COMPARISON OF DIFFERENT PREPROCESSING AND NEURAL NETWORK APPROACHES FOR THE INVESTIGATION OF THE FORECAST HORIZON OF TIME SERIES OF WIND SPEED

Ilias-Marios Sarris

University of Crete, Physics Department

Supervisor: Yannis Padazis

Date: June 2021

Abstract

Environmental concerns have encouraged the adoption of renewable energy alternatives to reduce greenhouse gas emissions across the world. As a consequence of that wind farms have been installed in several locations across Greece for the production of wind power. Wind farm management and control, power distribution planning, storage capacity management, and system's dependability, all benefit from reliable wind speed forecasts.

In this thesis, the problem of the wind speed forecast has been approached through different preprocessing techniques, that involve scaling, smoothening or data augmentation, as well as Artificial Neural Network models that are based on Gated Recurrent Units (*GRUs*). Moreover, comparisons were made for different Datasets with consisting of various sampling steps and Datasets, such as Jena Climate, that contain significantly more observations.

The comparisons that were made for the preprocessing techniques indicated that the smoothening approach managed to capture more accurately the dynamic structure of the data and perform robust predictions, even for up to 10 – 12 hours ahead. At the same time, the different Neural Network Architectures that were proposed had no significant differences with respect to their performance. Comparison between datasets with different sampling steps (1 hour and 10 minutes) indicated no systematic differences as a result of the stochastic nature of the timeseries. Transfer Learning strategy turned out to behave similarly to the original models for short term predictions, where the long term forecasts appeared to be more robust. Jena Climate dataset, may indicate a potential increase in accuracy of forecasts with the condition that more observations will be added in the datasets at hand.

Acknowledgments

I would like to express my sincere gratitude to my Supervisor Researcher Yannis Pantazis for the continuous support of my B.Sc. study and research, for his patience and his insightful comments. His guidance helped me though the whole stages of this research and writing of this thesis.

Content Table

Introduction

Timeseries is one of the most widespread types of data. They consist of a large amount of observations that possess the property of time progression. Every value corresponds to a specific moment in time and thus they are named timeseries. Time progressed data of certain variables is essential to plenty of human or natural processes, especially when forecasting is desirable. A typical example is weather forecast. Meteorologists collect timeseries of temperature, air pressure, wind velocity and direction, along with other variables that play a crucial role in the development of climatic models, and use them in order to make predictions into the future.

The advance of Machine Learning, resolved many real life complex problems in several fields. More specifically, it turned out to be very successful in timeseries forecast as it enabled the performance of more accurate predictions. Sophisticated models like Artificial Neural Networks are able to capture the underlying dynamics of complex systems and estimate their progression in time.

Furthermore, the demand of renewable energy has been increasing sharply over the recent years. Many industries have turned their investments into employing the necessary equipment for generating renewable energy, such as wind farms. Commercializing the energy that is produced, requires a determination of the price. In the example of the wind farms, it is expected the price should be related to the amount of energy that was generated within a day and by extension, the amount of air current that was provided in the location of the wind farm. Therefore, industries install meteorological stations in several locations and heights off the ground in order to predict the airflow of the wind farm for the following hours. Significant errors for such predictions are not desirable as they will result into immense variation of the cost.

The purpose of this work was to apply different ***Artificial Neural Networks*** (***ANNs***) into timeseries of wind velocity from several locations of wind farms in Greece and investigate the limits of the predictions that can be performed, as well as which ANNs model is more appropriate for this task. A significant part of the project was also to perform an exploratory data analysis and a necessary preprocessing in order to extract the structure of the data and feed it to the ANNs in a more sufficient way. That would enable more accurate predictions of the wind velocity and therefore determining the cost of the energy produced by the wind farms in a more precise way.

Generally, it is expected that the error of the predictions increases as we try to foresee further and further into the future. Moreover, as our experience with weather forecast or with the stock market indicates, the projections are never fully reliable. This is mainly due to the fact that the systems at examination have chaotic behaviors, meaning that the progression of such systems is very sensitive to the initial conditions. In the case of this thesis, time series of wind components are very unsystematic and contain very little seasonality, making the forecast process extremely challenging.

Time Series

Time Series Terminology

Time series is a type of data that describe processes that progress in time. In most of the processes of the real world, time is continuous. Time series are created by reconstructing the real continuous signal into a discrete one, which can be recorded and stored in a digital form. The sequenced data that are created are discrete and the values are separated by a regular time interval δt.

For terminology purposes, let us consider a variable $A(t)$ that progresses in time. A typical example of a timeseries for the variable A is shown in the *figure 1*.

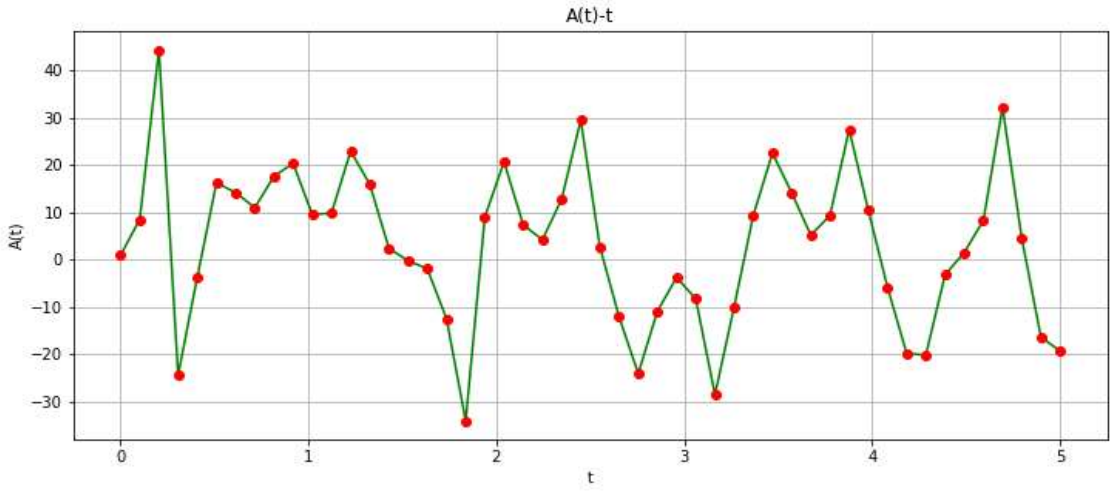


Figure 1. An Example of Time Series (Lorenz's System)

On the example above, the variable A progresses in a time interval $t\epsilon[0,5]$. The time distance between two adjacent values of the time series is $\delta t = 0.1$. The observation $A(t)$ corresponds to the value of A in the current time t. It is often desirable to describe observations that were made on a previous or on a subsequent moment in time. A moment prior in time is defined as $t - n\ \delta t$. Where n is the number of time steps that interpose from the current time. In addition, a subsequent moment in time is indicated as $t + n\ \delta t$. Therefore, the corresponding observations of $A(t)$ are defined as $A(t - n\ \delta t)$ and $A(t + n\ \delta t)$ respectevly.

Time Series Analysis & Forecasting

Estimating the progression of certain values in time has a major significance in many fields. The prediction process involves fitting models into recorded data and then use them for extrapolation. The prediction process is mostly known as *Time Series Forecasting* and it is precisely what we are going to employ on this thesis. In time series forecasting, the future values are always inaccessible and the accuracy of the forecasting models depends on the deviation of the estimations from the ensuing observed data. For optimum forecasting models, one has to determine confidence intervals and most of all, the underlying dynamics of the system in study.

Time series analysis plays a crucial role on the forecasting process. The analysis of the time series aims to extract useful information about the structure of the data, the underlying dynamics of the system and several statistical characteristics. This is made possible mostly through classical statistics. Time series analysis often involves determining Histograms, Frequency Spectrums through Fourier Analysis, Mean Values, Standard Deviations, Feature Importance and Outliers.

Time series analysis produces descriptive models of the dataset at hand that offer a sensitive insight of some important components of time series. Some of the most useful components are presented below:

- **Level**: It is the mean value of the time series. It shows the reference point of the fluctuation that takes place.
- **Seasonality**: A repeating pattern that makes its appearance over time.
- **Noise**: It is often linked with the variability of the time series that it caused by external and undetermined factors.
- **Correlation**: It indicates the level of association of two adjacent values.

These components will guide us into what method of forecasting may be more sufficient to apply. They will also provide plausible arguments about the limits and the divergence of the predictions.

Components of Time Series Analysis

Fourier Analysis (Frequency Spectrum)

The Fourier Transform (FT) is a tool that is commonly used in spectral estimation and signal processing in general. FT transforms a signal from the time domain to the frequency domain. The idea behind this transformation is that every periodic function is a superposition of an infinite sum of cosine functions. Non periodic function can in turn be represented using a continuous set of frequencies, i.e. through an integral representation (FT).

At first let us define the Fourier Transform. Consider an integrable, piecewise continuous function $x(t), t \in R$. The FT of this function is defined as:

$$F\{x(t);\omega\} = X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \quad (1)$$

The function $X(\omega)$ is a continuous, complex-valued function, and it is called *"the Fourier transform of x(t)"* .

The Inverse Fourier Transform (IFT) of the function $X(\omega)$ results in $x(t)$:

$$F^{-1}\{X(\omega);t\} = x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}d\omega \quad (2)$$

Although most signals in the real world (e.g. voice signals) are continuous, the way in which they are measured, in order to be processed, is digital. Therefore, it is a discrete version of the FT, the Discrete Fourier Transform (DFT) that is applied on them, where the integral is replaced by a sum. The reduction of a continuous time signals into a discrete one, is called sampling.

At first, let us consider a continuous signal $a(t)$ sampled at intervals Δt (the sampling period). The rate at which the signal is sampled is $f_s = 1/\Delta t$ and it is called sampling frequency. If the signal's length is T and N is the number of samples, then we have:

$$T = N \cdot \Delta t \quad (3)$$

The sampled signal can now expressed as $a_n = a(t_n)$, $t_n = n\Delta t$, $n = 0,1, \dots . N-1$.

The resulting discrete version of the FT, the Discrete Fourier Transform (DFT), has the form:

$$A_k = \sum_{n=0}^{N-1} a_n e^{-j\frac{2\pi kn}{N}} \quad (4)$$

This is a complex valued sum. While the discrete values a_n correspond to times $t_n = n\Delta t$, $n = 0,1, \dots , N-1$. The discrete values of A_k correspond to frequencies $f_k = k\Delta f$, $k = 0,1, \dots ,$ $N-1$, $where\ \Delta f = \frac{1}{T}$, or in other words to circular frequencies $\omega_k = k\Delta\omega$ where $\Delta\omega = \frac{2\pi}{T}$.

The inverse of the Discrete Fourier Transform is in turn defined as:

$$a_n = \frac{1}{N}\sum_{k=0}^{N-1} A_k e^{\frac{j2\pi kn}{N}} \quad (5)$$

At this point we are in a position to make an approximation of the discrete sample autospectrum, which is designated as:

Let us consider a discrete time window $w_n = w(t_n)$ and $x_n = x(t_n)$ $where$ $t_n = n\Delta t\ and\ f_s = \frac{1}{\Delta t}$, $n = 0,1, \dots . N-1$

$$A_k = \sum_{n=0}^{N-1} w_n x_n e^{-\frac{j2\pi nk}{N}} \quad (6)$$

$$\hat{S}_{xx}^{w}(\omega_k) = \Delta t^2 |A_k|^2 \quad , \quad k = 0,1,\dots,\frac{N}{2} \quad (7)$$

The calculation of $\hat{S}_{xx}^{w}(\omega_k)$ can be easily and quickly executed using the FFT algorithm.

Histogram

In many cases it is desirable to have an insight about the frequency distribution of the dataset at hand, as long as the variables are continuous. One of the most convenient ways to achieve that is through a histogram. Histograms are also very valuable for the detection of outliers and skewness.

In order for a histogram to be constructed, one has to group the data into chosen intervals that are called *bins*. Bins hold the number of occurrences of each value that is included in a given interval. The construction of the histogram requires that the bins are not too big, so they are able to distinguish significant differences of the values, and at the same time, not too small, so they can capture the underlying distribution.

It is important to note that frequency of the occurrences is given by the area and not the height of each bin. However, in many cases bins are equally spaced and therefore the height is an accurate indicator of the frequency.

Machine Learning & Artificial Neural Networks

Introduction

Machine learning (***ML***) is a computational process that learns the structure of the data which are provided to the computer by adaptive algorithms in order to perform desirable tasks. Machine Learning is a part of a more generic field, ***Artificial Intelligence*** (***AI***), that describes the process of automating, predicting and optimizing tasks that are typically performed by humans. ***Artificial Neural Networks*** (***ANNs***) consist a type of model that is used in machine learning.

Machine Learning algorithms employ several approaches into the learning process. The most important ones are presented below:

- In *Supervised Learning* algorithms the data that are used for training are paired with corresponding target data, called *labels*. In these kind of tasks, the goal is to train the computer to match up the given input with the corresponding *label*. Typical examples of supervised learning algorithms are regression and classification tasks.
- In *Unsupervised Learning* algorithms the data that are used for training are unlabeled. The model is constructed using only the inputs. Typical examples of unsupervised learning algorithms are Dimensionality Reduction and Clustering.
- In *Reinforcement Learning* algorithms *agents* are used for performing a certain task. Agents are rewarded every time they perform the desired task, otherwise they are given penalties. Agents attempt to learn the best routine that will guarantee the largest amount of rewards over time.

Challenges of Machine Learning

In a Machine Learning project there are two independent parts that have a particular interest. The first one is the data and the other is the ML model. Regarding the data, there are a few challenges that may arise and need to be addressed before moving to the selection of a suitable model:

- *Insufficient Amount of Data*: One of the disadvantages of the existing ML algorithms is that they need large amount of data in order to be trained. In many cases the data at hand are not adequate for the model to learn successfully from them and perform the anticipated task. A typical way to approach this is by implementing *data augmentation* techniques that basically generate pseudo data from the existing ones. Another approach is to apply *transfer learning* from other models with similar tasks that ensured a sufficient amount of data.

- *Non Representative Data*: In some cases, some variables of the dataset at hand are not strongly related to the desirable task and therefore complicate the training process. Moreover, outliers may detune the learning procedure and prevent it from being generalized. Typically, it is advisable to remove these non-representative data.
- *Poor Quality Data*: Acquiring data with sufficient quality is not always trivial. There are many factors that may contribute to the deterioration of the quality of the observations such as defects of the detector, noise of the system, etc. One approach is to remove the erroneous observation or replace them with expected values.

Creating or choosing an appropriate model for training remains a challenging task, one that only with proper insight of the dataset's statistical structure and the underlying dynamics of the system at study, could be possible to unravel.

Dataset split and Overfitting

At this point it is important to state that the significance of Machine Learning algorithms lays on their ability to create generalized models that perform the desirable tasks. This means that the models are able to approach new instances that were not included in their training process.

If the model is not sufficiently generalized then *overfitting* has occurred, meaning that the model is unable to perform for new data from the ones that it was trained on. On the other hand, if the model is not able to capture the underlying structure of the data, it is called *undrefitting*. Both pose an equal threat to the accuracy of the model employed.

In order to have an insight of the training process and detect potential abnormalities, like the ones described, the dataset is split into three components: the *training set*, the *validation set* and the *test set*. The model is initially fitted on a training dataset, which is a set of data used to fit the parameters (e.g. weights) of the model. The fitted model is used to make predictions of the elements of the validation. The validation dataset provides an unbiased evaluation of a model fit on the training dataset. Finally, the test set is a dataset used to provide an unbiased evaluation of a final model fit on the training dataset. Validation set offers us an insight of the performance of the model during the training process.

A clear indication of overfitting is when the error of the training set reduces while the error of the validation set rises. Overfitting typically occurs when there is not sufficient amount of data at hand, when the parameters of the model are excessively more than the amount of data at hand, and more. Conversely, if both the training and validation error have not dropped adequately indicates undrefitting.

A representative example of overfitting is presented on the figure below:

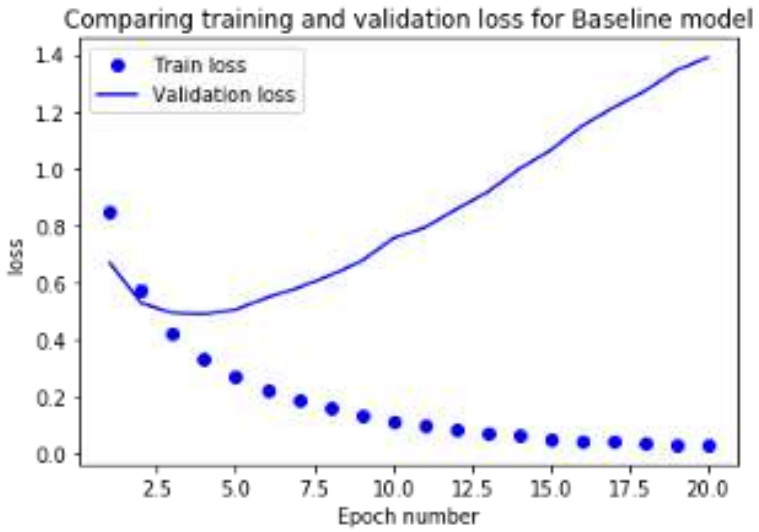


Figure 2. Example of Overfitting [ ]

Undrefitting is usually resolved by extending the training process for more iterations. Regarding the Overfitting there are many computational techniques that could assure a satisfactory generalization of the model. *Early Stopping* is a very convenient way to elude overfitting as it interrupts the training process when the validation error starts rising and then saves the parameters of the models that ensured the best performance. Implementing *L1 and L2 Regularizations* is another approach. The Regularization technique penalizes the loss function of the model and in the case of ANNs, it limits the weight connections of Neurons. *Dropout*, is another technique commonly used for ANNs, in which there is a probability *p* for each neuron at every training step to be temporarily excluded from the training process and therefore reducing the complexity of the model. Finally, *Max-Norm regularization* is another way to combat overfitting in ANNs. In Max-Norm regularization, an upper bound of the weight of each neuron is imposed such that $\|w\|_2 \leq r$, where r is the max-norm hyperparameter and $\|\cdot\|_2$ is the ℓ_2 norm.

Preprocessing and Feature Engineering

In many cases raw data contain many inconsistencies and errors or lack of certain necessary trends. As a result, these properties prevent the ML model to properly assimilate the structure of the data and therefore complete their desirable task. Preprocessing is the practice of handling such abnormalities and therefore transforming the raw data into a more meaningful set. This includes *Normalizing* and *Scaling* the raw data, dealing with missing values or applying *Filters* and implementing *Data Augmentation* techniques. Feature Engineering, on the other hand, is the process of using domain knowledge in order to transform existing features into new ones that would contain more meaningful

information for the model. A brief introduction to some of the most common techniques of Preprocessing and Feature Engineering will be discussed below.

Normalization

Normalization is a scaling technique in which the data are scaled to have a unit norm. Algorithms that do not assume a specific type of distribution (e.g. Neural Networks) may benefit significantly from that process. There are two common methods for normalization:

- *Maximum Normalizer*: Through this method the data are normalized by applying the transformation:

$$x_{norm} = \frac{x}{\max(x)}$$

  Where x_{norm} is the normalized value and x is the value of the sample before normalization.

- ℓ_1 *Normalizer*: Through this method the data are normalized by applying the transformation:

$$x_{norm} = \frac{x}{\sum_{i=1}^{N}|x_i|}$$

  Where N is the length of the vector that contains all the instances, x_i, and it is about to get normalized.

Scalers

Scaling is a technique that aims to set a common scale to features without altering differences in the ranges of values. In some cases, this is extremely helpful as it allows a more accurate comparison of the corresponding data. Models that employ Gradient Descent as an optimizing method, such as ANNs, may require scaling since it provides an important boost in the convergence towards the minimum. Some of the most representative scalers are introduced below:

- *Min-Max Scaler:* Through this method the values of the samples are confined in a range between 0 and 1 by applying the transformation:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

  Where x_{scaled} is the scaled value, x is the value of the sample before scaling and x_{max}, x_{min} are the maximum and minimum values of the dataset before scaling.

This transformation conserves the distribution of the data and is has a high sensitivity with outliers.

- *Absolute Maximum Scaler:* Similar to the *Min-Max Scaler*, the data are ranged between -1 and 1 by applying the transformation:

$$x_{scaled} = \frac{x}{\max(|x|)}$$

This transformation is based on the absolute maximum, as the name implies. It also preserves the distribution of the data and is has a high sensitivity with outliers.

- *Standard Scaler (z-score Normalization):* This scaler sets the mean value of the data to zero and its standard deviation to one by applying the following transformation:

$$x_{scaled} = \frac{x - \bar{x}}{\sigma}$$

Where $\bar{x}$ is the mean value and σ is the standard deviation of the data before scaling.

This transformation does not set a certain range to the values of the data as it is based on the σ of the dataset. It can be useful for datasets that are approximately described with a Gaussian distribution.

- *Robust Scaler:* Similar to *Standard Scaler*, this scaler sets the mean value of the data to zero and its standard deviation to one while neglecting the effect of outliers. It accomplishes that by applying the following transformation:

$$x_{scaled} = \frac{x - median(x)}{Q_{75} - Q_{25}}$$

Where $median(x)$ is the median value. Q_{75} and Q_{25} are the 75th and 25th percentiles of the distribution of the data before scaling. The corresponding subtraction is the known as *interquartile range* (*IQR*).

This transformation has not a predetermined range of values and it can be a very powerful tool when outliers are present.

As a general rule, it is preferable to fit the Scaler into the training set and then apply it on the test set, in order to avoid any leakage of information that would affect the model's evaluation of the training.

Numerical and Categorical Features

There are two general categories that features of the dataset are often classified into: Numerical and Categorical Features. If a house for example is an apartment or a detached house, that would be a categorical feature. On the other hand, the price of a given residence is a numerical feature.

Categorical features are often considered problematic in their raw form, as they cannot be processed as word statements from ML models. Thus, they need to be transformed into numerical features. The most popular approach is to employ the *one-hot-encoder* strategy, in which every category is represented by a vector as shown below:

$$\text{Apartment} \rightarrow \begin{pmatrix}1\\0\\0\end{pmatrix} \quad \text{Detached House} \rightarrow \begin{pmatrix}0\\1\\0\end{pmatrix} \quad Villa \rightarrow \begin{pmatrix}0\\0\\1\end{pmatrix}$$

In this way the model is able to differentiate the provided categorical features and incorporate them for training.

Numerical features, on the contrast, can be implemented into the ML model in their raw form. However, there are cases were the corresponding features require certain transformations in order to be best grasped by the algorithm. Typical examples are features with periodic conditions such as the date, time and angles. The key idea is that certain transformations, like the $\sin(x)$ and $\cos(x)$, may capture the periodicity of these variables as presented below:

$$x_{time} = \begin{pmatrix}0{:}\,05\\10{:}\,35\\\vdots\\23{:}\,50\end{pmatrix} \rightarrow \begin{pmatrix}5\text{ min}\\10\cdot 60+35\ min\\\vdots\\23\cdot 60+50\ min\end{pmatrix} \rightarrow \begin{pmatrix}5\text{ min}\\635\ min\\\vdots\\1430\ min\end{pmatrix} \xrightarrow{\sin\left(x[min]\ \frac{2\pi}{24\cdot 60}\right)} \begin{pmatrix}0.022\\0.362\\\vdots\\-0.043\end{pmatrix}$$

Therefore, the time is now expressed in a way the relative variances of the periodic values are revealed in a clearer way.

Data Augmentation

Data augmentation is a regularization technique that consist of generating new training instances from existing ones. This method can be very beneficial for the training process as it extends the number of data in our possession and prevents overfitting by forcing the model to be more tolerant to minor alterations of the instances.

For timeseries one suggested technique for data augmentation is to implant white noise into the observations. Let us consider a timeseries $A(t) = \{a_i\}$ with $1 \leq i \leq n$ and suppose a variable ε that meets the condition:

$$0 < \varepsilon < |a_{i+1} - a_i| \forall i \in \{1, \dots, n\}$$

The new timeseries, $B(t) = \{b_i\}$, is generated by adding a term r_i to every instance of $A(t)$, where r_i is a realization of the distribution $N\left(0, \frac{\varepsilon}{2}\right)$. [ ]

$$B(t) = \{a_i + r_i\}$$

Then the ML model is then fitted into both timeseries to perform training.

Smoothing

Timeseries often consist of many anomalies and noise that conceal the valuable information. This prevents the extraction of useful patterns or trends that are required for the timeseries forecasting. As a result, a common method for dealing with this is to smooth the signal by applying filters. The two most frequent choices are the *Gaussian* and *Moving-Average filter*.

Let us consider a series $A(n)$ with $1 \leq n \leq N$ and a filter $H(k)$ with $1 \leq n \leq L$, where L represents the width of the filter. Supposing an impulse response of the filter $h(k)$ that is applied on the signal, the output value $G(n)$ is expressed as:

$$G(n) = \sum_{m=1}^{L} h(k)A(n-m)$$

Note that applying the filter $H(k)$ into the signal in to perform convolution of the filter with the given series.

A moving average is calculated by generating a new series whose values are the average of the raw observations in the original time series. A moving average requires the selection of a window size, which specifies how many instances are used to determine the moving average value. Then, to compute the average values in the new series, the window is slid along the time series. The impulse response of the moving average filter is the following:

$$h_{ma}(k) = \frac{1}{L}$$

In other words, the moving average filter can be seen as a rectangular pulse with height $\frac{1}{L}$.

Similarly, the Gaussian filter is defined by a window size L and it is slid along the series in order to generate the new one. However, the Gaussian filter does not set the same weights for each instance

of the series. The weights are determined based on the Gaussian distribution and the given standard deviation σ. In this case the impulse response is given by:

$$h_G = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

For both cases the larger the window width, the more observations are involved in the convolution and the more effective the smoothing is going to be.

Occasionally, smoothing may not be a preferable choice as it comes with a price of removing information of the signal. Forecasting might be less challenging, but it also might be less accurate.

Introduction to Artificial Neural Networks

Linear Threshold Unit & Multi-Layer Perceptrons

Inspired by the structure of a biological neural network that consists of many interacting neurons, the fundamental unit of an artificial neural network is a *Linear Threshold Unit* (*LTU*). An LTU takes one or more numbers as inputs, calculates the weighted sum and passes the argument through a step function, which then outputs the final result. The placeholder of each value is called a *neuron*. A *Perceptron* is simply composed of a single layer of LTUs with each neuron connected to all the inputs. By stacking Perceptrons together a *Multi-Layer Perceptron* (*MLP*) is created. An MLP is composed of one input layer, one or more layers of LTUs, called *hidden layers*, and one final layer of LTUs called the *output layer*. Every layer except the output layer includes a *bias neuron* and is fully connected to the next layer. When NN has two or more hidden layers, it is called a *Deep Neural Network* (*DNN*). An illustration of an LTU is given in the figure 3.

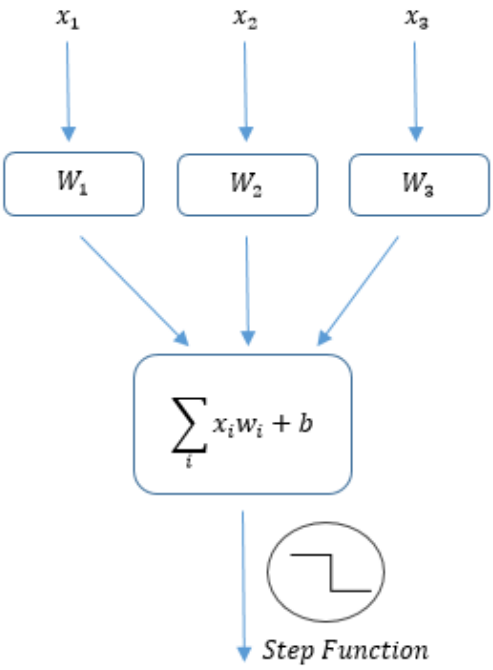


Figure 3. Linear Threshold Unit (LTU)

Backpropagation and Training Process of ANNs

In the training process the Neural Network has to determine the best values of weights in order to optimize the prediction given an input x. In a forward pass the network performs the below operation for each of the neurons:

$$z^l = W^l X^{l-1} + b$$

$$X^l = \sigma(z^l) = \sigma(W^l X^{l-1} + b) = \sigma(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b)$$

Where the X^{l-1} is the vector of the input values (the output of the previous layer, X^{l-1}), X^l is the vector of the output values of the layer l, σ is the step function, z^l is the weighed sum of the inputs for layer l that later passes through the step function, W is a vector the carries the weights of the layer l and b is the bias of the layer.

The output of the whole model after the forward pass is denoted by $\hat{y}$, and to measure its variance from the target value y, that is the desirable outcome, a loss function is calculated. The most common choices regarding the loss function are Mean Squared Error (*MSE*) and Mean Absolute Error (*MAE*):

$$MSE: \qquad L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

$$MAE: \qquad L = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

The *backpropagation* algorithm is commonly used in the training process. According to this algorithm, the NN makes a prediction (forward pass), calculates the error, goes through each layer in reverse to measure the error contribution from each connection (reverse pass), and then adjusts the connection weights to reduce the error in each training instance (epoch). The ratio of how much each weight influences the cost function is calculated by employing the partial derivatives of the cost function with respect to each parameter:

$$\frac{\partial L}{\partial w^L} = \frac{\partial L}{\partial x^L}\frac{\partial x^L}{\partial z^L}\frac{\partial z^L}{\partial w^L}$$

$$\frac{\partial L}{\partial b^L} = \frac{\partial L}{\partial x^L}\frac{\partial x^L}{\partial z^L}\frac{\partial z^L}{\partial b^L}$$

Where, L denotes the parameters of the last layer.

Each partial derivative from the weights and biases is saved in a *gradient vector*, that has as many dimensions as you have weights and biases:

$$-\nabla L(w_1, b_1, w_2, b_2, \dots, w_L, b_L) = \begin{bmatrix} \frac{\partial L}{\partial w^1} \\ \frac{\partial L}{\partial b^1} \\ \vdots \\ \frac{\partial L}{\partial w^L} \\ \frac{\partial L}{\partial b^L} \end{bmatrix}$$

The gradient is computed using mini-batches (subsets) of the data. The performance is calculated for each weight and bias for each observation in the mini-batch.

The new, optimized weight values are given from the following equation:

$$w^l = w^l - \eta \frac{\partial L}{\partial w^l}$$

$$b^l = b^l - \eta \frac{\partial L}{\partial b^l}$$

Where η is a constant between 0 and 1 which expresses the learning rate for Gradient Descent Optimizer, which is the optimization algorithm used for the minimization of a cost function.

An illustration of a fully connected network of artificial neurons is presented below:

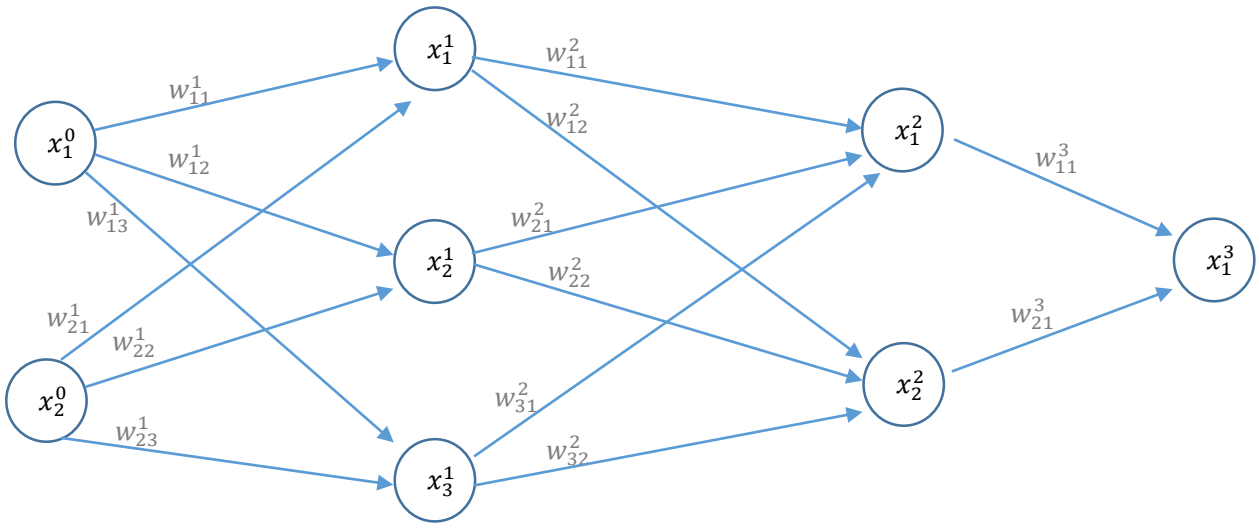


Figure 4. Multi-Layer Perceptron (MLP)

In the figure above we use the notation x_n^l to imply the value of the n-th neuron in the l-th layer and w_{nm}^l to imply the value of the weight in the l-th layer that connects the neurons x_n^{l-1} with x_m^l. Note that for every neuron x_n^l there is a bias b_n^l respectively.

For example, the values of the first layer are computed by the following equations:

$$x_1^1 = \sigma\left(\sum_{j=1}^{2}\left(w_{1j}^1 x_j^0 + b_1^1\right)\right) = \sigma(w_{11}^1 x_1^0 + w_{12}^1 x_2^0 + b_1^1)$$

$$x_2^1 = \sigma\left(\sum_{j=1}^{2}\left(w_{2j}^1 x_j^0 + b_2^1\right)\right) = \sigma(w_{21}^1 x_1^0 + w_{22}^1 x_2^0 + b_2^1)$$

$$x_3^1 = \sigma\left(\sum_{j=1}^{2}\left(w_{3j}^1 x_j^0 + b_3^1\right)\right) = \sigma(w_{31}^1 x_1^0 + w_{32}^1 x_2^0 + b_3^1)$$

Similarly, for the second layer the computed values of the neurons will be:

$$x_1^2 = \sigma\left(\sum_{j=1}^{3}\left(w_{1j}^2 x_j^1 + b_1^2\right)\right) = \sigma(w_{11}^2 x_1^1 + w_{21}^2 x_2^1 + w_{31}^2 x_3^1 + b_1^2)$$

$$x_2^2 = \sigma\left(\sum_{j=1}^{3}\left(w_{2j}^2 x_j^1 + b_2^2\right)\right) = \sigma(w_{21}^2 x_1^1 + w_{22}^2 x_2^1 + w_{32}^2 x_3^1 + b_2^2)$$

The output of the x_1^3 is given through analogous calculations.

After the forward pass, the backpropagation algorithm takes place. The determination of the impact of each feature in the parameters of the model is described by the following partial derivatives, starting from the weights and biases of the last layer and moving towards the first layer:

For the 3rd layer of the network:

$$\frac{\partial L}{\partial w^3} = \frac{\partial L}{\partial x^3}\frac{\partial x^3}{\partial z^3}\frac{\partial z^3}{\partial w^3} \quad and \quad \frac{\partial L}{\partial b^3} = \frac{\partial L}{\partial x^3}\frac{\partial x^3}{\partial z^3}\frac{\partial z^3}{\partial b^3}$$

And by extend:

$$\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial x_1^3}\frac{\partial x_1^3}{\partial z_1^3}\frac{\partial z_1^3}{\partial w_{11}^3}$$

$$\frac{\partial L}{\partial w_{21}^3} = \frac{\partial L}{\partial x_1^3}\frac{\partial x_1^3}{\partial z_1^3}\frac{\partial z_1^3}{\partial w_{21}^3}$$

$$\frac{\partial L}{\partial b_1^3} = \frac{\partial L}{\partial x_1^3}\frac{\partial x_1^3}{\partial z_1^3}\frac{\partial z_1^3}{\partial b_1^3}$$

Regarding the 2nd layer of the network:

$$\frac{\partial L}{\partial w^2} = \frac{\partial L}{\partial x^3}\frac{\partial x^3}{\partial z^3}\cdot\frac{\partial z^3}{\partial x^2}\frac{\partial x^2}{\partial z^2}\frac{\partial z^2}{\partial w^2} \quad and \quad \frac{\partial L}{\partial b^2} = \frac{\partial L}{\partial x^3}\frac{\partial x^3}{\partial z^3}\cdot\frac{\partial z^3}{\partial x^2}\frac{\partial x^2}{\partial z^2}\frac{\partial z^2}{\partial b^2}$$

Where the first part represents the contribution of the w^3 and the second part represents the contribution of the w^2.

The contribution of the weights and biases of the 1st layer towards the cost function is:

$$\frac{\partial L}{\partial w^2} = \frac{\partial L}{\partial x^3}\frac{\partial x^3}{\partial z^3}\cdot\frac{\partial z^3}{\partial x^2}\frac{\partial x^2}{\partial z^2}\cdot\frac{\partial z^2}{\partial x^1}\frac{\partial x^1}{\partial z^1}\frac{\partial z^1}{\partial w^1} \quad and \quad \frac{\partial L}{\partial b^1} = \frac{\partial L}{\partial x^3}\frac{\partial x^3}{\partial z^3}\cdot\frac{\partial z^3}{\partial x^2}\frac{\partial x^2}{\partial z^2}\cdot\frac{\partial z^2}{\partial x^1}\frac{\partial x^1}{\partial z^1}\frac{\partial z^1}{\partial b^1}$$

Finally, the weight will be updated according to following the relations:

$$w_{11}^3 = w_{11}^3 - \eta\frac{\partial L}{\partial w_{11}^3} = w_{11}^3 - \eta\frac{\partial L}{\partial x_1^3}\frac{\partial x_1^3}{\partial z_1^3}\frac{\partial z_1^3}{\partial w_{11}^3}$$

$$w_{21}^3 = w_{21}^3 - \eta\frac{\partial L}{\partial w_{11}^3} = w_{21}^3 - \eta\frac{\partial L}{\partial x_1^3}\frac{\partial x_1^3}{\partial z_1^3}\frac{\partial z_1^3}{\partial w_{21}^3}$$

$$b_1^3 = b_1^3 - \eta\frac{\partial L}{\partial b_1^3} = b_1^3 - \eta\frac{\partial L}{\partial x_1^3}\frac{\partial x_1^3}{\partial z_1^3}\frac{\partial z_1^3}{\partial b_1^3}$$

The same goes for update of the rest of the parameters of the model.

Activation Functions

The activation function of a node in an artificial neural network (ANN) determines the output of that node given an input or a set of inputs. The next node receives this output as input (in the next layer). The non-linearity of ANNs is determined by activation functions, since if they are non-linear, the entire network is non-linear too. Activation functions may be thought of as binary classifiers, with the options of "trigger" or "not activate." If no activation function is used, the output signal is just a simple linear function, and the system is doomed to learn complex functional mappings from data with less power. As a result, they will be unable to accurately reflect the output.

The selection of an appropriate activation function is a very important step. As the activation function mediates in the passing of arguments between the neurons, it plays a essential role in the training process of the model. Let us suppose that we use the MSE cost function. Then the above calculations of the partial derivatives are given as:

$$\frac{\partial L}{\partial w^L} = \frac{\partial L}{\partial x^L}\frac{\partial x^L}{\partial z^L}\frac{\partial z^L}{\partial w^L} = 2(X^L - y) \cdot \sigma'(z^L) \cdot X^{L-1}$$

Where $\mathrm{X^L}$ represent the output of the last layer, i.e. the output of the whole model ($\hat{y}$) and σ' is the derivative of the step function.

In case that the value of the derivative of the activation function σ' is very close to zero the modification of the parameters' values will be negligible and therefore the ML model will be trained insufficiently. This is known as *Vanishing Gradient Problem*. On the opposite course, if the value of σ' increases sharply, the updates of the parameters will be unreasonably high resulting into a defective training process too. This is known as *Exploding Gradient Problem.*

For these particular reasons, Data Scientists have put in a lot of effort to develop activation functions that do not have the issues listed above and even boost the performance of ANNs.

The *Sigmoid activation function* is the basic choice. The sigmoid function is basically a logistic function that scales the output in a range between zero and one. It is described through the following equation:

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

The graph of the sigmoid function and its derivative is presented in the figure below:

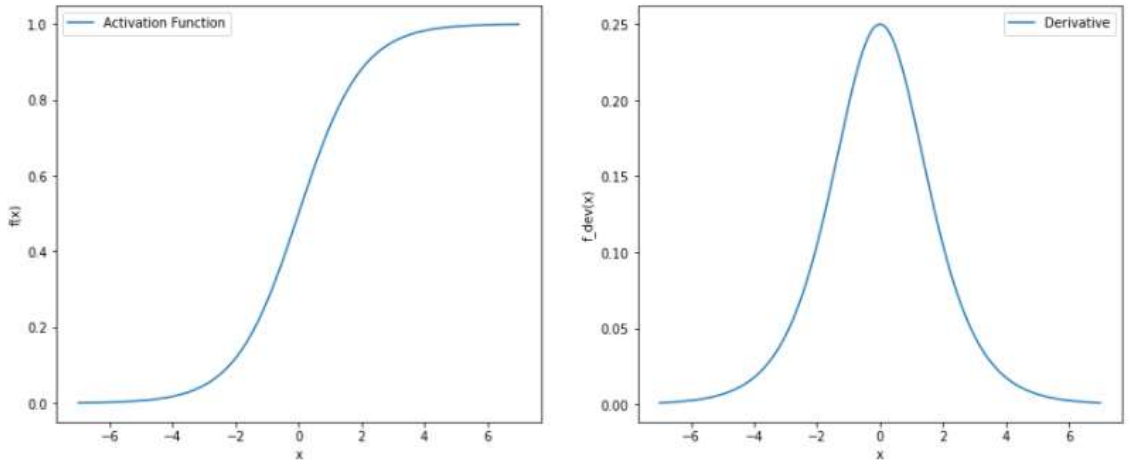


Figure 5. Sigmoid Function and Derivative

In the derivative graph, it is noticeable that for large absolute values of input the derivative approaches to zero and therefore produces the Vanishing Gradient Problem.

The most robust choice regarding the activation function is the *Rectified Linear Unit (ReLu).* Anything with an x-value less than zero has a y-value of zero, but anything with a value greater than zero is mapped to its own y-value. The equation of ReLu is given below:

$$\sigma = \begin{cases} x & for\ x > 0 \\ 0 & for\ x \leq 0 \end{cases}$$

The graph of the ReLu function and its derivative is presented in the figure below:

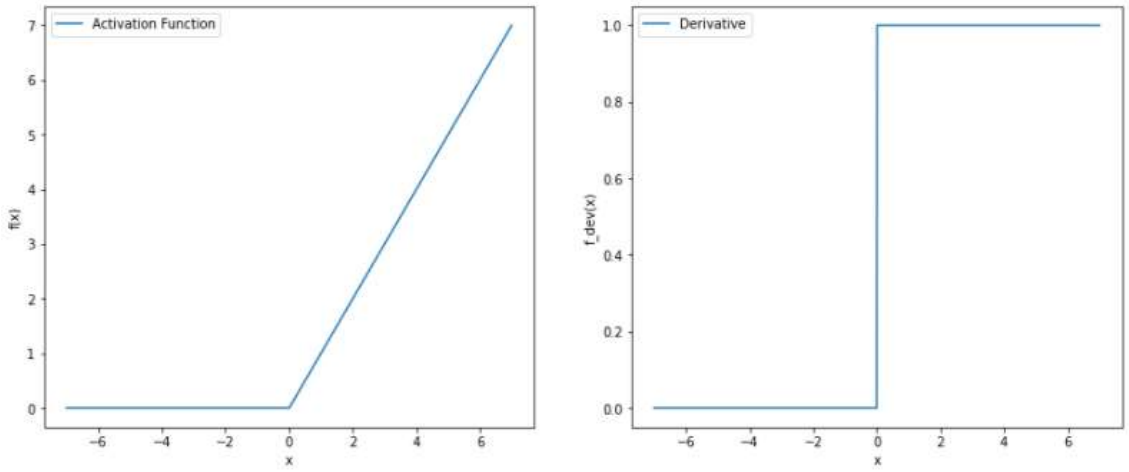


Figure 6. ReLu Function and Derivative

The ReLu activation is one of the most stable options for solving the Vanishing Gradient Problem due to the constant value of the derivative. It does not, however, provide a solution to the Exploding Gradient Problem.

Gradient Descent

Gradient Descent is an optimization method capable of finding optimal solutions to a variety of problems. The general idea to minimize a given cost function $J(\theta)$, parameterized by the model's parameters θ, by changing its parameters iteratively. A determination of the learning process η, is required, which will control the size of the steps that will be taken for approaching the minimum.

Gradient Descent has three modifications that vary in the amount of data used to calculate the gradient of the objective function. Achieving the most accurate calculation of the gradient, and by extend the most accurate update of the parameters of the model, requires the involvement of all the data in our disposal. However, in this case, the most accurate solution is not the most efficient one, regarding the time of the computation. Compromising between a lesser precision and a faster update of the models parameters is preferable.

Batch Gradient Descent is the simplest form of Gradient optimizer. It computes the grad of the loss function with respect to the parameters θ for the entire training set:

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

Although this method assures the convergence to the global minimum, it is very time consuming and it is not recommended.

Mini Batch Gradient Descent updates the parameters for every mini-batch of n training samples, ensuring a safe trade-off between the accuracy of the parameters' update and the execution time.

$$\theta = \theta - \eta \nabla_\theta J\left(\theta; x^{i:i+n}; y^{i:i+n}\right)$$

The convergence towards the global minimum is less stable than the *Batch Gradient Descent*, but the number of data that are included in a mini batch can seriously affect this stability as they increase. *Mini Batch Gradient Descent* is a common choice.

Adam's optimizer is another approach that computes adaptive learning rates for each parameter. It does so by combining different techniques from other algorithms, such us *Momentum []* and *Adagrad []* and *RMSprop*. In the Momentum algorithm the approach of the minimum is accelerated by adding a fraction γ of the update vector of the past time step to the current update vector (gradient calculation):

$$\theta_t = \theta_t - \eta \nabla_{\theta_t} J(\theta_t) + \gamma \sum_{\tau=1}^{t} \eta \nabla_{\theta_\tau} J(\theta_\tau)$$

If the momentum becomes too heavy, it may lead the model to swing back and forth between the local minima.

Adagrad's key characteristic is that reduces the learning rate relative to the features frequency of the data, at every epoch. It assigns low learning rates to parameters linked to regularly occurring features and high learning rates to parameters linked to infrequently occurring features:

$$\theta_{t+1,\iota} = \theta_{t,\iota} - \frac{\eta}{\sqrt{\varepsilon + \sum_{\tau=1}^{t} \left(\nabla_{\theta_\tau} J\left(\theta_{\tau,i}\right)\right)^2}} \nabla_{\theta_t} J\left(\theta_{t,i}\right)$$

ε it's just a small value that ensures that we don't divide by zero.

The decaying learning rate ensures a faster convergence as it avoids the overstepping of the local minimum with big steps. At some point the gradients become so small that momentum becomes stale.

Root Mean Squared Propagation (*RMSprop*) is similar to Adagrad in that it provides an exponentially decaying average rather than the sum of the gradients. One interesting property of RMSprop is that it is not limited to the number of previous gradients, but rather to gradients of the most recent time stages:

$$\theta_{t+1,\iota} = \theta_{t,\iota} - \frac{\eta}{\sqrt{\varepsilon + E[g^2]_t}} \nabla_{\theta_t} J(\theta_{t,i}) \ , \ \ where\ E[g^2]_t = (1-\gamma)g^2 + \gamma E[g^2]_{t-1}$$

The combination of all the ideas above construct the *Adams* optimizer, which is probably one of the most reliable methods that is employed so far. The equations that describe the Adam optimizer are the following:

$$\theta_{t+1} = \theta_t - \frac{\eta \cdot \widehat{m_t}}{\sqrt{\widehat{u_t}} + \varepsilon}$$

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$$

$$\widehat{u_t} = \frac{u_t}{1-\beta_2^t}$$

$$m_t = (1-\beta_1)g_t + \beta_1 m_{t-1}$$

$$u_t = (1-\beta_2)g_t^2 + \beta_2 u_{t-1}$$

Where ε is a small term ($\sim 10^{-8}$), Learning rate η (the recommended default value is $\eta = 0.001$).

Adam's optimizer is the most typical and reliable choice in ANNs applications. β_1 and β_2 represent the forgetting factors for gradients and second moments of gradients and they are the first and the second momentum terms respectively, which are set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The value of β_1^t is given by raising the value of β_1 to the power of the time step t.

In essence, the hyperparameters of a neural network are the parameters that do not change during the training process as they describe fundamental parts and of the ANN. The most basic hyperparameters are listed below:

- Number of Hidden Layers
- Number of Neurons in Each Layer
- Activation Function (Step Function):
- Optimizer: Executes Gradient Descent
- Number of Epochs: It defines the number of repetitions that will be performed by backpropagation algorithm
- Loss function: Cost Function quantifies the error between predicted values and expected values. Depending on the problem the form of the Cost Function can vary.
- Batch Size: The number of data that will be fed into the NN

Transfer Learning

Transfer learning is a predictive modeling technique that can be used to speed up training and enhance the performance of a model on a different but quite similar problem. In deep learning, this means reusing the weights in one or more layers from a previously trained model in a new model. The new model will either holding the weights constant, fine tuning them, or completely adapting the weights during training.

Weight Initialization and Feature Extraction are the two primary methods for applying transfer learning. The weights in re-used layers can be used to start the training process and then adapted to the new task. Transfer learning is referred to as a weight initialization scheme in this context. When the first associated problem has a lot more labeled data than the problem of interest, and the structure of the problem is identical in both cases, this can be beneficial. Conversely, the weights of the network will not be adjusted in response to the new challenge, and only new added layers may be trained to analyze the output after the reused layers have been trained. Transfer learning is referred to as a feature extraction scheme in this case. Variations on these scenarios include not initially training the model's weights on the new issue, but later fine-tuning all weights of the trained model with a low learning rate.

Artificial Neural Networks for Timeseries Forecast

The timeseries forecasting is at its core a regression task, meaning that the model tries to best fit the inputs and make predictions. The goal is to approximate the predictions from new instances that were not included in the training.

Sliding Window

In most cases the input data are introduced into the model in a sliding window format. The window is an object that holds a certain number of instances by preserving their time sequence. The target values are also in a window format with a different number of instances. At the end of the day, the timeseries forecasting task is defined by the length of the input window, the length of the target window and their distance in between which is defined by the number of observations that mediate from the end of the input window to the beginning of the target window.

Let us examine a basic example for simplicity. Assuming a timeseries $a(t) = a_t$. In order to create the dataset that will be fed into the NN we are going to employ the sliding window strategy. Firstly, we set the desirable window lengths and distances. Let us suppose that the input window has a length equal

to 3 and the target window has a length equal to 2. The corresponding distance between them will be 2. The first instance in the new dataset will be an input vector (window) that holds the values a_0, a_1, a_2 and a target vector that holds the values a_5, a_6. The next instance in the new dataset, assuming that we set a *step=1*, is an input vector (window) that holds the values a_1, a_2, a_3 and a target vector that holds the values a_6, a_7, and so on.

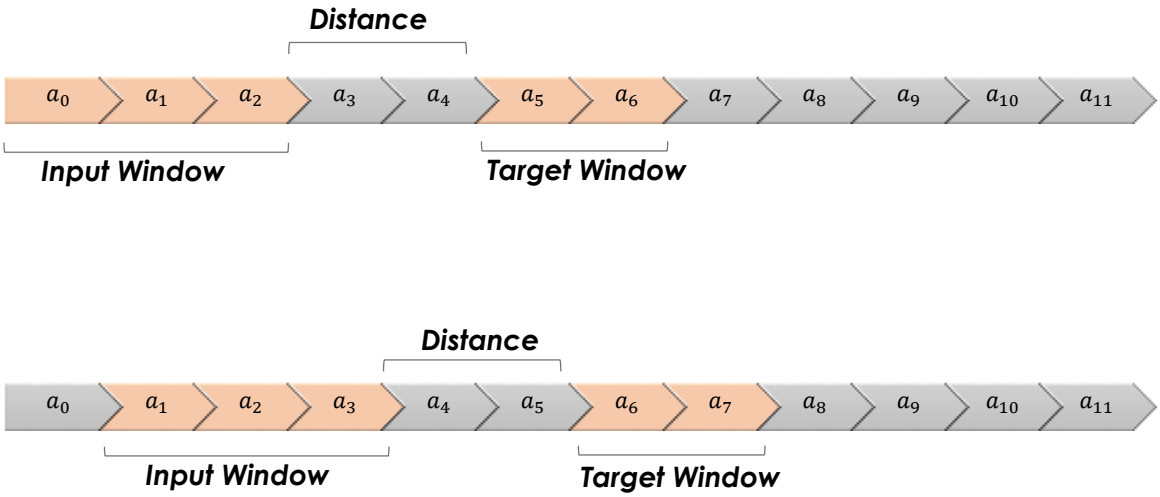


Figure 7. Example of sliding window

Neural Network Architectures

There are a variety of neural network architectures available, each with its own set of features that are ideally suited to specific applications. Below are some of the most well-known architectures, especially in the context of time series forecasting.

Feed Forward Neural Networks

The model that was previously introduced as a standard model for Neural Networks in the figure 4 is in fact a specific type of NN model, called *Feed Forward Neural Network* or *Multilayer Perceptrons (MLPs)*. The goal of a feed forward network is to approximate a function f. According to this architecture, the information moves in only forward direction without loops in between. The information passes from the input nodes, through the hidden nodes (if any) and to the output nodes.

For a variety of purposes, this particular ability is useful for time series. Neural networks are resistant to noise in the input data and the mapping function, and they can also learn and make predictions in

the absence of data. Furthermore, neural networks make no firm assumptions regarding the mapping function and can learn linear and nonlinear interactions easily.

More precisely, in the approximated function, neural networks can be programmed to support an arbitrary specified but fixed number of inputs and outputs. This implies neural networks can handle multivariate inputs and multi-step forecasts directly. Multivariate Inputs refers to the ability to specify an integer number of input elements, allowing for direct support for multivariate forecasting. Multi-step Forecasts, on the other hand, refers to an arbitrary number of output values that can be specified, providing direct support for multi-step and even multivariate forecasting.

For these capabilities alone, feedforward neural networks may be useful for time series forecasting. The assumption of a reasonable mapping from inputs to outputs is implicit in the use of neural networks.

Convolutional Neural Networks

CNNs, or Convolutional Neural Networks, are a version of neural network that was created to process image data efficiently. They also shown their effectiveness on difficult computer vision issues, producing state-of-the-art outcomes on challenges such as image classification and object detection. CNNs have been proven to be also very effective in timeseries data. CNNs provide many of the advantages of Multilayer Perceptrons for time series forecasting, such as multivariate input, multivariate output, and understanding random and dynamic functional relationships. At the same time, they can process the information through the innovative approach of *Feature Learning*. Feature Learning is the automatic identification and extraction of relevant features from raw input data that pertain directly to the prediction problem that is being modeled.

Similar to MLPs, CNNs may contain one or more hidden layers, however each neuron is connected only to neurons that are contained within its receptive field. The receptive field is a region of the input timeseries where a filter (of the same size) can be applied. The distance between two receptive fields is called a stride. A 1D function map is generated by applying (convolving) the same filter to the entire timeseries. Different filters provide different feature maps, which are then combined to form a convolutional layer. Each convolutional layer is an entity that contains all of the previous layer's feature maps. The weight of a layer is the value of a feature map's pixel.

A pooling layer is needed in between each convolutional layer due to the computational complexity resulting from this architecture. A pooling layer is created by adding a pooling kernel to each of the function maps and reducing the layer's dimensions (length). Let us consider a max pooling kernel of length 2 and a stride of 2, which inputs 2 values of the time series and outputs the maximum. This simple kernel will reduce the size of the sequence by a magnitude of 2. A CNN usually has many convolutional and pooling layers that are added one after the other. As a consequence, the convolutional layers' spatial dimensions are reduced while their depth dimensions' increase.

Finally, a MLP with a few completely connected layers is introduced, and the prediction is output by the final layer. CNNs perform better in timeseries classification tasks since they are better at identifying low and high level features in sequences. Nonetheless, their ability to recognize complex patterns may be beneficial for forecasting.

A typical convolutional Neural Network is presented in the figure below.

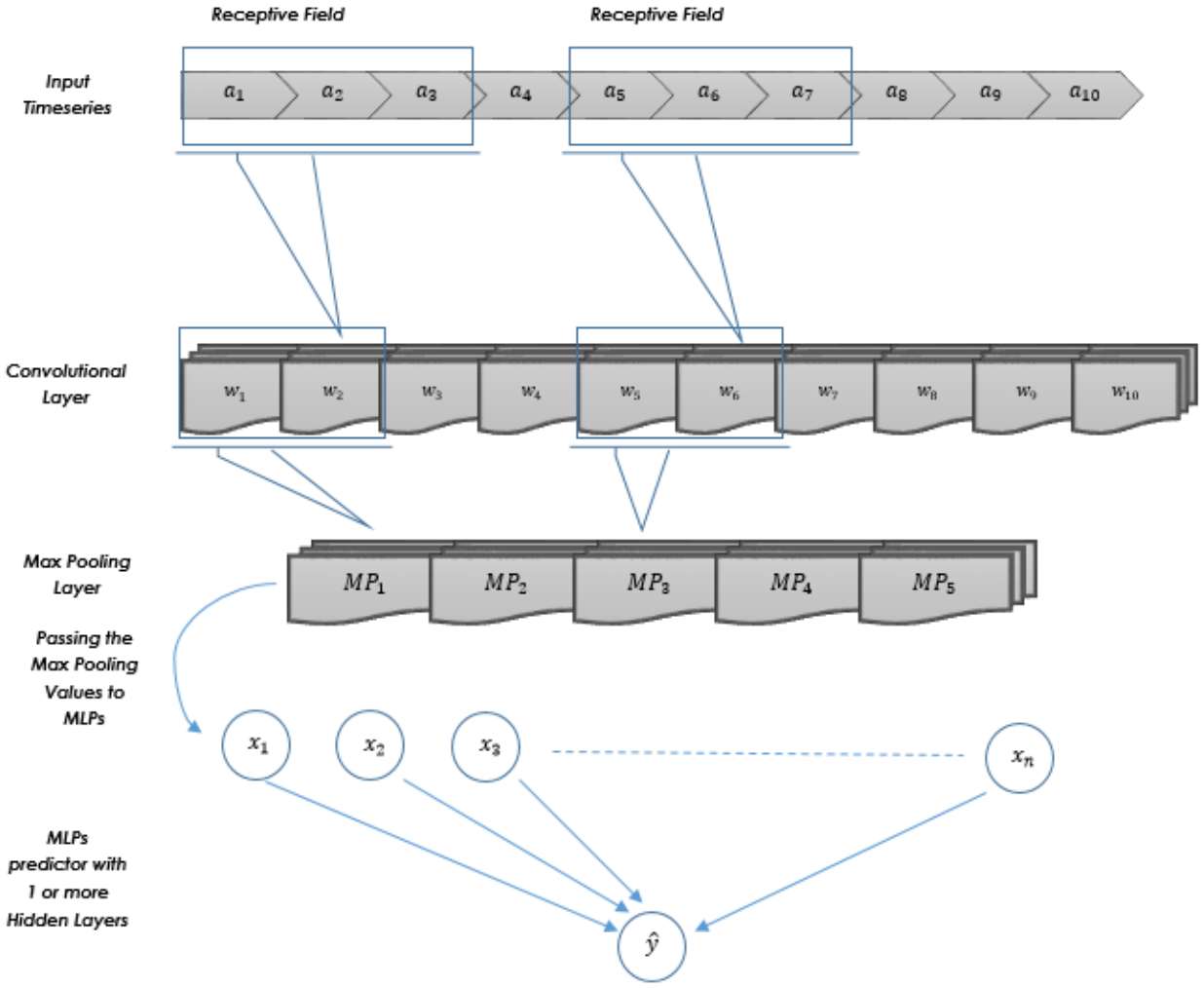


Figure 8. A Typical representation of Convolutional Neural Network model

Recurrent Neural Networks

Recurrent Neural Networks (RNN) are networks of neuron-like nodes arranged into subsequent layers, analogous to regular Neural Networks in architecture. Neurons are classified into input, hidden, and output layers, much as in standard Neural Networks. Each neuronal connection has a trainable weight associated with it.

The distinction is that each neuron is allocated to a particular time step. The neurons in the hidden layer are also forwarded in a time-dependent direction, which ensures that each of them is totally linked only to the neurons in the hidden layer with the same allocated time step, and is connected to any neuron assigned to the next time step by a one-way link. This results that the activation of the neurons is calculated in time order so the output of one-time step's hidden layer is part of the input of the next time step. At any given time-step, only the neurons assigned to that time step compute their activation.

The architecture of a typical RNN is described below:

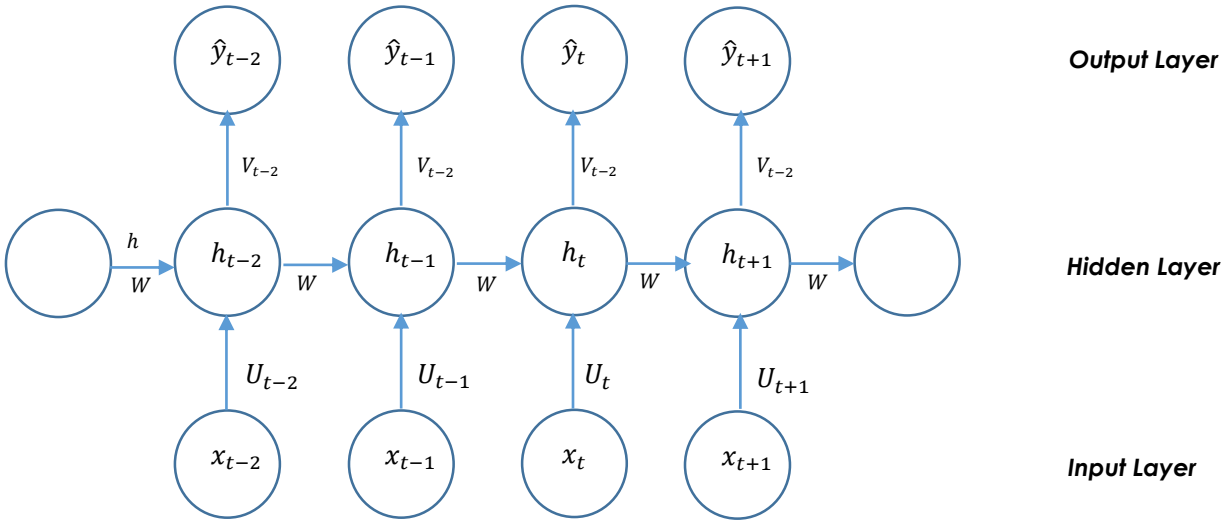


Figure 9. A Typical representation of a Recurrent Neural Network

Where x_t is the values of the input timeseries that are connected only through the weights U_t with the hidden layer that correspond to the same instance t. $\hat{y}_t$ denotes the output of the network. The hidden layer is linked with the output layer through the weight V_t and it is connected with its neighbor nodes of the hidden layer through the weights W. The hidden state of the network is represented as h_t, which marks a type of memory of the network and it is computed from all previous values, up to the current moment t.

The hidden state and the output of the network are given from the equation below:

$$h_t = \sigma_1(W h_{t-1} + U_t x_t)$$

$$\hat{y}_t = \sigma_2(V_t h_t)$$

The learning process' aim is to find the best weight matrices U, V, and W that provide the best prediction of $y(t)$ of the real value $y(t)$ starting from the input $x(t)$. Similar to feed forward Neural Networks, the way to achieve that is through backpropagation algorithm. Let us consider a cost function L. The model will calculate the contribution of its weight for the output value, by employing partial derivatives and then adjust the weights accordingly, in order to minimize L. The partial derivatives that will be computed are the followings:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial W} = \frac{\partial L}{\partial \hat{y}_t} \cdot (\sigma_2^{'} \cdot V_t)\ (\sigma_1^{'} \cdot h_{t-1})$$

$$\frac{\partial L}{\partial U_t} = \frac{\partial L}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial U_t} = \frac{\partial L}{\partial \hat{y}_t} \cdot (\sigma_2^{'} \cdot V_t)(\sigma_1^{'} \cdot x_t)$$

$$\frac{\partial L}{\partial V_t} = \frac{\partial L}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial V_t} = \frac{\partial L}{\partial \hat{y}_t} \cdot (\sigma_2^{'} \cdot h_t)$$

The corresponding updates of the weights will be given from:

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$

$$U_{new} = U_{old} - \eta \frac{\partial L}{\partial U_{old}}$$

$$V_{new} = V_{old} - \eta \frac{\partial L}{\partial V_{old}}$$

Recurrent Neural Networks often suffer from exploding gradient or vanishing gradient problems as a result of their dependence on several partial derivatives, especially the $\frac{\partial L}{\partial W}$ that is related to h_{t-1} and by extend to previous inputs.

Gated Recurrent Units - GRUs

The evolution of RNNs came with the implementation of *Long-Short Term Memory (LSTMs)*. A variation of LSTM that effectively managed to elude the issues regarding to the vanishing gradient is the Gated Recurrent Units (GRUs) and concerns the model that will be employed on this thesis.

The innovation of this model is to use a GRU unit in place of the hidden layer. GRU employs the update and reset gates to solve the vanishing gradient problem of a regular RNN and LSTM. These two gates determine what data should be sent to the output. These two gates can be trained to hold information from several time steps before the actual time step without removing it through time, or to exclude unnecessary information from the forecast.

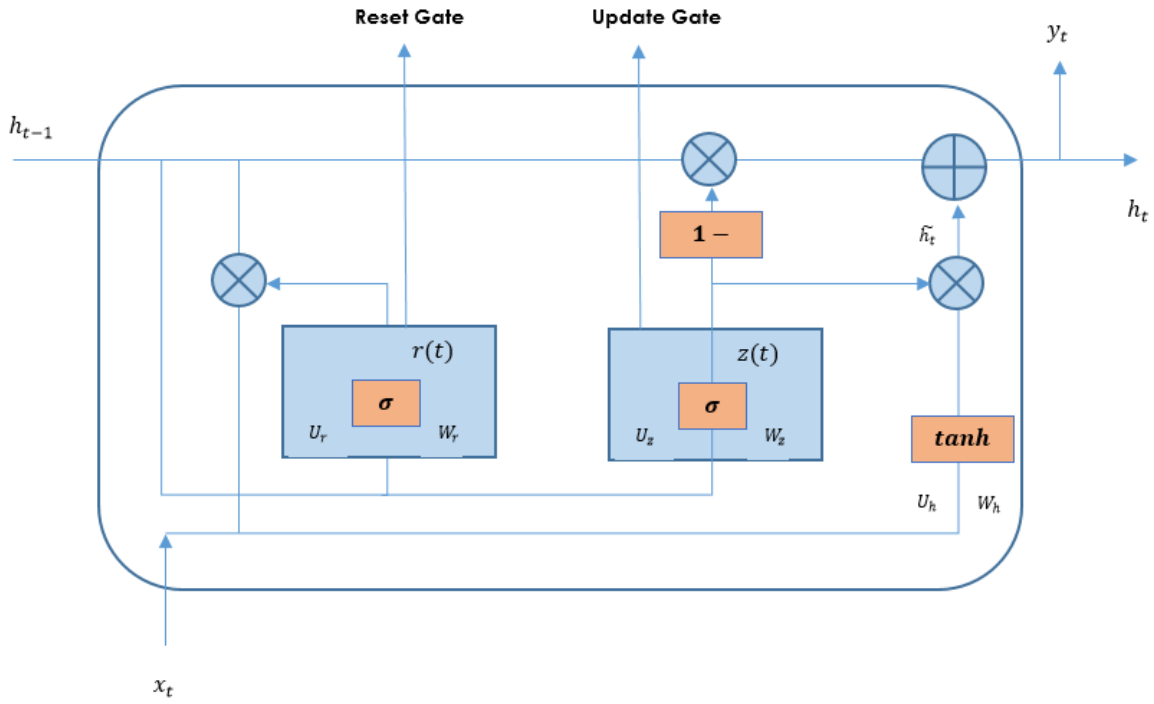


Figure 10. GRU unit

The reset gate manages what amount of information should be forgotten whereas update gate manages what amount of information should be saved. The responses of the reset and update gate are given the weighed sum of the input x_t and the memory from the previous unit h_{t-1}:

$$r_t = \sigma(x_t U_r + h_{t-1} W_r)$$

$$z_t = \sigma(x_t U_z + h_{t-1} W_z)$$

The memory h_t of the unit is calculated firstly by computing a "*current memory*" parameter ($\widetilde{h_t}$) which ignores the response z_t, and then take it into account for the calculation of the final memory h_t.

$$\widetilde{h_t} = \tanh(\mathrm{x_t U_h} + (\mathrm{r_t} * \mathrm{h_{t-1}})\mathrm{W_h})$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \widetilde{h_t}$$

Where $*$ indicates a multiplication element by element.

Methodology

Dataset

For the purposes of this thesis 4 different datasets were employed. Both datasets concern timeseries of wind elements and other related meteorological parameters that were estimated from meteorological stations located for several heights and positions in wind farms across Greece. The dataset's main differences are traced in the sample step, in the locations of the wind farms and in the amount of parameters possessed.

Dataset 1 – Loupounaria (1 hour)

The first dataset concerns a wind farm located in Loupounaria in Greece. The values were measured with a sampling step of 1 hour and the total amount of instances reaches 7768. The timeseries involve a time period that extends from 29th of May 2019 (00:00) to 31th of May 2020 (22:00).

There are 38 variables in the dataset and they are the followings:

- ***Time1***: The moment for each observation in a DD-MM-YY HH:MM format
- ***(W1Speed10 - W1Dir10) - (W5Speed10 – W5Dir10)***: the Wind speed $[\mathrm{m/s}]$ and direction [Degrees] estimated from meteo services. The first number next to W (1-5) denotes a different location and the number at the end (10,80) denotes the height of a given meteorological station in meters.
- ***(W1Speed80 - W1Dir80) - (W5Speed80 – W5Dir80)***
- ***(D1flux, W1Dens) - (D5flux, W5Dens)***: The airflow flux and density estimated from meteo services
- ***LouNRG:*** The energy generated from the wind farm
- ***LouDIR, LouWSav***: The average of the actual direction and wind speed measured by anemometers installed in the wind turbines.
- ***NRGsc, LouWSsc***: Scaled values of *LouNRG, LouWSav* between 0 and 1.
- ***LouDIR2, LouWSsc2, NRGsc2***: Scaled values of *LouDIR, LouWSav and LouNRG* between 0 and 1.

Some basic statistical properties of the dataset are presented in the table below:

	mean	*Std*	*min*	*max*
W(1-5)Speed10	5.476527	3.954184	0.058727	23.95064
W(1-5)Dir10	167.8802	127.7819	0.05477	359.9649
D(1-5)flux	218.3796	310.6143	0	1032.608
W(1-5)Dens	1.113748	0.034887	1.033094	1.20655
W(1-5)Speed80	7.614136	5.091163	0.07226	29.72372
W(1-5)Dir80	167.8281	127.7683	0.027986	359.9705
LouNRG	166.4486	188.2134	-7.58333	550.3333
LouDIR	121.6993	104.1129	-6.5	550.1667
LouWSav	7.063066	4.572681	0.241667	25.01667
NRGsc	0.33698	0.324506	0.036925	0.998851
LouWSsc	0.281397	0.182179	0.009628	0.99668

Table 1. Table of statistical properties of the Dataset Loupounaria, Sampling Step: [1 hour]

Note that the prefix $W(1-5)$ represents the average of a certain variable over all the different locations.

Dataset 2 – Trikorfo (1 hour)

The second dataset concerns a wind farm located in Trikorfo in Greece. The values were measured with a sampling step of 1 hour and the total amount of instances reaches 7745. The timeseries involve a time period that extends from 29th of May 2019 (00:00) to 31th of May 2020 (23:00).

There are 38 variables in the dataset and they are the followings:

- ***Time1***: The moment for each observation in a DD-MM-YY HH:MM format
- ***(W1Speed10 - W1Dir10) - (W5Speed10 – W5Dir10)***: the Wind speed $[\mathrm{m/s}]$ and direction [Degrees] estimated from meteo services. The first number next to W (1-5) denotes a different location and the number at the end (10,80) denotes the height of a given meteorological station in meters.
- ***(W1Speed80 - W1Dir80) - (W5Speed80 – W5Dir80)***
- ***(D1flux, W1Dens) - (D5flux, W5Dens)***: The airflow flux and density estimated from meteo services
- ***TriNRG:*** The energy generated from the wind farm
- ***TriDIR, TriWSav***: The average of the actual direction and wind speed measured by anemometers installed in the wind turbines.

- ***NRGsc, TriWSsc***: Scaled values of *LouNRG, LouWSav* between 0 and 1.
- ***TriDIR2, TriWSsc2, NRGsc2***: Scaled values of *LouDIR, LouWSav and LouNRG* between 0 and 1.

Some basic statistical properties of the dataset are presented in the table below:

	mean	***std***	***min***	***max***
W(1-5)Speed10	6.718375	4.278848	0.085311	24.39026
W(1-5)Dir10	116.4134	117.1046	0.037314	359.944
D(1-5)flux	216.0778	308.7591	0	1023.651
W(1-5)Dens	1.159659	0.031934	1.087758	1.241597
W(1-5)Speed80	9.459414	5.363492	0.108887	30.15923
W(1-5)Dir80	116.4219	117.1067	0.022345	359.9198
TriNRG	223.7224	185.8438	0	533.3611
TriDIR	160.6577	167.6646	0	531.3611
TriWSav	9.713655	5.671272	0.436973	36.41643
NRGsc	0.435728	0.32042	0.05	0.969588
TriWSsc	0.266128	0.155377	0.011972	0.99771
TriWSsc2	0.266331	0.155121	0.011972	0.99771
NRGsc2	0.435682	0.320426	0.05	0.969588

Table 2. Table of statistical properties of the Dataset Trikorfo, Sampling Step: [1 hour]

Note that the prefix $W(1-5)$ represents the average of a certain variable over all the different locations.

Dataset 3 – Flabouro (1 hour)

The second dataset concerns a wind farm located in Flabouro in Greece. The values were measured with a sampling step of 1 hour and the total amount of instances reaches 7768. The timeseries involve a time period that extends from 29th of May 2019 (00:00) to 31th of May 2020 (22:00).

There are 38 variables in the dataset and they are the followings:

- ***Time1***: The moment for each observation in a DD-MM-YY HH:MM format
- ***(W1Speed10 - W1Dir10) - (W5Speed10 – W5Dir10)***: the Wind speed $[\mathrm{m/s}]$ and direction [Degrees] estimated from meteo services. The first number next to W (1-5) denotes a different location and the number at the end (10,80) denotes the height of a given meteorological station in meters.
- ***(W1Speed80 - W1Dir80) - (W5Speed80 – W5Dir80)***
- ***(D1flux, W1Dens) - (D5flux, W5Dens)***: The airflow flux and density estimated from meteo services

- ***FlampNRG:*** The energy generated from the wind farm
- ***FlampDIR, FlampWSav***: The average of the actual direction and wind speed measured by anemometers installed in the wind turbines.
- ***NRGsc, FlampWSsc***: Scaled values of *LouNRG, LouWSav* between 0 and 1.
- ***FlampDIR2, FlampWSsc2, NRGsc2***: Scaled values of *LouDIR, LouWSav and LouNRG* between 0 and 1.

Some basic statistical properties of the dataset are presented in the table below:

	mean	***std***	***min***	***max***
W(1-5)Speed10	5.820632	3.848273	0.063286	22.41084
W(1-5)Dir10	105.072	98.7736	0.104822	359.945
D(1-5)flux	188.9153	282.6418	0	1013.407
W(1-5)Dens	1.148394	0.037633	1.064799	1.240231
W(1-5)Speed80	8.23178	4.904303	0.079013	27.53922
W(1-5)Dir80	105.0339	98.73798	0.097787	359.957
FlampNRG	216.056	200.1024	-4.97681	575.119
FlampDIR	122.6803	100.8688	0	575.119
FlampWSav	7.437408	4.072574	0.45625	28.00417
NRGsc	0.42251	0.345004	0.041419	1.041585
FlampWSsc	0.264676	0.144931	0.016237	0.99659
FlampWSsc2	0.264846	0.146087	0.016237	0.99659
NRGsc2	0.422457	0.345008	0.041419	1.041585
FlampDIR2	122.6369	100.7378	0	575.119

Table 3. Table of statistical properties of the Dataset Flampouro, Sampling Step: [1 hour]

Note that the prefix $W(1-5)$ represents the average of a certain variable over all the different locations.

Dataset 4 – Loupounaria (10 minutes)

Regarding the second dataset the values were measured with a sampling step of 10 minutes and the total amount of instances reaches 50960.

There are 139 variables in the current dataset and they are the followings:

- ***S4WSpeed80ave – S129WSpeed80ave***: The averaged wind speeds at 80m height for consecutive 10 min instants and for different locations in the grid (S4, S8, etc).
- ***S4WSpeed80ave5 – S129WSpeed80ave5***: The averaged wind speeds raised to the fifth power.
- ***S4WSpeed80diff – S129WSpeed80diff***: The differences in wind speeds at 80m for different locations in the grid (S4, S8, etc)

- ***sDIR80S1WSpeed80ave - sDIR80S129WSpeed80ave***: The averages of sines of predicted wind direction at 80m for different locations in the grid (S4, S8, etc)
- ***cDIR80S1WSpeed80ave - cDIR80S129WSpeed80ave***: The averages of cosines of predicted wind direction at 80m height for different locations in the grid (S4, S8, etc)
- ***sDIR80S1WSpeed80 - sDIR80S129WSpeed80***: The differences of sines of predicted wind direction at 80m height for different locations in the grid (S4, S8, etc)
- ***cDIR80S1WSpeed80 - cDIR80S129WSpeed80***: The differences of cosines of predicted wind direction at 80m height for different locations in the grid (S4, S8, etc)
- ***DENSS1WSpeed80ave***: The averaged wind density at location 1
- ***DENSS1WSpeed80diff - DENSS129WSpeed80diff***: The differences of wind densities for different locations in the grid (S4, S8, etc)
- ***y1***: the scaled energy output
- ***z1***: the scaled average wind speed in the wind farm

Some basic statistical properties of the dataset are presented in the table below:

	mean	***Std***	***min***	***max***
S(4-129)WSpeed80ave	0.011811	1.005798	-1.49344	5.198519
S(4-129)WSpeed80ave5	0.005787	1.014549	-0.23054	31.2948
S(4-129)WSpeed80diff	-0.0002	0.998691	-23.8821	22.83789
sDIR80S(4-129)WSpeed80ave	0.006278	0.562414	-0.99998	0.999972
cDIR80S(4-129)WSpeed80ave	0.007398	0.561204	-0.99997	0.999949
sDIR80S(4-129)WSpeed80	-1.19E-05	0.858858	-1.99976	1.999797
cDIR80S(4-129)WSpeed80	-3.69E-05	0.857486	-1.99978	1.999843
DENSS(4-129)WSpeed80diff	0.001014	0.998608	-22.0729	28.41918
DENSS1WSpeed80ave	0.016659	1.009039	-2.36356	2.675584
y1	0.291565	0.341919	2.10E-05	0.996396
z1	0.232304	0.152418	0.003333	0.931667

Table 4. Table of statistical properties of the Dataset Loupounaria, Sampling Step: [10 minutes]

Neural Network Architectures

For the purposes of this work 3 different Neural Network Architecture were proposed.

Model 1 - GRUs

The first model is composed out 100 GRUs, a Dropout layer and an output node as it is shown below:

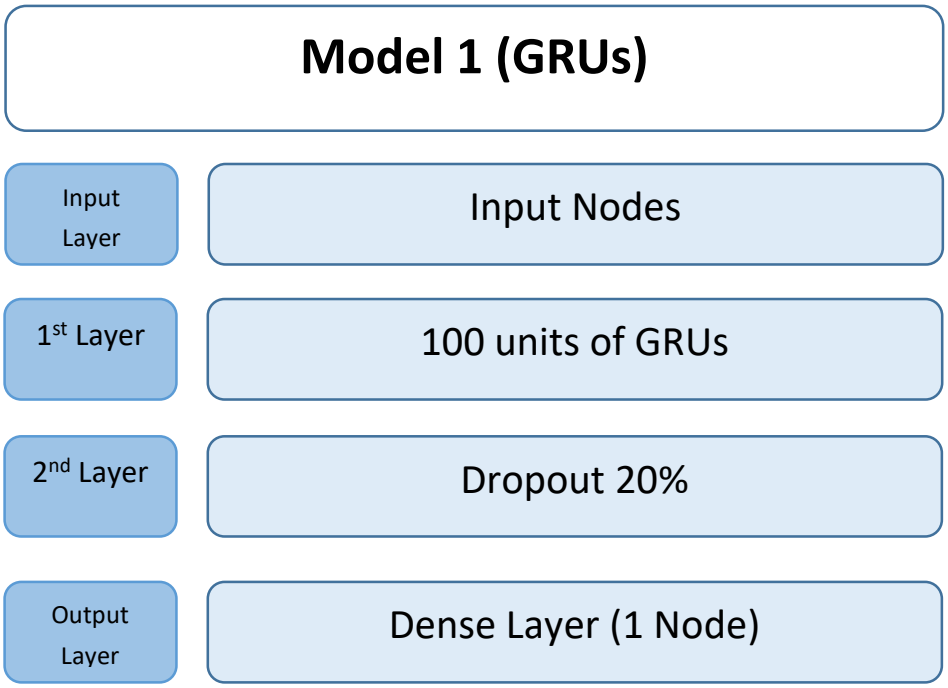
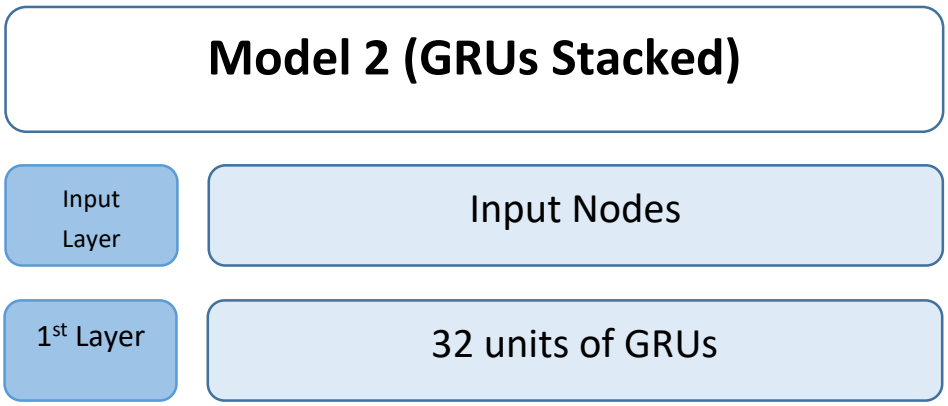


Figure 11. Representation of the NN Architecture: GRUs

Model 2 – GRUs Stacked

The second model is composed out of 4 layers of GRUs and Dropout layers in between them, with 1 output node at the end. The model is presented below:



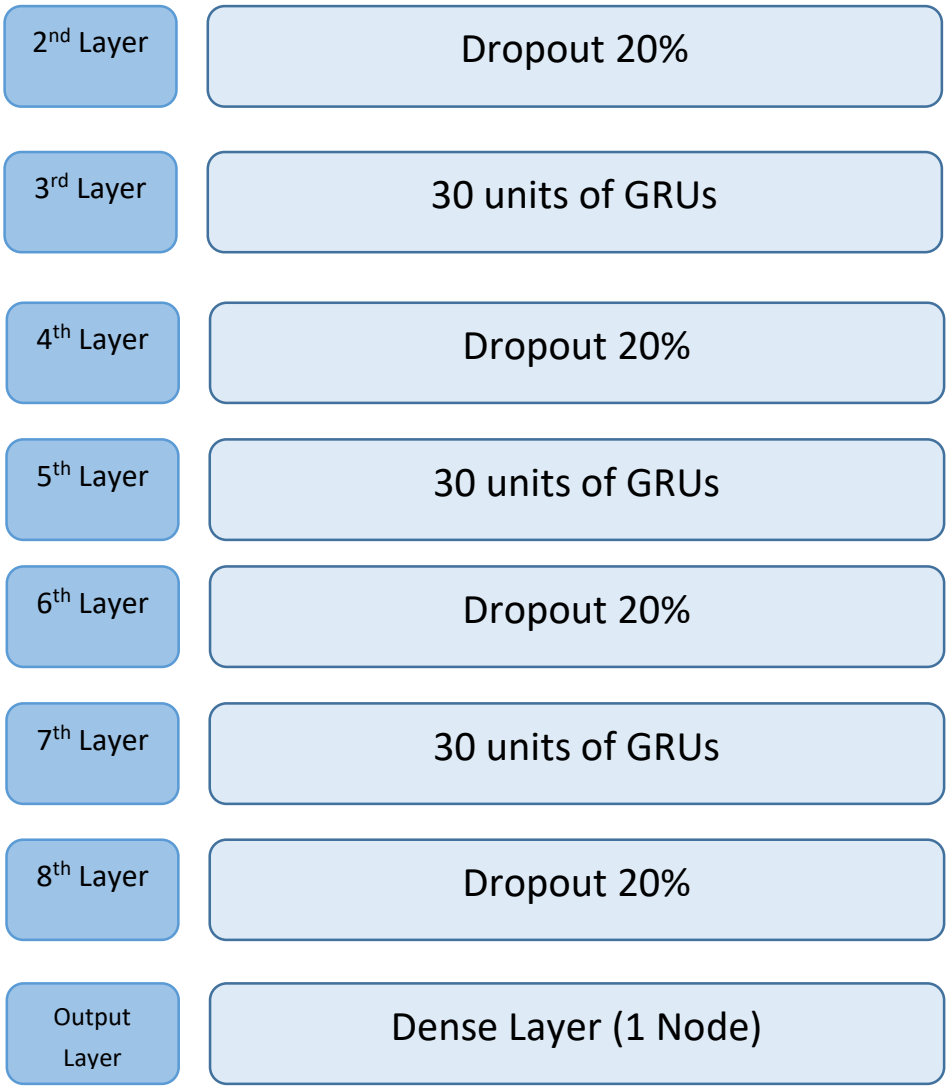
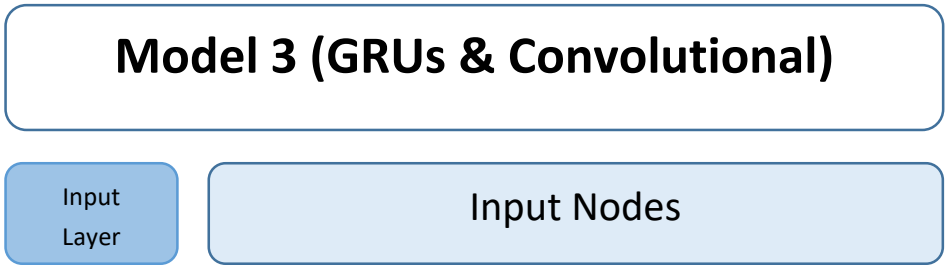


Figure 12. Representation of the NN Architecture: GRUs Stacked

Model 3 – GRUs Convolutional

The third model is composed out of a 1Dimensional Convolutional Layer, a max pooling layer, a GRU Layer, a Flatten Layer, 2 Dense Layers and the output Layer. The model is presented below:



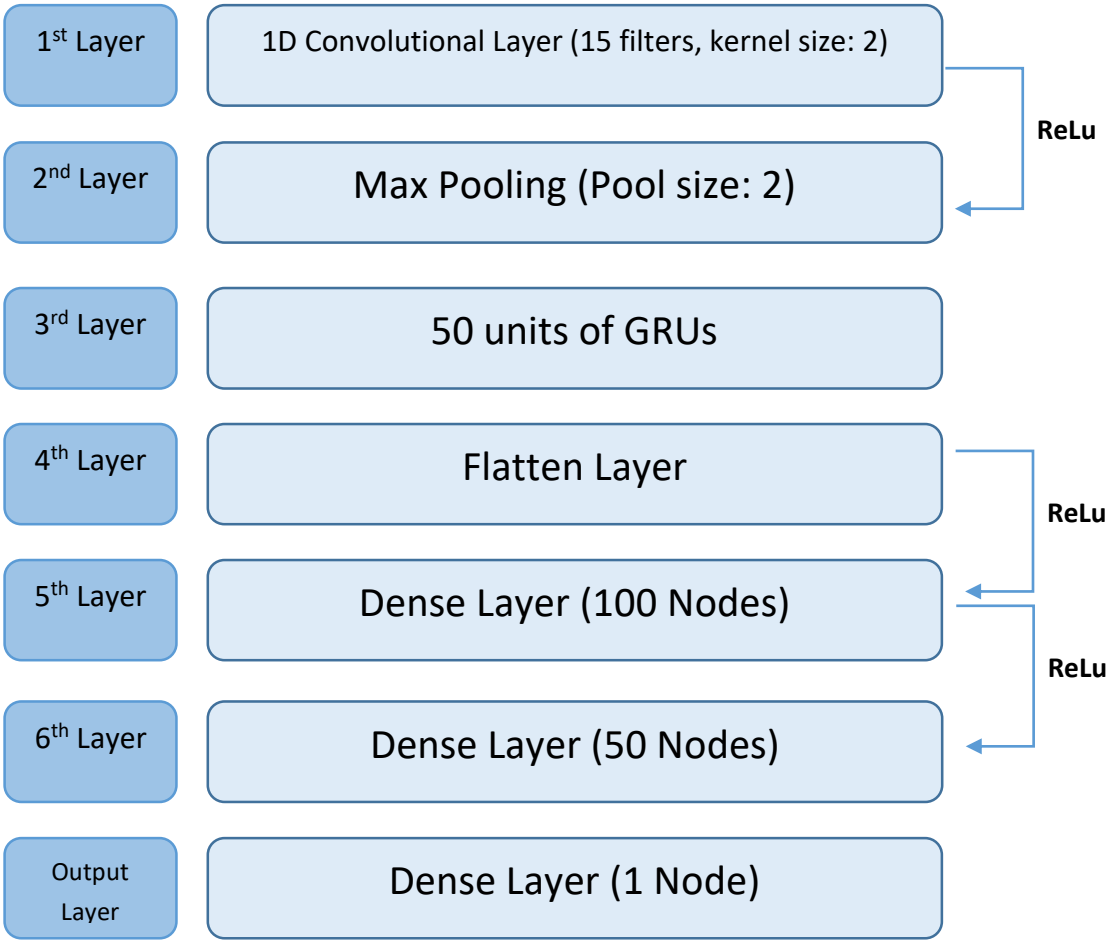


Figure 13. Representation of the NN Architecture: GRUs Convolutional

Study

The aim of this thesis is to forecast the scaled wind speed and investigate the limits of the estimations regarding the prediction horizon. We will employ different datasets and implementing different preprocessing techniques and Neural Network Architectures in order to study the effects of each of those parameters in the forecast and make comparisons.

The first step is to select a dataset and apply preprocessing and feature extraction techniques. For the preprocessing we considered 3 different paths: (a) perform robust scaler and leave the data as they are (*Vanilla*), (b) perform robust scaler and then apply a Gaussian filter with a standard deviation of 4 (*Smoothed*), or (c) perform robust scaler and then implement data augmentation by adding noise as proposed in the section above (*Data Augmentation*). Note that for dataset 4 the variables were previously scaled.

Regarding feature engineering, previous studies of the current problem have shown that by introducing some additional variables of the 3rd or the 5th power of the wind speed improves the performance of the NN models. Moreover, variables other than the wind speed have been proven to have a minor contribution to the quality of the prediction. Wind turbines are usually programmed to rotate with respect to the wind direction of the current moment, thus the effect of the wind direction is insignificant as well. Nevertheless, for the datasets 1 – 3 we took the variable of the wind direction into consideration, in contrast with dataset 4. For the datasets 1 – 3 we substituted the variables of wind speed and wind direction into a vector format as shown below:

$$w_{Speed}\left[\frac{m}{s}\right] = \begin{pmatrix} 3.6 \\ 4.2 \\ \vdots \\ 1.9 \end{pmatrix} \quad \theta_{Direction}[rad] = \begin{pmatrix} 0.5 \\ 4.3 \\ \vdots \\ 5.9 \end{pmatrix} \quad \left.\begin{array}{l} w_x = w_{speed} \cdot \cos(\theta) \\ w_y = w_{speed} \cdot \sin(\theta) \end{array}\right\} \longrightarrow \quad w_x = \begin{pmatrix} 3.1 \\ -1.7 \\ \vdots \\ 1.7 \end{pmatrix} \quad w_y = \begin{pmatrix} 1.7 \\ -3.8 \\ \vdots \\ -0.7 \end{pmatrix}$$

The time variable was implemented only for the first 3 datasets.

For Datasets 0 – 3 the target variables are ***LouWSsc, TriWSsc*** and ***FlampWSsc*** respectively**.** The training variables are the followings:

$W1\_x\_10$	$W1\_y\_10$	$W1\_pow3\_10$	$W1\_x\_80$	$W1\_y\_80$	$W1\_pow3\_80$	…	$W5\_x\_80$	$W5\_y\_80$	$W5\_pow3\_80$

For Datasets 4 the target variable is ***z1***, whereas the training variables are the followings:

$S4WSpeed80ave$	…	$S129WSpeed80ave$	$S4WSpeed80ave5$	…	$S129WSpeed80ave5$

The next step is to slice the dataset into windows in order to feed it to the neural network models. The input window was determined to contain values of a whole day (24 hours). Therefore, for datasets 1 – 3 the input window contained 24 values whereas for dataset 4 the window contained 144 values $(6[10min] \cdot 24[hours])$. The target window contained only 1 value. Then for a given distance between the input and the output windows, we trained the NN models. The distance defines the depth of the prediction horizon, i.e. how far into the future the forecast is performed. By changing the distance and retraining the models, and then saving the relative and absolute errors of each prediction, we investigate the limits of our forecasting models. In this study the distance ranged from 0 to 24 hours.

Lastly, the hyperparameters that were used for training are presented below:

- Optimizer: Adams

- Number of Epochs: 30
- Loss function: Mean Absolute Error
- Batch Size: 50
- Early Stopping: Yes, patience: 15 epochs

The Loss function of the NNs is MAE, however a more realistic indication of the quality of the forecast will be given from the relative absolute error:

$$L_{relative} = \frac{MAE}{\frac{1}{N}\sum_{i=1}^{N}|y_i - \mu|}$$

Where μ indicates the mean value of the set.

Results

An Illustration of the Quality of the Predictions with Respect to the Relative Error

Before diving into the exploration of the predicted horizon we should first get familiar with the concept of Relative Error and how does it relate to the quality of the prediction. In this section we will present some examples of timeseries from the dataset Loupounaria (1 Hour) by the *GRU Stacked* Architecture, and its corresponding predictions for typical values of relative error. The predictions were performed on the test set of the dataset according to different preprocessing techniques that were employed (Vanilla, Smoothed and Data augmentation). The graphs illustrate the quality of the predictions with respect to the relative error. The relative error was calculated from the test set of each Dataset.

Firstly, the graphs with the *Vanilla* preprocessing are presented:

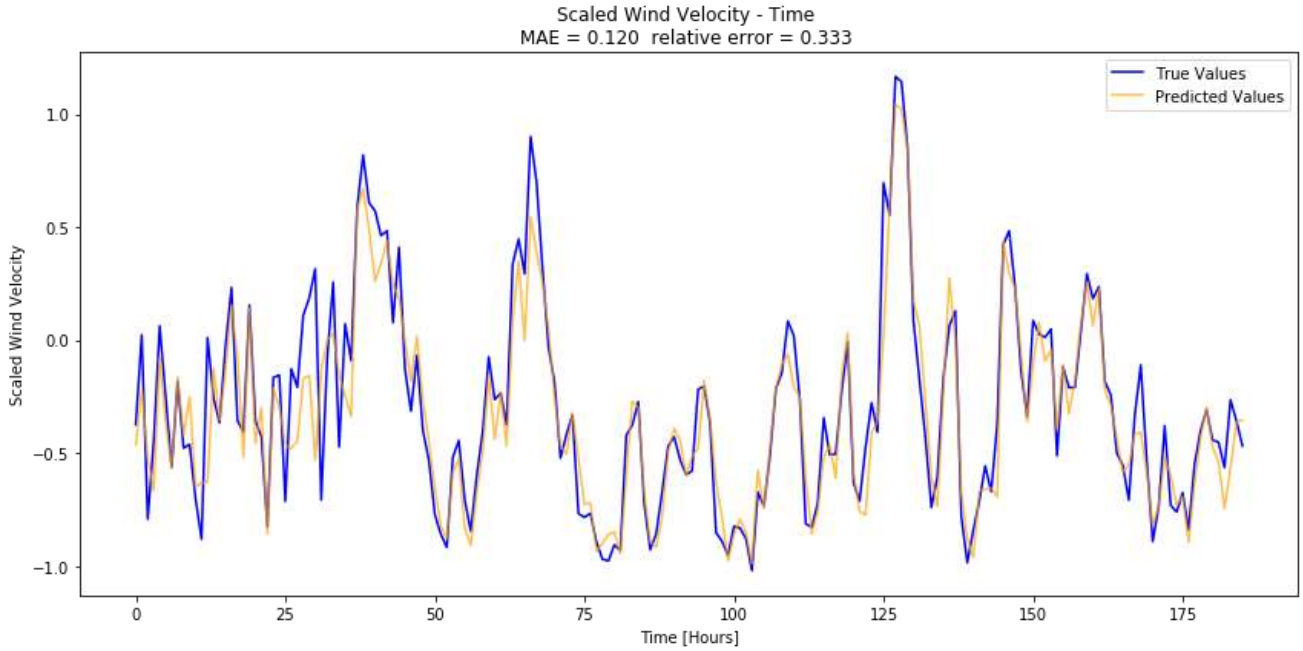


Figure 14. Forecast: 1 [Hours], Preprocessing Technique: Vanilla, NN Architecture: GRUs

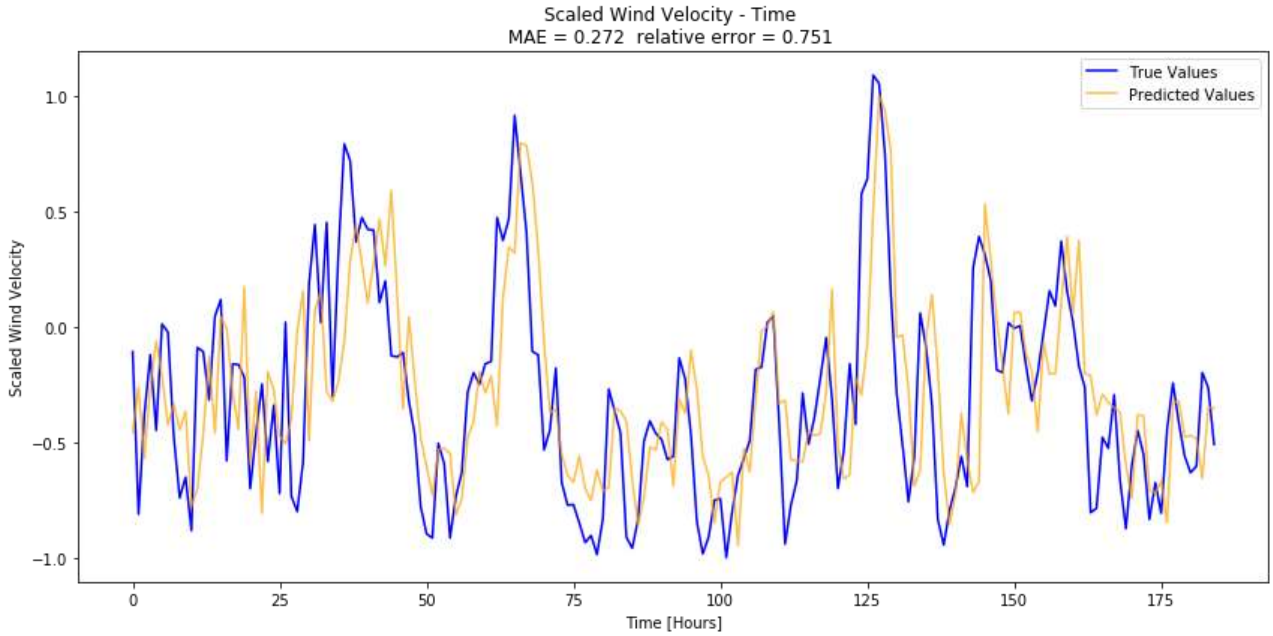


Figure 16. Forecast: 6 [Hours] , Preprocessing Technique: Vanilla, NN Architecture: GRUs

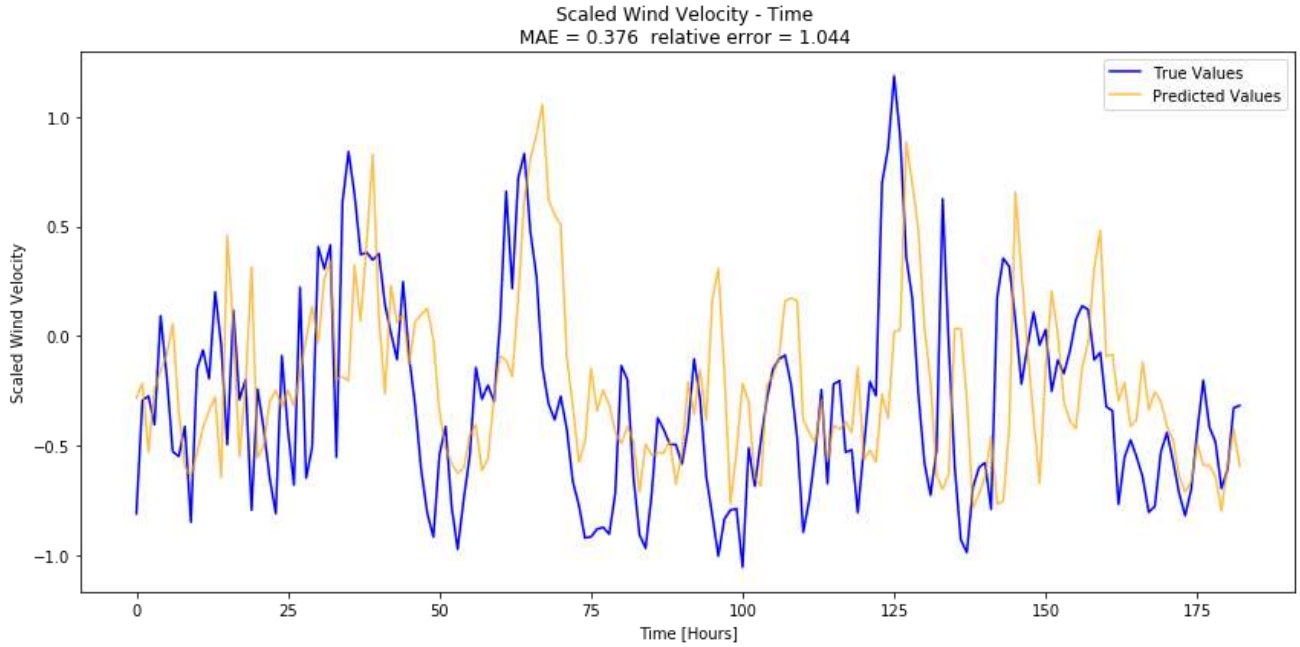


Figure 15. Forecast: 11 [Hours] , Preprocessing Technique: Vanilla, NN Architecture: GRUs

The blue line indicates the true values of the timeseries, whereas the orange line represents the predictions.

The graphs that correspond to the *Smoothed* Preprocessing are depicted below:

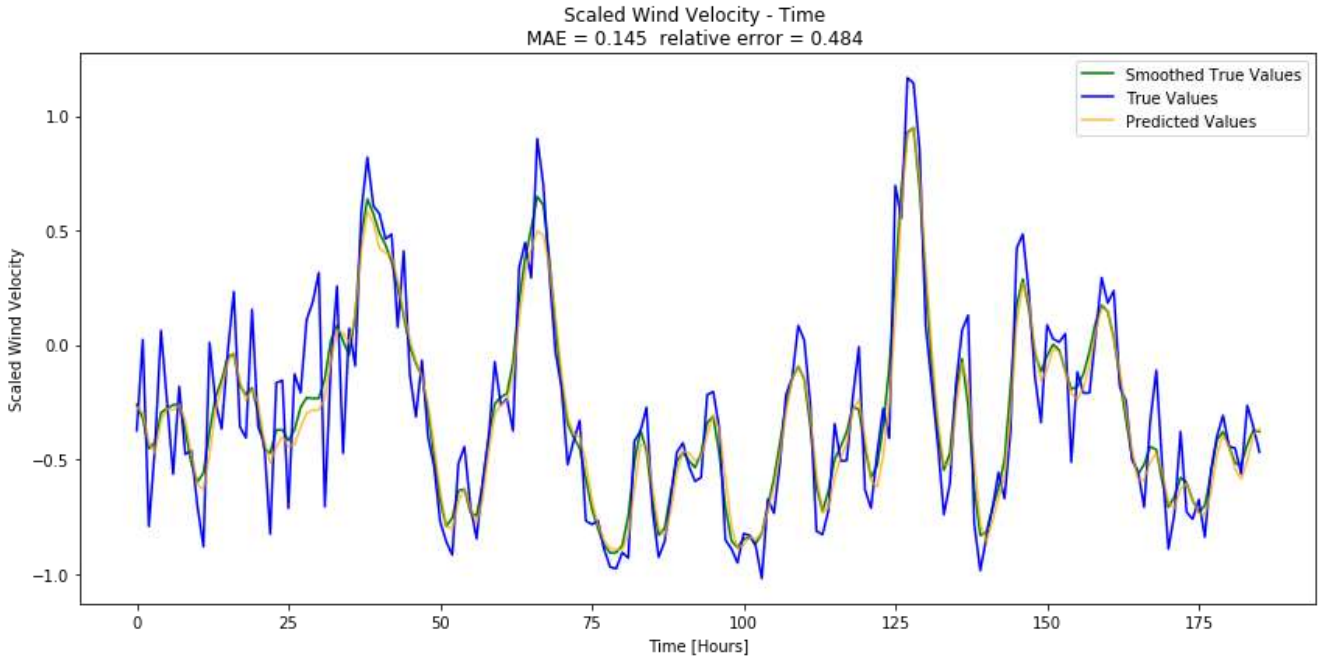


Figure 17. Forecast: 1 [Hours] , Preprocessing Technique: Smoothed, NN Architecture: GRUs

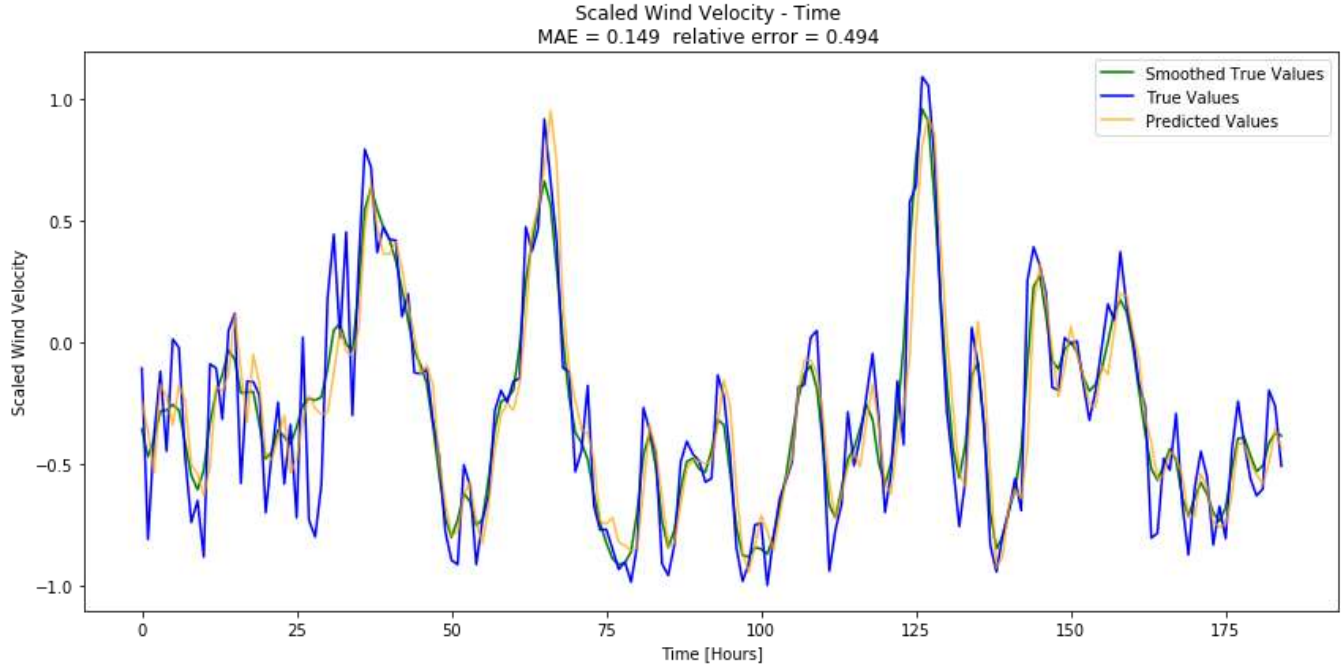


Figure 18. Forecast: 6 [Hours] , Preprocessing Technique: Smoothed, NN Architecture: GRUs

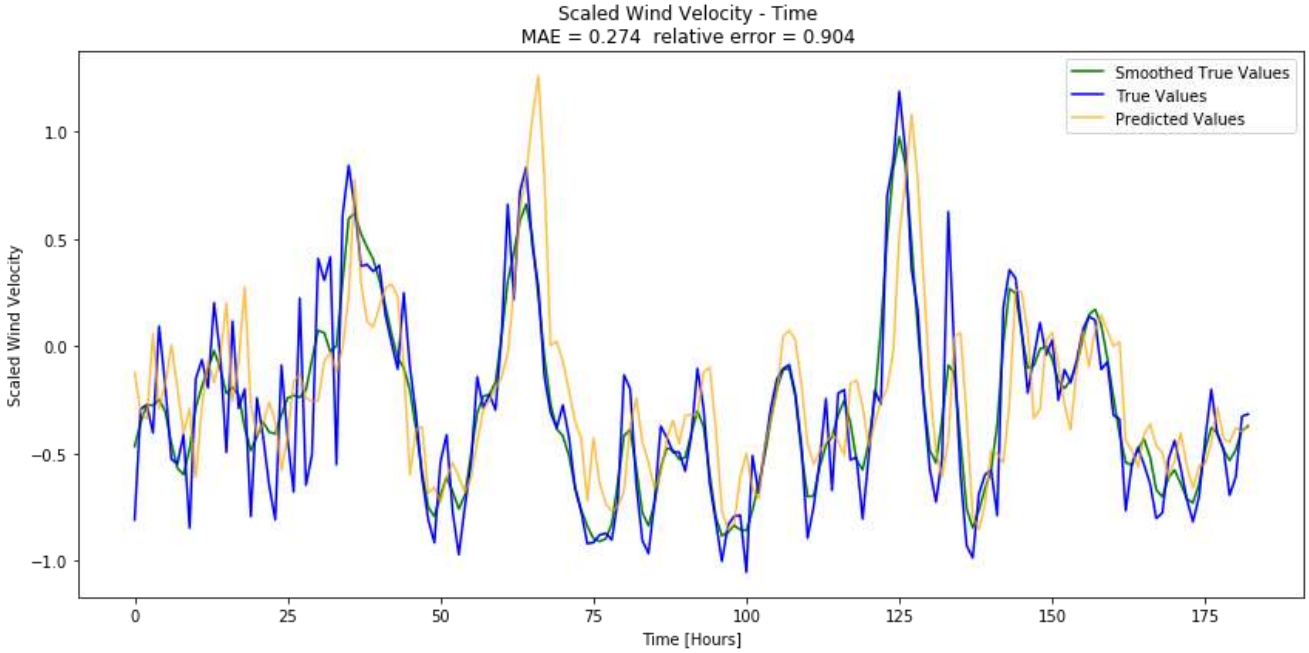


Figure 19. Forecast: 11 [Hours] , Preprocessing Technique: Smoothed, NN Architecture: GRUs

Note that for the graphs that correspond to the *smoothed* preprocessing technique, the model was initially trained in a smoothed timeseries Therefore, the predictions (*orange*) were produced by receiving as input the smoothed true values (*green*) as well. However, the error was calculated with respect to the original true values (*blue*) with no smoothing implemented.

The graphs that correspond to the *Data Augmentation* preprocessing technique are presented below:

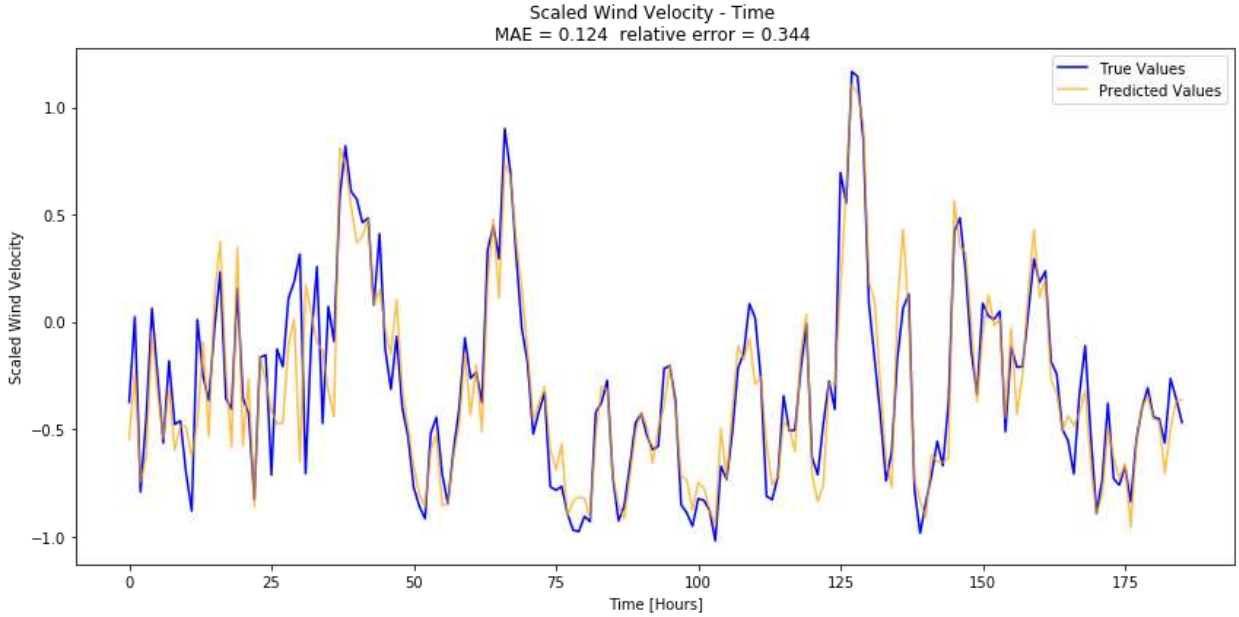


Figure 20. Forecast: 1 [Hours] , Preprocessing Technique: Data Augmentation, NN Architecture: GRUs

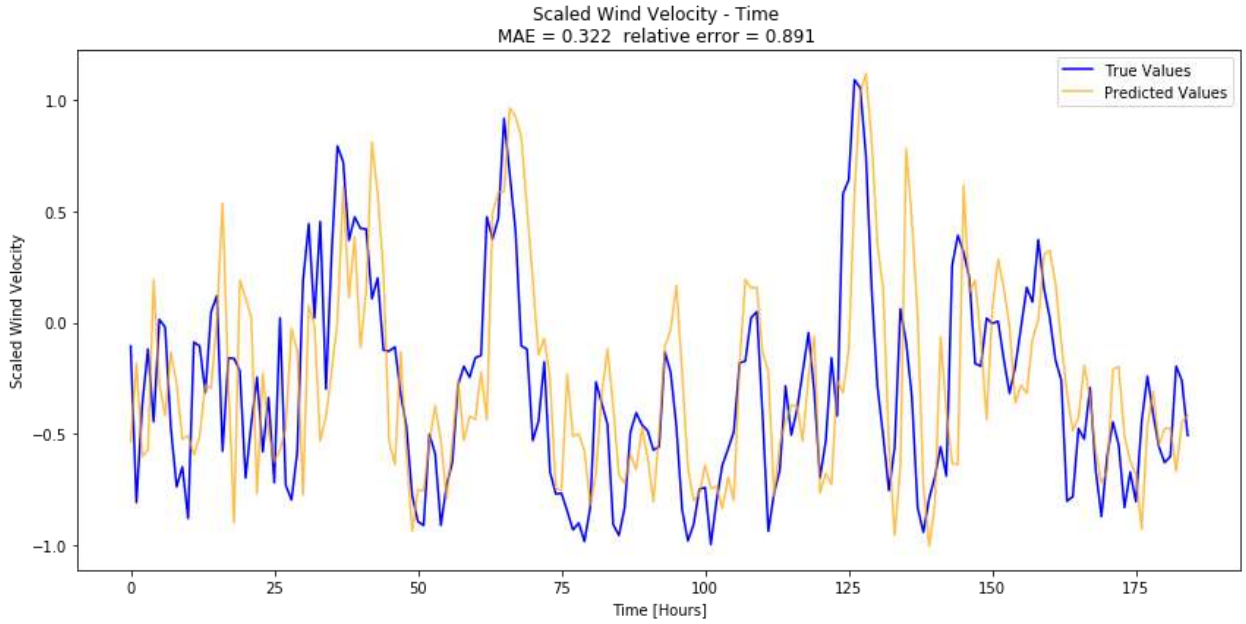


Figure 21. Forecast: 6 [Hours] , Preprocessing Technique: Data Augmentation, NN Architecture: GRUs

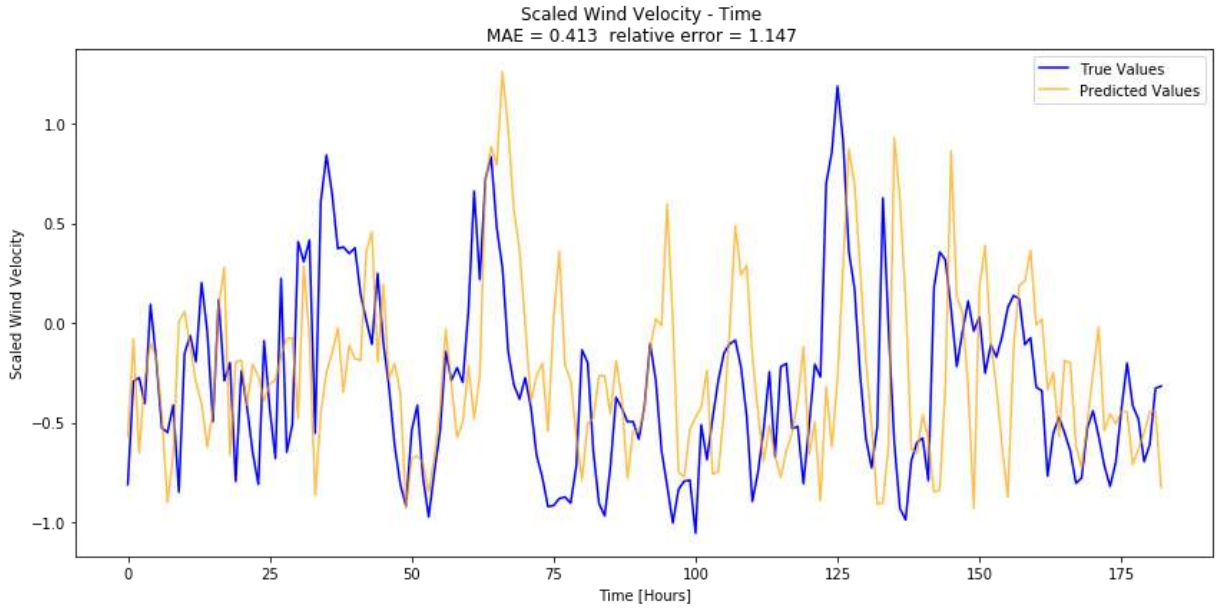


Figure 22. Forecast: 11 [Hours] , Preprocessing Technique: Data Augmentation, NN Architecture: GRUs

For the graphs that correspond to the *Data Augmentation* preprocessing technique, the model was initially trained in a combined set of original and augmented data. The Augmented Data were only used to increase the amount of the training set and they have no purpose whatsoever in evaluating the predictions. Therefore, the test set only contained the true values (*blue*) with no data

augmentation implemented. Given that as an input the predictions (*orange*) were evaluated with respect to the original true values (*blue*).

Comparison of Preprocessing Techniques

We will first present the graphs of Relative Error in relation to Prediction Horizon that illustrate a clear comparison between the preprocessing techniques that were employed. Each set of graphs represents a different dataset (Flampouro, Loupounaria or Trikorfo) and contains 3 graphs which each represents a different NN Architecture (GRU, GRUs Stacked or GRUs Convolutional). In every graph, 3 lines are depicted for each preprocessing technique respectively (Vanilla, Smoothed and Data augmentation).

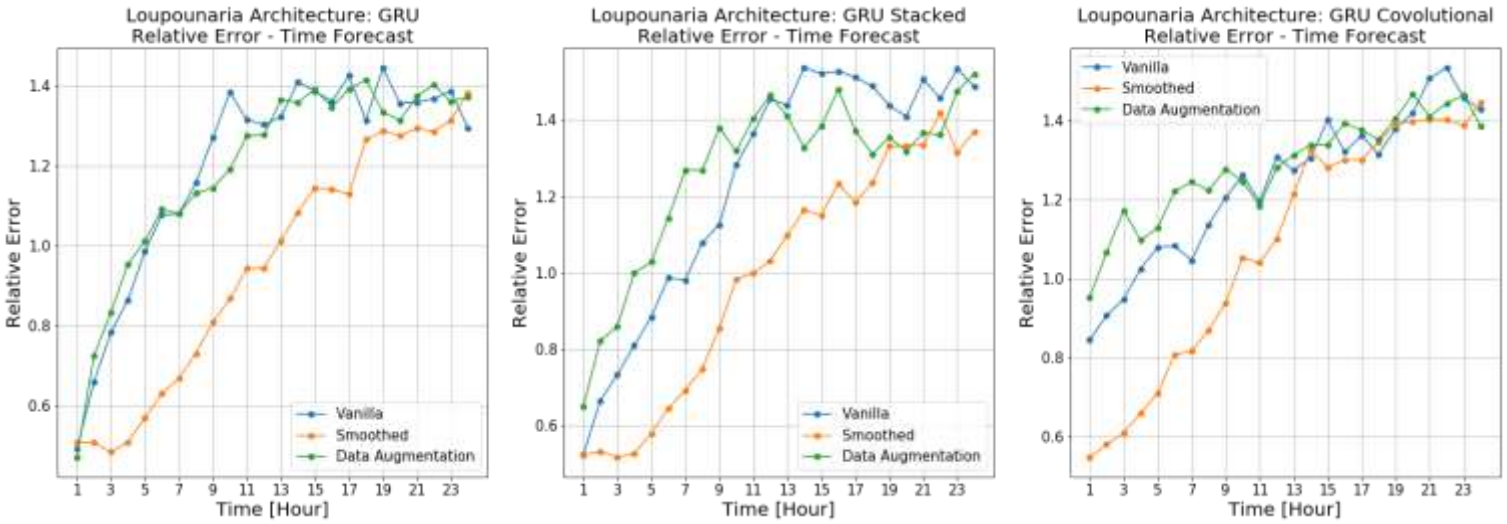


Figure 23. Comparison of Preprocessing Techniques: Dataset Loupounaria

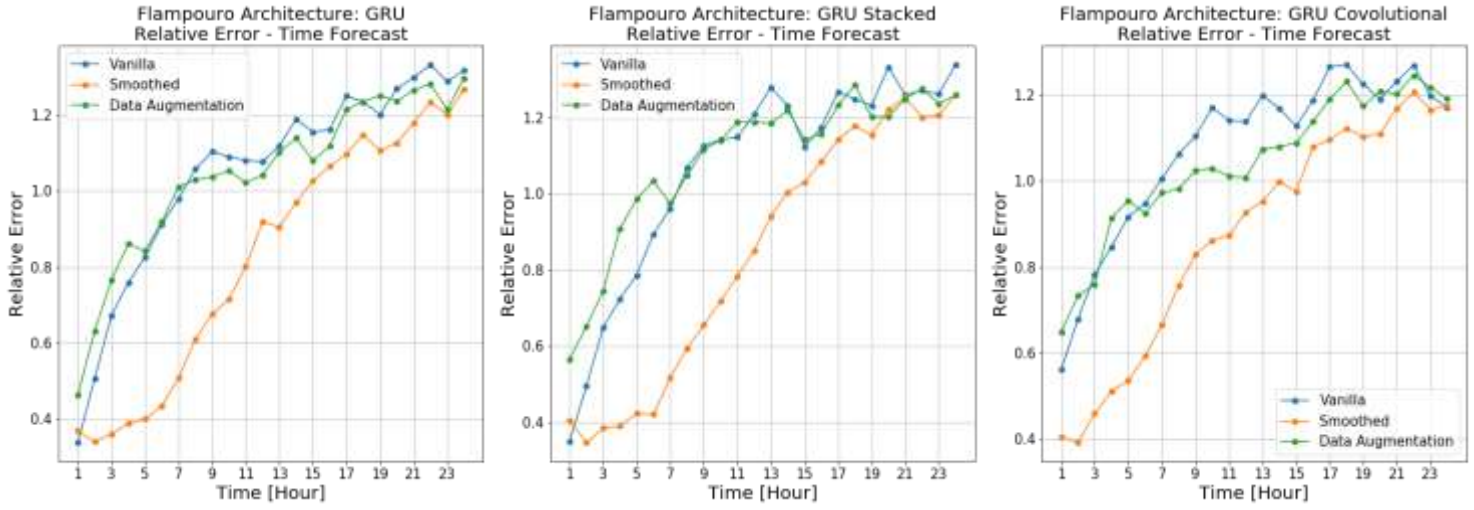


Figure 24. Comparison of Preprocessing Techniques: Dataset Flampouro

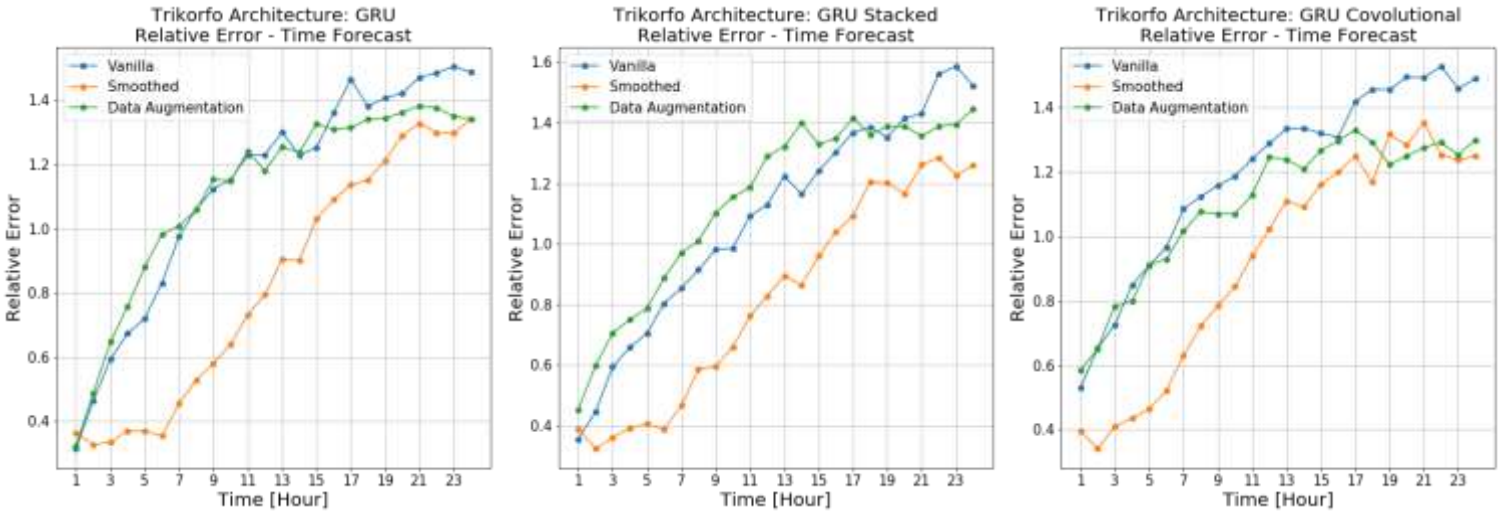


Figure 25. Comparison of Preprocessing Techniques: Dataset Trikorfo

The index *Time* of x-axis denotes the time of the forecast. If the distance between the target value and the window is zero, then the prediction has been made for 1 Hour ahead into the future.

Comparison of NN Architectures

In this section the graphs of Relative Error in relation to Prediction Horizon will be presented by comparing the different NNs Architectures (GRU, GRUs Stacked or GRUs Convolutional). Each set of graphs represents a different dataset (Flampouro, Loupounaria or Trikorfo) and contains 3 graphs which each represents a different preprocessing technique respectively (Vanilla, Smoothed and Data augmentation). In every graph, 3 lines are depicted for each NN Architecture respectively (GRU, GRUs Stacked or GRUs Convolutional).

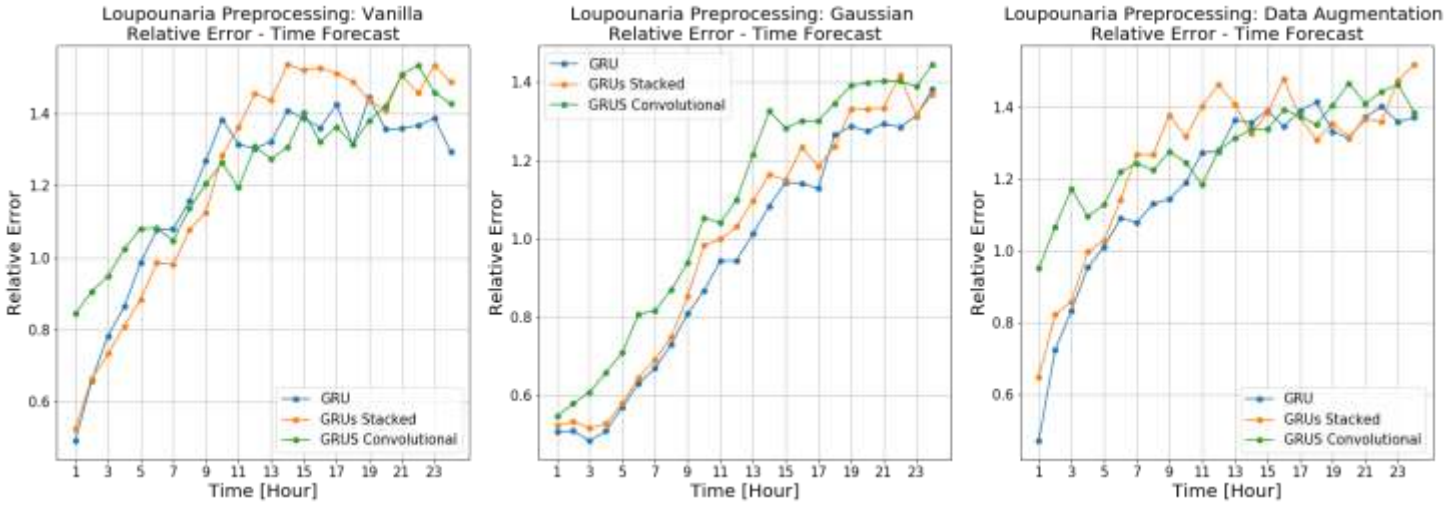


Figure 26. Comparison of NN Architectures: Techniques: Dataset Loupounaria

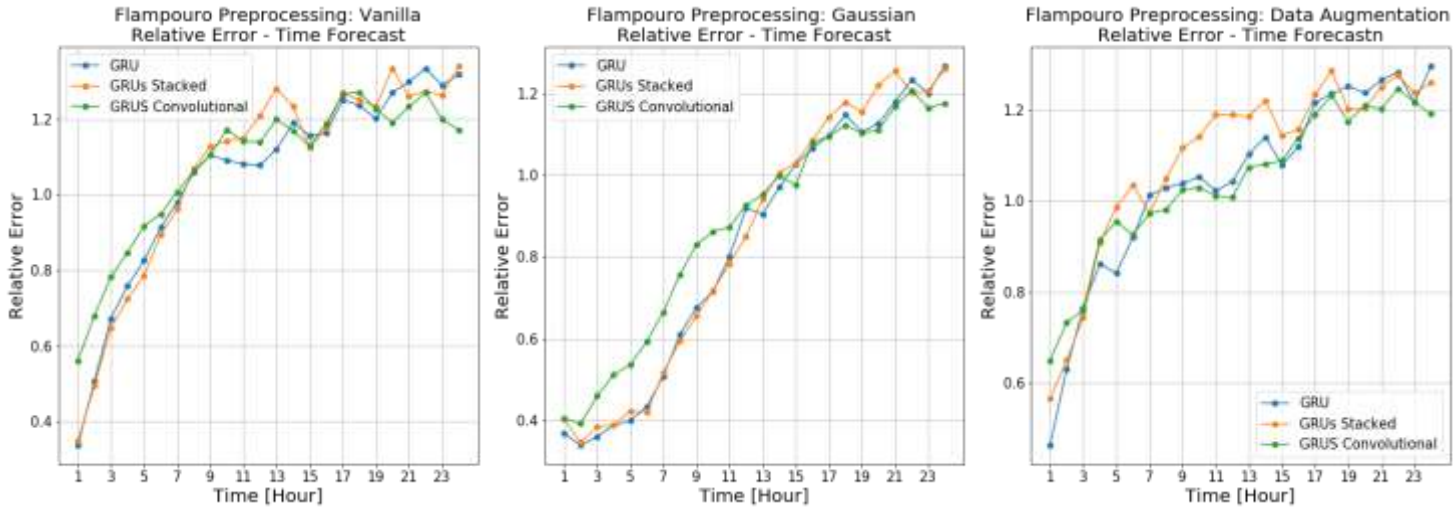


Figure 28. Comparison of NN Architectures: Techniques: Dataset Flampouro

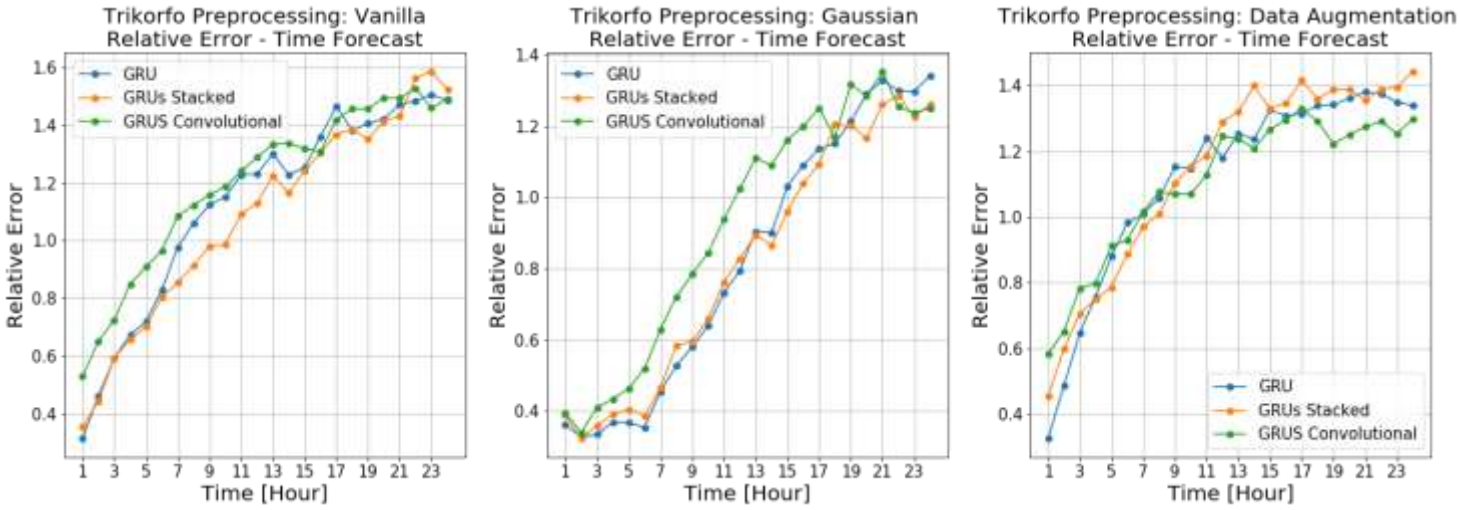


Figure 27. Comparison of NN Architectures: Techniques: Dataset Trikorfo

Comparison of Sampling Steps

In this section a comparison between Datasets with different sampling steps will be presented. For this purpose, let us consider the Dataset – 4, Loupounaria (Sampling Step: 10 min). The examination of the forecast horizon will be made by employing 3 different NN Architectures (GRU, GRUs Stacked or GRUs Convolutional) and 2 different preprocessing techniques (Vanilla and Smoothed). Moreover, the same experiments will be conducted by considering a new dataset which will be created by calculating the average of every 6 instances of wind speed from Dataset – 4. This will simulate a transformation in the dataset's sampling step, from 10 min to 1 hour.

Firstly, two sets of graphs will be presented each corresponding for a different preprocessing technique. Each set will contain 3 graphs, each representing a different NN Architecture respectively. In every graph, 2 lines are depicted symbolizing the forecast horizon for Loupounaria (Sampling Step: 10 min) and Loupounaria (Sampling Step: 1 Hour).

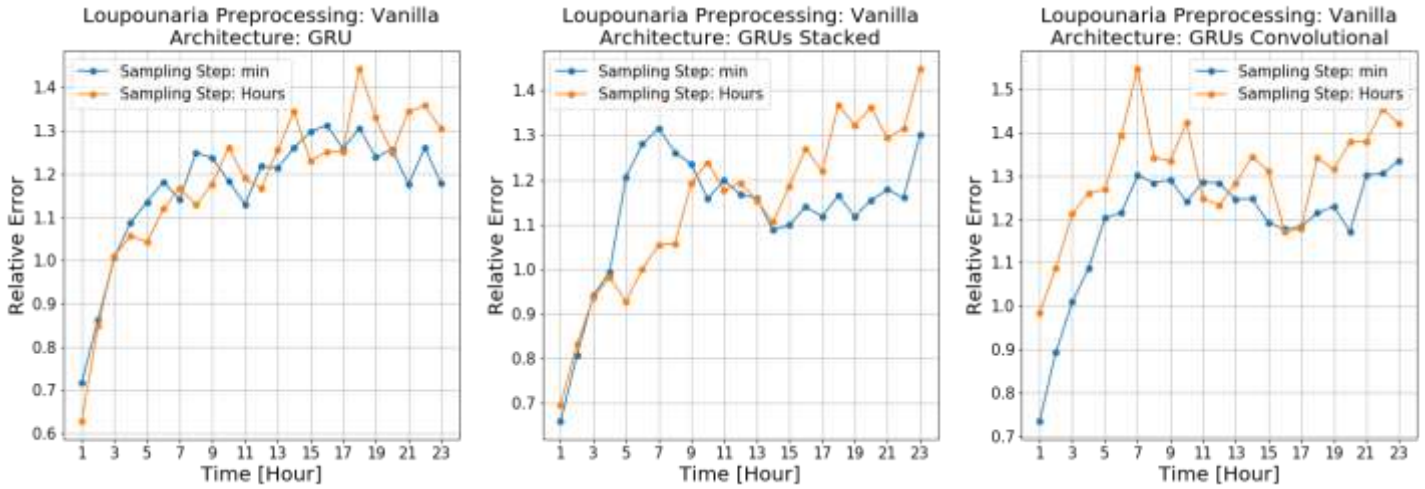


Figure 29. Comparison of Datasets with different Sampling Steps: Preprocessing Technique - Vanilla

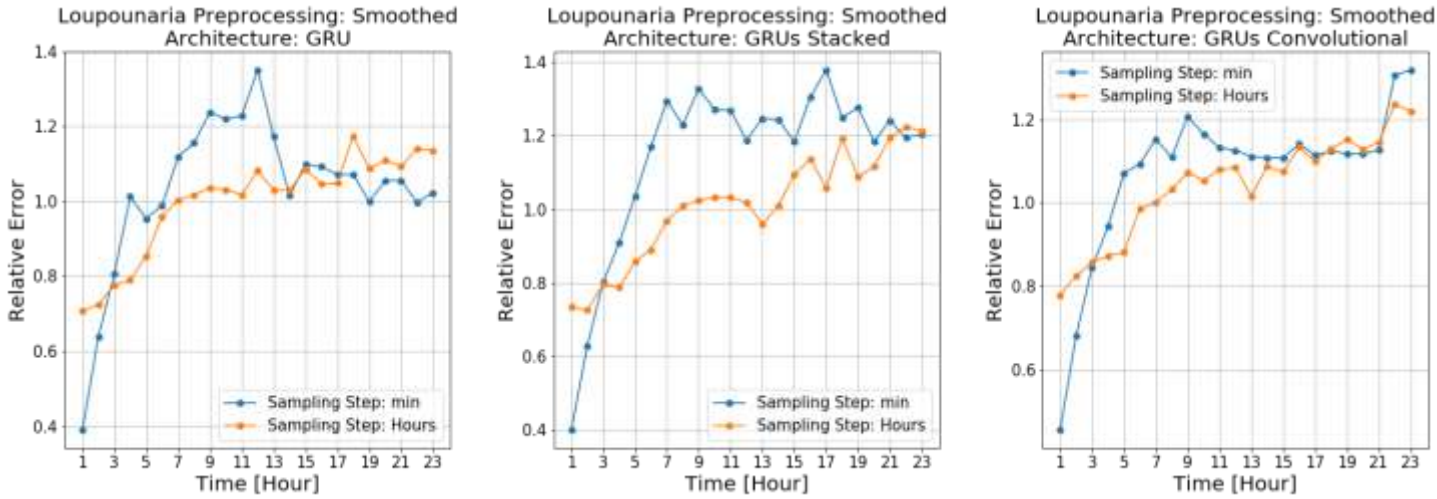


Figure 30. Comparison of Datasets with different Sampling Steps: Preprocessing Technique - Smoothed

Transfer Learning

Another approach to the problem is to consider implementing transfer learning. For this purpose, Datasets 1 – 3 will be sliced in a window format and then they will be combined to create a Generic type of dataset. This will be used to train a Generic model according to GRUs Stacked Architecture. Afterwards, the first 6 layers of the generic model, with their corresponding trained weights, will be used for a specialized model. The parameters of the layers of the generic model will be set to be constant and not part of training. At the same time 3 more layers (30 GRUs, Dropout – 0.2, Dense Layer – 1 node) will be added to the specialized model to complete the GRUs Stacked Architecture. The specialized model will be then train only the last 3 added layers based on a specific dataset (Loupounaria, Flampouro, Trikorfo).

This strategy aims to first capture a generic dynamic of the timeseries and then train use is to tackle a more specialized problem of just on dataset. Moreover, the generic model contains 3 times more instances and therefore has a greater chance of capturing a more generic structure of the data. For this experiment only Vanilla preprocessing was implemented to Datasets 1 – 3.

Finally, 3 graphs will be presented, each corresponding for a different Dataset (Datasets 1 – 3, Loupounaria, Flampouro, Trikorfo). In every graph, 2 lines are depicted symbolizing the forecast horizon of the transfer learning model and another model that was trained according to the same preprocessing technique (Vanilla) and NN Architecture (GRUs Stacked) but without a implementing Transfer Learning. The Second model is the one that was used for the sections “comparison of preprocessing techniques” and “comparison of NN Architectures”. The graphs will show the effects Transfer Learning in the quality of the predictions, in relation to the initial forecasts.

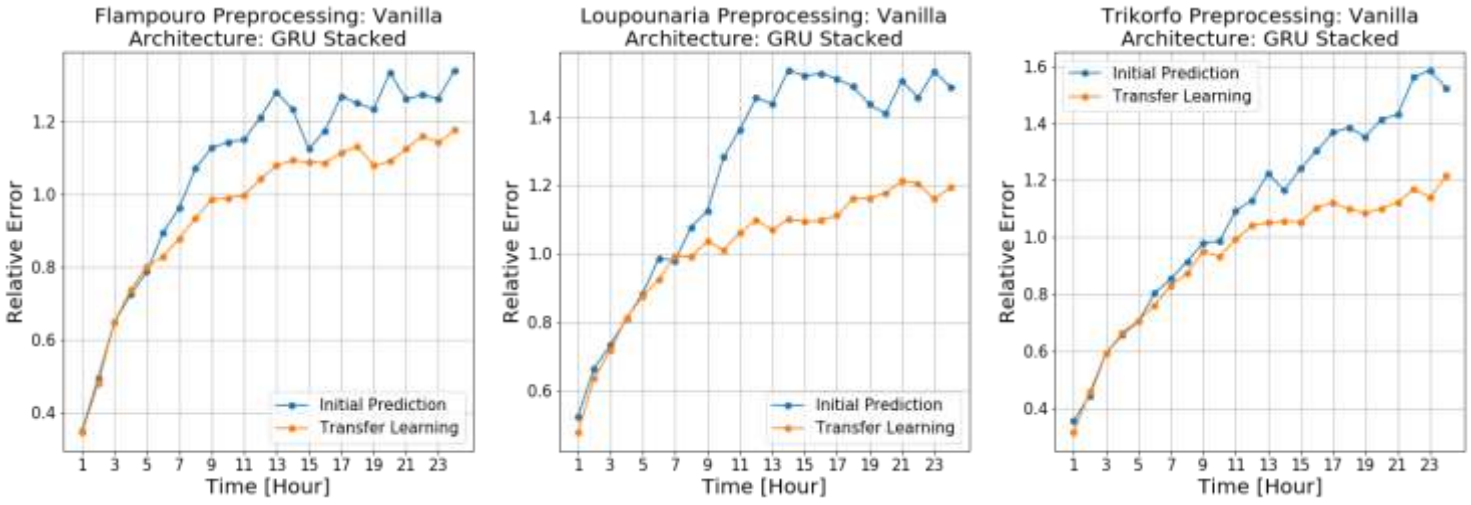


Figure 31. Transfer Learning: NN Architecture – GRUs Stacked, Preprocessing Technique - Vanilla

Comparison to Jena Climate Dataset

In this section we will consider the Jena climate Dataset. Jena climate Dataset is a dataset recorded at the Weather Station at the Max Planck Institute in Jena, Germany. It contains plenty of atmospheric features such as Temperature, Pressure, Wind Speed and Direction and more. The timeseries involve a time period that extends from 2009 to 2016 with a sampling step of 1 hour. The main purpose of the dataset was to perform Forecast exercises with ANNs approaches. The advantage of this dataset is that it contains a total amount of 70091 instances, compared to only 7500 in the datasets 1 – 3.

A comparison between Datasets with different amount of observations will be presented. For this purpose, only the features of wind velocity and direction were considered, from the Jena Climate Dataset and they were transformed into a vector format ($w_x, w_y, \|w\| = \sqrt{w_x^2 + w_y^2}$). Moreover, 2 preprocessing technique were introduced (Vanilla and Smoothed).

Despite the fact that Jena Climate Dataset involves a completely different location than the ones at study, it would still be interesting to investigate how the models perform with datasets that contain 10 times more data. Then, a new dataset (*Sliced*) will be produced, by taking into account only 7500 instances of the Jena Climate Dataset's and then comparing the models performance with the performance of the dataset 1 – 3. Supposing that Dataset 1 – 3 perform similar to the *Sliced* Dataset, then Jena Dataset with all 70091 observations may be an indicator of how Datasets 1 – 3 would perform, given enough data.

Finally, 4 graphs will be presented each corresponding for a different preprocessing technique (Vanilla or Smoothed) and for a different Jena Dataset (Full or Sliced). In every graph, 4 lines are depicted representing the forecast horizon for Datasets 1 – 3 and Jena Climate Dataset.

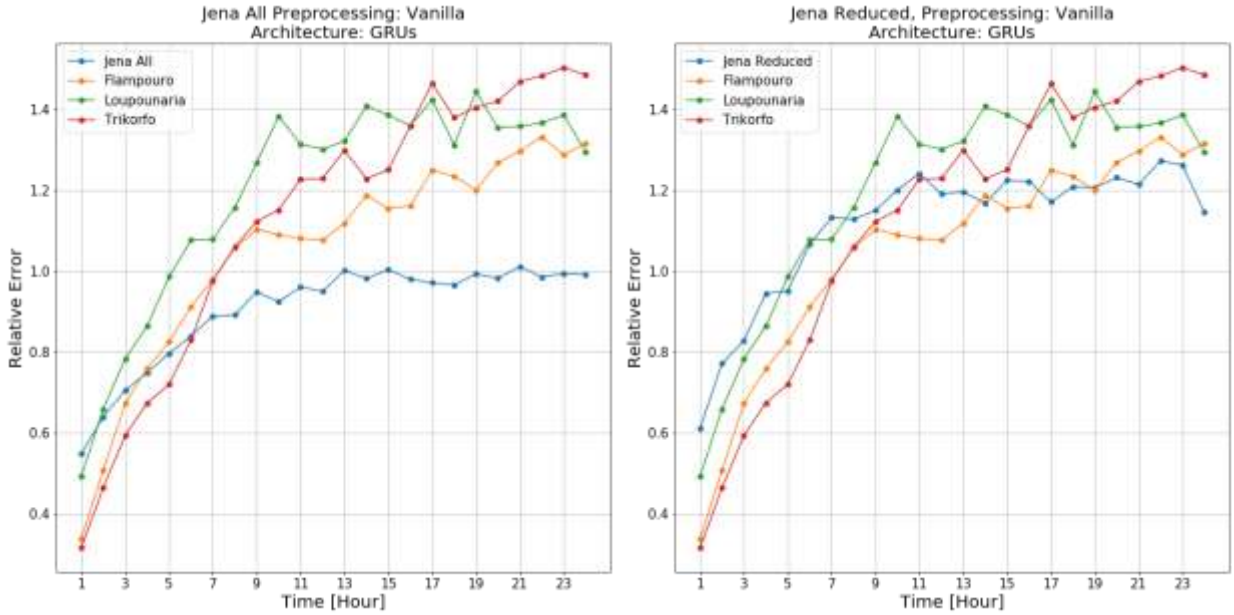


Figure 32. Comparison of Jena Climate Dataset with Datasets 1 – 3: NN Architecture – GRUs, Preprocessing Technique - Vanilla

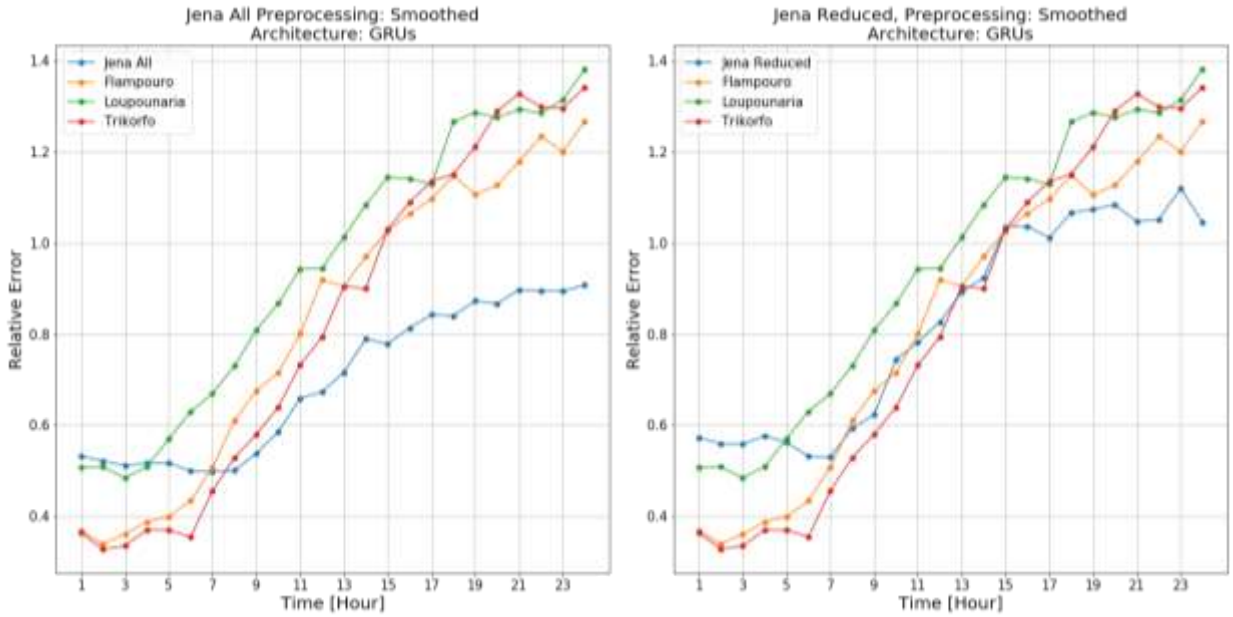


Figure 33. Comparison of Jena Climate Dataset with Datasets 1 – 3: NN Architecture – GRUs, Preprocessing Technique - Smoothed

Summary & Discussion

From the graphs presented in the section "*An Illustration of the Quality of the Predictions with Respect to the Relative Error*" it has been made clear that the relative error is a great indicator of the quality of the prediction made by the model. In fact, if the relative error is anything above 1, then it seems that the prediction has no apparent correlation with the True Values timeseries. For the purposes of this study consider that any forecast with a relative error of 8.5 and higher is considered unreliable.

By examining the Forecast horizon graphs, it is obvious that in all cases the relative error increases with respect to the depth of the prediction. The rate of the increase may vary depending on different Datasets, preprocessing techniques, the amount of data and more. This was expected due to the stochastic and chaotic behavior of the window velocity's timeseries. According to the graphs, almost all models were unable to predict accurately the wind velocity approximately after 10 – 12 hours.

Regarding the Comparison of Preprocessing Techniques, it seems that the Smoothed strategy performed significantly better that the other two. When the smoothed technique was implemented, the model was able to capture better the Dynamical Structure of the Data, even for long term forecasts (10 – 12 hours) and by extend to perform more robust predictions. The relative errors increased significantly slower compared to the other Preprocessing techniques, from $RE = 0.4 - 0.6$ for one to six hours ahead, to $RE = 8.5$ for 10 – 12 hours ahead. As for the Vanilla and Data Augmentation strategies, the relative error also begins at 0.4 - 0.6, however it soon rises to 8.5 for 4 – 6 hours into the future. Overall, it seems that Vanilla preprocessing technique performs slightly better. This indicates that the augmented data may be similar enough to the non-augmented data, which leads to overfitting.

By comparing the NN Architectures, it seems that GRUs and GRUs Stacked perform very similar to each other for short and long term forecasts. On the other hand, GRU Convolutional Architecture generates a higher relative error by a factor of 1, for short term predictions, in contrast with the other 2 models. It could be possible that the max pooling layer removes important information about the structure of the timeseries or alternatively that the amount of data that are contained in the input window are not enough for a convolutional NN to function accurately. For long term predictions (after 12 hours) all models generated equally poor forecasts.

By examining the graphs that compare the Datasets 1 and 4 with different sampling steps, no clear conclusions can be extracted as it seems to be no systematic differences between the performances of the specific models. The initial expectation that a smaller sampling step should correspond to better quality predictions, appears to be mistaken. For stochastic timeseries, a smaller sampling step may lead to greater content of noise. In that case the model may need more data in order to sufficiently capture the dynamics of the timeseries.

Concerning, the transfer learning approach, it is shown that both the transfer learning model and the original model, that did not involve transfer learning, perform exactly the same for a forecast horizon up to 7 hours. Predictions subsequent to 7 hours have been considered unreliable as the relative error is above 8.5. Nonetheless, transfer learning seems to be more robust for long term forecasts by retaining the relative error between 1 and 1.2, relative to original predictions that show a significant increase in the relative error, up to 1.2 or higher. This is anticipated due to the fact that transfer learning model had access to considerably more data, and therefore approached more accurately the stochastic nature of the timeseries.

Finally, regarding the graphs that present the forecast horizon of the Jena climate Dataset indicate a great similarity with the performance of the Datasets 1 -3 when Jena Dataset's observations were reduced to 7500, similar to the amount of data contained in the available datasets. When Jena Dataset had all the observations at its disposal (70091) the model performed noticeably better, producing more robust long term forecasts. This could suggest a potential increase in forecast accuracy if more observations are added to the datasets at hand.

With the advances in the field of Machine Learning more progressive models are created to cope with problems that involve timeseries forecasts. Architectures like *Encoder-Decoder GRUs with attention technique* or *Transformers* promise more accurate predictions even for chaotic and stochastic data. A future prospect of the project is to employ more recent models and compare their performance with the models at study. Moreover, it would be interesting to experiment more in the prospect of transfer learning by testing the effects of Smooth preprocessing technique, or including more datasets (Jena Climate Dataset, Dataset – 4) in the generic model. Nevertheless, further research should be conducted on the matter, as it is crucial not only for meteorological data but also for other time series measurements.

Bibliography

Abbod, Maysam F. 2007. "Application of Artificial Intelligence to the Management of Urological Cancer." *The Journal of Urology* 1150–1156.

Abiodun, Oludare Isaac, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat Abdelatif Mohamed, and Humaira Arshad. 2018. "State-of-the-art in artificial neural network applications: A survey." *Heliyon.*

Bird, Jordan J., Michael George Pritchard, Antonio Fratini, Aniko Ekart, and Diego Faria. 2021. "Synthetic Biological Signals Machine-generated by GPT-2 improve the Classification of EEG and EMG through Data Augmentation." *IEEE Robotics and Automation Letters* 3498–3504.

Bottou, Léon. 1998. "Online Algorithms and Stochastic Approximations." *Online Learning and Neural Networks. Cambridge University Press.*

Bottou, Léon, and Olivier Bousquet. 2012. "The Tradeoffs of Large Scale Learning." *Cambridge: MIT Press* 351–368.

Brownlee, Jason. 2018. *Deep Learning for Time Seris Forecasting.*

—. 2020. *Introduction to Time Series Forecasting with Python.*

—. 2017. *Long Short-Term Memory Networks with Python.*

Carremans, Bert. 2018. *Towards Data Science.* https://towardsdatascience.com/handling-overfitting-in-deep-learning-models-c760ee047c6e.

Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. " Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation."

Courville, Ian Goodfellow and Yoshua Bengio and Aaron. 2016. "Deep Learning." *MIT Press* 326.

D Bloice, Marcus, Christof Stocker, and Andreas Holzinger. 2017. " Augmentor: An Image Augmentation Library for Machine Learning." *The Journal of Open Source Software* 432.

Drakos, Georgios. 2019. *GDCoder What is a Recurrent Neural Networks (RNNS) and Gated Recurrent Unit (GRUS).* https://gdcoder.com/what-is-a-recurrent-neural-networks-rnns-and-gated-recurrent-unit-grus/.

Duchi, John, Elad Hazan, and Yoram Singer. 2011. "Adaptive subgradient methods for online learning and stochastic optimization." 2121–2159.

Farley, B.G., and W.A. Clark. 1954. "Simulation of Self-Organizing Systems by Digital Computer." *IRE Transactions on Information Theory* 76–84.

Géron, Aurélien. 2017. *Hands-On Machine Learning with Scikit-Learn and |Tensor Flow.* O'Reilly.

Gers, Felix, Jürgen Schmidhuber, and Fred Cummins. 1999. " Learning to Forget: Continual Prediction with LSTM." *Proc. ICANN'99, IEE* 850–855.

Graves, Alex, Marcus Liwicki, Santiago Fernandez, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. 2009. " A Novel Connectionist System for Improved Unconstrained Handwriting Recognition." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 855–868.

Grus, Joel. 2015. *Data Science from Scratch.* Sebastopol: O'Reilly.

HANSEN, CASPER. 2019 . *Optimizers Explained - Adam, Momentum and Stochastic Gradient Descent.* https://mlfromscratch.com/optimizers-explained/#/.

Hebb, Donald. 1949. "The Organization of Behavior."

Hinton, Geoffrey. 2020. "Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude." 26.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long short-term memory." *Neural Computation* 1735–1780.

Ioffe, Sergey, and Christian Szegedy. 2015. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift."

Kingma, Diederik, and Jimmy Ba. 2014. "Adam: A Method for Stochastic Optimization."

Kleene, S.C. 1956. "Representation of Events in Nerve Nets and Finite Automata." *Annals of Mathematics Studie* 3–41.

Li, Xiangang, and Xihong Wu. 2014. "Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition."

Maas, Andrew L., Awni Y. Hannun, and Andrew Y. Ng. 2013. "Rectifier nonlinearities improve neural network acoustic models." *Proc. ICML.* 30 .

McCulloch, Warren, and Walter Pitts. 1943. "A Logical Calculus of Ideas Immanent in Nervous Activity." *Bulletin of Mathematical Biophysics* 115–133.

Miljanovic, Milos. 2012. " Comparative analysis of Recurrent and Finite Impulse Response Neural Networks in Time Series Prediction."

Mills, Terence C. 2019. *Applied Time Series Analysis.* Loughborough.

O'Haver, T. 2012. *Smoothing.*

P, Pieter. 2020 . *Simple Moving Average.* https://tttapa.github.io/Pages/Mathematics/Systems-and-Control-Theory/Digital-filters/Simple%20Moving%20Average/Simple-Moving-Average.html.

RUDER, SEBASTIAN. 2016. *An overview of gradient descent optimization algorithms.* https://ruder.io/optimizing-gradient-descent/.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. 1986a. "Learning representations by back-propagating errors." *Nature* 533–536.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. 1986. "Learning representations by back-propagating errors." *Nature* 533–536.

Russell-Puleri, Sparkle. 2019. *Gated Recurrent Units explained using matrices: Part 1 Towards Data Science .* https://towardsdatascience.com/gate-recurrent-units-explained-using-matrices-part-1-3c781469fc18.

Sak, Hasim, Andrew Senior, and Francoise Beaufays. 2014. "Long Short-Term Memory recurrent neural network architectures for large scale acoustic modeling."

Shorten, Connor, and Taghi M Khoshgoftaar. 2019. "A survey on Image Data Augmentation for Deep Learning." *Mathematics and Computers in Simulation.*

Simonoff, Jeffrey S. 1998. *Smoothing Methods in Statistics.* 2nd edition. Springer.

2019. *Stack Exchange.* https://stats.stackexchange.com/questions/320952/data-augmentation-strategies-for-time-series-forecasting.

Valueva, M.V., N.N. Nagornov, P.A. Lyakhov, G.V. Valuev, and N.I. Chervyakov. 2020. "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation." *Mathematics and Computers in Simulation. Elsevier .*

Williams, Ronald J., Geoffrey E. Hinton, and David E. Rumelhart. 1986. "Learning representations by back-propagating errors." *Nature* 533–536.