into 2 types of independent models, autoregressive model (*AR*) and moving average model (*MA*)[5].

Autoregressive model is a representation of a random process, in which it is specified that the output variable depends linearly on its predecessors by satisfying equation 4.1.

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t \qquad (4.1)$$

Where X_t represents the output variable, $\varphi_i \; and \; c$ are constant values defined by the model, X_{t-i} represents previous values used ($p \geq i$) and the white noise is represented by the coefficient ε_t. Likewise, the parameter 'p' represents the number of previous values to be used for model prediction.

Moving average model represents an invariant time series, in which the output variable is specified to depend linearly on the current and past values of an unpredictable

constant, thus the model directly accumulates the error of previous predictions. The output can be expressed by the following equation (4.2) where 'q' parameter determines the influence of the past values in the forecasting.:

$$X_t = \mu + \varepsilon_t + \theta_t \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{q-1} \tag{4.2}$$

The *ARIMA* model adds to the *ARMA* model the possibility of dealing with non-stationary dataM the integral part 'I' of ARIMA takes careo f it. *ARIMA* models with non – stationary data has always this constants that Will vary depending on the dataset:

$$p \rightarrow order\ of\ the\ Autoregressive\ term$$
$$q \rightarrow order\ to\ make\ the\ data\ stationary$$
$$d \rightarrow order\ of\ the\ Moving\ Average\ Term$$

Equation for generic ARIMA (p, d, q):

$$X_t = \emptyset_1 \cdot X_{t-1} + \emptyset_2 \cdot X_{t-2} + \cdots + \emptyset_p \cdot X_{t-p} + \beta_1 \cdot \varepsilon_{t-1} + \beta_2 \cdot \varepsilon_{t-2} + \cdots + \beta_q \cdot \varepsilon_{t-q} + \varepsilon_t$$
$$where\ p, q \leq t \tag{4.3}$$

4.2.1. SEASONALITY AUTO REGRESSIVE INTEGRATED MOVING AVERAGE (SARIMA)

Seasonality must be taken in care in time series forecasting; so, if it's known that in a dataset some values are higher or lower in a regular cadence this means there is seasonality. SARIMA (*Seasonality Auto Regressive Integrated Moving Average)* models take seasonality into account by essentially applying an ARIMA model to lags that are integer multiples of seasonality (previously identified). Once seasonality is modelled with seasonality parameters (P, D, Q and s), ARIMA is applied to the non-seasonal structure. Seasonality can be identified by applying the differencing operator on different points of the time series. Next table shows an explanation of the parameters:

Parameter	Definition
P	Seasonal Autoregressive Order
D	Seasonal Difference Order

Q	Seasonal Moving Average Order
s	Number of time steps for a single seasonal period

To define the parameters P, D and Q we have used the methods studied previously in the ARIMA section such as Box & Jenkins Methodology combined with ACF and PACF. The data for the good parameterization must be previously modified, so differential operator is applied between seasonal periods to perform the study on the difference values. As an extension of ARIMA same parameters *p, d and q* are applied for forecasting with SARIMA.

4.3. Criterions to find p, d, q values

There is different combination of values that can be assigned to p, d, q. There are different approaches to find the best p, d, q values that minimize the prediction error. In this section the most common criterions and methodology are shortly described.

4.3.1. Box and Jenkins Methodology

Box and Jenkins methodology consists in an iterative process developed by Box & Jenkins in 1976. This methodology is used over year to determine the coefficients (p, d, q) in ARIMA for an accurate prediction; the requirements for using this methodology consists in having a stationary time series or a time series that is stationary after one or more differencing degrees. The next graph shows us the methodology applied during the project for determining ARIMA p, d, q coefficients:

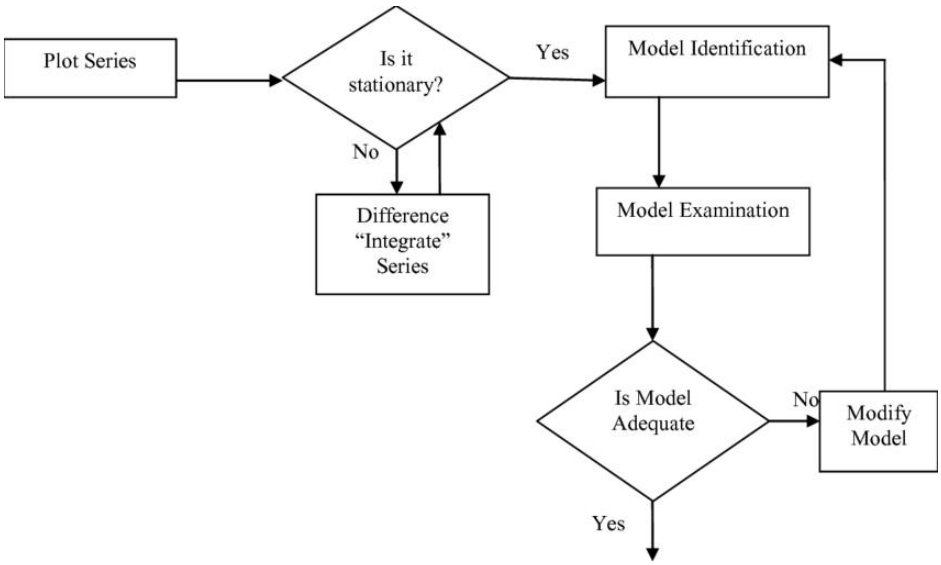

Fig.1 Box & Jenkins Methodology [1]

Step 1: Identifying stationary of the data.

Assuming stationarity of the model is necessary to provide a valid basis for forecasting. In Mathematical approach we call a stochastic process stationary in the broad sense if it satisfies equation 4.4:

$$E(x_t) = \mu; E[(x_t - \mu)^2] = \sigma_x^2; Cov(x_t, x_{t+k}) = \gamma(k) \quad (4.4)$$

Plotting your data gives us a first impression if the data is stationary or has a clear trend. Trend can be identified by measuring the slope of the linear regression equation. In addition to graphical (visual) methods, there are different tests that we can pass through our data that clearly identify if data is stationary or must be corrected such as 'Augmented Dickey-Fuller Unit Root Test'.

Augmented Dickey-Fuller Unit Root Test is a type of statistical test that defines how strongly a time series is defined by a trend (non-stationary), that uses an autoregressive model and optimizes an information criterion across multiple lag values. Test is formed by two hypotheses:

- H_0 (Null Hypothesis): Suggests time series has a unit root, meaning is non-stationary and has some time dependent structure.
- H_1 (Alternate Hypothesis): Suggests time series does not have a unit root, meaning it is stationary and does not have a time dependent structure.

Result of the test is interpreted using a p-value given by the test and a threshold (normally 5%) determined by the user. A p-value below threshold means H0 is rejected, and data is stationary; a p-value above the threshold suggests we fail to reject the null hypothesis and data is non-stationary.[1]

Step 2: Determine p and q through Autocorrelation: ACF and PACF.

Next steps consist in developing an accurate prediction of *p* and *q*. ACF and PACF[4] functions helps us to determine the order of each parameter. Both samples of the Autocorrelation Plot and Partial Autocorrelation plot are compared to the theoretical behaviour of these plots when the order is known. [2][3]

Autocorrelation Function [9] defines how data points in a time series are related, on average, to the preceding data points (it measures the self-similarity of the signal over different delay times). A scatter plot of $\hat{u}_t$ with $\hat{u}_{t-1}$ can be defined as simple regression equation without constant as follows:

$$\hat{u}_t = \emptyset_t \cdot \hat{u}_{t-1} + e_t \quad t = k+1, \dots, T \quad (4.5)$$

Where $\widehat{\emptyset}_k \cong \hat{p}_k = \frac{\sum_{t=k+1}^{T} \hat{u}_t \cdot \hat{u}_{t-k}}{\sum_{t=k+1}^{T} \hat{u}_{t-k}^2}$ and takes values from -1 to 1. If we make the estimation for different time steps k=1, 2, … K, we obtain a number of simple correlation coefficients $\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_k$ and it's representation through time steps it's the AutoCorrelation Function. To compare zero hypothesis $(\emptyset_k = 0)$ in front of the alternative $\emptyset_k \neq 0$ we calculate the t rate where:

$$t = \frac{\widehat{\emptyset}_k}{\sqrt{VAR(\widehat{\emptyset}_k)}} \rightarrow that\ follows\ a\ N(0,1)\ distribution\ where\ VAR(\widehat{\emptyset}_k) = \frac{1}{T} \qquad (4.6)$$

The hypothesis is denied if:

$$\left| \frac{\widehat{\emptyset}_k}{\sqrt{VAR(\widehat{\emptyset}_k)}} \right| > z_{\frac{\alpha}{2}} \leftrightarrow \left|\widehat{\emptyset}_k\right| > z_{\frac{\alpha}{2}} \cdot \sqrt{VAR(\widehat{\emptyset}_k)} \qquad (4.7)$$

$$where\ z_{\frac{\alpha}{2}}\ is\ the\ critical\ value \mid Prob\left(N(0,1) > z_{\frac{\alpha}{2}}\right) = \frac{\alpha}{2}$$

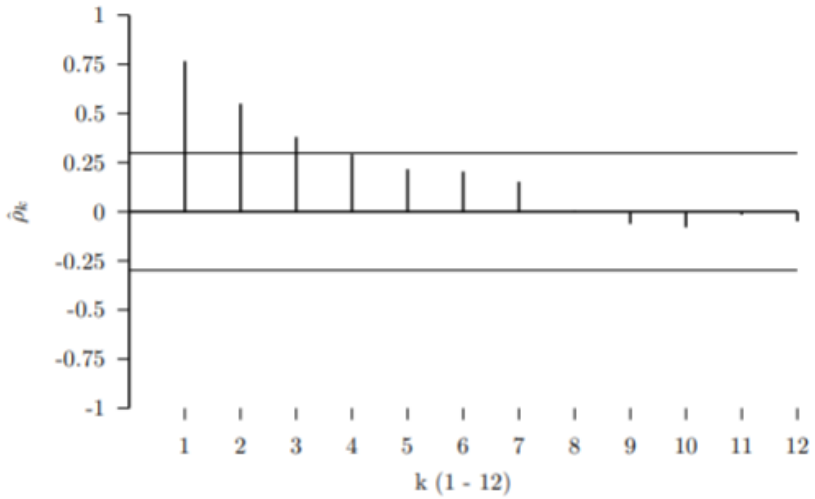

Fig 2. Autocorrelation Function [9]

Partial Autocorrelation Function as a complement of the ACF is constructed by estimating successive equations of multiple regression shown as follows:

$$\begin{aligned}
&\hat{u}_t = \emptyset_{11} \cdot \hat{u}_{t-1} + e_t \\
&\hat{u}_t = \emptyset_{21} \cdot \hat{u}_{t-1} + \emptyset_{22} \cdot \hat{u}_{t-2} + e_t \\
&\hat{u}_t = \emptyset_{31} \cdot \hat{u}_{t-1} + \emptyset_{32} \cdot \hat{u}_{t-2} + \emptyset_{33} \cdot \hat{u}_{t-3} + e_t \\
&\vdots \\
&\hat{u}_t = \emptyset_{k1} \cdot \hat{u}_{t-1} + \cdots + \emptyset_{kk} \cdot \hat{u}_{t-k} + e_t
\end{aligned} \qquad (4.8)$$

The coefficients $\emptyset_{11}, \emptyset_{22}, \emptyset_{33}, \ldots, \emptyset_{kk}$ is named PACF sample. This terminology becomes from the interpretation of the multiple regression coefficients as partial regression coefficients. The coefficients $\sqrt{T\emptyset_{kk}}$ follow a N(0,1) distribution, so, the zero hypothesis $\emptyset_{kk} = 0$ is denied in front of $\emptyset_{kk} \neq 0$ when $\left|\widehat{\emptyset}_{kk}\right| > 2/\sqrt{T}$.

1. *Order of Autoregressive Process (p):* For an AR (1) process, the sample autocorrelation function should have an exponentially decreasing appearance. However, with higher-order AR processes are often a mixture of exponentially decreasing and damped sinusoidal components.

   For higher – order autoregressive process, we must calculate the PACF that will become to zero of an *AR (p)* at lag *p+1*; so, we examine if there is evidence of a departure from zero. A 95% confidence interval on the sample partial autocorrelation plot is usually determined.

2. *Order of Moving Average Process (q):* The autocorrelation function of a MA(q) process becomes zero at lag q+1 and greater, so we examine the sample autocorrelation function to see where it essentially becomes zero. We do this by placing the 95 % confidence interval for the sample autocorrelation function on the sample autocorrelation plot.

   For determining the order of moving average process the PACF is usually not useful.

With the shape of Autocorrelation function we are capable to identify the model as explained before. Next table summarises how to determine coefficients p and q with the plot shape:

Table 1. Summary of Autocorrelation Shape

SHAPE	INDICATED MODEL
Exponential Shape, decaying to zero	*Model is autoregressive, to determine it's order we use the PACF plot.*
Alternating positive and negative, finally decaying to zero	*Model is autoregressive, to determine it's order we use the PACF plot*
One or more spikes, rest of them are essentially zero	*Model is moving average, order identified by where plot becomes to zero.*
Decay starts after a few lags	*Mixed autoregressive and moving average model*

All zero or close to zero	*Dataset is random*
High values at fixed invervales	*Include seasonal autoregressive term (SARIMA)*
No decay to zero	*Series is not stationary.*

Step 3: Check the accuracy of the model.

Once coefficients are determined, we do a prediction with a real time series data. Using different error performance indicators such as *(MAPE, MAE, RMSE, etc.).* If the results given for the model are satisfactory, we determine that the model is 'robust' and we can continue with the forecasting.

4.3.2. Akaike´s Information Criterion (AIC): Find the models with the lowest AIC values

Aikaike-s Information Criterion (AIC) is an estimator of prediction error and thereby relative quality of statistical models for a given set of data. AIC is founded on information theory that means that when statistical models are used to represent the process that generate the data, this representation will never be exact and there will always be some information that is lost. The AIC parameter estimates the relative amount of information that is lost by a model, the less information a model loses, higher is the value and the quality of the model[6].

There will always be information lost when a 'true data' is represented so the model is an estimation, not an exactly 'replica'. The main purpose of AIC is to determine how well a model fits the data it was generated from. AIC is calculated from:

- *K:* Number of independent variables used to build the model.
- *L:* Maximum likelihood estimate of the model (how well the model reproduces the data.

Using the two parameters mentioned before we can calculate the AIC coefficient:

$$AIC = 2 \cdot K - 2 \cdot \ln(L) \qquad (4.9)$$

K for default is always 2 so if your model use one independent variable your K will be 3, if it uses two independent variables your K will be 4, and so on.

For determining which one is better, we need to calculate for each model the AIC Coefficient; if a model is more than 2 AIC units lower than another, it is considered significantly better. [6]

The main problem for calculating the AIC coefficient is that calculating the log-likelihood is difficult so statistical software is used for this calculation, in this project Python libraries are used for calculating the log-likelihood for the models.

4.3.3. Schwarts Bayesian Information Criterion (BIC) models with the lowest BIC values.

Bayesian Information Criterion or Schwarts Criterion is a criterion for model selection among a finite collection of models. It is closely related with AIC (*Akaike Information Criterion*) due to it is based, in part, on the likelihood function.

Increasing the likelihood when fitting models can be easy by adding more parameters but can produce overfitting. BIC solve this problem by introducing a penalty term in the equation for the number of parameters in the model; it's term Is larger in BIC than in AIC (*that's the main difference).*

The BIC balances the number of model parameters and number of data points against the maximum likelihood function as follows:[7]

$$k = number\ of\ model\ parameters$$
$$n = numer\ of\ data\ points$$
$$L = maximum\ likelihood\ function$$

$$BIC = k \cdot \ln(n) - 2 \cdot \ln(\hat{L}) \tag{4.10}$$

Maximum likelihood function is defined as:

$$\hat{L} = p(x|\hat{\theta}, M) \tag{4.11}$$

This is explained as how likely our data *x* is explained by the model *M*, which has certain model parameters θ. Usually, the calculation of $\hat{L}$ parameter, is done by statistical software, in this project we use python libraries[8].

4.4. PROBABILISTIC DEMAND FORECASTING – NEURAL NETWORKS

The origins of neural networks (NN) go back to the late 1950s where Rosenblatt conceived the idea of *perceptron*, a simple mathematical model of how neurons in the brain operate. From the earliest ideas to the present day, interest in NN's has oscillated considerably from great interest from the scientific community to periodic abandonment. [10]

Neural networks are made up of cells (which act as neurons) connected to each other; the type of cell, the organization of the cells (number of layers), the connections between them and how the neural network is trained determines the type of neural network. There are many types of cells, some of which are shown below:

- ***Perceptron****:* A single-layer perceptron is the basic unit of a neural network, it consists on input values, weights and a bias, a weighted sum and activation function.

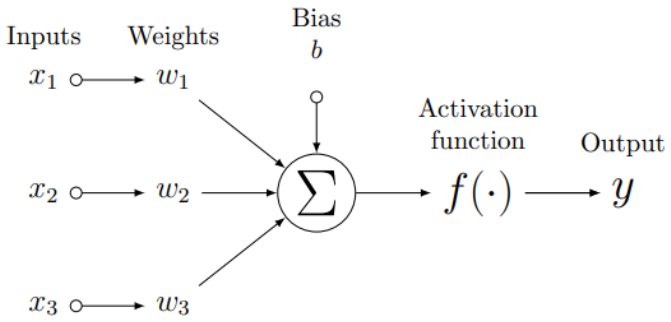

Fig 3. Single – layer Perceptron [10]

- ***Long Short-Term Memory (LSTM):*** LSTM is composed by a cell, an input gate, an output gate and a forget gate. It remembers values over arbitrary time intervals and the tree *gates* regulate the flow of information into and out of the cell.

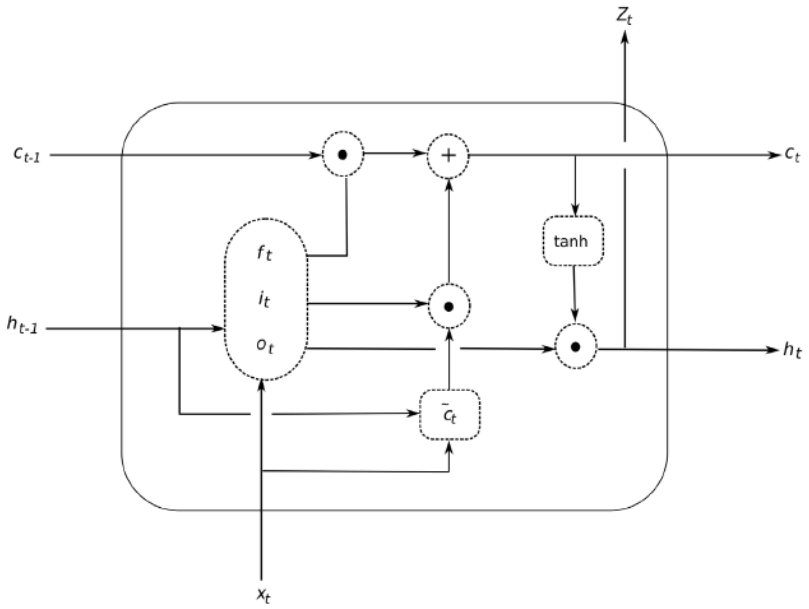

Fig 4. Basic LSTM unit. [11]

- ***Elman recurrent unit (ERU):*** *Elman Recurrent Neural Network (*ERNN) are networks with hidden state but without any advanced gating mechanisms. ERNN cell suffers from the well-known vanishing gradient and exploding gradient problems over very long sequences. This implies that are not capable of carrying long-term dependencies to the future.

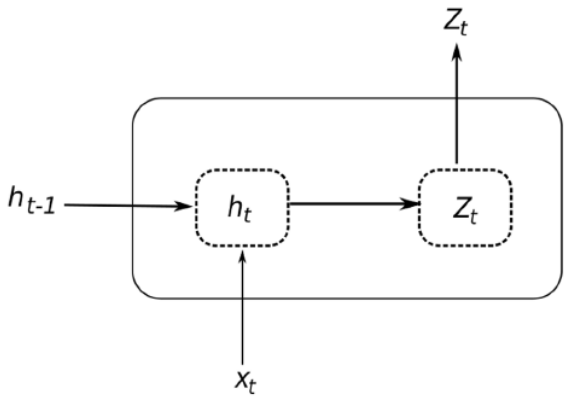

Fig 5. Elman recurrent Unit [11]

4.4.1. Learning Algorithms

In order for NNs to learn and improve results, so-called learning algorithms are used.*The AdaGrad optimizer (AdaGrad), Root Mean Square Propagation (RMSProp) and The Adam optimizer (Adam).* All of them require and hyperparameter learning rate, that must be correctly tuned for not perturbing the learning process.

- ***The AdaGrad optimizer (AdaGrad):*** Adapts de learning rate to the parameters, performing smaller updates for parameters associated with frequently occurring features, and larger updates for parameters associated with infrequent features[13].
- ***Root Mean Square Propagation (RMSProp):*** Maintain per – parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight. This algorithm works well in online and non-stationary problems.
- ***The Adam optimizer (Adam):*** Is an algorithm that can be used instead of the classical stochastic gradient descent procedure for updating network weights iterative based in training data. It's been described as combining the best features of *AdaGrad* and *RMSProp* and famous for it's computational efficiency and little memory requirements, also, have an intuitive interpretation and typically require little tuning[12].

4.4.2. Activation function

A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks. Activation function defines how the sum

of the input is transformed into an output from a node or nodes in a layer of the network.

Normally the activation function for the nodes of a hidden layer is the same and can vary from one layer to another. The activation functions can be classified according to the layer in which they are to be used.

Hidden layers receive the input from another layer (Input Layer or Hidden Layer) and provides output to another layer (Output Layer or Hidden Layer). Basically, there are 3 main activation functions used in hidden layers:

- *Rectified Linear Activation (ReLu):* ReLu is the most common activation function used in NN design, its calculated as $f(u) = \max(0.0, x)$; this means that if x has a negative value the output is 0, otherwise the output will be the value of x (as shown in Figure 6) [14][15].

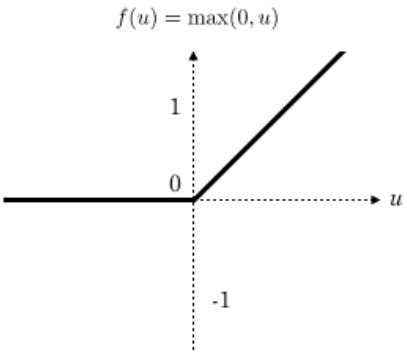

Fig 6. ReLu Function [18]

- *Sigmoid:* It's also called *logistic function* and it's calculated as $(u) = \frac{1}{1-e^{-x}}$. Output range is between 0 to 1 and more positive is the value is closer to 1, otherwise ($x < 0$) output is closer to 0.

- *Hyperbolic tangent (tanh): Tanh* is remarkably similar to the sigmoid function for its S-shape, output values are from -1 to 1. The main difference between the *Sigmoid* and *Tanh* is that Sigmoid values goes from 0 to 1 instead of -1 to 1, furthermore, near input values close to zero in *tanh* will be centred in zero.

$$y = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4.12}$$

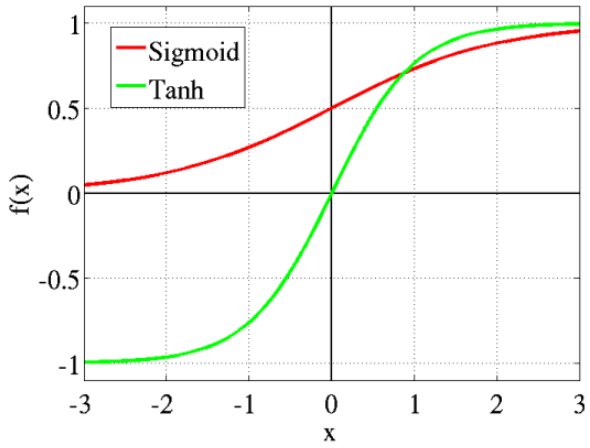

Fig 7. Sigmoid vs Tanh [19]

Output layers receive the input from the hidden layers or input layer and directly outputs the prediction of the model, depending on the type of NN and the data, the activation function will be different; a study must be carried out to determine which is the most appropriate in each case.

- *Linear Activation Function*: Multiplies the weighted sum of the input per 1.0 so there is no transformation; due to it's naturality is also called '*identity'* or '*no activation'*. Linear activation function is useful for regression problems and usually, the target values used to train the model had scaled prior using normalization or standardization transformations.

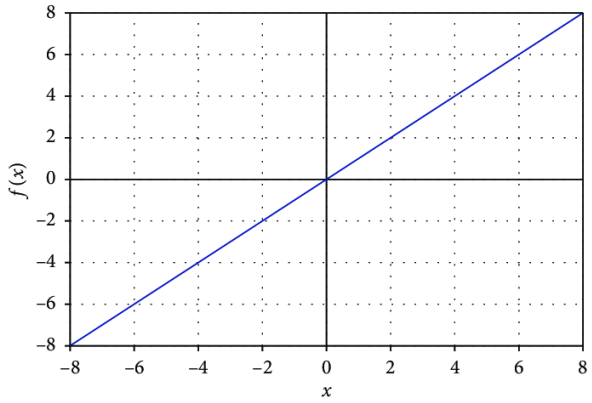

Fig 8. Linear Activation Function [20]

- *Sigmoid Activation Function*: As explained before Sigmoid outputs goes from 0 to 1. Sigmoid is used when the output accepts multiple 'true' answers, that means that the sum of outputs is not necessary to be equal to 1 performs well with problems of binary classification or multi-label classification.

- *Softmax Activation Function*: Outputs a vector of values that converts the vector of input values into probabilities, probabilities are proportional to each value in the input vector. Softmax is often used in NN that provide $n\ (n > 1)$ outputs, one for each class in the classification task (Multi-class Classification); the activation function normalizes the output, converting them from weighted sum values into a vector of probabilities that sum to one.

  Softmax activation is related to argmax function that outputs a 0 for all option and 1 for the chosen option, the main difference is that instead of 0 values gives probability values that sum 1.

$$p, x_i = \frac{e^{x_i}}{\sum_{i=1}^{N} e^{x_i}}\ i = 1, I, N \tag{4.13}$$

4.4.3. Multilayer perceptrons (MLP)

MLPs also known as *feed-forward* neural networks are the most basic form of artificial NNs. The name of *"Feed-Forward"* comes from the fact that the inputs are feed forward, through the hidden nodes (if any) of the network and to the outputs nodes only in one direction. There are no cycles or loops in the network.

A complex NN is composed by *Input layer, hidden layer(s) and output layer*. The input features are denoted as the *input layer*. Output nodes are denoted as *output layer* and *hidden* layers are layers between *input layer* and *output layer*, nodes in each layer of the network are fully connected to all the nodes in the previous layer. A fully connected layer is called a *dense* layer. Figure 9 (a) shows a generic example of a SLP and Figure 9 (b) shows a MLP:

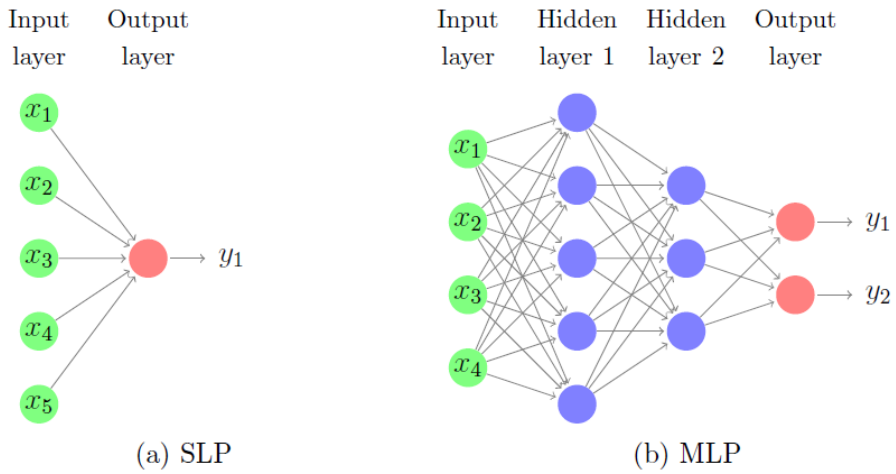

Fig 9. Single layer Perceptron and Multiple Layer Perceptron [10]

One of the main limitations of MLP is that the number of input and output values is fixed,

which makes it inapplicable to problems where the size of the input and output values vary as in the case of time series. In the hidden layers the main activation function is *Rectified Linear Activation (ReLu).*

4.4.4. Recurrent Neural Networks (RNN)

Recurrent neural networks (RNNs) are designed specifically to handle sequential data that arise in applications such as time series, natural language processing and speech recognition. A direct approach to address the sequential nature of the data is to use *recurrent* connections between nodes that connect the neural networks hidden units back to themselves with a time delay[11].

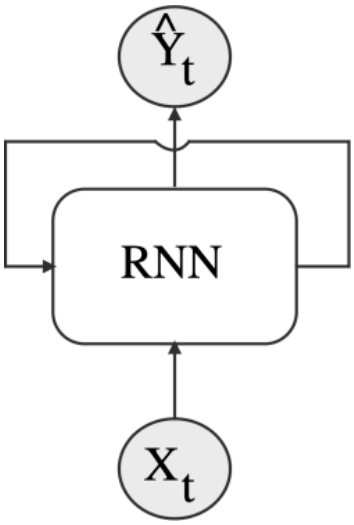

Fig 10. Folded RNN [11]

Since the hidden units learn some kind of feature from the raw input, feeding hidden units back to themselves in each time step can be interpreted as providing the network with a kind of dynamic memory. A crucial detail Is that the same network is used for all time steps; this weight-sharing allows the RNNs to handle sequences of varying length during training and, generalize to sequence lengths not seen during training.

Training RNN is quite difficult given that they are typically applied to very long sequences of data. LSTM were proposed to address the problem of training, instead of using a simple network at each time step, LSTMs use a more complicated architecture that controls the flow of input to the cell as well as decide on what information should be kept inside and what information should be propagated through next time step. One simplest variant of LSTMs is *Gated Recurrent Units (GRU)* that do not use a separate memory cell and consequently they are computationally more efficient.

Stacked Architecture: Figure 11 shows the structure of one layer in stacked architecture. The feedback loop of the cell helps the network to propagate the state h_t to the future time steps; for generalisation purposes in the diagram, it's only shown h_t but for an LSTM cell, h_t should be accompanied by the cell state C_t. Stacking means that multiple LSTM layers can be stacked on top of one another (shown in Figure 12), the output from every layer is directly fed as input to the next layer immediately above, and the final forecasts are retrieved from the last layer. X_t denotes the input to the cell at time step t and $\hat{Y}_t$ corresponds to the output; once stacked X_t and $\hat{Y}_t$ are vectors instead of single data points.

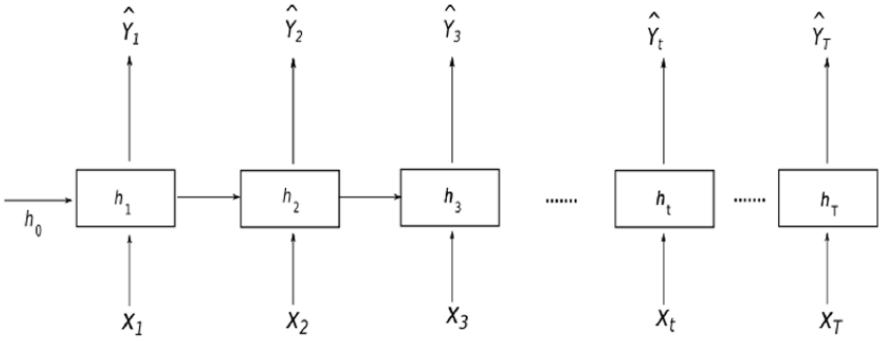

Fig 11.1-Layer Stacked Architecture [11]

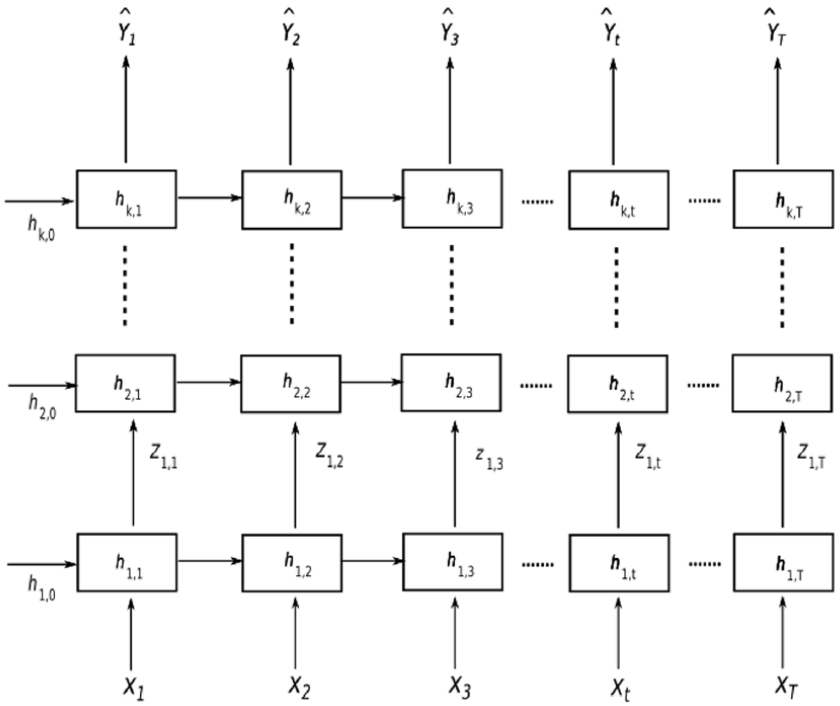

Fig 12. Multi-Layer Stacked Architecture [11]

Sequence-to-sequence architecture: Figure 13 illustrates the generalised S2S architectures. The input $x_t$fed to each cell instance of this network is a single data point. *Encoder* is the part of the RNN where cells keep getting input at each time step and consequently build the state of the network. However, in contrast to *stacked architecture*, the output is not considered for each time step; rather, only the forecast produced after the last input point of the encoder is considered.

The *Decoder* is the component that produces the outputs in this manner, every y_t corresponds to a single forecasted data point in the forecast horizon. The initial step of the *decoder* is the last step of the *encoder* which is also known as the context vector, and it contains autoregressive connections from the output of the previous time step into the input of the cell of the next one. During training, those autoregressive connections are disregarded and externally fed with the actual values of the output of the previous time, this is called teacher forcing – helps the *decoder* to see how much it should be corrected. During testing, since the actual values are not available (forecasting is made), auto regressive connections are used instead as the actual value because it is unknown.

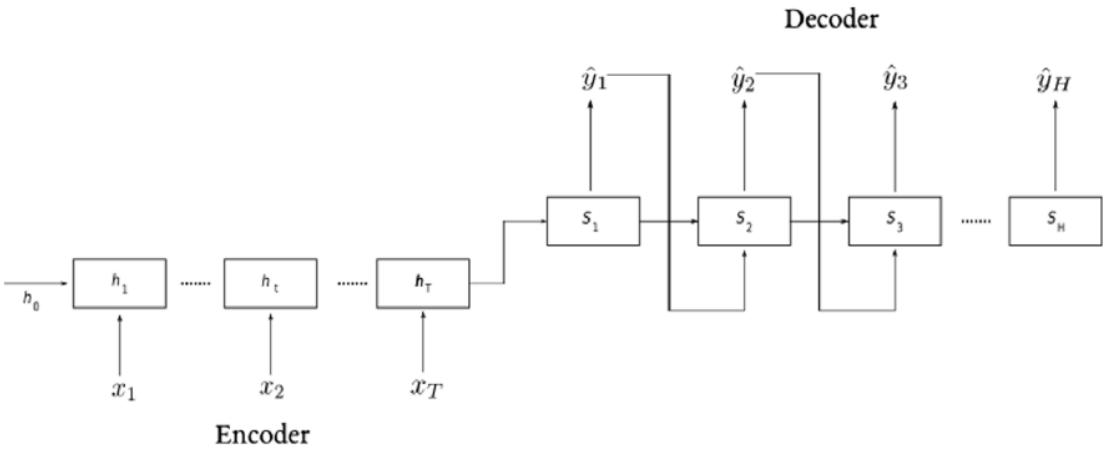

Fig 13. Sequence-to-sequence architecture [11]

Table 2. RNN summary

Architecture	Output component	Input component	Error computation
Stacked	Dense layer	Moving Window	Accumulated error
Sequence-to-sequence	Decoder	Without moving window	Last step error

Usually Sigmoid and Tanh (explained above) are the activation functions used hidden layers in RNN.

4.5. ERROR PERFORMANCE INDICATORS

In order to determine the performance of the forecasting the following error indicators between the actual values and forecasting will be calculated.

- ***Root Mean Square Error (RMSE):*** Is the square root of the average of squared errors. The effect of each error on RMSE is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSE.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(\widehat{y_t} - y_t)^2}{T}} \tag{4.14}$$

- ***Mean Squared Error (MSE):*** The mean squared error (MSE) tells you how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the 'errors') and squaring them.

$$MSE = \frac{\sum_{t=1}^{T}(\widehat{y_t} - y_t)^2}{T} \tag{4.15}$$

- ***Mean Absolute Error (MAE):*** It's the mean of the absolute errors between the forecast and the actual values.

$$MAE = \frac{\left|\sum_{t=1}^{T}(\widehat{y_t} - y_t)\right|}{T} \tag{4.16}$$

- ***Mean Percentage Absolute Error (MAPE):*** is the mean or average of the absolute percentage errors of forecasts. Error is defined as actual or observed value minus the forecasted value. Percentage errors are summed, and a positive percentage is given.

$$MAPE = 100 \cdot \frac{\left|\frac{\sum_{t=1}^{T}(\widehat{y_t} - y_t)}{y_t}\right|}{T} \tag{4.17}$$

5. ALGORITHM DEVELOPMENT

5.1.1. DATA SET STRUCTURE

Data has been taken on 5567 Households of UK Power Network, during the period from November 2011 to February 2014 at 30-minute intervals. Of these 5567 values there are 1100 customers on dynamic tariffs (Low, Normal, High) and 4467 on constant tariffs.

The database contains specific energy consumption information, in kWh per half hour, a household identifier, the date and time of data collection.

Step 1: Data Cleaning

The First step consists of preparing data for analysis by removing or modifying data that is incorrect. Data frame is created using pandas, dates and times column is set to index and transformed to *datetime* object and energy values are set to type *float64.*

tstp	LCLid	energy(kWh/hh)
2012-10-12 00:30:00	MAC000002	0.000
2012-10-12 01:00:00	MAC000002	0.000
2012-10-12 01:30:00	MAC000002	0.000
2012-10-12 02:00:00	MAC000002	0.000
2012-10-12 02:30:00	MAC000002	0.000
...	...	...
2014-02-27 22:00:00	MAC005492	0.182
2014-02-27 22:30:00	MAC005492	0.122
2014-02-27 23:00:00	MAC005492	0.140
2014-02-27 23:30:00	MAC005492	0.192
2014-02-28 00:00:00	MAC005492	0.088

1222620 rows × 2 columns

Fig 14. Step 1 Dataframe.

Step 2: Reduction of dataset

Since there is a big amount of Data, it is essential to reduce the dimensionality of the dataset in order to study some properties such as stationarity, autocorrelation and seasonality. The Data set will be reduced to smaller periods of time in order to minimizethe compilation time. Next step consists in selecting a range of Day – Time (initial date – final date)to focus in order to study properties mentioned before indicating dates of the period that wants to be analysed ; a data frame is created with 4 columns (*Cumulative Sum, Max Load, Mean Load, and Number of smartmete–s* - calculated for each date and time) and

DayTime set to index. These 4 parameters are used for forecasting in later sections.

	DayTime	Cumulative Sum	Mean Load	Max Load	Smart Meters	Cumulative Sum Corrected
0	2013-01-01 00:00:00	23.360	0.486667	1.646	48	23.360
1	2013-01-01 00:30:00	19.959	0.415813	1.580	48	19.959
2	2013-01-01 01:00:00	19.833	0.413188	1.760	48	19.833
3	2013-01-01 01:30:00	17.434	0.363208	1.424	48	17.434
4	2013-01-01 02:00:00	17.102	0.356292	1.952	48	17.102
...	...	...	...	...	...	...
667	2013-01-14 21:30:00	32.918	0.685792	3.805	48	32.918
668	2013-01-14 22:00:00	31.231	0.650646	2.961	48	31.231
669	2013-01-14 22:30:00	31.641	0.659187	3.418	48	31.641
670	2013-01-14 23:00:00	28.187	0.587229	4.244	48	28.187
671	2013-01-14 23:30:00	24.422	0.508792	3.145	48	24.422

672 rows × 6 columns

Fig 15. Example of a Dataframe from 01-01-2013 00:00:00 to 14-01-2013 23:30:00

5.1.2. PREPARATION OF SCENARIOS TO FORECAST

In order to test the algorithms, it is essential to create a test scenario for developing the long-term forecast; since the season with the highest peak demand (due low temperatures) is Winter, a '*Winter scenario'* will be carried out.

The Forecast will be made using 'January 2012 – January 2013' for model training and 'January 2014' for model validation.

The original dataset contains the energy consumption of each smart meter at specific time but instead of focusing in several smart meters, it has been decided to work with the aggregated demand, meaning that, all residential smart meters have been summed (same time intervals) in order to model a single aggregated load curve. Some of identified issues in this process are the homogeneity of smart meter data. It is common to find metering errors of energy consumptions due external factors (non-technical losses, SM failures). In this case, the number of smart meters (load curves) were not consistent over all thedataset, therefore a correction has been applied to each point adding the average load to reach the same number of smart meters for each period of time (January 2012 – 2013 – 2014).

5.1.3. STATISTICAL METHODS FORECASTING

Long term forecasting with *ARIMA* and *SARIMA* has been developed, during the following sections *Box & Jenkins Methodology* is applied for parameter definition and Long-Term forecasting results are presented.

5.1.3.1. ARIMA parameters - Box & Jenkins Methodology

In order to define ARIMA parameters *p, d and q* has been decided to focus in 1 month analysis, decision has been made to analyse a smaller period and has been assumed that the behaviour of this period represents in a general way the behaviour of the data in the whole data set. Month decided for this analysis is January 2013.

Step 1: Identifying stationary data.

To identify whether the data are stationary time series of the *'Cumulative Sum Corrected' (Aggregated Load Demand) and Mean Load vs 'Dates'* is plotted. In this way its easy to identify if any transformation of the data is necessary, for instance converting non-stationary data into stationary. Once the data is plotted, linear regression equation is calculated, which its slope indicates whether the data is stationary or not; the closer the slope is to 0 the more stationary the data is, for a final check, the Augmented Dickey Fuller test is run through data.

Figures 16 and 17 show the graphs of our data for the month of study (January 2013) in which can be seen that there is no significant trend and appears to be stationary; Linear regression equations have been calculated for checking the previous hypothesis:

- *Cumulative Sum Corrected vs Dates*: $y = 0.09x - 4061.9$
- *Mean Load vs Dates:* $y = 0.002x - 84.623$

As can be seen, the values of the slope in the linear regression equation are close to zero, so the data is probably stationary and would not need any transformation. Furthermore, can be identified that there are some peaks every 48 values that represent 1 day (resolution each 30 min), so seasonality in the data can be expected.

Augmented Dickey Fuller test gives the following results:

```
ADF Statistic: -11.895104
p-value: 0.000000
Critical Values:
        1%: -3.435
        5%: -2.864
        10%: -2.568
```

With that information, can be confirmed that data is stationary so *d* parameter has will be set to 0, and dataset does not need any transformation for the *ARIMA* execution.

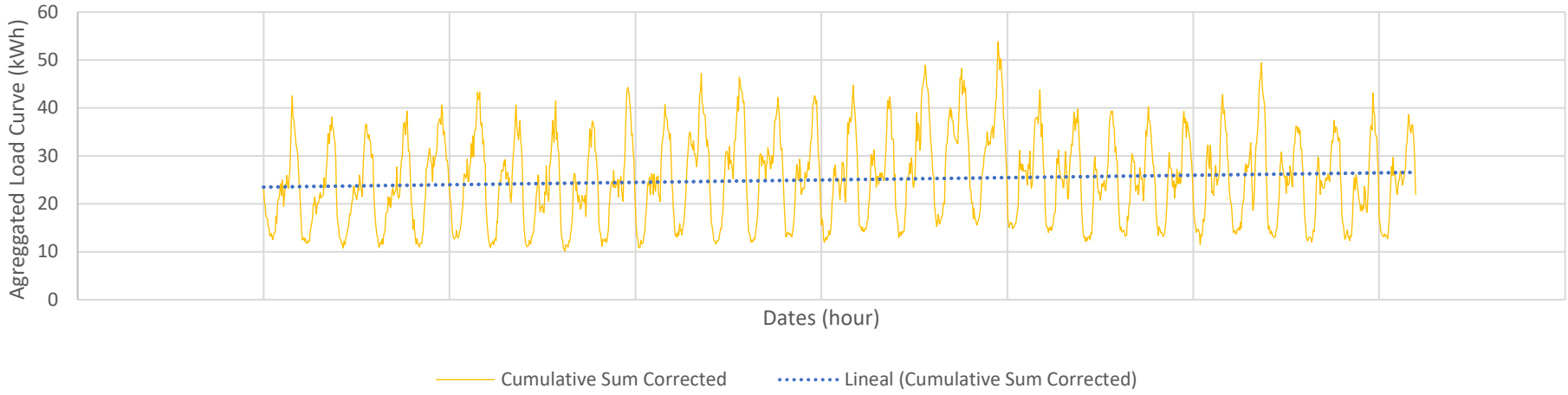

Figure 17. Aggregated Load Curve January 2013

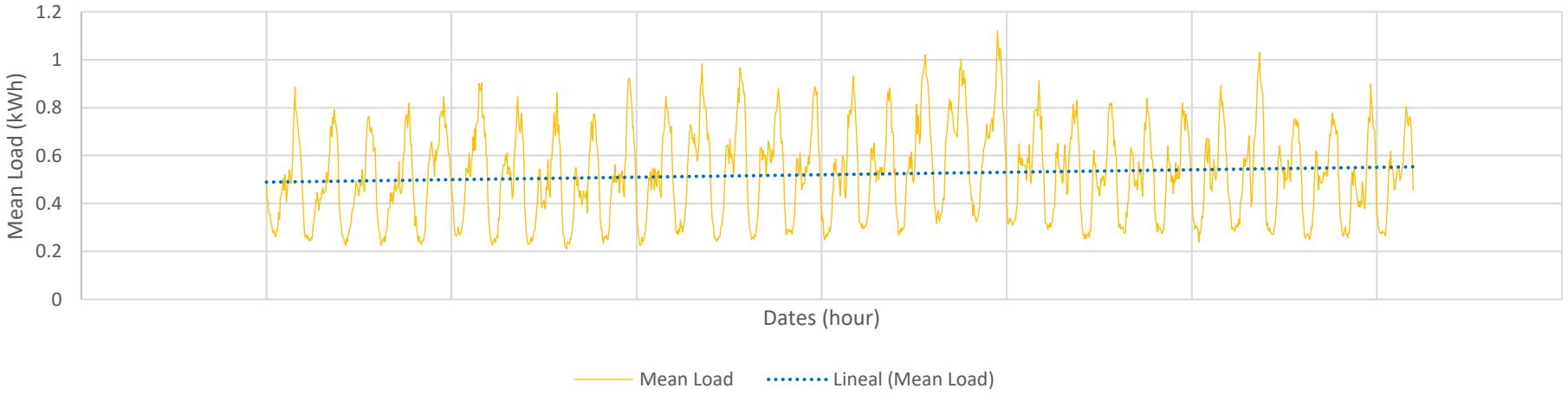

Fig 18. Mean Load January 2013

Step 2: Determine p and q through Autocorrelation: ACF and PACF.

Through this step the ACF and PACF has been developed for the *data frame* created in previous steps.

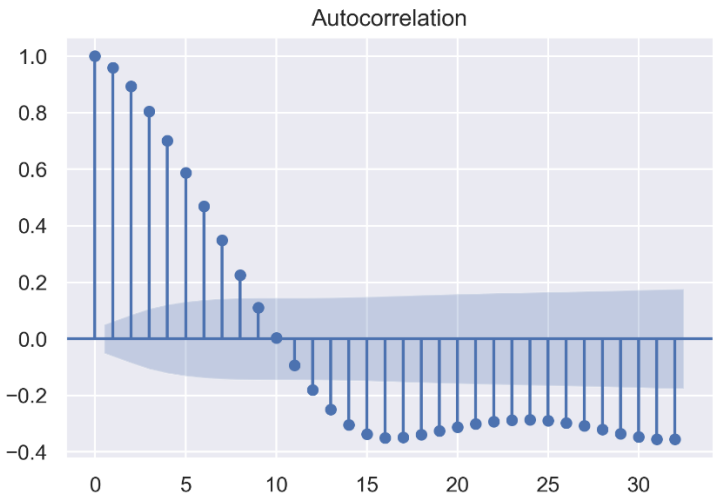

Fig 19. Autcorrelation Function Plot

ACF gives us valuable information about the behaviour of our time series dataset. Also, it helps us to define the order of the parameters *p* and *q*. These values are taken as an initial indicative and must be used as a starting point of the iterations. Analysing Figure 19 (ACF) we conclude the following statements:

- Strong decay starts after a few lags (2 or 3) that means dataset might be forecast with both parts Auto Regressive and Moving Average Parameters
- Autocorrelation enters to non-significative zone at 8^{th} lag.
- After 11^{th} lag there appears to be some significant values, but they appear later and seem to be part of noise and show us periodicity in our data set. Must be taken in care later for Seasonality.

ACF analysis concludes that parameter *q* has an order of 8 because it enters to non-significance band at 9^{th} lag. Order of the *p* parameter (AR part) must be defined after analysing PACF plot.

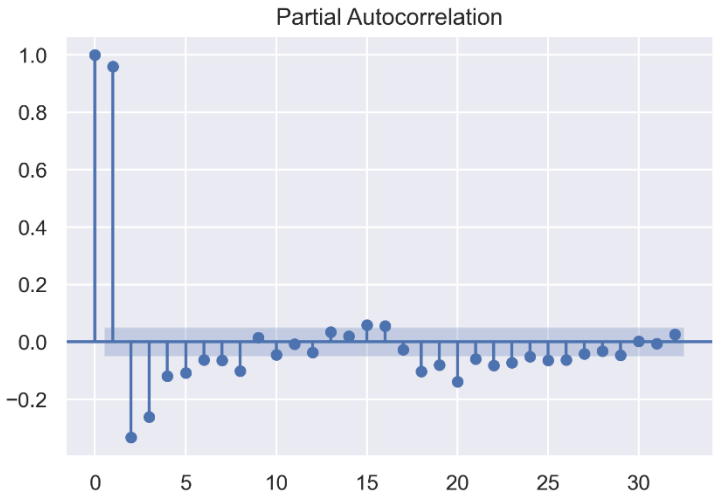

Fig 20. Partial Autocorrelation Function Plot

PACF gives the information to define exactly the order of *p* and *q,* in this case its concluded that after 5th lag there is no values greater than *0,05* in exception of values from 18th to 20th; however, values from 18th to 20th lag might be noise of the data set because they appear after a big number of values that are between the significancy band and are not considered. Furthermore, applying rules and characteristics explained in section. 4.2.1 has been decided that parameter *p* have an order of 6 (5 + 1).

P, d, and q values defined before will be used in the project as hypothesis for appliance of *ARIMA* model.

Step 3: Check Accuracy of the model.

In this step accuracy of the model defined in previous sections (*p=8, d=0, q=6)* is checked, to do so, error performance indicators (RMSE, MAPE, MSE and MAE), Aikaike's Information Criterion (AIC) and Bayesian Information Criterion (BIC) are calculated. A study has been made varying values of *p* and *q* in a range of ± 2, so different options has been studied.

One month time compilation (January 2013) is very high, and volume of data must be reduced for studying the performance of different *ARIMA* parameters. So, it has been decided to focus in the first 2 weeks of January 2013. From these two weeks it has been decided that the first 10 days (01/01 - 10/01) will be used for training the *ARIMA* model and the 4 resting days (11/01 - 14/01) will be used for model validation.

Table 3 Study of different ARIMA Distribution performance indicators

ARIMA COEFS	RMSE	MAPE	MSE	MAE	AIC	BIC
(6, 0, 5)	2,35	8,85	5,53	1,76	3000,93	3059,55
(6, 0, 6)	2,33	6,73	5,42	1,75	3011,98	3075,11
(6, 0, 7)	2,33	6,60	5,42	1,71	3009,63	3077,26
(7, 0, 4)	2,29	6,53	5,24	1,70	3005,48	3064,09
(7, 0, 5)	2,31	6,78	5,35	1,73	3003,01	3066,14
(7, 0, 6)	2,38	6,76	5,64	1,77	3004,45	3072,08
(7, 0, 7)	2,35	6.66	5,51	1,74	3012,19	3084,34
(8, 0, 6)	2,41	6.91	5,80	1,80	3009,46	3081,61
(8, 0, 7)	2,36	6,73	5,55	1,75	3015,73	3092,38
(9, 0, 7)	2,335	6,71	5,43	1,74	3013,42	3094,58

Red colour numbers represent the minimum values of each performance indicator; ARIMA coefficients with the lowest error indicator is *(7, 0, 4).* In the case of AIC and BIC the lowest value is *3000,93 (6, 0, 5)* and *3059,55 (6, 0, 5)* very similar to the value of *(7, 0, 4)* distribution *3005,48* and *3064,09*. So it's been decided that ARIMA coefficient that will be used for future forecasting will be *(7, 0, 4).*

5.1.3.2. ARIMA FORECASTING

'Statsmodels' is the Python3 package that includes ARIMA model used for the forecasting in this project. The parameters for the long term ARIMA forecasting are (7, 0, 4) defined in the previous sections; run time for the case study applying ARIMA with 2 months of training (January 2012 – 2013) and performing the 1-month forecasting (January 2014) is *11574,4* seconds which corresponds to approximately *3,5* hours.

ERROR PERFORMANCE INDICATORS:

ARIMA COEFS	RMSE	MAE	MSE	MAPE
(7, 0, 4)	*2,98*	*2,30*	*8,91*	*9,70 %*

ARIMA MODEL SUMMARY:

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                          y   No. Observations:                 4462
Model:                 ARIMA(7, 0, 4)   Log Likelihood              -15148.245
Date:                Mon, 24 May 2021   AIC                          30322.491
Time:                        19:27:04   BIC                          30405.734
Sample:                             0   HQIC                         30351.835
                                 - 4462
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         20.5319      0.884     23.220      0.000      18.799      22.265
ar.L1          0.2355      0.705      0.334      0.738      -1.146       1.618
ar.L2          0.4857      0.833      0.583      0.560      -1.147       2.119
ar.L3          0.1602      0.491      0.326      0.744      -0.803       1.123
ar.L4          0.0362      0.421      0.086      0.931      -0.788       0.861
ar.L5         -0.0119      0.174     -0.068      0.945      -0.353       0.329
ar.L6         -0.0787      0.024     -3.326      0.001      -0.125      -0.032
ar.L7         -0.0602      0.049     -1.234      0.217      -0.156       0.035
ma.L1          0.5402      0.706      0.766      0.444      -0.843       1.923
ma.L2         -0.0808      0.338     -0.239      0.811      -0.743       0.581
ma.L3         -0.1270      0.267     -0.475      0.635      -0.651       0.397
ma.L4         -0.0330      0.206     -0.160      0.873      -0.436       0.370
sigma2        52.0131      0.502    103.543      0.000      51.029      52.998
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):             59399.48
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):               0.06   Skew:                             1.76
Prob(H) (two-sided):                  0.00   Kurtosis:                        20.52
===================================================================================
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
TIME ARIMA 11574.403470993042
```

Autoregressive constants (L1, L2, L3, L4, L5 and L6) and Moving Average constants (L1, L2, L3 and L4) are non-significative (P-value > 0.05) and ARIMA model can be optimised to obtain more accurate results. However, *error performance indicators* are low and acceptable since a long-term forecast does not need an extremely high accuracy but must be able to foresee the highest demand peaks and changes in consumption. As shown below, this model can foresee these demand pics and shapes itself to the energy demand. Furthermore, error is stationary and is considered white noise because p-value in Dickey Fuller test is 0,00:

```
ADF Statistic: -13.157155
p-value: 0.000000
Critical Values:
        1%: -3.435
        5%: -2.864
        10%: -2.568
```

The following figures show the results obtained from Week 1 and Week 2 of January 2014:

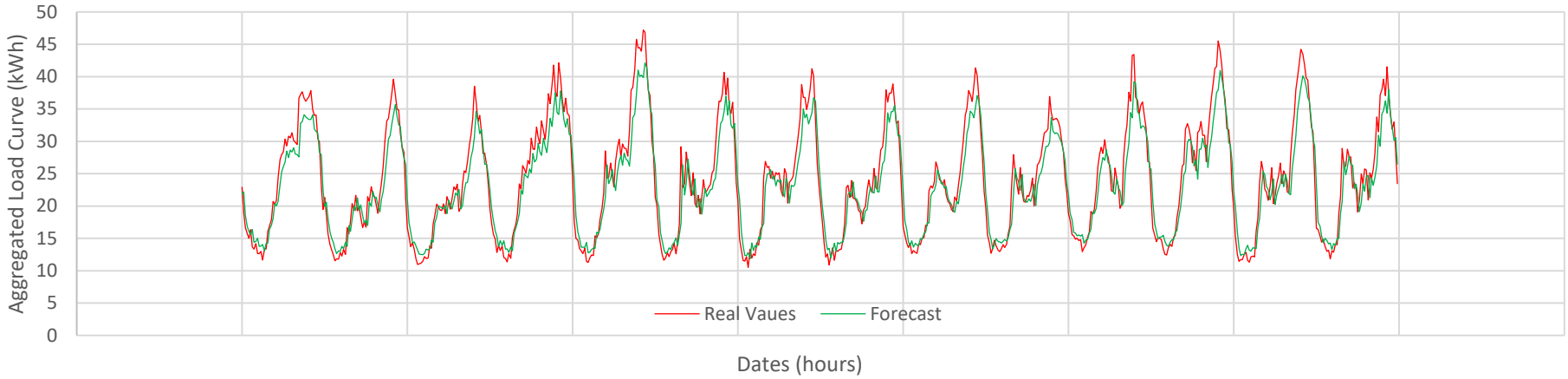

Fig 21. ARIMA Forecast Week 1 – 2 January 2014

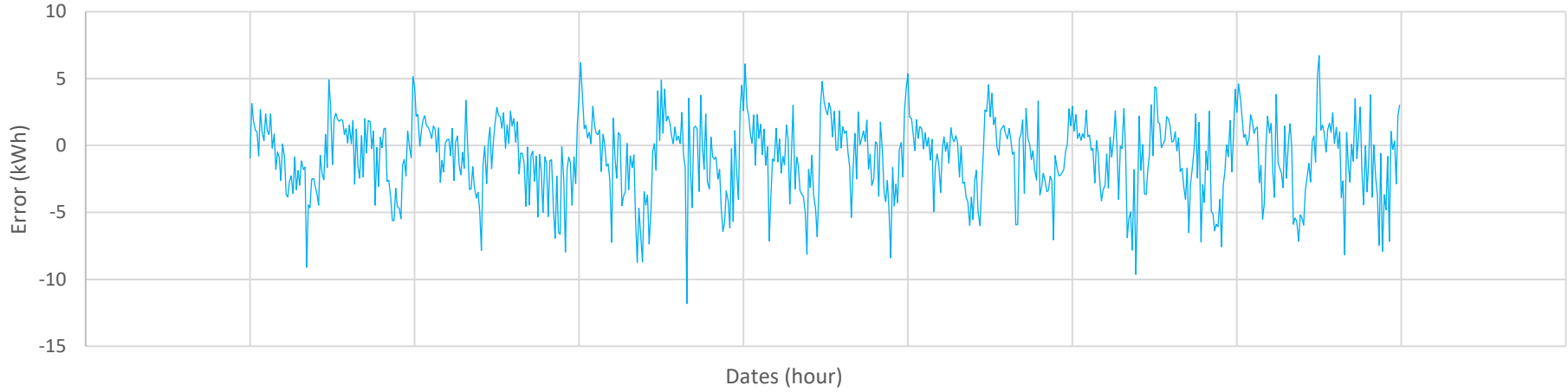

Fig 22. ERROR ARIMA Forecast Week 1 – 2 January 2014

5.1.3.3. SARIMA – (P, D, Q) s PARAMETER DEFINITION

For an accurate forecast is needed to study seasonality in the dataset. Seasonality can be studied with the results of *ACF* and *PACF* plots in a longer period. However, in ARIMA Box & Jenkins methodology was focused in a period of 2 weeks to estimate parameters p, d and q; those parameters will be used for SARIMA forecast as *p, d, q* coming from ARIMA. Period taken for seasonality analysis is 2 months, and for ACF and PACF has been taken 60 lags, study is performed to the Aggregated Load Demand.

One of the main troubles when performing long-term forecasting with *SARIMA* is that is difficult converge for its complex equations and high number of parameters; the resources in this project are limited in terms of computer calculation capacity so its been decided that for SARIMA forecasting to reduce the data resolution in the dataset from 30 minutes to 2 hours.

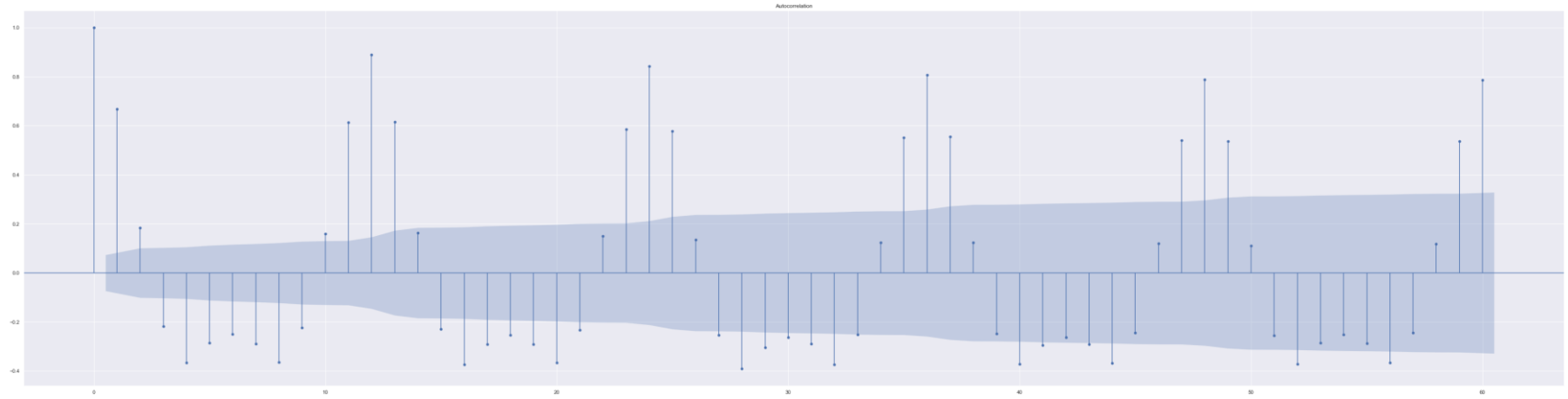

Fig 23. Autocorrelation Plot J-F 2013

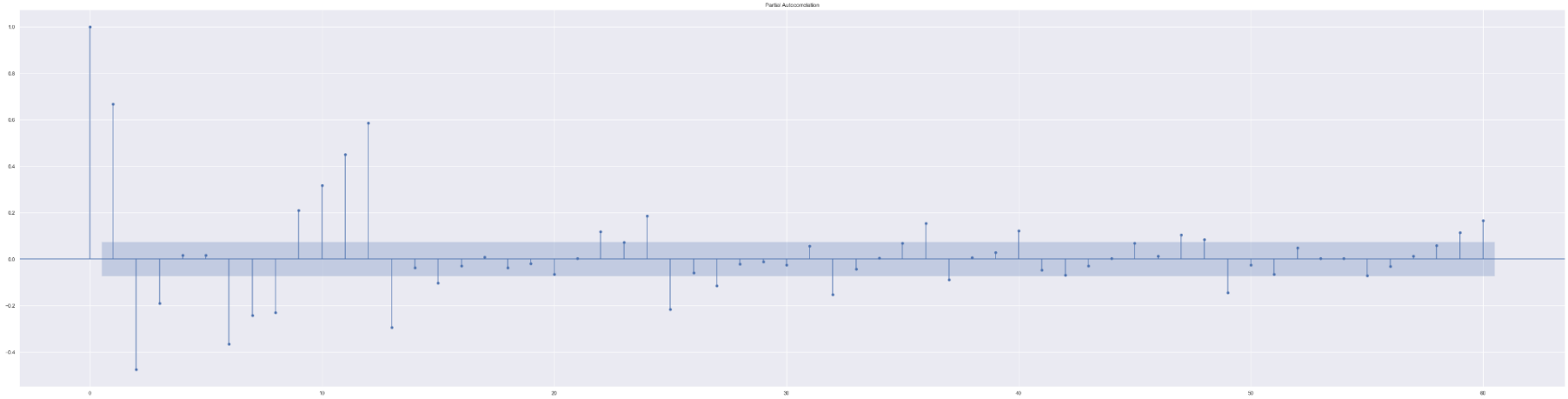

Fig 24. Partial Auto Correlation Plot J-F 2013

As shown in Autocorrelation plot there is evidence of seasonality each 12 lags, that corresponds a period of 1 day, so there is a measure 2 hours, so the parameter *s* will take value of 12.

To determine the auto regression, moving average and integration parameters, a 12 -value differencing is applied to the initial data. Once the differencing is done, the study is repeated using the Box & Jenkins technique to determine the P, D and Q values of the seasonality.

Auto-Regressive Parameter (P) and Moving Average Parameter (Q): In order to determine both parameters it has been analysed the ACF and PACF of the data with the differentiation applied.

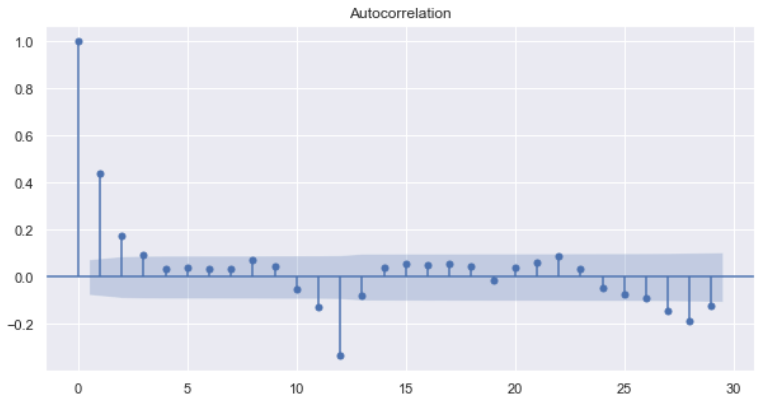

Fig 25. Autocorrelation Plot

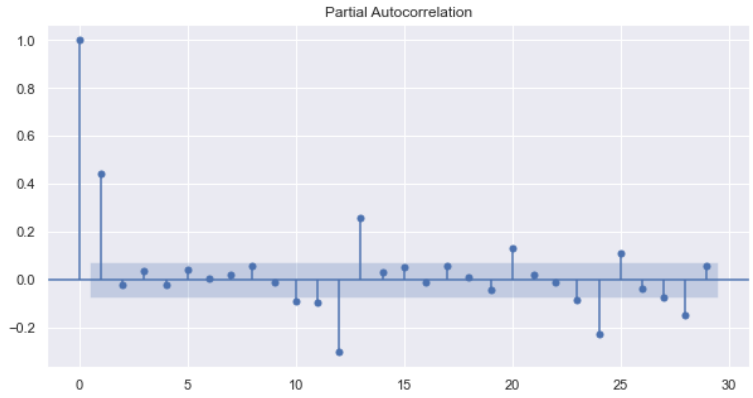

Fig 16. Partial Autocorrelation Plot

- Autocorrelation Function: Very strong decay and enters to non-significance band at 3th lag. That means $Q = 2$.
- Partial Autocorrelation Function: After 1st lag there is no values outside the non-significance band so $P = 2$.

Integration parameter (D): Figure 27 shows the graph with the values of *'Aggregated Load Demand' vs 'Dates'*. At first glance it seems that the data is stationary, and it is checked with the linear regression equation and with the Dickey fuller test:

- *Linear Regression equation: y = -0.0014x + 59.524*
- *Augmented Dickey Fuller Test:*

```
ADF Statistic: -4.858278
p-value: 0.000042
Critical Values:
        1%: -3.440
        5%: -2.866
        10%: -2.569
```

The slope of linear regression equation is remarkably close to zero and Augmented Dickey Fuller Test with a p-value of 0.000042 confirms that data is stationary so D parameter will be 0.

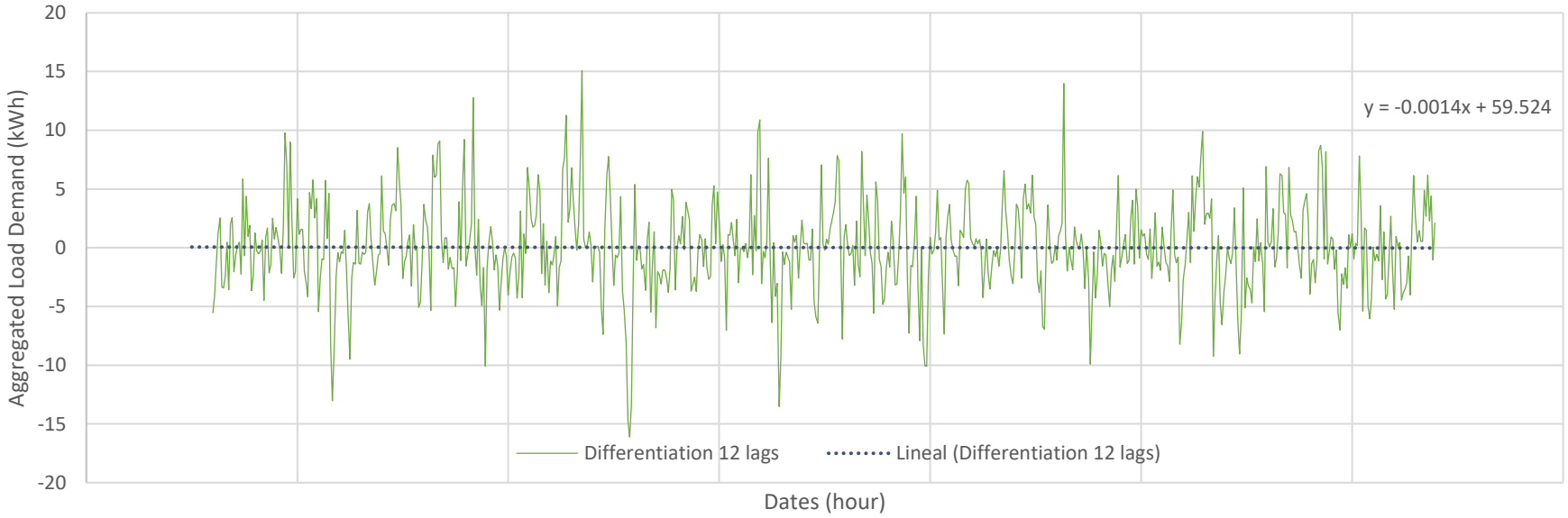

Figure 27. Differentiation 12 lags Stationarity Study January - February 2013

5.1.3.4. SARIMA FORECASTING

'*Statsmodels'* package in python3 is the package used for developing the forecast in this project. Model *SARIMA (7, 0, 4) x (2, 0, 2)$_{12}$* defined in the previous sections is used for forecasting, the execution time of this model with data of January 2012 – 2013 as a training and January 2014 for testing is *4408,93 s* what is approximately 1 hour and 13 minutes. We must consider that the amount of data has been reduced to 2 hours instead of 30 minutes that is why the execution time is lower than *ARIMA*.

ERROR PERFORMANCE INDICATORS

SARIMA COEFS	RMSE	MAE	MSE	MAPE
(7, 0, 4) x (2, 0, 2)$_{12}$	3.00	2.37	8.99	10.50%

SARIMA MODEL SUMMARY

SARIMAX Results

Dep. Variable:	y	No. Observations:	1115
Model:	ARIMA(7, 0, 4)x(2, 0, [1, 2], 12)	Log Likelihood	-4024.985
Date:	Fri, 18 Jun 2021	AIC	8083.971
Time:	22:43:11	BIC	8169.253
Sample:	0 - 1115	HQIC	8116.212
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	20.3979	5.010	4.071	0.000	10.578	30.218
ar.L1	0.4353	0.203	2.146	0.032	0.038	0.833
ar.L2	0.5723	0.062	9.293	0.000	0.452	0.693
ar.L3	0.4975	0.083	5.980	0.000	0.334	0.661
ar.L4	-0.7780	0.155	-5.026	0.000	-1.081	-0.475
ar.L5	0.2273	0.041	5.594	0.000	0.148	0.307
ar.L6	-0.0509	0.042	-1.217	0.224	-0.133	0.031
ar.L7	0.0400	0.026	1.516	0.130	-0.012	0.092
ma.L1	-0.1696	0.204	-0.833	0.405	-0.569	0.230
ma.L2	-0.6615	0.110	-5.992	0.000	-0.878	-0.445
ma.L3	-0.6337	0.065	-9.790	0.000	-0.761	-0.507
ma.L4	0.6473	0.180	3.592	0.000	0.294	1.001
ar.S.L12	0.2770	0.561	0.494	0.622	-0.823	1.377
ar.S.L24	0.6978	0.552	1.265	0.206	-0.384	1.779
ma.S.L12	-0.1393	0.553	-0.252	0.801	-1.223	0.945
ma.S.L24	-0.6210	0.469	-1.324	0.185	-1.540	0.298
sigma2	78.0490	1.804	43.256	0.000	74.513	81.585

Ljung-Box (L1) (Q):	0.02	Jarque-Bera (JB):	3395.47
Prob(Q):	0.88	Prob(JB):	0.00
Heteroskedasticity (H):	0.04	Skew:	1.39
Prob(H) (two-sided):	0.00	Kurtosis:	11.08

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

There are some constants of the model that are non – significative (*p-value > 0,05),* the algorithm can be optimised and should have similar results with less constants that will

reduce the execution time. However, *RMSE* of the model equals to *3,00*, so it is considered a good model for long-term demand forecasting. As shown below, the model can foresee demand pics and shapes itself to the energy demand. Furthermore, error is stationary and is considered white noise because p-value in Dickey Fuller test is *0,000017*:

```
ADF Statistic: -5.061181
p-value: 0.000017
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
```

The following figures show the results obtained from Week 1 and Week 2 of January 2014:

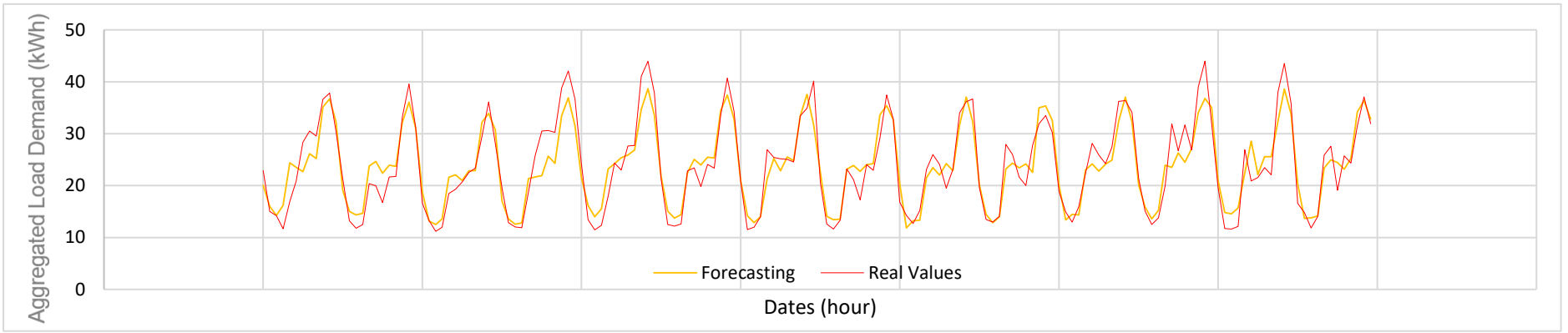

Fig 28. SARIMA Forecast Week 1 - 2 January 2014

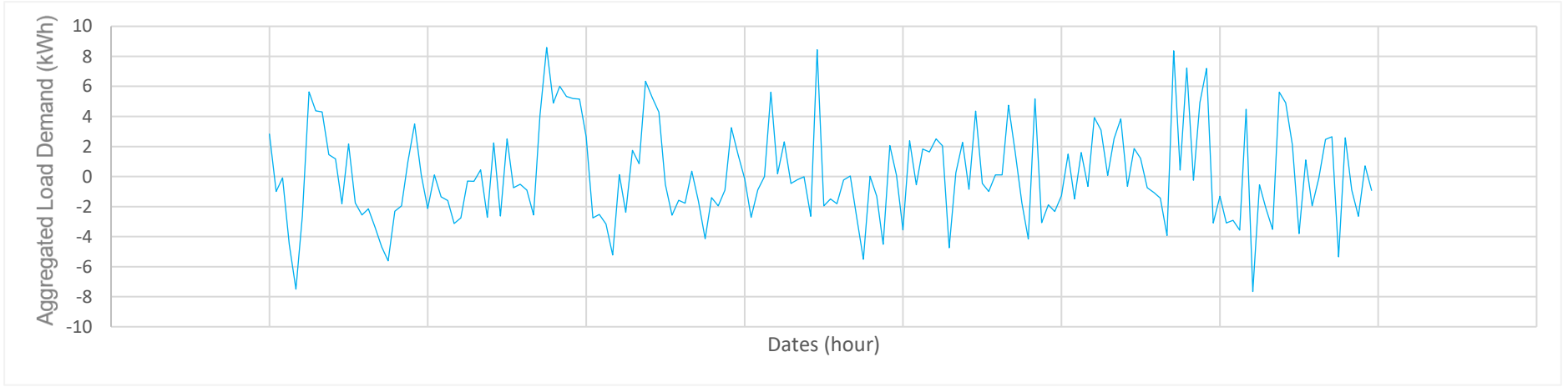

Fig 29. ERROR SARIMA Forecast Week 1 - 2 January 2014

5.1.4. NEURAL NETWORKS FORECASTING

For the development of long-term energy demand forecasting with neural networks, an MLP model and an RNN have been developed. The development of both neural networks is based on the aforementioned data.

Neural networks can offer a lot of help when data scientists deal with more complex but still very common problems such as, time series forecasting. Many different studies have been published during last few years about performance of NN depending on the type of cell, activation functions and optimization algorithms applied to short term and long-term demand forecasting. Recurrent neural networks — specifically long short-term memory (LSTM) networks and gated recurrent units (GRUs) — are good at extracting patterns in input data that span over relatively long sequences[16].

In order to test different NNs, three *Neural Networks* have been developed. Two *RNN* based on *LSTM and a simple RNN* and *1 MLP.* The main purpose consists in comparing the results of a similar structure between both *RNN* and *MLP* and study the performance of the *NN* depending on the activation function, type of cell and optimizer.

In the experiment the number of steps *'n'* (*n* values used to determine the *'n+1'* value) has been determined to *20* and a learning rate of *0,005*.

5.1.4.1. RECURRENT NEURAL NETWORK – DEFINITION.

Development of *Recurrent Neural Networks* has been based on the study of *Long Term Load Forecasting with Hourly predictions based on LSTM* developed in the Delhi Technological University[17]. This study proposes a 3-layer with 15 *LSTM cells* tuned by an exhaustive search approach for hyper parameters applying a learning rate of *0,01* using the activation function '*ReLu'*.

For the purpose of this work, two types of recurrent neural network have been developed, in both cases the *Keras* library within the *tensorflow* framework have been used, applying a reduction of cells between the hidden layers without *Dropout* and one *Dense Layer* as output layer:

1. ***Simple RNN Cells:*** Made up of 1 input layer (*400 cells*), two hidden layers *(300 and 100 cells)* and 1 output layer *(Dense layer).*

2. ***LSTM Cells:*** Made up of 1 input layer (*400 cells*), two hidden layers *(300 and 100 cells)* and 1 output layer (*Dense layer*).

DATA PREPARATION

The initial data frame has a resolution of 30 minutes, for the simplification of the problem and to be able to correctly run the neural network models (*Simple RNN and LSTM*) the data has been organized in a matrix form with a resolution of 1 hour, thus obtaining a row per day - month - year and a column for each hour of the same day.

Out[2]:

	loads													
hours	0	1	2	3	4	5	6	7	8	9	...	14	15	16
date														
2012-01-01	24.432000	12.144000	44.640000	4.944000	5.328000	3.792000	3.120000	4.944000	3.696000	3.024000	...	17.088000	52.560000	32.8800
2012-01-02	38.832000	30.096000	4.608000	2.880000	2.544000	4.896000	3.312000	4.944000	6.240000	6.096000	...	14.592000	2.352000	1.9680
2012-01-03	23.040000	6.528000	4.656000	5.136000	6.384000	4.416000	2.640000	6.288000	3.792000	17.568000	...	1.824000	2.016000	4.3680
2012-01-04	12.672000	21.408000	5.472000	4.464000	4.944000	5.904000	5.424000	5.136000	5.184000	2.880000	...	4.704000	2.736000	3.7440
2012-01-05	64.224005	44.496000	5.952000	4.944000	4.320000	4.944000	5.952000	5.712000	4.656000	5.856000	...	19.872000	3.936000	4.5600
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2014-01-27	19.596279	14.260465	13.892093	12.617302	11.995535	12.704372	13.309395	18.275721	29.091349	26.780651	...	24.891907	24.564837	23.3023
2014-01-28	17.697488	14.262698	13.706791	12.820465	11.858233	13.111814	14.384372	21.566512	27.120000	24.617302	...	21.395721	23.953116	24.8550
2014-01-29	15.368930	11.739907	11.163907	12.031256	10.722977	11.113674	12.832744	18.606140	28.236279	27.506233	...	25.045954	23.128186	19.4266
2014-01-30	15.464930	12.535814	12.975628	11.359256	12.490047	12.392930	13.934512	18.090419	27.024000	23.841488	...	23.632744	22.184930	22.5298
2014-01-31	18.049116	14.129860	12.506791	12.116093	12.244465	12.162977	14.078512	18.574884	25.724651	23.034419	...	25.424372	19.062698	20.2537

93 rows × 24 columns

Fig 30. Matrix form of data - Aggregated Load Demand (kWh)

Next step consists in separate data between *train data (January 2012 – January 2013)* and *test data (January 2014)*, once data is separated its normalized to improve neural network performance using *MinMaxScaler()* from *sklearn.*

As mentioned, it has been decided to use the previous 20 values to determine the 21st, train and test data frames are splitted and transformed to sequences. A requirement of *LSTM and SimpleRNN cells* is that the input data is a 3D array *'(batch_size, time_steps, seq_len)'.* In this case *X* is the input of the Neural Network and *Y* is the data validation:

```
Training Set X (2, 25, 24) and Y (2, 24)
Dimensions of the test set with training data needed for predictions:
(91, 24)
Testing Set X (31, 25, 24) and Y (31, 24)
```

Methodology of study

This section describes the methodology used to study the performance of the *RNNs*; in this case, the structure of the neural network is already defined and the parameters that will vary are the activation function and the learning algorithm (optimization). In this algorithm for all the examples a learning ratio of *0,005* is applied.

Three activation functions (*ReLu, Tanh and Sigmoid*) and three learning algorithms (*Adam, AdaGrad and RMSprop*) are considered and a maximum of 2000 epochs are

permitted. The performance is measured considering the errors *(RMSE, MAE, MAPE, MSE)* of each model applied in each type of *neural network*, the number of epochs without overfitting and the total execution time.

To avoid overfitting an early stopping function has been applied with a patience of 20 values, that means that if in 20 values the *mean squared error* is stationary it stops, apart from avoiding overfitting it improves the total time execution.

Furthermore, graphical representation of the values has been evaluated to avoid repetitive models.

RNN FORECASTING RESULTS

Table 4 shows the values of the performance indicators for the *Simple RNNs.*

		RMSE	MAE	MSE	MAPE	EXEC TIME (seconds)	EPOCHS
RNN SIMPLE	Adam 'relu' 20	8.23	6.70	67.71	13.59%	38.85	516
	Adam 'tanh' 20	8.16	6.58	66.64	10.53%	47.76	724
	Adam 'sigmoid' 20	8.39	6.85	70.38	11.26%	93.12	1581
	AdaGrad 'relu' 20	7.60	6.27	57.83	14.22%	111.54	2000
	AdaGrad 'tanh' 20	8.56	6.78	73.29	15.19%	31.66	436
	AdaGrad 'Sigmoid' 20	8.15	6.56	66.47	10.48%	116.90	2000
	RMSProp 'relu' 20	7.30	6.12	53.33	13.86%	14.56	92
	RMSProp 'tanh' 20	8.45	6.68	71.33	10.89%	10.58	67
	RMSProp 'Sigmoid' 20	6.89	5.69	47.42	14.09%	10.99	70

Table 4. Simple RNN performance indicators

Green values have a normal behaviour and are considered for the comparation with other methodologies; however, white ones show a strange behaviour when are plotted and are omitted. Next figure shows the strange behaviour (all days are practically identical) of RMSProp 'Sigmoid' during the first week test:

Best results in *SimpleRNN* are performed by the learning algorithm *RMSProp* and with the activation function 'ReLu' with a *RMSE* value of *7,30*, an execution time of *14,56* seconds and 92 epochs. Next graphs show the behaviour of the green ones the first two weeks of the month:

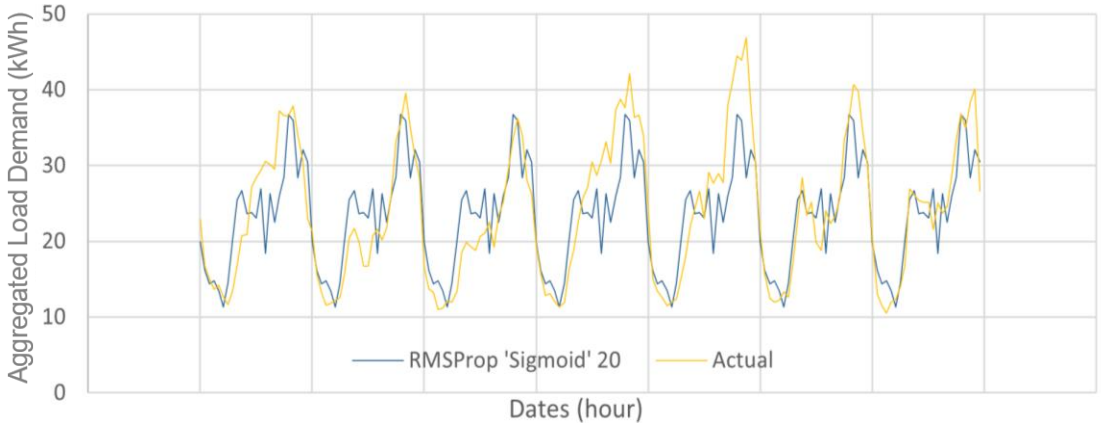

Fig 31. RMSProp Sigmoid W1

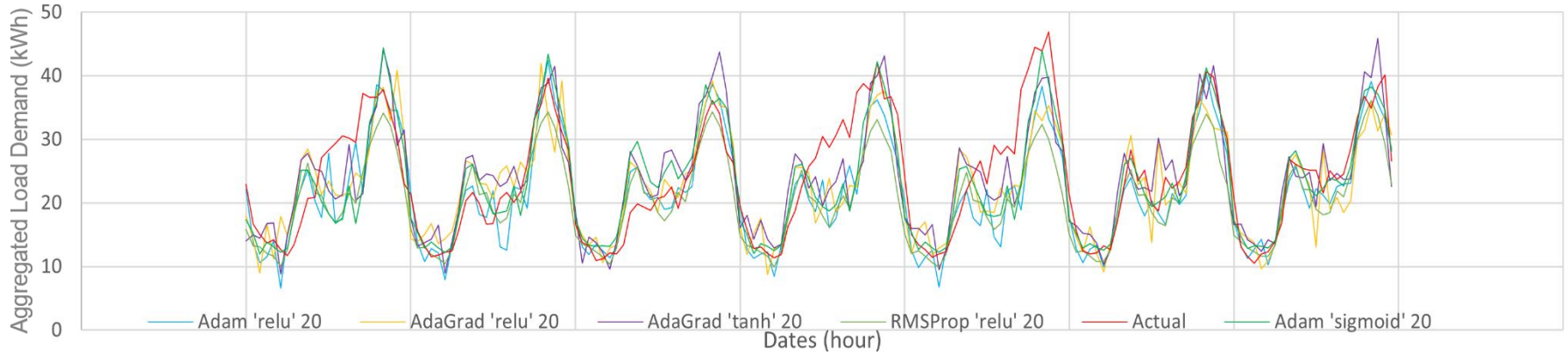

Fig 32. Simple RNN Forecast Week 1.

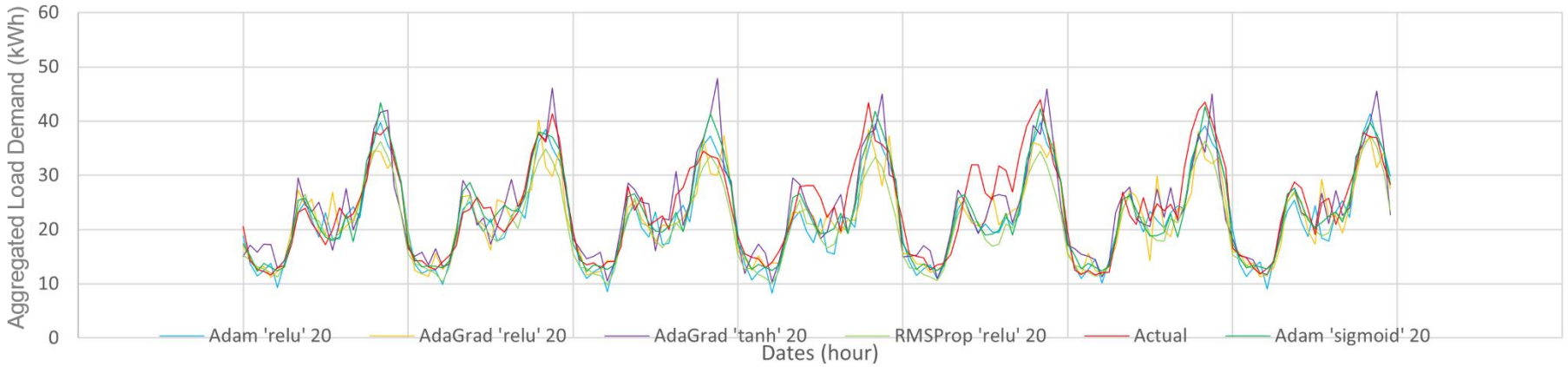

Fig 33. Simple RNN Forecast Week 2.

Table 5 shows the results of the performance indicators for *LSTM.*

		RMSE	MAE	MSE	MAPE	EXEC TIME (seconds)	EPOCHS
LSTM CELLS	Adam 'relu' 20	7.62	6.32	58.07	10.81%	80.13	527
	Adam 'tanh' 20	8.19	6.83	67.12	11.30%	103.33	638
	Adam 'sigmoid' 20	8.43	6.90	71.05	11.42%	182.25	1240
	AdaGrad 'relu' 20	7.80	6.42	60.89	10.68%	263.39	2000
	AdaGrad 'tanh' 20	7.87	6.37	61.90	10.42%	260.48	2000
	AdaGrad 'Sigmoid' 20	8.15	6.56	66.40	10.49%	86.66	658
	RMSProp 'relu' 20	NA	NA	NA	NA	NA	NA
	RMSProp 'tanh' 20	8.19	6.88	67.00	12.00%	40.82	109
	RMSProp 'Sigmoid' 20	8.64	7.05	74.71	11.84%	24.03	65

Table 5. LSTM Performance Indicators

Same analysis performed in *SimpleRNN* have been done for *LSTM*. As can be seen for *LSTM* the execution time is higher than *SimpleRNN* due to the high number of epochs performed during the test. *Adam* is the only learning algorithm that gives good results in terms of the behaviour of the model once it is plotted*; RMSprop* '*ReLu'* gave bad results it is not taken into account, so the optimization didn't perform correctly. Figure 34 shows the strange behaviour of *AdaGrad 'tanh'* during the first week:

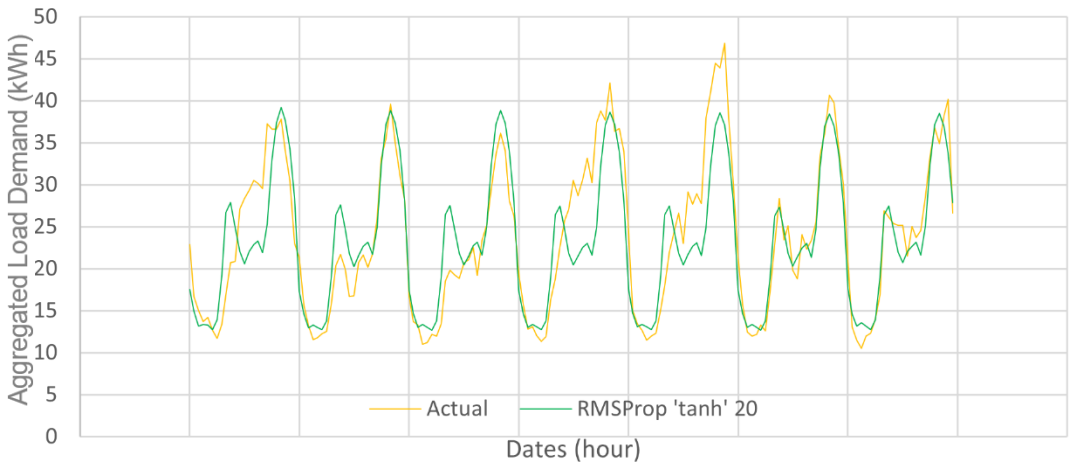

Fig 34. AdaGrad 'tanh' strange behaviour

Best results in *LSTM* are a rmse value of *7,62*, and execution time of *80,13* seconds and *527* epochs, those are given by the learning algorithm *Adam* with the activation function 'ReLu'. Next figures show the behaviour of the green ones the first two weeks of the month:

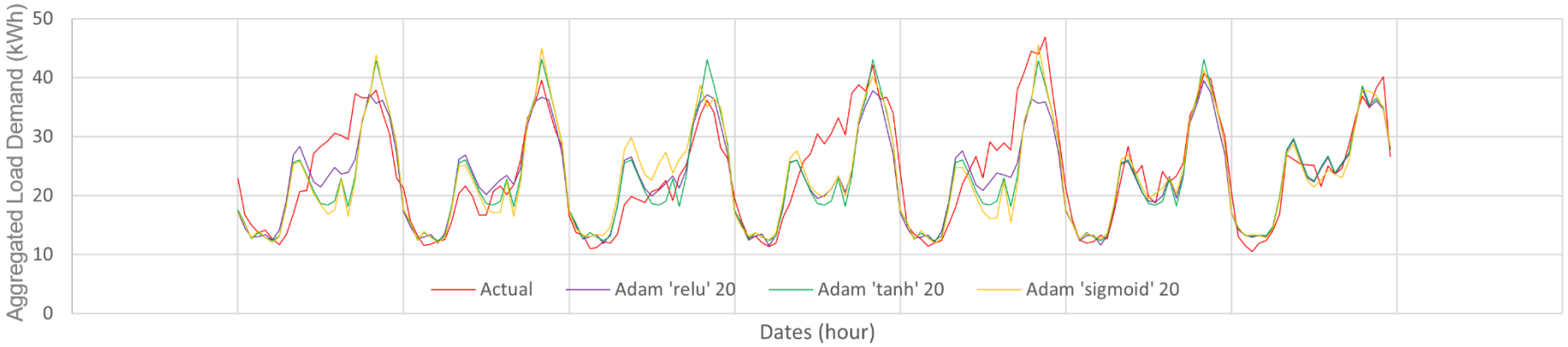

Fig 35. LSTM Normal Behaviour W2

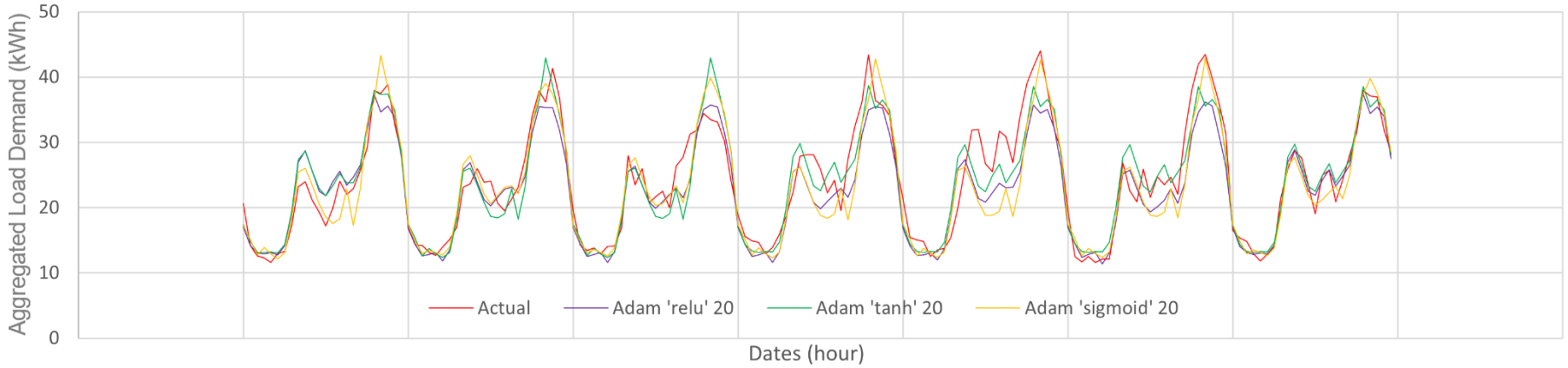

Fig 36. LSTM Normal Behaviour Week 2

5.1.4.2. MULTIPLE LAYER PERCEPTRON NEURAL NETWORK

Multiple Layer Perceptron is simpler than recurrent neural networks in its structure. One of the objectives is to compare the results obtained by the *MLP* with *RNN*, so the *RNN* previously designed has been adapted to a *MLP*. To do so, the same number of layers has been used; 1 input layer *(400 cells)*, 2 hidden layers *(300 and 100 cells)* and an output layer (Dense layer), the main difference is that all layers are fully connected, so it has been done with 3 dense layers.

This way it will be possible to measure the computational speed and performance of the different neural networks. Learning rate and number of previous steps is the same decided in *RNN* with values of *0,005 and 20.*

DATA PREPARATION

In the case of the MLP the data has not been normalized since its typical activation function as mentioned above is *ReLu* and performs well without data normalization.

From the initial data frame we use the *'Cumulative Sum Corrected'* column and split it between the values according to the date with those to be used for training *(January 20–2 - January 2013)* and those to be used for the test part *(January 2014).* The output before transforming it to an array:

```
Train dataset
DayTime
2012-01-01 00:00:00    24.432
2012-01-01 00:30:00    15.216
2012-01-01 01:00:00    12.144
2012-01-01 01:30:00    11.952
2012-01-01 02:00:00    44.640
                         ...
2013-01-31 21:30:00    36.472
2013-01-31 22:00:00    35.004
2013-01-31 22:30:00    32.580
2013-01-31 23:00:00    28.110
2013-01-31 23:30:00    21.943
Name: Cumulative Sum Corrected, Length: 2976, dtype: float64


Test dataset
DayTime
2014-01-01 00:00:00    22.890546
2014-01-01 00:30:00    19.056000
2014-01-01 01:00:00    16.698545
2014-01-01 01:30:00    15.674182
2014-01-01 02:00:00    15.021818
                           ...
2014-01-31 21:30:00    32.216930
2014-01-31 22:00:00    33.353302
```

```
2014-01-31 22:30:00    28.708465
2014-01-31 23:00:00    23.103628
2014-01-31 23:30:00    23.840372
Name: Cumulative Sum Corrected, Length: 1487, dtype: float64
```

Input shape in Dense Layers is (batch_size, units), then the data frame values should be divided into sequences of 20 values to predict the 21st. In this case, due to the nature of the sequences, it is not necessary to modify the resolution of the problem to one hour and it is kept at 30 minutes, which benefits since we have more data both for training and for the test.

```
Training Set X_train (2956, 20) and Y_train (2956,)
Testing Set X_test (1487, 20) and Y_test (1487,)
```

Those are the shapes of the sequences introduced in the *MLP* for training and testing.

Methodology of study

For the study case, RELU is the only activation function used in the MLP, and the performance of the NN with the different learning algorithms it's been measured. Same parameters used in the study performed for RNN are used (learning rate applied = *0,005* , maximum number of epochs = 2000).

The performance is measured considering the errors *(RMSE, MAE, MAPE, MSE)*, the number of epochs without overfitting and the total execution time. To avoid overfitting an *early stopping* function has been applied with a patience of 20 values, which means that if in 20 values the *mean squared error* is stationary it stops, apart from avoiding overfitting it improves the total time execution.

Furthermore, graphical representation of the values has been evaluated to avoid repetitive models.

MLP FORECASTING RESULTS

Table 6 shows the performance indicators of MLP study case:

		RMSE	MAE	MSE	MAPE	EXEC TIME (seconds)	EPOCHS
MLP	Adam 'Relu' 20	2.27	1.71	5.16	7.20%	19.29	124
	AdaGrad 'Relu' 20	2.32	1.76	5.40	7.04%	84.49	577
	RMSprop 'relu' 20	2.13	1.62	4.52	7.05%	39.62	206

Table 6. MLP performance indicators

As can be seen all three optimization models gave good results in terms of error and execution time, clearly the worst is *AdaGrad 'relu' 20* with an execution time of *84,49 s* and an *RMSE* of 2,32. The more accurate model is *RMSprop 'relu' 20* with an *RMSE* of *2.13* and an execution time of *39,62 s* but *Adam 'relu' 20* is very fast and forecasting is also accurate so we consider the best model to be *Adam 'relu' 20.* Following graphs show values

predicted during the first 2 weeks of January 2014; as can be seen prediction clearly follows the trend of the actual values apart from *AdaGrad* that is not accurate in the periods that demand is low:

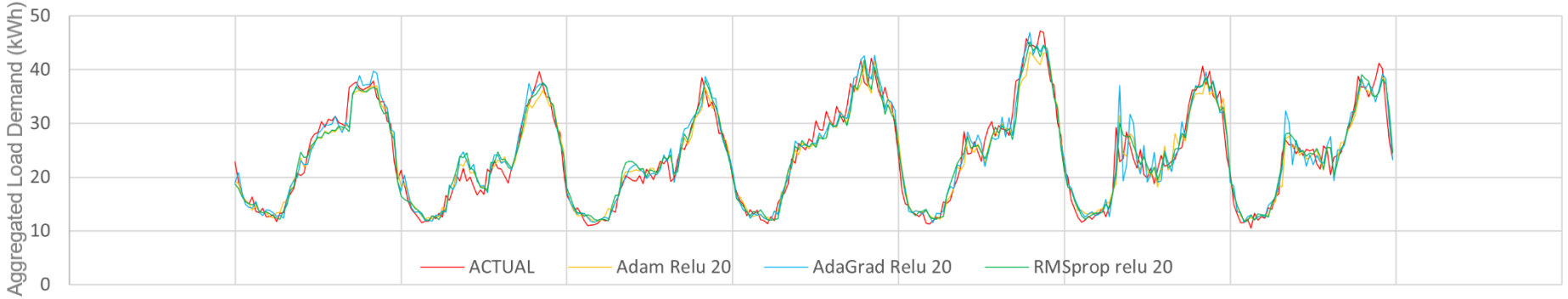

Fig 37. MLP Forecast Week 1

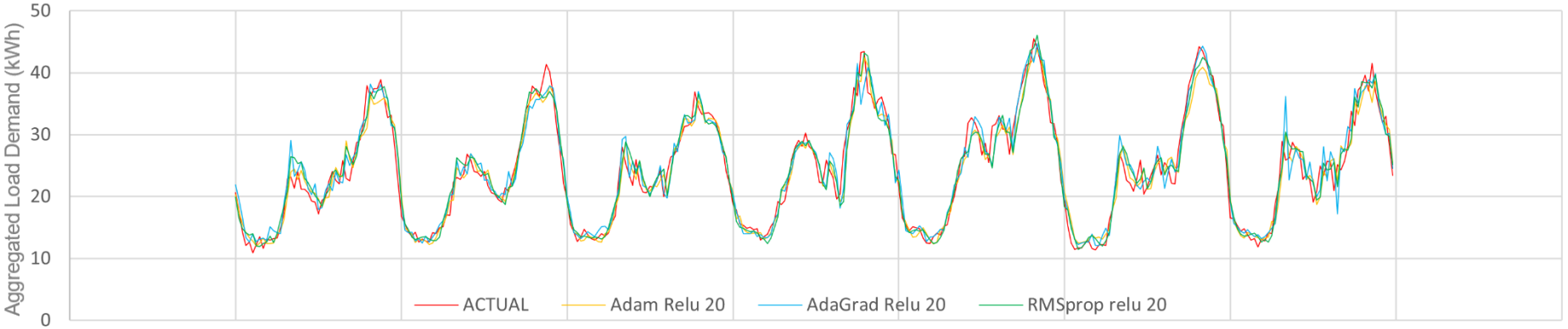

Fig 38. MLP Forecast Week 2

5.1.4.3. NEURAL NETWORKS COMPARISON

To make a comparison of which between different neural networks, best configuration within the type of neural network (*SimpleRNN, LSTM and MLP*) has been selected.

NEURAL NETWORK	CONFIGURATION	RMSE	MAE	MSE	MAPE	EXEC TIME	EPOCHS
RNN SIMPLE	RMSProp 'relu' 20	7.30	6.12	53.33	13.86%	14.56	92
LSTM CELLS	Adam 'relu' 20	7.62	6.32	58.07	10.81%	80.13	527
MLP	Adam 'Relu' 20	2.27	1.71	5.16	7.20%	19.29	124

Table 7. Best configurations of Neural Networks

Best configurations are selected considering the lowest *RMSE, execution time and number of epochs.* From the performance indicators we can see how *MLP* networks perform better than *RNN* having a much lower error ($RMSE_{MLP} = 2,27$) compared to *RNNs* ($RMSE_{RNN} = 7,30$). *RNNs* are more complex than *MLPs,* one of the causes of the differentiation of performance between them can be the size of the data used; RNNs work very well with large amount of data, and this is not the case since 2 months of training and 1 month of testing were used. The week-by-week forecasting for best configurations during January 2014 are plotted below:

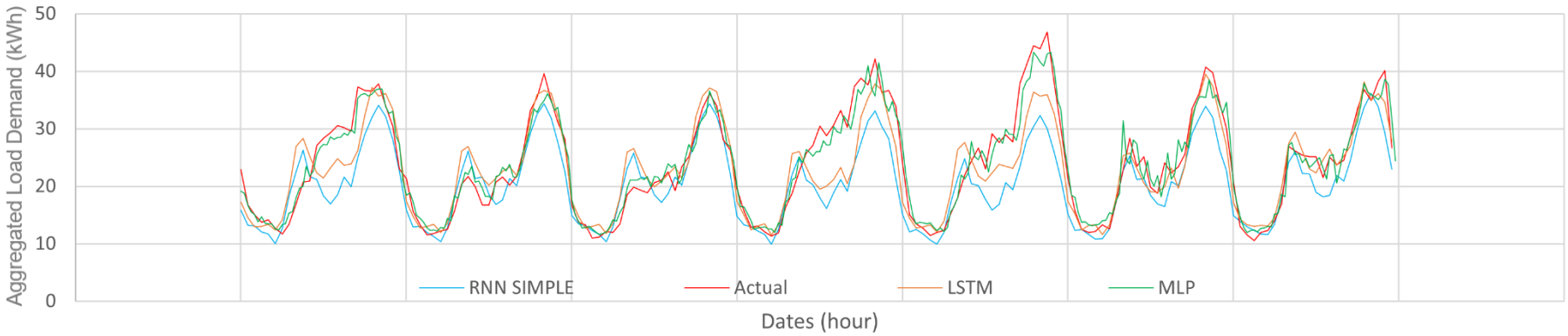

Fig 39. Neural Networks Comparison Week 1

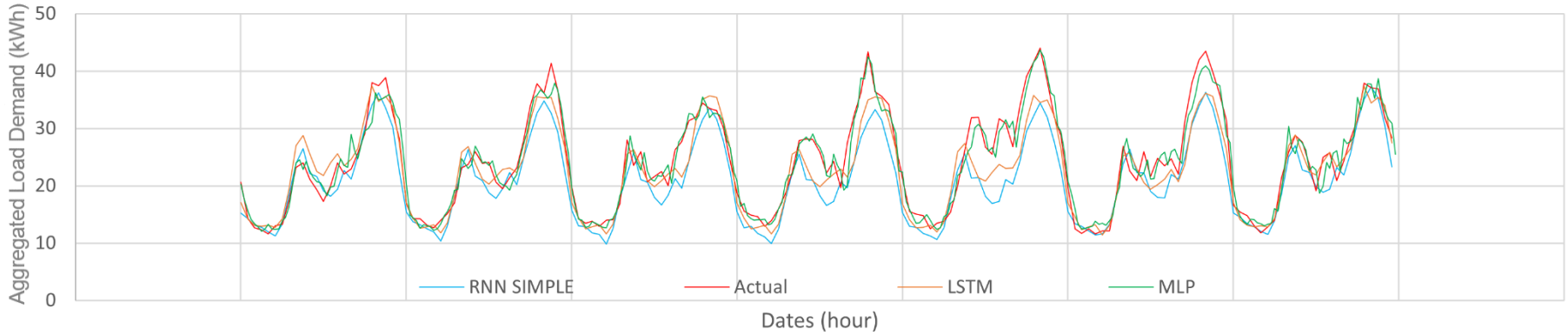

Fig 40. Neural Networks Comparison Week 2

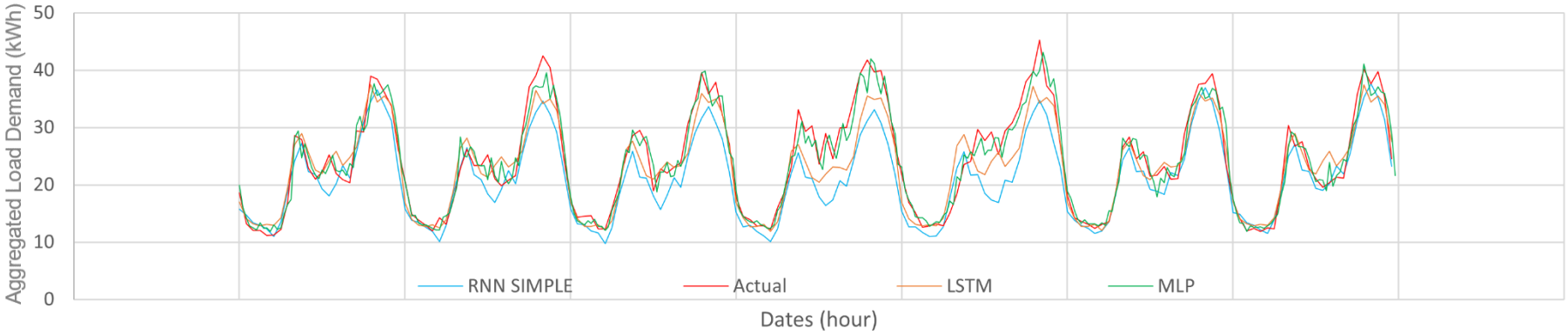

Figure 41. Neural Network Comparison Week 3

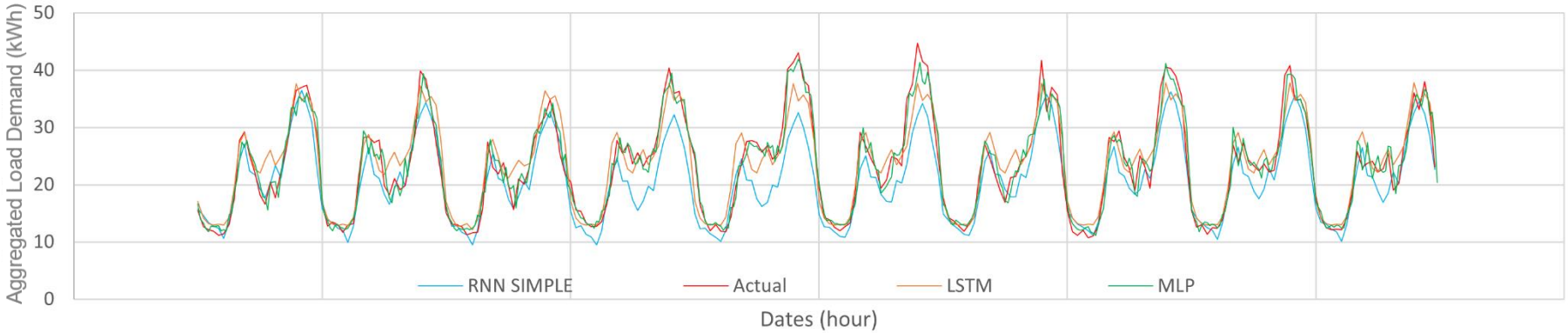

Figure 42. Neural Network Comparison Week 4

5.1.5. DISCUSSION

In chapter 5 forecast of Aggregated Load Demand using 2 Statistical methods (*ARIMA, SARIMA)* and 3 Neural Networks (*SimpleRNN, LSTM, MLP)* in a winter scenario have been created to compare the qualitative differences between models and study the performance. The winter scenario designed corresponds into using January 20–2 – 2013 as training data, and January 2014 for testing.

The results obtained from Neural Networks has been compared in section 5.1.4.3 where *MLP* shows a better performance than *RNN (*$RMSE_{MLP}$*=2,27,* $RMSE_{RNN}$ *= 7,30)* so in order to compare *NN* with Statistical methods, *MLP* has been selected to be compared with ARIMA an SARIMA to define the Advantages and Disadvantages of each one.

Forecasting results obtained in section 5.1.2, 5.1.3 and 5.1.4 show significant differences for every model in terms execution time, in terms of accuracy statistical models are quite similar *(RMSE 2,98 – 3,00)*, NN show an accuracy a bit better than statistical methods *(RMSE 2,27).* Furthermore, as can be seen in the following graphs all 3 methods adapt correctly to the *sharper* changes of the aggregated load demand in the peak hours from 18:00 to 22:00.

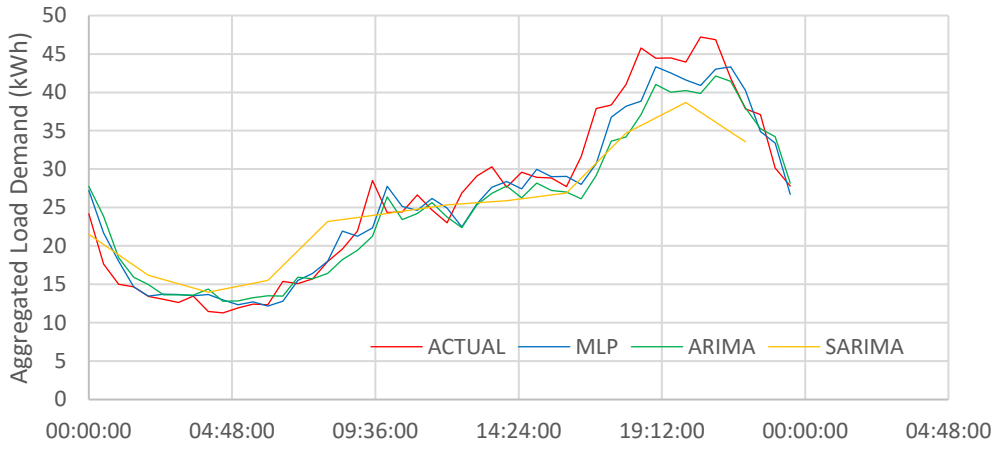

Fig 43. Aggregated Load Demand on 05-01-2014

	RMSE	MAE	MSE	MAPE	EXECUTION TIME (s)	Resolution
ARIMA	*2,98*	*2,30*	*8,91*	*9,70 %*	*11574,4*	*30 min*
SARIMA	*3,00*	*2,37*	*8,99*	*10,50%*	*4408,93*	*2 hours*
MLP	*2,27*	*1,71*	*5,16*	*7,20%*	*19,29*	*30 min*

Table 8. Comparison NN vs Statistical Methods

However, in terms of *Execution Time NN,* performs much better than statistical methods. *MLP* in *19,29 s* is capable of being training with the amount of data given and give a more accurate prediction than Statistical Methods with the same resolution (*11574,4 s)*. In the case of SARIMA it has not been possible to perform a forecasting with the resolution of 30 minutes due to the computational restrictions. *SARIMA* is a more complex model than ARIMA and using ¼ *(2 hours resolution)* of the data used in *ARIMA* or *MLP (30 min resolution)* it has an execution time of *4408,93 s.*

Depending on the type of cell we are using, the input data used in *NN* need to be transformed or modified, so needs an extra time of data cleaning that in Statistical methods is not necessary. Nevertheless, the computational efficiency and accuracy of *NN* in big amounts of data, gives possibilities that cannot be performed in Statistical methods.

The following figures show week-by-week long-term energy demand forecasts for the month of January 2014. As can be seen the *SARIMA* values do not adhere to the energy demand curves as accurately as *MLP or ARIMA* although the error is similar, that is because SARIMA does not have the same data resolution as the other forecasts:

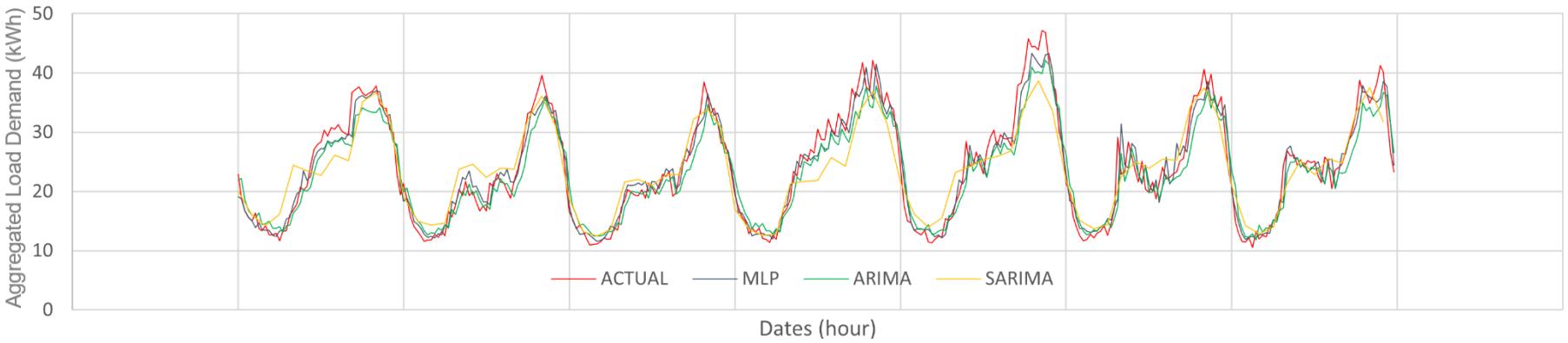

Fig 44. Stastistical models vs Neural Networks Week 1

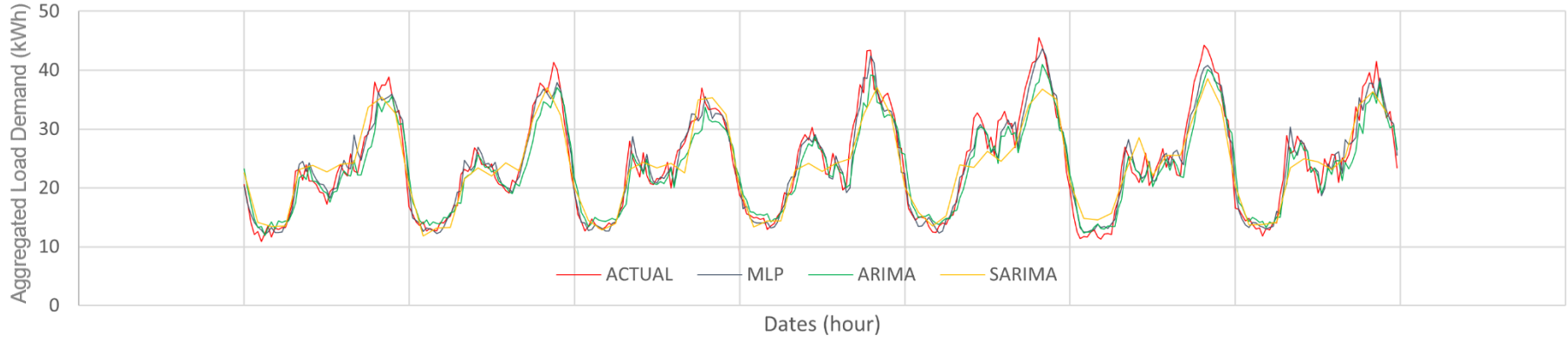

Fig 45. Stastistical models vs Neural Network Week 2

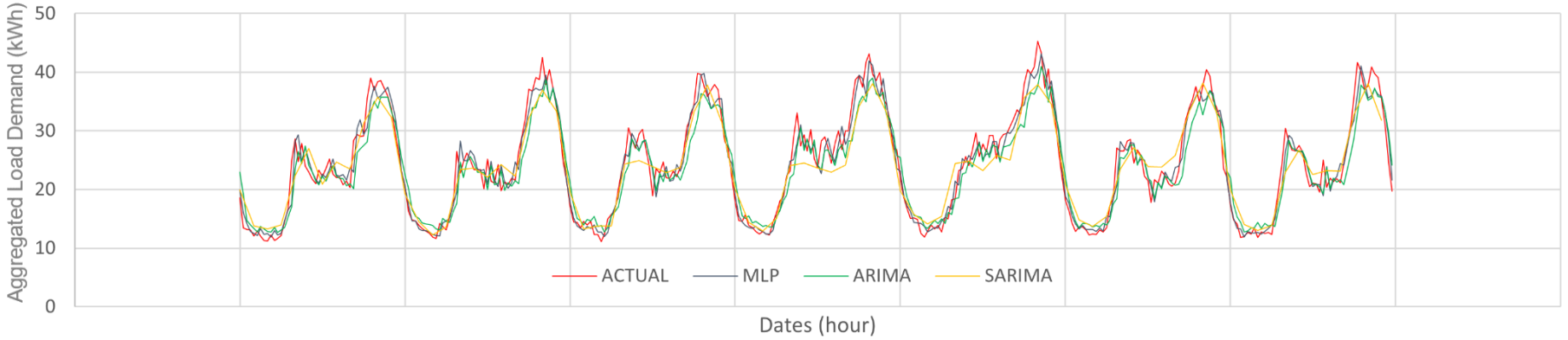

Fig 46. Statistical models vs Neural Network Week 3

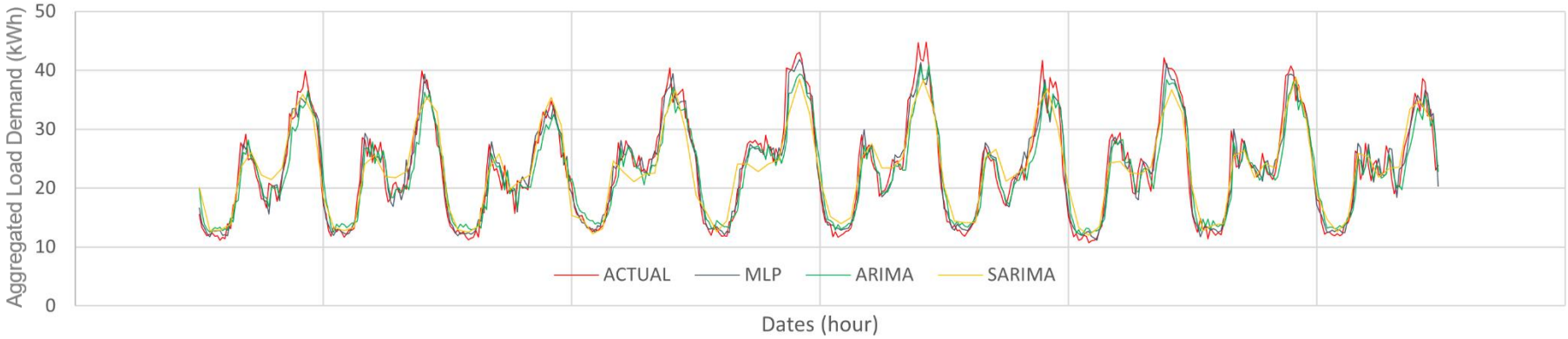

Fig 47. Statistical models vs Neural Network Week

6. OPEN RESEARCH QUESTIONS

During the work of this thesis additional interesting questions arose, but, due to the scope of the work some of them have to remain subject to future research. The focus has been on the development of artificial intelligence methodologies to be applied to long-term demand forecasting, in order to compare the statistical methods with the Neural Networks.

Dataset used has only data of energy load demand, so it would be interesting to combine this data with other datasets such as economic factors, or meteorological influence in order to determine the influence in the energy demand. Furthermore, Recurrent Neural Networks developed in the project could be and tested with bigger amounts of data to determine the computational efficiency and be also compared with MLP in a big data environment.

Finally, Convolutional Neural Networks combined with RNN could be developed and tested with other datasets to determine its performance since they haven't been included in the framework.

Conclusions

Long-Term forecasting is crucial for planning and operations in the power sector. There are not many published studies on long-term prediction as it is something new. However, there are several published studies on short-term energy demand prediction. For the development of the work different articles and studies have been analysed to finally develop a neural network capable of making a long-term prediction in a reasonable time with the data set provided for further comparisons with statistical methods.

Statistical methods (*ARIMA, SARIMA)* have proven to be reliable and capable of giving a good long-term forecast *(RMSE = 2,98 – 3,00)*. Nevertheless, the run time is high and a previous study of the data must be performed in order to select the parameters correctly. In the study case data size is not very large and yet the run times have been quite large i.e. *ARIMA 3,5* hours. Therefore, if we were to use large data sizes the run time would increase or model the might not converge.

Neural Networks have proven to be better in accuracy by *25,56%* than statistical methods, and also in the execution time (*NN: 19,29s, ARIMA: 11574,4s, SARIMA: 4408,93s)*. Several published studies such as [11] or [17] refer to the great advantages of RNNs in terms of accuracy and speed of calculation for long-term energy demand forecasts. According to the results obtained, it has been seen how *MLP* led to better results than *RNN*, this is since RNN are more complex. The size of the data used in this work is quite small, so in a bigger dataset *RNN* would have a different behaviour and a better performance.

In conclusion, all statistical models and neural networks tested have a good accuracy and can be used for long-term demand forecasting. However, the simplicity of the Neural Networks, along with its numerical and computational efficiency add great benefits to the results.

Acknowledgments

I would like to express my sincere gratitude to Mònica Aragües Peñalba and Antonio Emmanuel Saldaña for all guidance in this project, it could not be better. Moreover, I would like to thanks to my family, friends, and partner for all the support and motivation received.

Bibliography

[1] ARIMA by Box Jenkins Methodology for Estimation and Forecasting Higher Education. Marilena Aura Din. Romanian – American University, January 2015. Page 4-12

[2] Engineering Statistics Handbook, 6.4.4.6.Box-Jenkins Model https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc446.htm - April 2012

[3] Engineering Statistics Handbook, 1.3.3.1.Autocorrelation Plothttps://www.itl.nist.gov/div898/handbook/eda/section3/autocopl.htm – April 2012

[4] Engineering Statistics Handbook, 6.4.4.6.3. Partial Autocorrelation Plot https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc4463.htm - April 2012

[5] SERIES TEMPORALES: MODELO ARIMA, Santiago de la Fuente Fernandez, Universidad Autónoma de Madrid – May 2021

[6] An introduction to the Akaike information criterion – March 2020 https://www.scribbr.com/statistics/akaike-information-criterion/

[7] An Intuitive Explanation of the Bayesian Information Criterion – July 2020 https://towardsdatascience.com/an-intuitive-explanation-of-the-bayesian-information-criterion-71a7a3d3a5c5

[8] Maximum Likelihood Estimates: Class10,18.05 Jeremy Orlo and Jonathan Bloom – April 2014

[9] Capítulo 10: Autocorrelación. Prof. Dr. José Luis Gallego Gómez – Universidad de Cantabria – 2008

https://ocw.unican.es/pluginfile.php/1285/course/section/1583/tema10.pdf

[10] Neural forecasting: Introduction and literature overview – Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Bernie Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella,Laurent Callot, Tim Januschowski - April 2020

[11] Recurrent Neural Networks for Time Series Forecasting: Current status and future directions – Hansika Hewamalage, Christoph Bergmeir, Kasun Bandara - 2021 Faculty of Information Technology, Monash University, Melbourne, Australia

[12] A Gentle Introduction to the Adam Optimization Algorithm for Deep Learning – Jason Brownle – January 2021

https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[13] An overview of gradient descent optimization algorithms – Adagrad. Sebastian Ruder – January 2016

https://ruder.io/optimizing-gradient-descent/index.html#adagrad

[14] Activation functions in Neural Networks – Geeks for Geeks. – October 2020

https://www.geeksforgeeks.org/activation-functions-neural-networks/

[15] A Gentle Introduction to the Rectified Linear Unit (ReLU) – Jason Brownle – August 2020

https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[16] Machine Learning for Time Series Forecasting with Python

[17] Long Term Load Forecasting with Hourly Predictions based on Long-Short-Term-Memory Networks – Rahul Kumar Agrawal, Frankle Muchahary, Madan Mohan – 2021.Tripathi. Dept. of Electrical Engineering, Delhi Technological

[18] Figure 6. https://www.researchgate.net/figure/ReLU-activation-function_fig3_319235847

[19] Figure 7. https://www.researchgate.net/figure/tanh-vs-logistic-Sigmoid_fig6_338655469

[20] Figure 8. https://abhigoku10.medium.com/activation-functions-and-its-types-in-artifical-neural-network-14511f3080a8

[21] *Big Data for Open Innovation Energy Marketplace (BD4OPEM H2020)*

https://cit.upc.edu/es/portfolio-item/bd4opem/