University of Liège
Faculty of Applied Sciences

Wind Power Forecasting

Master's thesis carried out to obtain the degree of Master of Science in Data Science and Engineering by

Dachet Victor

Supervised by

Prof. Ernst Damien

Academic year 2021-2022

Contents

Abstract

Renewable energies are challenging to forecast due to their intermittence. However, it is crucial for the energy transition to predict accurately what is going to be produced at different temporal resolution (short, mid or long term) to integrate them in the network. In this work, we investigate the short term horizon. We work in the practical setting of the day-ahead forecast for wind farms. The aim of this work is twofold: to help the transmission system operator (TSO) in its task of balancing the network and the market participants of the day-ahead spot market. Both tasks require to know what is going to be produced for the next day. In this work, we will try new Artificial Intelligence (*i.e.* AI) models for wind energy forecasting. We explore state-of-the-art Machine Learning and Deep Learning models like Random Forest, Extra Trees, Recurrent Neural Network (*i.e.* RNN) and Transformers. We also investigate new RNN cells (*e.g.* BRC, nBRC and hybrid). We create original architectures of RNNs and Transformers. To compare the models and assess the results, we use two datasets: the ORES and the Gefcom2014 dataset. The first dataset is built from ORES recording productions of wind farms located in Belgium and weather data produced by the MAR (Modèle Atmosphérique Régional) developed at the University of Liège. The second dataset is often used in the scientific community. Then, we perform a deep analysis of the results given by the best models on both datasets. Additionally, we provide perspectives of improvement and we discuss other interesting techniques to investigate further.

Acknowledgement

In this part, I would like to thank the following persons:

- My supervisor Prof. Damien Ernst for proposing me to work with him since a few years now. I learnt a lot in writing problem statements with his intransigent remarks. I also particularly appreciate his support when my computer was down. What a stressful moment quickly solved! I still have a lot to learn with him in the upcoming years.
- ORES for providing me the data of wind farms in Belgium. The wind farm data is difficult to get. So, I thank them for their help in research. I also would especially thank Mr David Vangulick for the answers he provided to my questions about the data and the electricity market
- I would like to thank Mathias Berger who gave me some good pointers in order to go in the right direction.
- Prof. Xavier Fettweis who agreed that I enrolled to its class about the MAR model last year. He learnt me a lot of things without those I would not have been able to retrieve the data from the climatological server. Moreover, he developed the MAR which is a fantastic model. I admire the wonderful work he did to develop it
- Prof. Gilles Louppe for the discussion we had about the transformer but also because he introduced me to a lot of technical aspects that I used in this work. I would also generally thank the professors I had during my studies and especially during the master for the wonderful training courses they gave me.
- Dr. Antonio Sutera from Haulogy which gave me valuable advice week after week for this work. He spent a lot of time with me discussing the results and the research directions. He was a wonderful guide throughout this demanding work.
- My friends for the wonderful time we had during these years of study. Moreover, I greatly thank my family and particularly my parents for the support they gave me during these studies

Chapter 1

Introduction

Today, our societies heavily rely on electricity. In the last decades, the trend is to go towards more and more electrical devices, making electricity essential for our daily life. Nowadays, in Europe, it seems normal to have electricity at any time. However, this common access to electricity hides a very complex system which requires continuously producing what we consume. Indeed, it is currently impossible to store a high volume of electricity at a good price.

Europe is slightly transitioning towards renewable energy sources. This transition requires to find new solutions in order to adapt the network to these new energy sources.

Renewable wind and solar energy sources are intermittent and their production is difficult to forecast. Nevertheless, it is useful to know previously how much electricity is available in the future. For example, on a short time horizon, it is important for the transmission system operator (TSO) to know in advance the power production that will be available in order to balance the network but also for the market participants of the day-ahead spot market which need to set up a price for their bids on the day ahead wholesale market. On a long time horizon, it is necessary to have an idea of the power generation capacity of our grid to plan the construction of other means of production or not. All these reasons explain why advanced forecasting techniques are of great help in order to integrate these new technologies in the existing network.

The rising of artificial intelligence offers new tools to forecast energy production. In this work, we will explore machine learning and deep learning techniques in order to apply them on this renewable energy forecasting problem. Especially, we will use recurrent neural networks (*i.e.* RNN) and transformers which are powerful techniques to deal with time series but also more classic algorithms like random forest.

First, we will forecast the energy production of wind farms located in Belgium. Thanks to meteorological data simulations produced by the MAR[1] [1] developed at the University of Liège in the Laboratory of Climatology and production data from ORES[2]. Then, we will use the GEFCom2014 dataset [2] which is a well known dataset used in the scientific community. It is composed of several tasks. Among them, one is the wind energy forecasting. We will explore the capacity of machine learning and deep learning techniques to predict renewable energy production.

[1] Modèle atmosphérique régional

[2] ORES is a belgian distribution company

We will work in the practical setting of the day-ahead spot market which requires to forecast the energy production at midday for the next day. Thus, we will adapt both datasets mentioned above to this practical framework.

The research questions on which we will attempt to shed light will be:

- Are new artificial intelligence methods better than naive or simple regression models?
- Which is the most suited algorithm for point energy forecasting among RNNs, transformers and random forests? Especially at quarter and hour wind turbine day-ahead forecast.
- Are the new bistable recurrent neural networks [3], which allow long term memory, better than the other methods?

In order to answer these research questions, we will first describe the background necessary to assess the results as well as to understand the models (*e.g.* RNN) in chapter 2. Then, in chapter 4, we will dive into the different datasets. After that in chapter 5, we will define formally the deterministic problem statement as well as describing the architectures developed. Then, in chapter 6, we will compare the different approaches on the ORES dataset and lead a study of the results of our best model. After that, in chapter 7 we will apply the same methodology on the gefcom dataset. Finally, we will conclude in chapter 8.

Chapter 2

Background

In this chapter, we will first describe basic knowledge about wind turbine physics in section 2.1. Then, in section 2.2, we will explain what is supervised learning and the intuition why a forecasting problem can be formulated in this way. After that, in section 2.3, we will explicitly define all the building blocks and models used in this work. Afterwards, in section 2.4 we will explain how we train the deep learning models in this work. Finally, in section 2.5 and section 2.6, the different metrics and statistics will be defined.

2.1 Wind turbine physics

2.1.1 Output Power of a wind turbine

In the book [4] part III B., they introduce basic concepts of wind turbines physics. In Figure 2.1, one can see a little scheme with an area equals to A and a volume of wind equals to Avt with v the speed norm of the wind and t the time variable. The kinetic energy of the volume of air represented is equal to

$$\frac{1}{2}mv^2 = \frac{1}{2}\rho Avtv^2 = \frac{1}{2}\rho Atv^3. \tag{2.1}$$

The energy of the wind per unit time (power) is equal to

$$\frac{\frac{1}{2}mv^2}{t} = \frac{1}{2}\rho Av^3. \tag{2.2}$$

The physics rule which governs the power given by the wind is thus proportional to the cube of the speed norm.

In practice, the wind turbine power output has a typical shape like in Figure 2.2. One can see the four regimes of a wind turbine:

1. Below the cut-in speed, the wind turbine does not produce electricity.
2. Between v_c and v_r, the power production follows the physics rule as defined in Equation 2.2.
3. Between v_r and v_f, the wind turbine produces electricty at its maximum capacity.
4. Higher than v_f, the wind turbine is shut down to prevent damages.

According to [5], typical values ranges are $v_c = 3-4m/s$, $v_r = 11-17m/s$ and $v_f = 25m/s$.

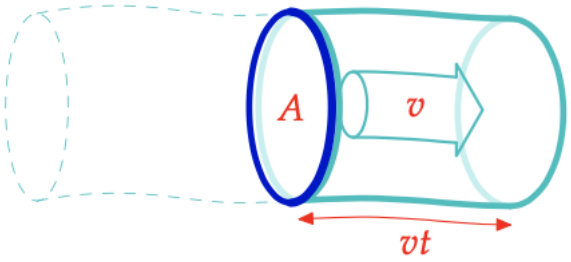

Figure 2.1: Wind Turbine Physics Scheme from [4] where we can see a volume of air passing through an given area of wind turbine.

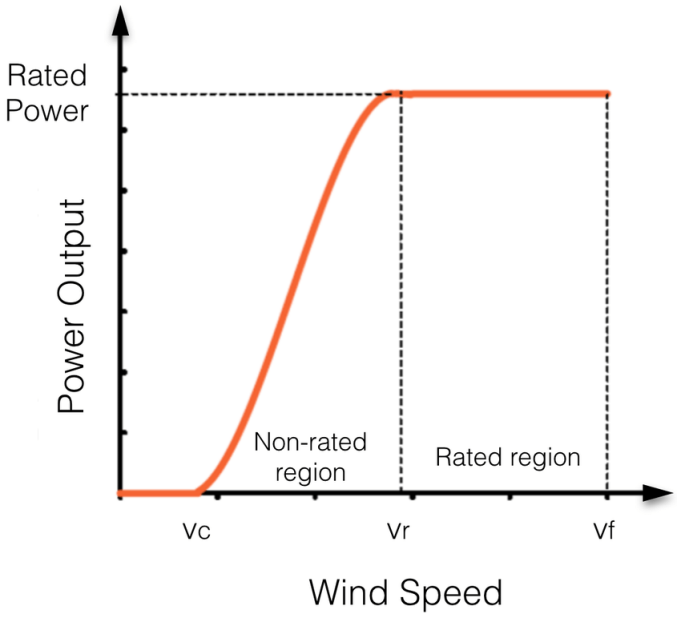

Figure 2.2: Typical Wind Turbine Output from [5] where we can see the four regimes of a wind turbine.

2.1.2 Load factor

The load factor of a wind farm is defined in [6] as:

$$\eta = \frac{\text{Average Load}}{\text{Maximum load in given time period}}.$$

This coefficient gives the efficiency of the means of production.

2.2 Forecasting as supervised learning

A forecasting problem is a problem where we try to predict the values of a given process based on currently available data. It can be seen as a supervised learning problem. There are two main types of tasks: classification and regression. This work only relates to regression so let us introduce this supervised learning task.

As defined in [7, Lecture 1: Fundamentals of machine learning], supervised learning consists in given training data *i.e.* pairs of input variables and target variables measured

$$(\mathbf{x}_i, \mathbf{y}_i) \sim P(X, Y) \tag{2.3}$$

with $P(X, Y)$ an unknown joint probability distribution, $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$, $i = 1, \ldots, N$. The training data is generated i.i.d. and can be of any finite size N. We usually do not have prior information on $P(X, Y)$.

Our inference task is that given our training data we want to infer for any new $\mathbf{x}$

$$\mathbb{E}[Y|X = \mathbf{x}]. \tag{2.4}$$

To achieve this goal, we perform an empirical risk minimization. Given a function $f : \mathcal{X} \to \mathcal{Y}$ the predictions of this function can be evaluated through a loss

$$l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+ \tag{2.5}$$

where $l(y, f(\mathbf{x})) \geq 0$ measures how close the prediction $f(\mathbf{x})$ is to y.

The function f belongs to an hypothesis space $\mathcal{F}$. We define the empirical risk as

$$\hat{R}(f, \mathbf{d}) = \frac{1}{N} \sum_{(\mathbf{x}_i, y_i) \in \mathbf{d}} l(y_i, f(\mathbf{x}_i)) \tag{2.6}$$

where $\mathbf{d}$ is the dataset.

The empirical risk minimization consists in finding the optimal model

$$f_* = \arg\min_{f \in \mathcal{F}} \hat{R}(f, \mathbf{d}). \tag{2.7}$$

A regression supervised learning problem can be seen as given $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ for $i = 1, \ldots, N$, we want to estimate for any new $\mathbf{x}$,

$$\mathbb{E}[Y|X = \mathbf{x}]. \tag{2.8}$$

Our power production forecasting problem can be seen as a supervised learning regression problem by creating pairs of input/output data. Therefore, given the data available $\mathbf{x}$ at a given moment, we will try to find $\hat{y} = f(\mathbf{x})$ as close as possible to the ground truth y.

A more detailed problem statement is given in section 5.1.

In order to assess that we effectively find the expected value of y given $\mathbf{x}$ as stated in Equation 2.8, it is important to create three distinct datasets: train, validation and test set. The train set is used in the empirical risk minimization process in order to find an optimal model. Then, using the validation set we will tune the hyper-parameters of the model. Finally, after several iterations of hyper-parameters tuning, we will assess the performance of the model on the test set.

In section 2.3, the different deep learning building blocks as well as the different typical models used in this work will be introduced.

2.3 Models and their building blocks

In this section, we will describe all the building blocks required to fully understand the different architectures that we will test in this work.

2.3.1 ReLu

The ReLu function is an activation layer defined as follows:

$$ReLu(x) = max(0, x)$$

which introduces useful non linearities in the model.

2.3.2 Sigmoid

Sigmoid function is an activation layer defined as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.9}$$

which has as great property to clip its output between 0 and 1. This property can be exploited when we want to approximate a probability or a normalized power production.

2.3.3 Linear Layer

A linear layer applies a linear transformation to the input $x \in \mathbb{R}^{in}$ as:

$$y = xA^T + b \tag{2.10}$$

with $A \in \mathbb{R}^{out \times in}$ and $b \in \mathcal{R}^{out}$.

2.3.4 Softmax Layer

A softmax layer applies to a vector $x \in \mathbb{R}^p$

$$Softmax(x_i) = \frac{\exp x_i}{\sum_j \exp x_j} \tag{2.11}$$

The nice property of this layer is that the sum of the elements in the output vector is equal to 1.

2.3.5 Attention Layer

The scaled dot-product attention mechanism as defined in [8]:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.12}$$

with queries Q and keys $K \in \mathbb{R}^{d_k}$ and $V \in \mathbb{R}^{d_v}$.

The scaling factor $\sqrt{d_k}$ is there to avoid suspected vanishing gradient for large values of d_k.

2.3.6 Multi-Head Attention

The multi-head attention mechanism is based on the concatenation of h attention mechanisms.

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O \tag{2.13}$$

$$\text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.14}$$

with $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}$ and $W_i^V \in \mathbb{R}^{d_{model} \times d_k}$ are the parameters to learn.

2.3.7 Positional Encoding

The positional encoding as defined in [8] consists in adding to a time multivariate time series input $\in \mathbb{R}^{S_x, p}$:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/dmodel}) \tag{2.15}$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/dmodel}) \tag{2.16}$$

to the features at place $2i$ and $2i + 1$ of the token with position $pos \in \{1, \ldots, S_x\}$.

2.3.8 Recurrent Layer

In classical supervised learning, we try to find a function $f : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X}$ is a p-dimensional space and $\mathcal{Y}$ a q-dimensional space. When we have to deal with time series, we get a sequence $S(\mathcal{X}) = \cup_{t=0}^{n-1} \mathcal{X}^t$ of observed variables. Recurrent Neural Networks (RNN) are well known deep learning models to deal with these time series. Indeed, they ingest one by one the elements of the input window sequence i_t while updating their recurrent states $h_t \in \mathbb{R}^q$ with q the size of the layer.

More formally, the recurrent state $\mathbf{h}_t \in \mathbb{R}^q$ with q the number of units in the recurrent layer is updated at each time step $t \in \{0, 1, \ldots, n-1\}$. Formally, for $t = 1, \ldots, n$,

$$\mathbf{h}_t = \phi(\mathbf{x_t}, \mathbf{h_{t-1}})$$

where $\phi : \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}^q$ and $\mathbf{h}_0 \in \mathbb{R}^q$.

It can output a prediction y_t at any time step t from the recurrent state

$$y_t = \psi(\mathbf{h}_t)$$

with $\psi = \mathbb{R}^q \to \mathbb{R}^C$. So, it finally outputs a tensor of size $n \times C$ with C which is equal to q or a given projected size. We will use four different types of RNN: LSTM, GRU [9], BRC [3] and nBRC [3]. We will formally describe each of these architectures. Indeed, the hypothesis space $\mathcal{F}$ of each RNN is completely defined by its number of layers, units, update function ϕ of its recurrent states and its prediction function ψ.

LSTM

The long short term memory (LSTM) introduced by [10] in 1997, has 2 recurrent states to update: $\mathbf{h}_t$ and $\mathbf{c}_t$. The updates are written as:

$$\begin{aligned}
\mathbf{f}_t &= \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\tilde{\mathbf{c}}_t &= \sigma(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \sigma_h(\mathbf{c}_t)
\end{aligned}$$

with $\mathbf{x}_t \in \mathbb{R}^p, W_f \in \mathbb{R}^{h \times p}, W_i \in \mathbb{R}^{h \times p}, W_o \in \mathbb{R}^{h \times p}, W_c \in \mathbb{R}^{h \times p}, U_f \in \mathbb{R}^{h \times h}, U_i \in \mathbb{R}^{h \times h}, U_o \in \mathbb{R}^{h \times h}, U_c \in \mathbb{R}^{h \times h}, \mathbf{b} \in \mathbb{R}^h$ and $\mathbf{c}_t \in \mathbb{R}^h$.

GRU

The Gated Recurrent Unit (*i.e.* GRU) was introduced by [9] in 2014 to simplify the LSTM and has similar performance than the previous one. Its update is given by:

$$\begin{aligned}
\mathbf{z}_t &= \sigma(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1}) \\
\mathbf{r}_t &= \sigma(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1}) \\
\mathbf{h}_t &= \mathbf{z}_t \odot tanh(W_h \mathbf{x}_t + U_h(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) + (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1}
\end{aligned}$$

with $\mathbf{x}_t \in \mathbb{R}^p$, vectors $\mathbf{b} \in \mathbb{R}^h, \mathbf{h} \in \mathbb{R}^h$ and matrices $W \in \mathbb{R}^{h \times p}$ and $U \in \mathbb{R}^{h \times h}$.

BRC

More recently in 2021, the bistable recurrent cell (BRC) introduced by [3], tries to mitigate the problem of long lasting memory in RNN. By imitating a neuronal mechanism which allows to store information at the cellular level for a long term, the BRC is able to better memorize information than classical GRU and LSTM on several tasks. This mechanism is called *bistability*.

The updates are written as:

$$\begin{aligned}\mathbf{c}_t &= \sigma\left(U_c\mathbf{x}_t + \mathbf{w}_c \odot \mathbf{h}_{t-1}\right)\\ \mathbf{a}_t &= 1 + \tanh\left(U_a\mathbf{x}_t + \mathbf{w}_a \odot \mathbf{h}_{t-1}\right)\\ \mathbf{h}_t &= \mathbf{c}_t \odot \mathbf{h}_{t-1} + (1-\mathbf{c}_t) \odot \tanh\left(U\mathbf{x}_t + \mathbf{a}_t \odot \mathbf{h}_{t-1}\right)\end{aligned}$$

with $\mathbf{x}_t \in \mathbb{R}^p, U_c \in \mathbb{R}^{h\times p}, U_a \in \mathbb{R}^{h\times p}, U \in \mathbb{R}^{h\times p}$

nBRC

Introduced by [3] again, the neuromodulated bistable recurrent cell (nBRC) improve further the BRC by relaxing the constraint of *bistability*. They create a dependency of a_t and c_t on the outpout of other neurons of the layer. The difference with the BRC stands in the formula of a_t and c_t. The update rule is the same as in BRC. The updates are written as:

$$\begin{aligned}\mathbf{a}_t &= 1 + \tanh\left(U_a\mathbf{x}_t + W_a\mathbf{h}_{t-1}\right)\\ \mathbf{c}_t &= \sigma\left(U_c\mathbf{x}_t + W_c\mathbf{h}_{t-1}\right)\\ \mathbf{h}_t &= \mathbf{c}_t \odot \mathbf{h}_{t-1} + (1-\mathbf{c}_t) \odot \tanh\left(U\mathbf{x}_t + \mathbf{a}_t \odot \mathbf{h}_{t-1}\right).\end{aligned}$$

with $\mathbf{x}_t \in \mathbb{R}^p, U_c \in \mathbb{R}^{h\times p}, U_a \in \mathbb{R}^{h\times p}, U \in \mathbb{R}^{h\times p}, W_a \in \mathbb{R}^{h\times h}, W_c \in \mathbb{R}^{h\times h}$.

Hybrid cell An Hybrid RNN cell is the concatenation of several different cells into the hidden state of a Recurrent Neural Network. Instead of having only n GRU cells, you may for example concatenate $n/2$ GRU cells and $n/2$ BRC cells.

2.3.9 Transformer

In [8], they introduced a new architecture: the Transformer. It is almost only based on the attention mechanism. The performance was tested on a natural language processing task. However, this new architecture can be adapted to regression tasks like in [11].

In Figure 2.3, one can see the building blocks of the transformer which is a composition of N Encoder and Decoder. The Encoder block is itself composed of Multi-Head Attention mechanisms and Feed Forward neural networks. The Decoder block is built with masked multi-head attention, a Multi-Head Attention layer and a final Feed-Forward Neural Network. They also both use skip connections in the add and norm layer in order to avoid vanishing gradients problems.

Then, the output of the decoder passes through a linear and Softmax layer to get a probability vector over the vocabulary. It is important to notice some particularities. The input of the model is a sequence of size $\mathbb{R}^{S_x\times v}$. Then, this sequence is embedded into a sequence of size $\mathbb{R}^{S_x\times e}$. The embedding size e will remain constant through each Encoder/Decoder block.

Moreover, a positional encoding is applied on the input embedding this ensure that the position of the token in the sequence is kept in the input of the Transformer. The *dmodel* parameter of the positional encoding is equal to the dimension of the embedding e.

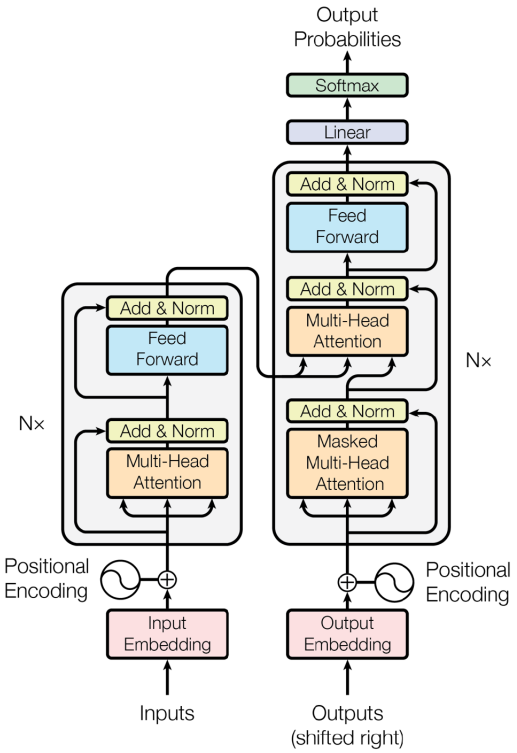

Figure 2.3: Transformer architecture introduced in [8] composed of N stacked encoder and decoder with positional encoding and a softmax output layer.

2.3.10 Random Forest

Random forest is an ensemble method which consists in building an ensemble of randomized regression trees. It belongs to the class of averaging methods and use the perturb and combine paradigm. In fact, it combines bagging and random attribute subset selection [12]. Bagging is the contraction of Bootstrap Aggregation and consist in reducing the variance of weak models (trees).

2.3.11 Extra Trees

Introduced in [13], the Extra Trees model is an ensemble method close to the Random Forest except in the choice of attribute subsets and in the choice of cut-off points. Indeed, Extra Trees do not use bootstrap whereas the Random Forest does. The choice of cut-off point is done randomly instead of using the optimal one.

2.3.12 Naive models: Persistence and climatology

Persistence and climatology are two naive methods in order to compare whether or not our model learns some patterns. The persistence model is defined by: given $y_0, \ldots, y_{k-1}$,

$$\hat{y}_{k-1+n} = y_{k-1} \forall n \in \mathcal{N}_0 \tag{2.17}$$

The climatology model is defined as the mean of the time series power production:

$$\hat{y}_{k-1+n} = \frac{1}{M} \sum_{i=0}^{m-1} y_i \tag{2.18}$$

We draw your attention to the fact that this mean must be computed on the training set only to avoid leaking information from the validation set to the model.

2.4 Training Neural Networks

2.4.1 Adam Optimizer

The Adam optimizer is considered as one of the default optimizer in deep learning. Indeed as described in [7, Lecture 4: Training neural networks], it performs mini-batch gradient descent while combining adaptive learning rate and momentum.

Mini-batch gradient descent consists in applying iteratively gradient steps on several samples:

$$g_t = \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta \ell \left(y_n, f\left(\mathbf{x}_n; \theta_t \right) \right) \tag{2.19}$$

$$\theta_{t+1} = \theta_t - \gamma g_t \tag{2.20}$$

where γ is the learning rate and N is the batch size. A pass on every sample in our batch is called an EPOCH. In order to optimize Neural Networks, several passes on the entire dataset are usually required *i.e.* the number of epoch is a key parameter.

The updates of the parameters θ with adaptive learning rate and momentum are performed as follows:

$$s_t = \rho_1 s_{t-1} + (1 - \rho_1)\, g_t \tag{2.21}$$

$$\hat{s}_t = \frac{s_t}{1 - \rho_1^t} \tag{2.22}$$

$$r_t = \rho_2 r_{t-1} + (1 - \rho_2)\, g_t \odot g_t \tag{2.23}$$

$$\hat{r}_t = \frac{r_t}{1 - \rho_2^t} \tag{2.24}$$

$$\theta_{t+1} = \theta_t - \gamma \frac{\hat{s}_t}{\delta + \sqrt{\hat{r}_t}}. \tag{2.25}$$

In this work, default parameters of the [14, pytorch] implementation are kept *i.e.* $\rho_1 = 0.9, \rho_2 = 0.999$. The learning rate was set to 0.0001.

2.5 Metrics

2.5.1 Forecast error

The forecast error is defined as the difference between the truth and the forecast error:

$$\epsilon = y_i - \hat{y}_i. \tag{2.26}$$

2.5.2 Bias

The bias is the mean over the forecasting horizon of the errors:

$$bias = \frac{1}{n}\sum_{i=0}^{n-1} y_i - \hat{y}_i \tag{2.27}$$

2.5.3 MAE

The mean absolute error (MAE) is defined as

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n}\sum_{i=0}^{n-1} |y_i - \hat{y}_i| \tag{2.28}$$

2.5.4 MSE

The mean square error (MSE) is defined as

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n}||\mathbf{y} - \hat{\mathbf{y}}||_2^2 = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2 \tag{2.29}$$

2.5.5 RMSE

The root mean square error (RMSE) is defined as

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \sqrt{\frac{1}{n}||\mathbf{y} - \hat{\mathbf{y}}||_2^2} = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2} \tag{2.30}$$

2.5.6 MAPE

The mean absolute percentage error (MAPE) is defined as

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{|y_t|} \tag{2.31}$$

2.5.7 SMAPE

The symmetric absolute percentage error (SMAPE) is defined as

$$\text{SMAPE} = \frac{2}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{|y_i + \hat{y}_i|} \in [0, 2] \tag{2.32}$$

2.6 Statistics

2.6.1 Pearson Correlation

As stated in [15] the Pearson coefficient is defined as

$$corr(x, y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.33}$$

with $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$.

2.6.2 Spearman's rank correlation

As stated in [16] the person coefficient is defined as

$$spearman(X, Y) = \rho_{\mathrm{rg}_X, \mathrm{rg}_Y} = \frac{\mathrm{cov}(\mathrm{rg}_X, \mathrm{rg}_Y)}{\sigma_{\mathrm{rg}_X}\sigma_{\mathrm{rg}_Y}}, \tag{2.34}$$

where

- ρ denotes the usual Pearson correlation coefficient, but applied to the rank variables,
- $\mathrm{cov}(\mathrm{rg}_X, \mathrm{rg}_Y)$ is the covariance of the rank variables,
- $\sigma_{\mathrm{rg}_X}\sigma_{\mathrm{rg}_X} and\ \sigma_{\mathrm{rg}_Y}\sigma_{\mathrm{rg}_Y}$ are the standard deviations of the rank variables.

Chapter 3

Related Work

We will structure this section as in [17] but focusing on deterministic forecasting. Moreover, we will rely on [18] which is a review only focused on deterministic wind power forecasting.

Deterministic forecasting models have been more investigated in the past but are still improved over the years thanks to new methods. On the other hand, probabilistic forecasting is nowadays a promising method in order to better catch the uncertainty around a single point forecast. It allows to have more information about the possible outcomes of the studied process.

Firstly, the authors introduced the most common metrics in order to assess the quality of a deterministic and a probabilistic forecast. Some of these metrics are also defined in this work in section 2.5. Secondly, they introduce the deterministic methods and classify them into three approaches: (1) *Physical*, (2) *Statistical*: also divided in: based on time series and based on Artificial Intelligence, (3) *Hybrid*. Third, they described the probabilistic approaches and classify them into two main categories: parametric and non-parametric approaches. The non-parametric approaches are also decomposed into five compartments: (1) QR (Quantile Regression) (2) KDE (Kernel Density Estimation) (3) Ensemble (4) Lube (Lower and upper bound estimation) and (5) Bootstrap. An overview of the different approaches is illustrated in Figure 3.1. In the appendix subsection A.4, some of these methods are mathematically defined. Now, let us dive into the deterministic forecast.

3.1 Deterministic forecast

The first approach was a physical approach, using numerical weather prediction (NWP). They use meteorological data like temperature, humidity, pressure, wind speed, wind direction but also topographical information in order to get an accurate estimation of the wind speed passing through the hub of the wind turbine. The NWP produce output of variable resolution. Depending on the resolution, they are well suited for long or short term energy forecasting.

The second approach is a statistical approach which covers different techniques. In the beginning, it was methods like ARMA, ARIMA and Grey Method which was developed. Those kind of methods are based on time series. Indeed, they try to find a recursive pattern in the data in order to predict the next values. For example, a typical ARMA model can be described as:

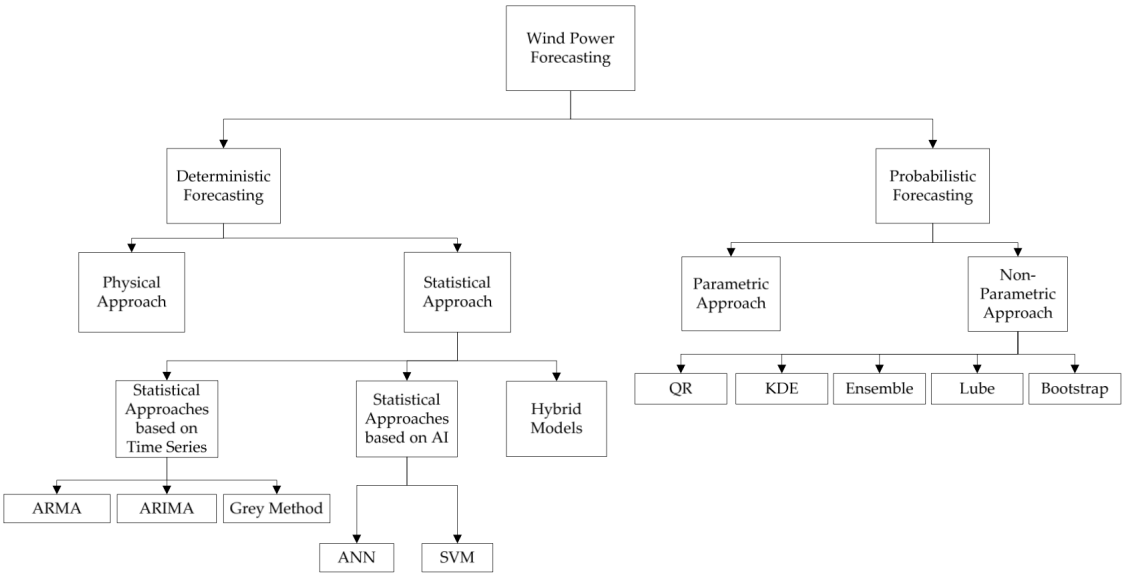

Figure 3.1: Taxonomy of wind power forecasting methodologies (extracted from [17]).

$$X_t = \sum_{i=1}^{p} \phi_i X_{t-1} + a_t - \sum_{j=1}^{q} \theta_j a_{t-j} \tag{3.1}$$

where X_t is the next value given the autoregressive model with parameter ϕ_i and the moving average model with parameters θ_j and a_t a white gaussian noise. The ARMA tends to be inaccurate if we do not have high-quality stationary data. To improve the capacity to deal with non-stationary data distribution they introduce ARIMA with an integrated part in order to tackle non-stationarity. Some alternatives or improvements of ARMA/ARIMA methods were developed like fractional-ARIMA (f-ARIMA), a combinaison of ARIMA and Autoregressive Conditional Heteroscedasticity (ARCH) model.

Another method used in statistical approach based on time series is the Grey Method which is useful for very-short-term predictions but tends to be highly inaccurate at longer time scales.

The second class in statistical approaches is the model based on AI. In this category, you can find well known algorithms like Artificial Neural Networks (ANN) and support vector machines. For the ANN, authors ([19], [20] and [21]) developed different architectures 2-3 layers, Convolutional Long Short Term Memory (Conv-LSTM).

The third class of statistical approaches is hybrid models which combines different models in order to improve the accuracy. For example, they can mix short-term and mid-term models or physical and statistical or just combining different AI based models. This approach tends to take the advantages from each model. For example, mixing ARIMA and ANN or SVM can improve the performance in comparison with single model. An example of article were the combination of all the models tested improved the accuracy is [22]. Indeed, they apply an MLP, a bi-directional LSTM, a Random Forest and a Gradient Boosting Decision tree on forecasting wind power time series. It is the ensemble *i.e.* the average of the four models which displayed the best results. Another type of hybrid model consists in combining algorithms like Wavelet Decomposition (WD) and LSTM. It shows higher accuracies but at an higher computational

cost. In this class of hybrid models, genetic algorithm and particle swarm optimization were also used to optimize Neural Networks parameters. Another new meta-heuristic called Dragonfly Algorithm was also tested to find the best parameters of an SVM model.

At the end of this literature review, we do not find any transformer model applied on wind power forecasting. Except in [11], where they applied a transformer model on energy load forecasting which is not exactly like in our problem focused on wind power.

Chapter 4

Datasets Exploration and Preparation

In this chapter, we will introduce the two datasets we use: the GEFcom 2014 and ORES dataset. Then, we will perform an exploration data analysis on both datasets. These latter are challenging however: the first one is a simulated dataset at an hourly resolution while the second comes from real wind farms in Belgium at the minute resolution. The second dataset will be used to predict at a quarter resolution the day ahead production of wind turbines. For both datasets, we will mimic the practical setting which consists in predicting before midday the entire next day at a hour or quarter resolution.

4.1 GEFCom2014 dataset

The GEFcom dataset 2014 [2] is a dataset introduced for a machine learning competition about probabilistic energy forecasting. This competition includes different tasks. One among them is wind energy forecasting. This competition was about probabilistic forecasting. However, we will use this dataset for deterministic forecast.

This dataset is composed of hourly measurements which cover more than one year. It starts on the 01/01/2012 and finishes on the 12/01/2013.

These hourly measurements contains as features:

1. `ZONEID` : integer between 1 and 10
2. `TARGETVAR` : Normalized power output
3. `U10`: zonal wind speed at 10 meters
4. `V10`: meridional wind speed at 10 meters
5. `U100`: zonal wind speed at 100 meters
6. `V100`: meridional wind speed at 100 meters

In Figure 4.1, one can see the histogram of the turbine output. There is an high peak around zero which is characteristic for wind turbines. Indeed, the main cause is that the wind speed is not high enough. There is not enough power in the wind to drive the turbine. Another reason might be that if the wind is too strong we may shut down the turbine. These two reasons are

easily discovered by looking at Figure 4.2 which shows the power curve of the wind turbine.

Again by crossing Figure 4.1 and Figure 4.2, we can also notice that around 0.95 percent of the power output a small increase of occurences. This may be due to the bridle of the wind turbines. When the wind is too strong, it is important to limit the velocity of the blades to avoid a technical problems.

Like we mentioned before, in Figure 4.2, one can see the power curve of the wind turbine in zone 1. For the x axis, we used the norm of the wind speed. We can see the characteristic cubic shape of the power output given the wind speed norm as explained in section 2.1. The difference is double: first, as explained before a wind turbine needs a minimum of wind power to drive the turbine. Second, the power output of the turbine is limited because the turbines are bridled by very strong wind.

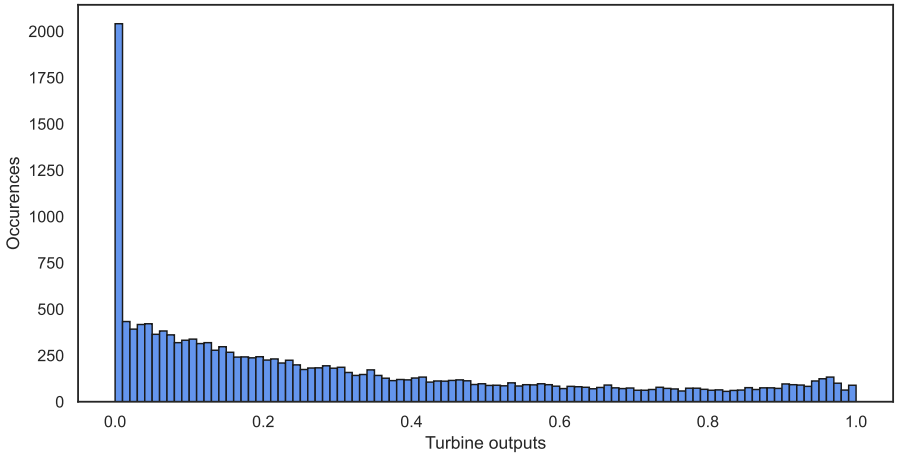

Figure 4.1: Histogram of the output of a typical wind turbine with a high peak in zero showing that a wind farm often does not produce electricity.

In Figure 4.4 and Figure 4.5, one can see two pair plots. In these plots, we subsample the entire dataset by taking randomly only 1000 records instead of the 16800 available for the first turbine. The first dataset consists in the raw data expressed in cartesian coordinates (also known as zonal and meridional directions in climatology) while the second is expressed in polar coordinates. In Table 4.2, we can notice the difference of correlation coefficient between the zonal and meridional variable and the polar coordinates expressions (H and A designate respectively the norm and the angle). The highest correlation coefficient is obtained for the norm in polar coordinates. It signifies that a simple linear relation would better fit the target variable in this case. Also in Figure 4.5, one can notice that the wind farm in zone 1 has low level of power outputs when the wind is blowing with an angle around 50 degrees. We may suggest that an obstacle like an hill, mountain or building keeps the wind from blowing. Another explanation could be a shadow effect from the other wind turbines in this particular configuration. Since we do not have access to the location of these wind turbines, it is difficult to draw a conclusion on the potential cause of this effect.

In Figure 4.3, we plot a realisation of the wind at both altitudes (normalized for the ease of comparison) and the power output. It allows us to understand the correlation better. Indeed, we clearly see an impact due to the wind power to the output of the turbine. However, there are

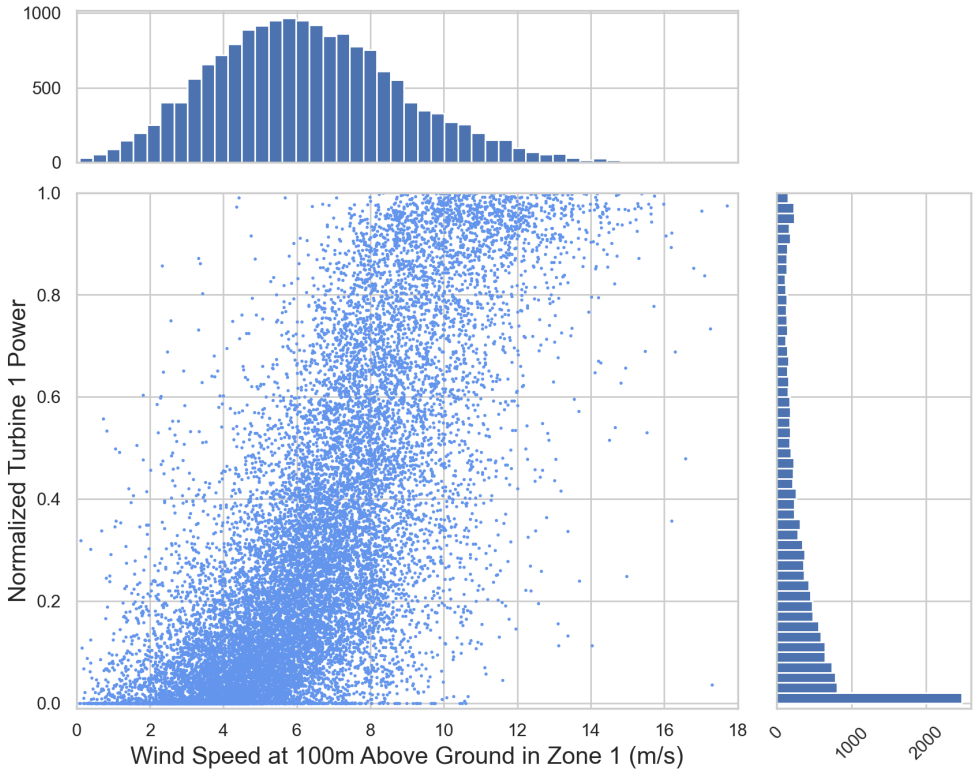

Figure 4.2: Empirical Power Curve of a wind turbine with a typical cubic shape.

a lot of discrepancies which occur. This implies that the simple physics explained in section 2.1 is clearly not sufficient to predict the power output.

By diving into the data, we can highlight another fact which is the load factor of the wind farms. The load factors as defined in subsection 2.1.2 for each wind turbine is listed in Table 4.1. The load factor for the aggregated wind farms is equal to 0.36 ± 0.31. We can interpret this by saying that the total power production installed produces only 30 percents of its maximal capacity in average.

For training our models, we only use the wind farm located in ZONEID 1 and we create train, validation and test sets of repectively 575, 59 and 59 samples.

Turbine	1	2	3	4	5
mean $\pm$ std	0.30 ± 0.29	0.32 ± 0.27	0.41 ± 0.30	0.35 ± 0.33	0.43 ± 0.34
Turbine	6	7	8	9	10
mean $\pm$ std	0.43 ± 0.34	0.30 ± 0.27	0.30 ± 0.28	0.30 ± 0.29	0.45 ± 0.34

Table 4.1: Gefcom Load factors per wind turbine.

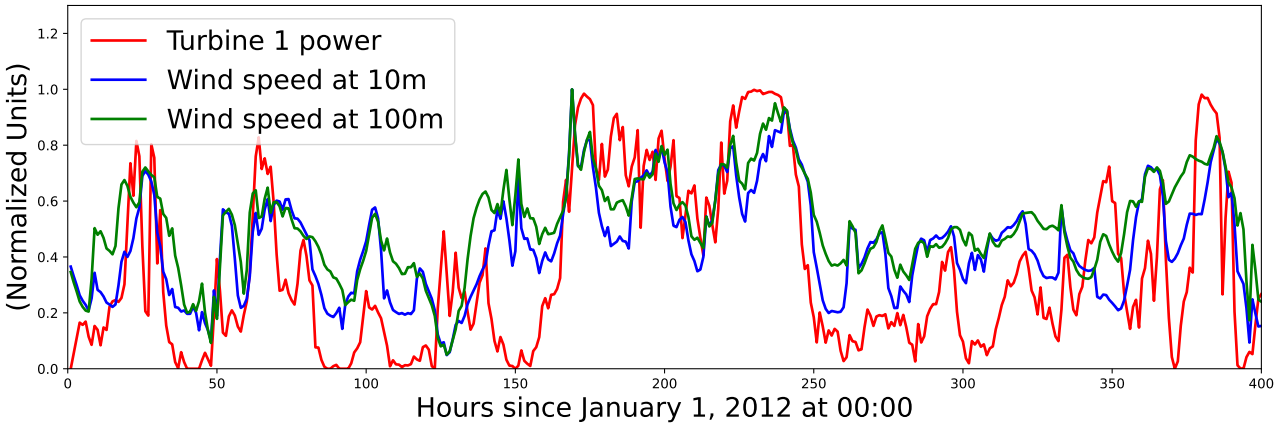

Figure 4.3: Realisation of 400 hours of production with wind speed each one normalized showing that those times series are correlated.

U10	U100	V10	V100
0.379	0.366	-0.193	-0.202
H10	H100	A10	A100
0.706	0.741	-0.153	-0.140

Table 4.2: Correlation coefficients with wind power output. We can see an higher correlation coefficient with the features expressed in polar coordinates.

4.2 ORES dataset

In this section, we will introduce the two different data sources used to create the dataset over Belgium. Besides, we will explain the main operations done to make the dataset usable in practice.

4.2.1 MAR data

First, the MAR data is composed of wind speed time series in Belgium simulated by the MAR[1]. We have two kind of features: an historical simulation of the MAR and a forecast simulation. The data cover the overall territory of Belgium. Thus, we only keep the time series data from the pixels[2] corresponding to the 3 wind farms we have. The 6 features that we will keep are:

- the wind speed at 80 and 100 meters of altitude in cartesian coordinates (components x and y)
- the temperature at 80 and 100 meters.

[1]Modèle Atmosphérique Régional
[2]The MAR defines a 2D grid of pixels over Belgium.

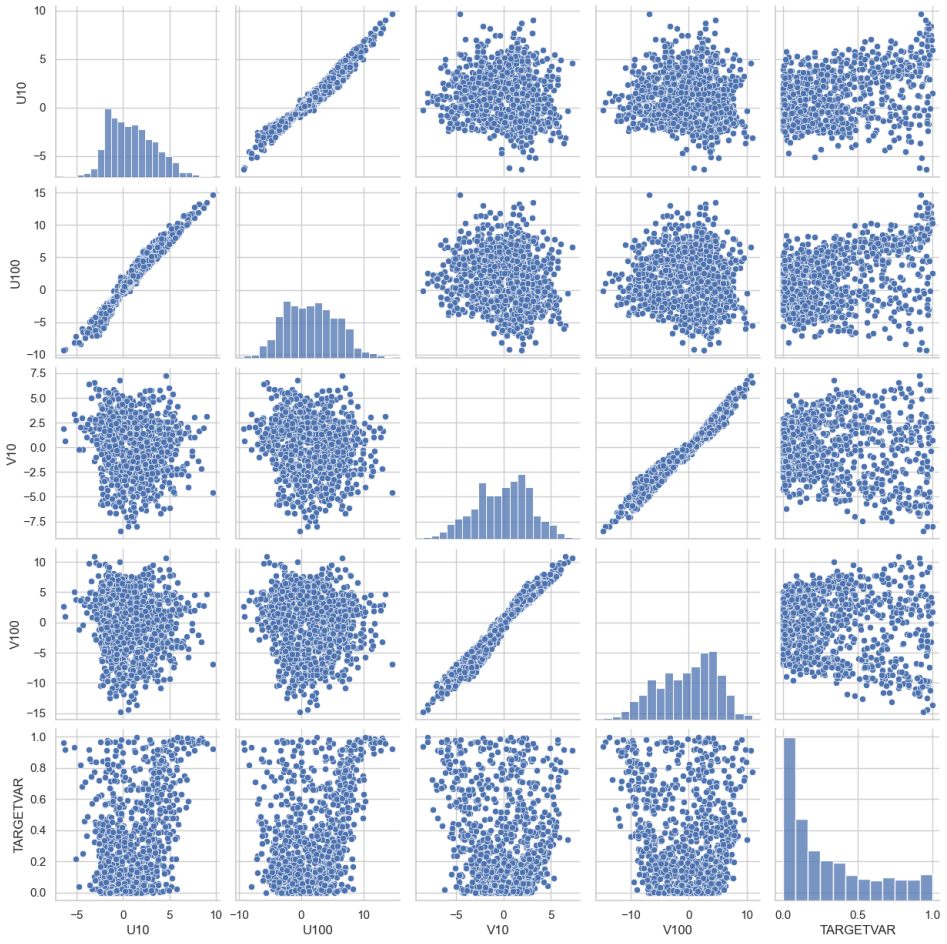

Figure 4.4: Pair plot in cartesian coordinates where it is difficult to see easily well known patterns between the targetvar and the other features.

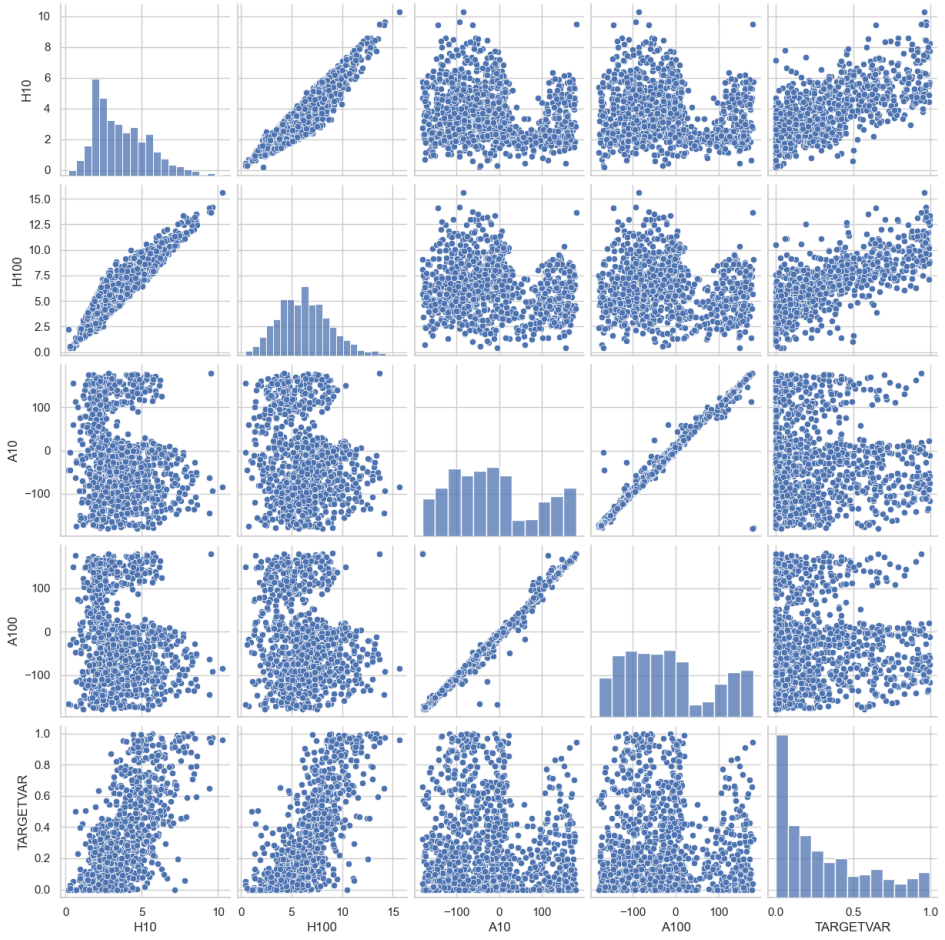

Figure 4.5: Pair plot in polar coordinates where the typical power curve of wind farms is visible.

Moreover, we transform these cartesian coordinates values into polar coordinates:

$$\mathbf{w} = \sqrt{\mathbf{w}_x^2 + \mathbf{w}_y^2} \in \mathbb{R}^n \tag{4.1}$$

$$\omega = \arctan(\mathbf{w}_y / \mathbf{w}_x) \in \mathbb{R}^n \tag{4.2}$$

where every operation is applied element-wise and $\mathbf{w}_x \in \mathbb{R}^n$ and $\mathbf{w}_y \in \mathbb{R}^n$ denote the cartesian wind speed.

The forecast data simulated by the MAR is produced each day at midnight, 6, 12 and 18 hour. Each file contains the forecast for 10 days ahead. Due to the practical setting in which we are we took as forecast the value of the next day coming with the files produced at midnight. For example: the 18 january 2021 at 00h00 we will take the forecast
for the entire 19 january 2021 from midnight to 23h59.

To retrieve the data from the climato server, we use the software [23, Ferret] useful to concatenate and manipulate netcdf[3] data files.

4.2.2 ORES data

Second, the ORES data is the production of 3 wind farms located in Belgium , more precisely in Wallonia. Until now, the period covered by the production data extends from January 2021 to the end of January 2022. The data contains 5 fields: the coordinates of the wind farm, the rated installed power in kVA, the contractual maximum power in kVA, the production power and the date-time. The most important feature will be the production power which is the target variable of our problem.

4.2.3 Time-Series matching

The production is recorded minutes by minutes while the MAR data is only produced by intervals of 7 minutes. To match the different time series, we made a linear interpolation on the MAR data to be at the resolution of the ORES records.

Let $i \in \{0, 1, 2\}$ be the index for the wind farm considered. Let $\mathbf{p}_i \in \mathbb{R}^n$ be the production vector from January 2021 to January 2022 for wind farm i and let $\mathbf{w}_i \in \mathbb{R}^n$ be the wind speed norm forecast for wind farm i. In Table 4.3, we can see the results of correlation coefficient[4] and Spearman's rank correlation coefficient[5] obtained between the production from the wind farms and the wind speed forecast norm at their locations *i.e.* $corr(\mathbf{w}_i, \mathbf{p}_i)$ and $spearman(\mathbf{w}_i, \mathbf{p}_i)$ for each wind farm. A correlation coefficient is always included between -1 and 1. A correlation of 0 depicts no linear correlation between the variables while a correlation of -1 or 1 depicts a perfect linear correlation between the two variables. The correlations obtained are greater than 0.7 which is a quite high linear correlation between the wind speed and the production from the wind farm. It implies that the data set is correctly made.

[3]Netcdf are special formatted files often used in climatology.
[4]See subsection 2.6.1 for a formal definition
[5]See subsection 2.6.2 for a formal definition

	$corr(\mathbf{w}_i, \mathbf{p}_i)$		$spearman(\mathbf{w}_i, \mathbf{p}_i)$	
Altitude (m)	80	100	80	100
Wind Farm 0	0.7410	0.7540	0.7268	0.7455
Wind Farm 1	0.7527	0.7548	0.7427	0.7490
Wind Farm 2	0.7585	0.7717	0.7637	0.7793

Table 4.3: Correlation and spearman coefficient for each wind farm in the ORES dataset with the wind speed features got by the MAR model.

4.2.4 Power normalization

To easily compare the different wind farms, each time series production are normalized by the maximum power installed of each plant. Let $P_i = \max(\mathbf{w}_i)$ be the maximum power production for wind farm i. We apply for each $i \in \{0, 1, 2\}$:

$$\mathbf{p}_i \leftarrow \frac{\mathbf{p}_i}{P_i} \tag{4.3}$$

where the division is applied element wise.

4.2.5 Visualization of the distributions

In Figure 4.6, one can see the box plot for each variable in our data set. We can see that there are a lot of outliers for the production and wind speed distributions. In fact, the distributions do not follow normal distributions. We try using log distribution for the positive variables but the decrease in the number of outliers was not significant.

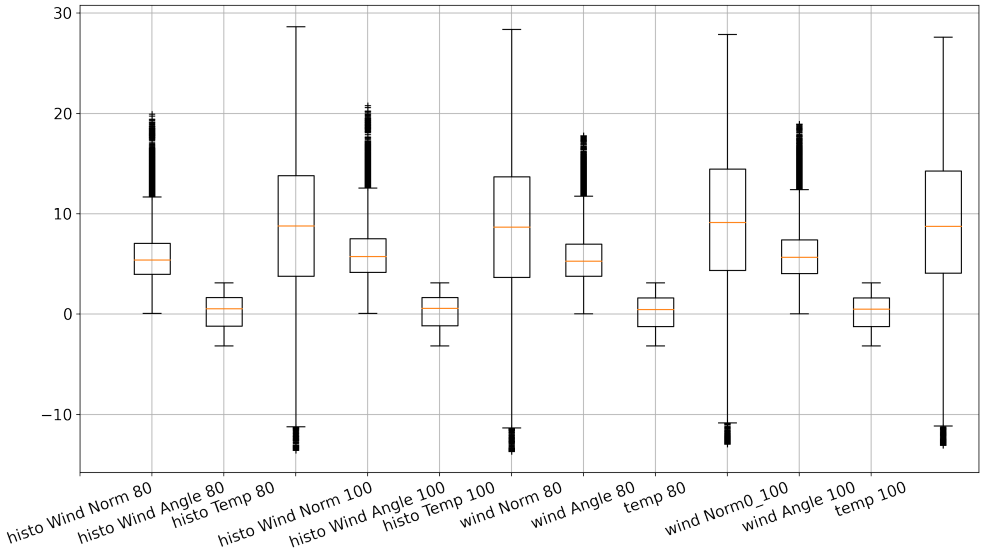

Figure 4.6: Boxplots of the features in the MAR dataset where we can see a lot of outliers for the diffenrent wind norms.

Set	Starting Date	Ending Date	Nbr of samples
Train	01-18-2021	11-17-2021	306
Valid	11-17-2021	12-25-2021	36
Test	12-25-2021	01-31-2022	36

Table 4.4: Division of the samples between the different sets.

4.2.6 Data set splitting

As explained in section 2.2 in order to assess the results, we need to split the data set into 3 distinct parts: train, validation and test set. In this case, we keep 80% of the data for the train set and 10% for the validation and test set. We wimply apply a split of the time series. These splits correspond to the dates in Table 4.4.

In Table 4.4, we can see that the number of samples is quite low. First of all, let us explain what is a sample: a sample is composed of one day of historic data from midday to midday of the previous day as well as some forecast features starting from midday to the next day midnight. We have a dataset covering a little bit more than one year precisely 378 days this is why we get a small dataset. It is also the reason why we try a second approach. We decided to create a bigger dataset where we did not try each day at midday to predict the next day. We opted for this approach: at each quarter of hour of the entire dataset, We took an historic of 96 quarters and a forecast of $48 + 96 = 144$ timesteps. This approach has one drawback: in many cases we get the forecast of two different files from the MAR data forecast. The dataset was far bigger the train, validation and test sets were composed of respectively 29342, 3458 and 3458 samples. However, when we trained our model with it and then assessed the performance on it or on the small dataset we did not get better performance. We also tried the way around: training on the small dataset and assessing the results on the big dataset and again models trained on the small dataset were better. We did not try to do again the retrieval of the data from the climato server because a huge amount of files hat to be download.

A last little note about the implementation: when making the input windows for the models, we concatenate the historical, the gap and the forecast wind speed. We fill with zeros the vectors into the gap and forecast matrices in order to match the size of the features window. See section 5.1 for more details about those matrices.

Chapter 5

Deterministic forecast

In this chapter, we will explicitly define the problem statement of deterministic forecast. It will be solved at an hourly and quarter resolution for the day-ahead and be focused on wind turbines output. Then, we will describe all the deterministic models from naive to machine learning and deep learning models.

5.1 Problem statement

Let $\mathbf{X} = \mathbf{x}_0\mathbf{x}_1 \dots$ represents a time series with $\mathbf{x}_t \in \mathbb{R}^p$ represents the p features at time t. We assume that every vector $\mathbf{x}_t$ can be decomposed into 2 vectors $y_t \in \mathbb{R}$ which represents the energy production at time t and the vector $\mathbf{z}_t \in \mathbb{R}^{p-1}$ represents the $(p-1)$ other features

$$\mathbf{x}_t = \begin{pmatrix} y_t & \mathbf{z}_t \end{pmatrix}^\top.$$

A multidimensional time serie from time step $t = 0$ until $t = m - 1$ is defined as

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_0 & \mathbf{x}_1 & \dots & \mathbf{x}_{m-1} \end{pmatrix} \in \mathbb{R}^{p \times m} \tag{5.1}$$

At time step $k-1$, the sequential data from time 0 to time $k-1$ is available and we seek to predict as accurately as possible the energy production in the forecasting window from time $k+T$ to $k+T+n-1$ with a fixed $T \in \mathcal{N}$ which represents the delay before the forecasting period. Moreover, we also have a forecast for the entire gap window of size $\mathbb{R}^{(p-1)\times T}$ and the forecasting window of size $\mathbb{R}^{(p-1)\times n}$ (see Figure 5.1). These gap ($\gamma \in \mathbb{R}^{(p-1)\times T}$) and forecast ($\phi \in \mathbb{R}^{(p-1)\times n}$) windows contain forecasts of the $\mathbf{z}_t$ vectors. By concatenating both of them, we form a multidimensional time series from time step $t = k$ to time step $t = k + T + n - 1$

$$\hat{\mathbf{Z}} = \begin{pmatrix} \gamma & \phi \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{z}}_k & \dots & \hat{\mathbf{z}}_{k+T+n-1} \end{pmatrix} \in \mathbb{R}^{(p-1)\times(T+n)} \tag{5.2}$$

Formally, we want to find a function

$$f : \mathbb{R}^{p\times k} \times \mathbb{R}^{(p-1)\times(T+n)} \to \mathbb{R}^n$$

with $k, T, n \in \mathcal{N}^3$ of fixed size. The function f takes two time series as input a historical and a forecast.
The function f outputs a vector $\hat{\mathbf{y}} \in \mathbb{R}^n$. The quality of these outputs is measured with the criterion

$$\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^+.$$

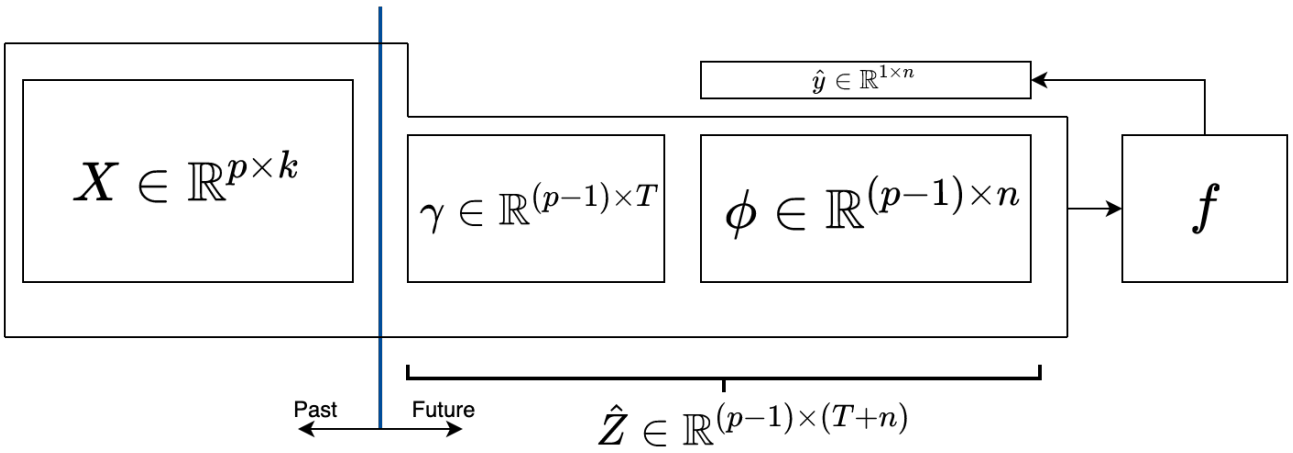

Figure 5.1: Input features matrices with the historical features X the gap matrix γ and the forecast matrix *phi*. The two last form a matrix of predicted features $\hat{Z}$. All the past and future features are given as input to f to forecast $\hat{y}$.

which measures the discrepancy between the 2 vectors $\mathbf{y}$ and $\hat{\mathbf{y}}$.

The time series $\mathbf{X}$ is built with previous wind power production and record of meteorological variables while the time series $\hat{\mathbf{Z}}$ is built with forecast of meteorological values (see chapter 4). The vector $\hat{\mathbf{y}} = f_k(\mathbf{i}_k, \mathbf{j}_{T+n})$ with $\mathbf{i}_k \in \mathbf{X}_{0:k}$ which represents the trajectory available before starting to forecast and $\mathbf{j}_{T+n} \in \hat{\mathbf{Z}}_{k:k+T+n}$ the forecast variables will be compared to the electricity production $\mathbf{y}$ measured on the renewable energy farm.

In this work, the main quality criteria will be the RMSE.

GEFCom2014 and ORES datasets

In the GEFCom2014 dataset, the difference between two timesteps is one hour. The history size k is equal to 24, the delay T before the forecasting window is equal to 12 and the forecasting window has a size of $n = 24$ timesteps. It is important to notice that there is only one data distribution for the value of wind speed and that we use this data distribution for both historical and forecast features.

In the ORES dataset, the difference between two timesteps is one quarter of hour. We will forecast at midday of the day $D-1$ with a historic of one day *i.e.* a history size equal to $k = 96$ the $n = 96$ quarters of hour of the next day D with a delay T equals to 48. In the ORES dataset we have two different data distributions one for the forecasts and one for the historical data.

5.2 Naive Method

As defined in subsection 2.3.12, the climatology and persistence models are two simple methods to compare with advances one to see if the advances methods perform better. The climatology is the mean of observed power production while the persistence is the last accessible power production.

5.3 Random Forest

The Random Forest model is in fact composed of $n = 96$ Random Forests. Each Random Forest is trained to predict a particular time-step of the next day. Let RF_i depict the i^{th} Random Forest trained on all the historic, gap matrix and forecast features for the timestep $i \in \{0, 1, \ldots, n-1\}$. Thus, the global model output is the concatenation of all $\hat{y}_i = RF_i(X, \gamma, \phi_{t \leq i})$. each Random Forest is composed of 50 regression trees with the criterion `squared error`. The other parameters were kept to the default values see [24] for an exhaustive list.

5.4 Extra Trees

The Extra Trees model is exactly defined as in section 5.3. It is composed of $n = 96$ models. It is composed of 50 estimators per Extra tree model.

5.5 RNN

In this section, we will describe all the RNN architectures used in this work. The first one, simple RNN, is a kind of encoder RNN while the second one called architecture 1 is more like an encoder decoder architecture. The third one history forecast context RNN is an original architecture. It is again a kind of encoder decoder architecture but it has a special way to deal with the gap matrix.

5.5.1 Simple RNN

This architecture is displayed in Figure 5.2. It consists in consuming the entire time series composed of the historical matrix and the forecast matrices in order to get an hidden state $\mathbf{h}$ considered as a context vector. Then, this context vector and the historical production $\mathbf{y}_{histo}$ are fed to a multilayer perceptron a.k.a. MLP.

5.5.2 Architecture 1

This architecture is displayed in Figure 5.3. It consists in giving the historical time series X (composed of historical features and production) to an RNN to get an hidden state $\mathbf{h}_{i-1}$. After that, sequentially the MLP will output a power production $\hat{y}_i$ based on $\mathbf{h}_{i-1}$ and the forecast features $\hat{\mathbf{z}}_i$. Then, the output $\hat{y}_i$ and again the forecast features $\hat{\mathbf{z}}_i$ are given to the RNN to update the hidden sate $\mathbf{h}_{i-1}$ to $\mathbf{h}_i$.

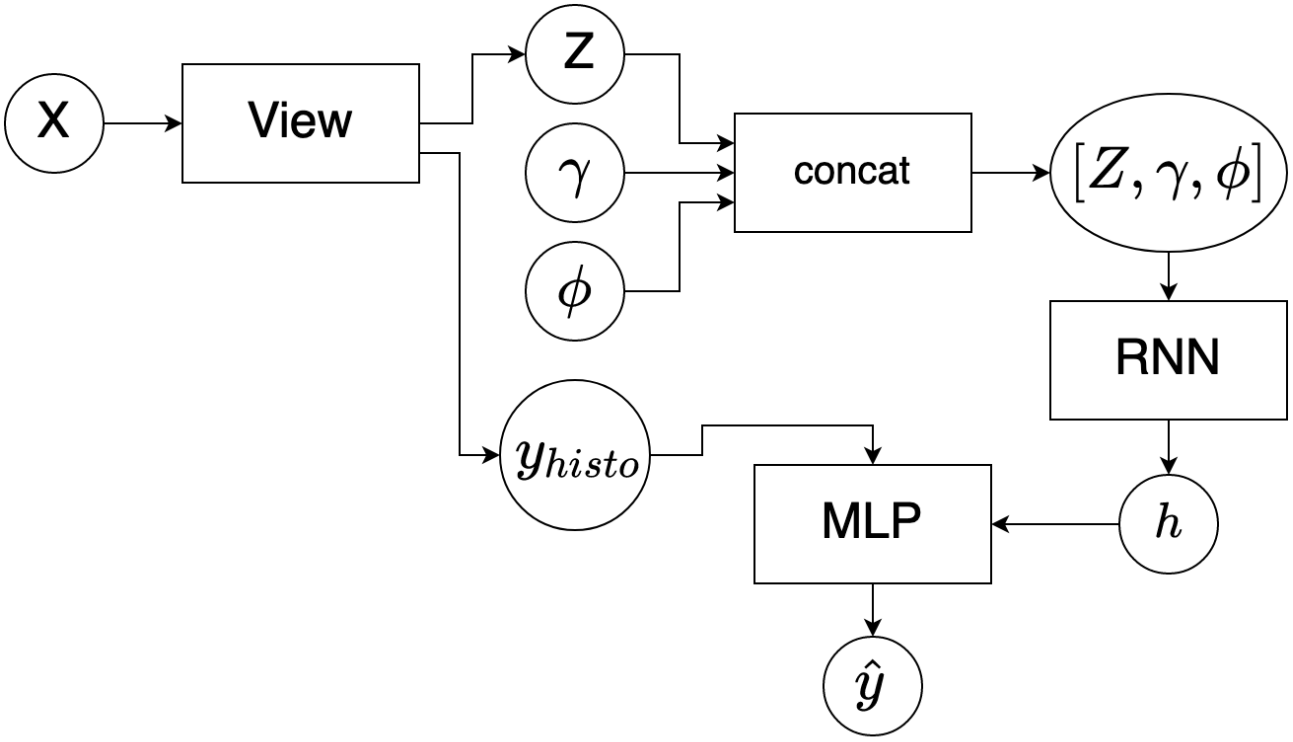

Figure 5.2: Simple RNN architecture: a view is applied to the historical matrix X to separate the fetaures Z and the production y_{histo}. All the $(p-1)$ features are concatenated into a matrix and fed to a RNN which produces an embedded vector h. From the embedded vector and the y_{histo} vector a MLP produces the forecast vector $\hat{y}$.

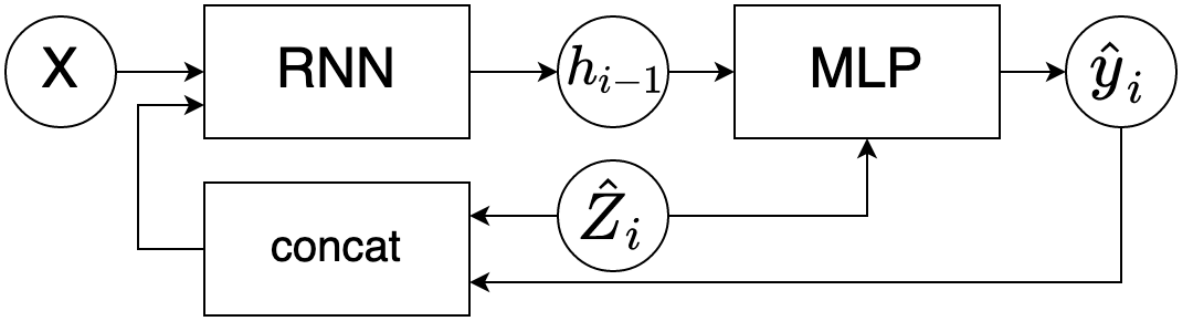

Figure 5.3: RNN architecture 1: the X features are consumed by the RNN. Then, from the predicted features it iteratively produces $\hat{y}_i$ and re-use the same RNN by updating its hidden state.

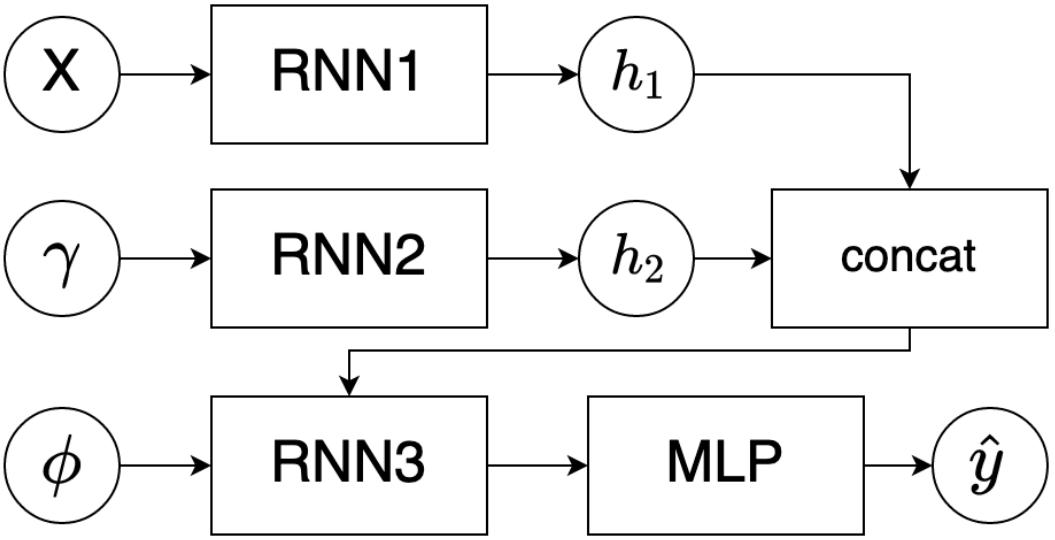

Figure 5.4: Context RNN architecture: It consumes the X and γ matrices by two distinct RNN. Then the concatenation of their last hidden states is fed as initial hidden state of a third RNN to consume the forecast window.

5.5.3 History Forecast Context RNN

This architecture is displayed in Figure 5.4. It consists in getting two hidden states $\mathbf{h}_1$ and $\mathbf{h}_2$ by respectively consuming the historical time series X and the gap matrix γ. Then, these two hidden states, by concatenating them, become the new hidden state of a third RNN consuming sequentially the forecast time series ϕ and gives its hidden state to an MLP which outputs $\hat{y}$.

5.6 Transformers

5.6.1 Encoder architecture

The Encoder architecture displayed in Figure 5.5, consists in using a positional encoding. In this case, the positional encoding simply consists in adding two features to the time series. Indeed, compared to [8], the dimension of the multidimensional time series is relatively low (The number of features is equal to 7 versus 512 of embedded size in the original paper). Therefore the input X is fed to the positional encoder then to a stack of *nlayers* Encoder. The stacked encoders output a kind of hidden states as in the RNN of the size of the input multidimensional time series. This representation is then given to an MLP in order to get the desired output feature size. After that, a view is applied to only keep the $\hat{\mathbf{y}}$ corresponding to the forecasting window.

Notice that the positional encoder is not strictly equivalent to those in the original paper.

Indeed, we made a kind of cyclical features encoding from the timestamp. Let x denotes the timestamp, it consists in concatenating

$$\sin(x \times k) \tag{5.3}$$

and

$$\cos(x \times k) \tag{5.4}$$

with $k = \exp\left(-log(10^4)/d\_model\right) \approx 0.5657$.

The choice of concatenating these two vectors instead of adding the positional encoding is motivated by the fact that we are working with a *d_model* equals to 7 and not 512. Therefore, adding two features do not increase too much the curse of dimensionality in our case.

5.6.2 Encoder/Decoder architecture

The Encoder Decoder architecture displayed in Figure 5.6, consists in almost exactly the same architecture as in [8] except for two things: (1) the softmax output layer is replaced by a Linear Layer (2) the input to the decoder is, instead of giving an embedding representation of the sequence of words, the concatenation of the forecast features $\hat{\mathbf{z}}$ with the last y_i predicted. The first input token is the last element of the gap window padded with a 0 as y_i production.

Given these two differences the architecture is similar to the original one, the encoder part is the same as in subsection 5.6.1 without the MLP. This output is given to the decoder also composed of *nlayers* stacked decoder. Then, the linear layer transforms the features into power production and a view comes to remove the first input token.

A big difference occurs in this model in comparison with the other ones. In fact, the decoder part requires the ground truth $\mathbf{y}$ during training. The forward method of the model which is used to train the model is consequently different from the method used at testing time called predict. The forward is done in one pass and is highly parallelized thanks to the GPU. We create a matrix:

$$Y_{fh \times fh} = \begin{bmatrix} y_1 & -\infty & -\infty & \dots & -\infty & -\infty \\ y_1 & y_2 & -\infty & \dots & -\infty & -\infty \\ \vdots & & & \ddots & & \vdots \\ y_1 & y_2 & y_3 & \dots & -\infty & -\infty \\ y_1 & y_2 & y_3 & \dots & y_{fh-1} & -\infty \\ y_1 & y_2 & y_3 & \dots & y_{fh-1} & y_{fh} \end{bmatrix}$$

A mask is applied to hide the ground truth which is not already available when predicting the next token. However, this method cannot be applied at testing time because we do not have directly access to the entire $\hat{\mathbf{y}}$. Indeed, we predict greedily one at a time each $\hat{y}_i$ in a sequential manner. The first input token called the starting token is in [8] a vector full of zeros but in our case we use the last output production recorded *i.e.* the last vector in the historical matrix X.

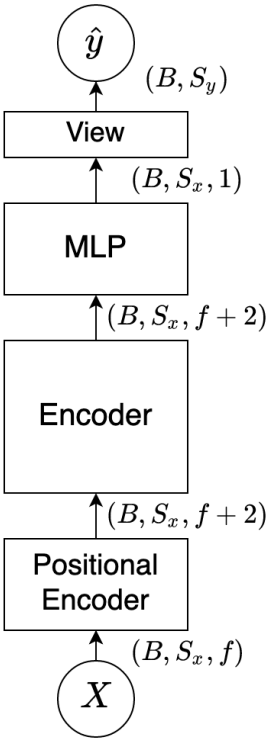

Figure 5.5: Encoder Transformer architecture: only composed of the encoder part as defined in [8] with a MLP on top of it to transform the embedded representation into production forecast.

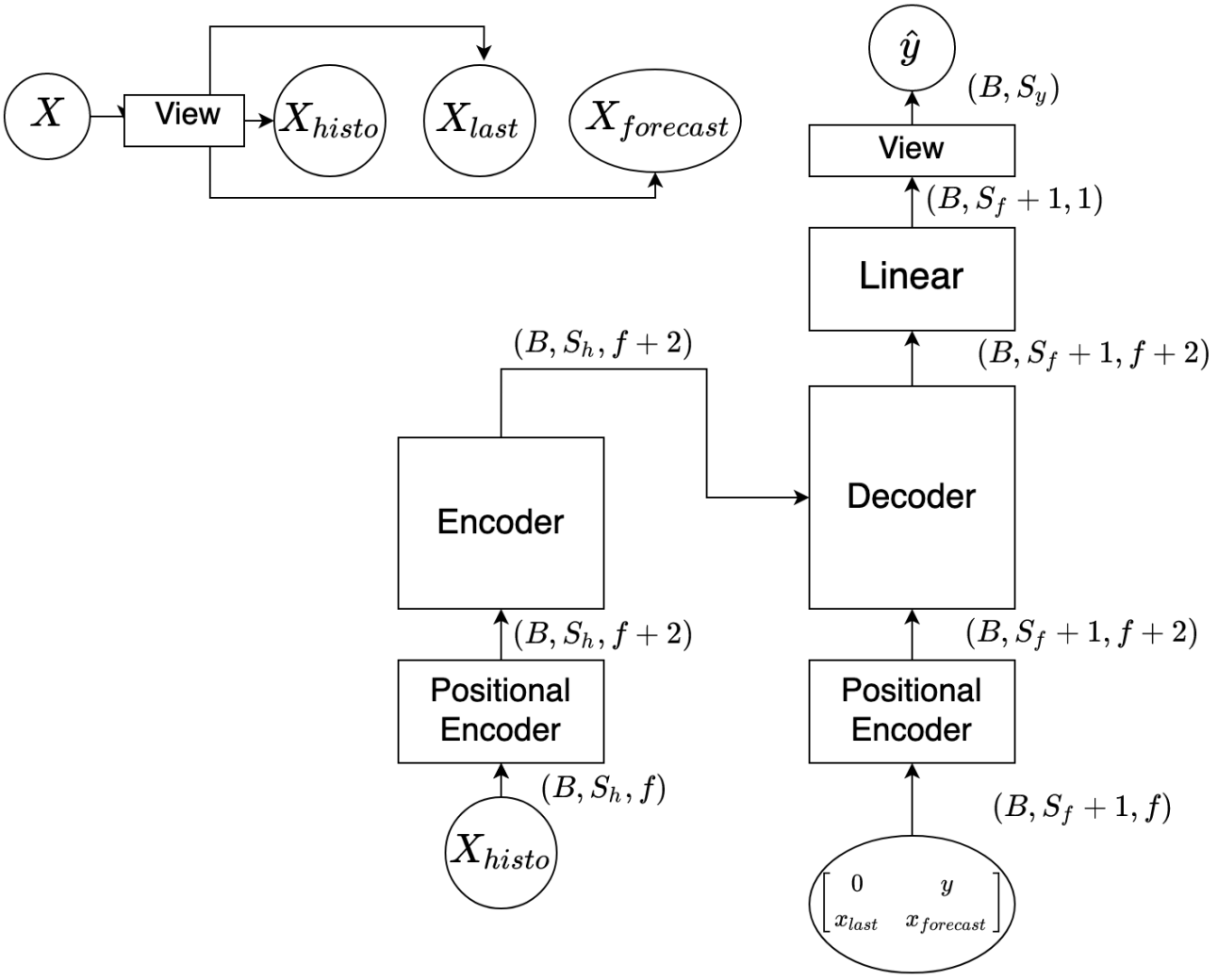

Figure 5.6: Encoder Decoder Transformer architecture: same architecture as in [8] except that we remove the softmax output layer by a linear one.

Chapter 6

Experiments on ORES Dataset

In this chapter, results are displayed for the training, validation and testing set. However, to select the different models the assessment was only done on the validation set. It implies that if some selections were done, the performance displayed in each table is the performance obtained by the model which performs the best on the validation set. When comparing the results between the tables and the curve losses, it is important to notice that the curve losses are expressed in MSE while the tables are most of the time expressed in RMSE.

6.1 Baselines

In this section, we will explore the results of the baselines described in subsection 2.3.12. In Table 6.1, the results for the training and validation set are displayed. What is important to remember is that the persistence and climatology method perform with a RMSE of approximatively 0.21 with a standard deviation of 0.15. We will analyse our results in the light of those results. If they do not perform better, the models do not learn anything and are useless.

Model	train	valid	test
Persistence	$\mathbf{0.1860 \pm 0.1487}$	$\mathbf{0.2102 \pm 0.1536}$	0.2241 ± 0.1828
Climatology	0.1893 ± 0.1098	0.2166 ± 0.1409	$\mathbf{0.2196 \pm 0.1219}$

Table 6.1: NRMSE ($\mu \pm \sigma$) on MAR train, validation and test set

6.2 Sklearn Models

In this section, we will discuss the results obtained with the Extra Tree model and the Random Forest. As mentioned in section 5.3, the model is in fact composed of fh models with fh the forecasting horizon. In Table 6.2, we can see that both models perform to around 7% better on the RMSE than the baselines.

The extra Tree weights $114.7MB$ and the Random Forest weights $113.7MB$ for the entire model.

Model	train	valid	test
ExtraTree	0.0443 ± 0.0231	0.1434 ± 0.0850	0.1463 ± 0.0640
Random Forest	0.0448 ± 0.0231	0.1412 ± 0.0860	0.1442 ± 0.0648

Table 6.2: NRMSE ($\mu \pm \sigma$) on MAR train, validation and test set

6.3 Comparison RNN

In this section, we will compare the results over our three architectures of RNN described in section 5.5: simple RNN, architecture and history forecast context RNN. In order to compare them, we ran 40 epochs of training with an MSE loss. Each model is trained with an *hidden_size* equals to 256 and 512. The curve losses are displayed in Figure 6.1 while the performance of the model which performs the best on the validation set is displayed in Table 6.4.

In Figure 6.1, we can clearly see that the simple RNN cannot achieve the performance of the other models either on the training or the validation set. Nevertheless, a bigger *hidden_size* helps to learn quicker. Indeed, the simple rnn with an *hidden_size* equals to 512 converges around 27 epochs while the simple RNN with *hidden_size* equal to 256 do not achieve any landing in the given 40 epochs.

As displayed in Table 6.4, the best performance is achieved by the history forecast context RNN with an *hidden_size* of 512 closely followed by itself with an *hidden_size* of 256 and the architecture RNN. Indeed, when looking at Figure 6.1, we can see the convergence of the 4 models (history forecast and architecture for *hidden_size* of 256 and 512) towards more or less the same MSE on the validation set. Then, we test the models with an *hidden_size* of 1024 but the performance is poorer *e.g.* 0.1372 ± 0.0825 on the validation set with history forecast.

An interesting behavior is the fact that the models with a bigger *hidden_size* (*i.e.* 512) have a quicker convergence than the smaller one. Indeed, the curve losses have the same kind of shape but stretched for smaller *hidden_size*.

Moreover, we can notice a big increase of performance compared to the baselines. The smallest RMSE worths 0.1355 which means an error in mean of 13.55% of the maximum power production. Consequently, this is 7.47% better than the baselines. We also notice a decrease in the standard deviation this means that the dispersion of poor and good predictions around the mean is decreased.

Lastly, let us discuss the weight of the models. Indeed, by looking at Table 6.3, we can see that history forecast is maybe heavier than the orther simple RNN and architecture RNN but simple RNN even with bigger *hidden_size* does not achieve good results and architecture RNN has a decrease in performance on the validation set from 512 to 1024 of hidden size.

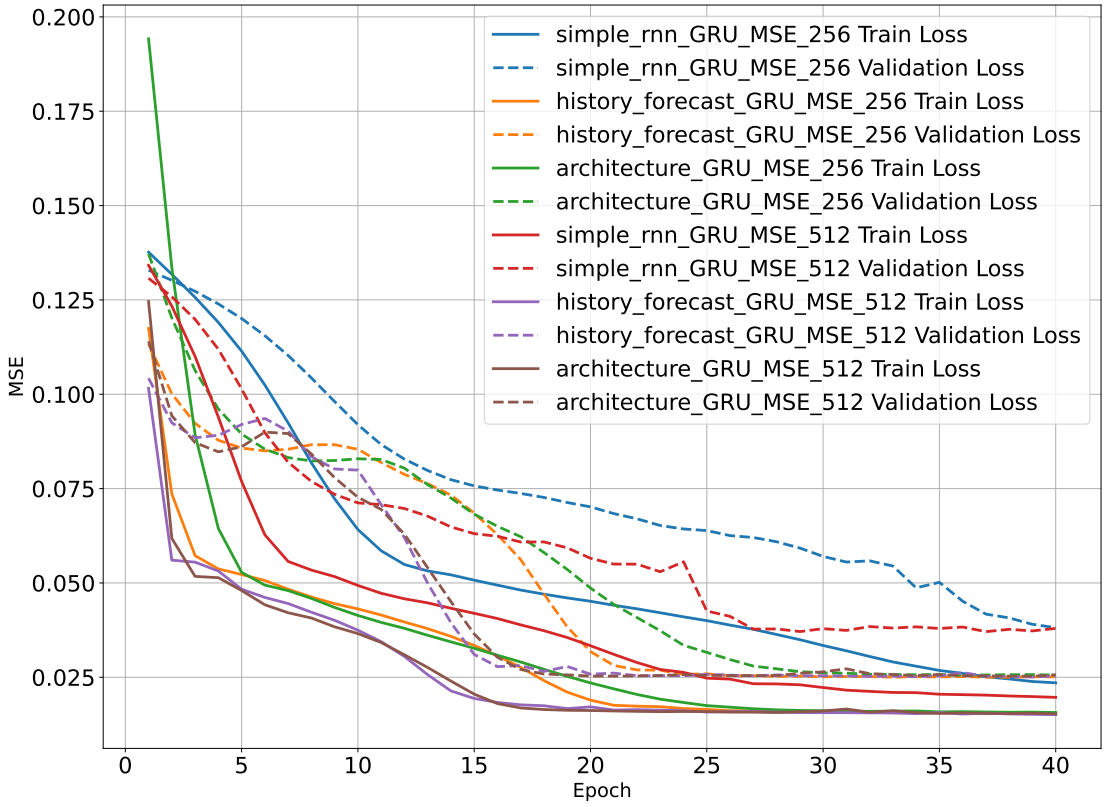

Figure 6.1: Curve loss of the RNNs

Model	simple RNN			architecture GRU			history forecast GRU		
Hidden Size	256	512	1024	256	512	1024	256	512	1024
Weights	1.3	4.6	17.7	1.1	4.3	16.9	5.4	21.2	84.3

Model	History forecast			
Cell	GRU	BRC	nBRC	Hybrid
Weight	21.2	2.3	14.9	10.2

Table 6.3: Weights in MB of the RNN

Model - Cell - Loss - Hidden Size	train	valid	test
simple RNN GRU MSE 256	0.1351 ± 0.0774	0.1832 ± 0.0991	0.1744 ± 0.0897
history forecast GRU MSE 256	$\mathbf{0.1055 \pm 0.0608}$	0.1357 ± 0.0841	0.1436 ± 0.0626
architecture GRU MSE 256	0.1092 ± 0.0602	0.1378 ± 0.0814	0.1464 ± 0.0627
simple RNN GRU MSE 512	0.1247 ± 0.0676	0.1675 ± 0.0964	0.1451 ± 0.0808
history forecast GRU MSE 512	0.1063 ± 0.0617	$\mathbf{0.1355 \pm 0.0831}$	$\mathbf{0.1352 \pm 0.0588}$
architecture GRU MSE 512	0.1074 ± 0.0614	0.1368 ± 0.0823	0.1393 ± 0.0600
simple RNN GRU MSE 1024	0.1167 ± 0.0636	0.1610 ± 0.0939	0.1421 ± 0.0787
history forecast GRU MSE 1024	0.1078 ± 0.0628	0.1372 ± 0.0825	0.1374 ± 0.0584
architecture GRU MSE 1024	0.1074 ± 0.0589	0.1380 ± 0.0820	0.1474 ± 0.0631

Table 6.4: NRMSE ($\mu \pm \sigma$) on MAR train, validation and test set

6.4 Comparison Cells in History forecast

In this experiment, we re-use the history forecast context model and we apply to it different cells of Recurrent Neural Network. We use GRU,BRC, nBRC and Hybrid cells. In Figure 6.2, we can see that the GRU is clearly the best one. Its training loss does not decrease as much as the other ones but the validation loss is better since the twentieth epoch. It seems that the other type of cells tends to overfit a little bit the dataset in comparison with the GRU. However, what is interesting to notice is the light weight of the BRC cell. Indeed, by looking at Table 6.3, we can see that the BRC history forecast weigths only $2.3MB$. Moreover, although the BRC cell does not perform really better than architecture RNN on the validation set but performs far better on the test set with an RMSE of 0.1355 ± 0.0671 which is the second better score behind the GRU history forecast with 0.1352 ± 0.0588.

In general, all the cells perform better than the baselines. There is also no significant differences in performance with respect to the type of cells.

Model -	train	valid	test
history forecast GRU MSE 512	0.1063 ± 0.0617	$\mathbf{0.1355 \pm 0.0831}$	$\mathbf{0.1352 \pm 0.0588}$
history forecast BRC MSE 512	$\mathbf{0.0991 \pm 0.0556}$	0.1398 ± 0.0866	0.1355 ± 0.0671
history forecast HybridRNN MSE 512	0.1016 ± 0.0562	0.1405 ± 0.0852	0.1356 ± 0.0658
history forecast nBRC MSE 512	0.1055 ± 0.0565	0.1400 ± 0.0832	0.1377 ± 0.0622

Table 6.5: NRMSE ($\mu \pm \sigma$) on MAR train, validation and test set

6.5 Comparison Training on different Losses

In this experiment, we re-use our best model on the RMSE validation metric *i.e.* the history forecast context RNN and we train it with different losses. We use (1) MSE, (2) MAE and (3) MSE plus SMAPE below denominated as MSEsMAPE.

In Figure 6.3, we can see the MSE evolution during the training with respect to the number of epoch. We can see better convergence properties with the loss combining the MSE and SMAPE losses.

In Table 6.6, we can notice that the best performance with respect to a given metric is always achieved by the training loss MSEsMAPE except for the test set where the best performance according to MAE and SMAPE is achieved by the model trained with the MAE metric. However the MSEsMAPE training is really close.

In general, if the results with respect to a given loss on the train set are not significantly better, the interest is on the validation and test set. Indeed, we achieve the best performance on the validation and test sets with the MSEsMAPE training. It seems that combining both metrics bring a better generalization property.

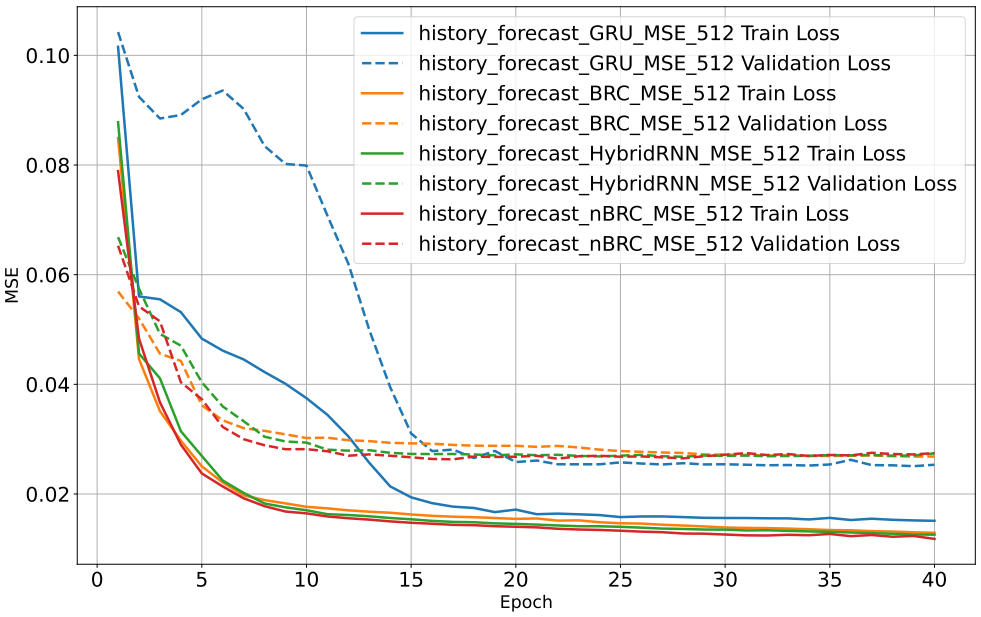

Figure 6.2: Comparison of cells: the GRU stays the best but the other cells have a smoother convergence.

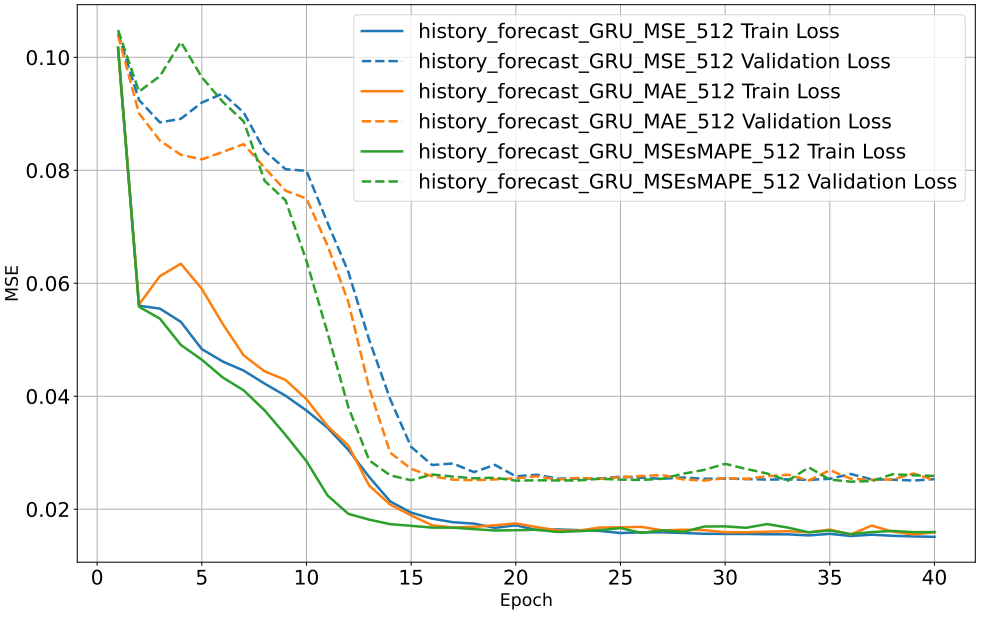

Figure 6.3: Comparison of Losses: All losses perform relatively closely in term of RMSE but the MSE and SMAPE loss achieve a better result on the validation set.

	Metric/Loss	MSE	MAE	MSEsMAPE
train	RMSE	0.1063 ± 0.0617	0.1068 ± 0.0660	$\mathbf{0.1060 \pm 0.0654}$
	MAE	0.0853 ± 0.0496	0.0837 ± 0.0537	$\mathbf{0.0837 \pm 0.0527}$
	SMAPE	0.7156 ± 0.3916	0.7153 ± 0.3915	$\mathbf{0.7016 \pm 0.3887}$
valid	RMSE	0.1355 ± 0.0831	0.1345 ± 0.0846	$\mathbf{0.1328 \pm 0.0863}$
	MAE	0.1128 ± 0.0700	0.1082 ± 0.0714	$\mathbf{0.1075 \pm 0.0722}$
	SMAPE	0.7024 ± 0.3873	0.7047 ± 0.3565	$\mathbf{0.6902 \pm 0.3640}$
test	RMSE	0.1352 ± 0.0588	0.1255 ± 0.0701	$\mathbf{0.1242 \pm 0.0681}$
	MAE	0.1141 ± 0.0520	$\mathbf{0.1012 \pm 0.0608}$	0.1014 ± 0.0572
	SMAPE	0.8720 ± 0.5637	$\mathbf{0.8207 \pm 0.5129}$	0.8297 ± 0.5293

Table 6.6: Loss Comparison where the vertical entries correspond to one training loss and the horizontal entries to a metric. We can see that the MSE plus SMAPE has good performance.

6.6 Comparison Transformer

In this section, we will compare the two transformers architectures developed *i.e.* Transformer (only composed of an encoder part) and Transformer Encoder Decoder.

6.6.1 Transformer

First, let us have a look at the curve losses displayed in Figure 6.4. We see a completely different behavior during the training in comparison with the training of RNNs. Indeed, there is more noise on the train and validation loss. This behavior is often observed when training transformers and we mitigate it by clipping the gradients absolute values when performing gradient descent. In Figure 6.4, we clearly see that the blue dashed line is under the other. The model Transformer with $N = 4$ times encoder block and a feedforward dimension of 256 seems to be the best. Without surprise by looking at Table 6.7, the best model according to the RMSE on the validation set is this later model.

In order to fine tune the hyperparameters it is quite difficult to extract obvious rules, for example, adding blocks is better or reducing the feedforward dimension. Indeed, in Table 6.7, we see a complex relationship between the embedded size and the number of encoder blocks.

In terms of performance, the Transformers beat the baselines but do not achieve so good results in comparison with simple Sklearn models.

Model - N - Loss - ff_dim	train	valid	test
Transformer 4 MSE 256	0.1085 ± 0.0532	$\mathbf{0.1553 \pm 0.0893}$	0.1435 ± 0.0854
Transformer 6 MSE 256	0.1149 ± 0.0611	0.1580 ± 0.0862	0.1595 ± 0.0972
Transformer 8 MSE 256	0.1115 ± 0.0568	0.1667 ± 0.0936	0.1544 ± 0.0931
Transformer 4 MSE 512	0.1107 ± 0.0558	0.1657 ± 0.0964	0.1448 ± 0.0742
Transformer 6 MSE 512	0.1080 ± 0.0586	0.1594 ± 0.1004	$\mathbf{0.1409 \pm 0.0857}$
Transformer 8 MSE 512	$\mathbf{0.1066 \pm 0.0538}$	0.1596 ± 0.0936	0.1502 ± 0.0886

Table 6.7: NRMSE ($\mu \pm \sigma$) on MAR train, validation and test set

6.6.2 Transformer Encoder Decoder

The training losses displayed in Figure 6.5 show the same kind of noisy training as the Transformer composed of only one encoder part. Only 4 models are displayed on the curve losses while 6 models are compared in Table 6.8 in order to not overload the figure. In this case we do not see with the validation losses a better model obviously. In Table 6.8, we can see that the best model selected on the validation set is the Transformer Encoder Decoder with $N = 8$ blocks of encoder and decoder with a feedforward dimension of 512.

Model - N - Loss - ff_dim	train	valid	test
TransformerEncDec 4 MSE 256	0.1111 ± 0.0591	0.1649 ± 0.1060	0.1859 ± 0.1314
TransformerEncDec 6 MSE 256	0.1095 ± 0.0583	0.1762 ± 0.1053	0.1703 ± 0.1031
TransformerEncDec 8 MSE 256	$\mathbf{0.1041 \pm 0.0523}$	0.1677 ± 0.0852	0.1716 ± 0.0955
TransformerEncDec 4 MSE 512	0.1088 ± 0.0533	0.1772 ± 0.1107	0.1735 ± 0.1044
TransformerEncDec 6 MSE 512	0.1057 ± 0.0488	0.1662 ± 0.1114	0.1896 ± 0.1235
TransformerEncDec 8 MSE 512	0.1038 ± 0.0519	$\mathbf{0.1593 \pm 0.1145}$	$\mathbf{0.1681 \pm 0.1112}$

Table 6.8: NRMSE ($\mu \pm \sigma$) on MAR train, validation and test set

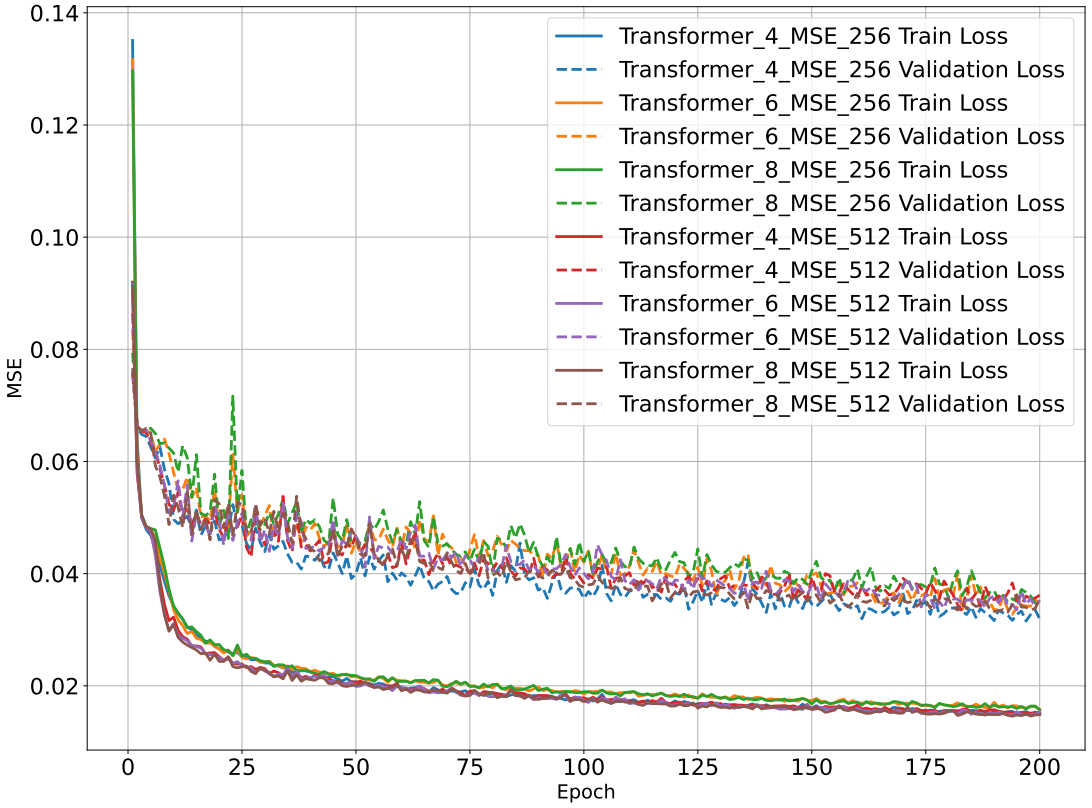

Figure 6.4: Curve loss of the Transformers: we can see that the Transformer composed with N=4 and 256 of feedforward size performs the best on the validation loss.

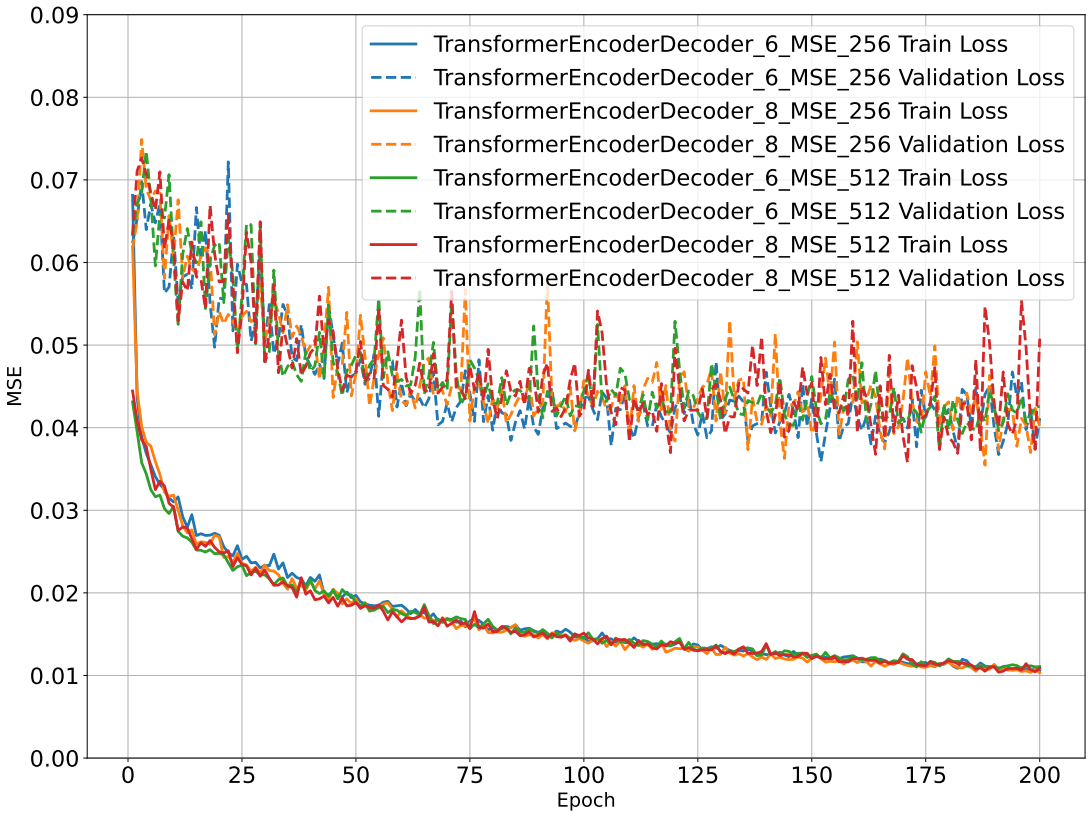

Figure 6.5: Curve loss of the Transformers Encoder Decoder: Difficult to see which one is the best because of the noisy behavior on the validation set.

6.6.3 Discussion

In the end, both models do not show spectacular performance. They are able to beat the baselines but do not perform better than the baselines. The problem may come from the embedded size of the vector which is smaller than in natural language processing (NLP) task[1] (7 instead of 512 with the default embedding in [14, pytorch]). which may limit the capacity of the model.

Another problem may be the overfitting. Indeed, here we are training the Transformers with a small dataset in comparison with the typical size of datasets in NLP tasks composed of billion words (see for example [25]). Moreover when looking at the training RMSE in Table 6.7 and Table 6.8, we can see that they are at the same order of magnitude than the losses obtained with RNN (see *e.g.* Table 6.4) but the validation are not which confirms a problem of overfitting the data.

Then, another aspect to mention is the difficulty to create and train transformers. Indeed, as already mentioned it was difficult to find patterns into hyper-parameters tuning in order to

[1]The embedded size is the size given to an encoded word in an n-dimensional space

find the best model. Besides, the need of a GPU in order to train or to make inference is also constraining because it is highly time-consuming while GPU is not required when training RNN in this work.

Nevertheless, other Transformer architectures can be imagined. For instance, instead of using the forecast matrix ϕ as described in section 5.1 in the decoder part maybe simply padding with zeroes the forecast features may be an option. Another idea could be to take inspiration from the history forecast context RNN and to have to encoder part: one for the history matrix and one for the gap matrix both connected into a new decoder of twice the embedded size.

6.7 Qualitative Results

In this section, we will perform a deep analysis of the results obtained with our best model: the history forecast context RNN trained with MSEsMAPE loss and with an hidden size of 512.

In Figure 6.12, we can see the best and worst results obtained with respect to the RMSE metric as well as those from a random sample: the sample six. Most of the time, the best and worst results with the MAE are the same but those with the SMAPE are not. The best results on validation and test set with respect to the SMAPE metric are displayed in Figure 6.17.

Let us first discuss the results according to the RMSE/MAE metric. The best results often lead to days when wind power production is low. It seems normal: if the model predicts low wind power for the next day, then the difference between zero and a small predicted power production will lead to small RMSE/MAE while the model may have difficulties to follow the power production by a day with stronger wind power. The worst results in term of RMSE tends to make this hypothesis. However, it is not really the case. To clarify this hypothesis, we display the results of a random sample (the sample 6) on the test and validation set in order to see that even with non-zero power production the model is able to follow the production.

Nevertheless, the worst results are really bad. This is why we conduct a deeper analysis by looking at the distribution of RMSE errors on the different sets. The distributions are displayed in Figure 6.21. In the histograms, we can see that the distribution is skewed towards 0.1 and we have an outlier with 0.46 of RMSE. We have the same problem on the validation set with the worst result which is really an outlier. In fact, most of the results on the validation and test set are around 0.10 and the worst results have a RMSE around 0.25.

Now, let us discuss the results obtained with respect to the SMAPE metric. First, it is interesting to notice that worst samples according to the SMAPE metrics leads to good results according to the RMSE or MAE. This must convince us of the importance of the loss/metric used. In this work, we focused ourselves most of the time on the RMSE loss but this latter cannot capture a kind of proportion error while the SMAPE can. However, for a dispatcher, we may wonder if it is really useful for him. Indeed, if the power production is close to zero and the prediction is thus a little bit higher leading to poor results in term of SMAPE but the important information - there will be not so much wind power tomorrow - is given then the dispatcher has the information needed for its task. However, the best results according to SMAPE are interesting to see. Indeed, even with the fact that they do not lead to the best RMSEs, we can see again that the model is able to follow the production with non-zero output power production.

Lastly, we conduct an analysis per timestamps (also known as lead time in [26, Chapter 2] see also in the appendix section A). The aim is to find some patterns *e.g.* the last lead times lead to poorer results. In Figure 6.25, the MAE and RMSE are displayed per timestamp *i.e.* the average is not done per sample but per timestamp. We also display the bias which needs to be equal to zero. We draw your attention to the fact that this is a necessary but not sufficient condition to get perfect results hence the RMSE and MAE also displayed. In the analysis per timestamp, we can see that for the training set everything looks homogeneous. Indeed, the mean of the MAE is equal to 0.0837 (see Table 6.6) on the train set. The results on the validation and test sets are not so homogeneous. Indeed, we can see an alteration of over and under estimations. The overestimation problem is worst on the test set in the beginning. In general we find more or less the same pattern as in the validation set anyway. The RMSE/MAE on the valid and test sets are not so flat than on the training set either. Nevertheless, no tendency appeared very clearly. Usually, when the records are available until the moment we predict *i.e.* there is no gap matrix, the first timestamps are better predicted as the later ones. We do not see this pattern in our case with a gap between the prediction time and the realisation.

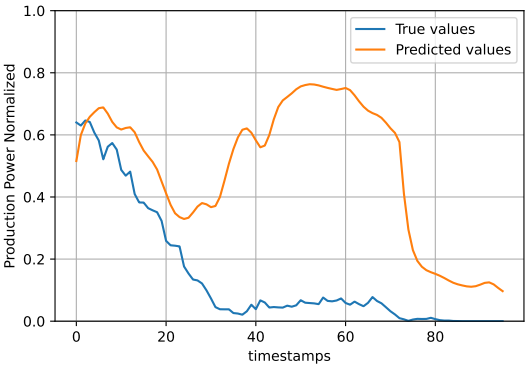

Figure 6.6: Worst on valid set w.r.t. the RMSE metric with RMSE 0.4236, MAE 0.3481 and SMAPE 1.3510

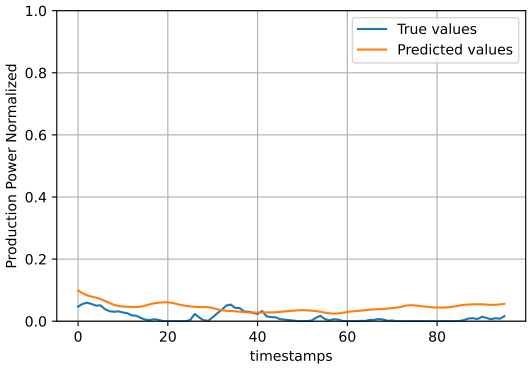

Figure 6.7: Best on valid set w.r.t. the RMSE metric with RMSE 0.0365, MAE 0.0334 and SMAPE 1.3179

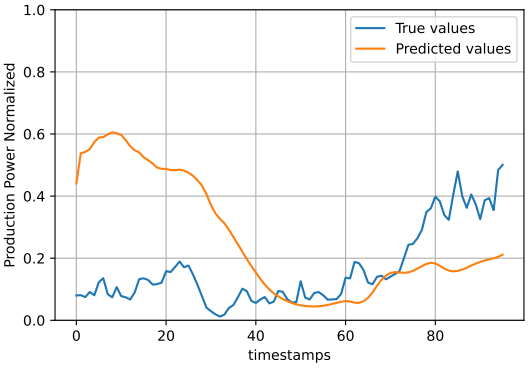

Figure 6.8: Worst on test set w.r.t. the RMSE metric with RMSE 0.2645, MAE 0.2085 and SMAPE 0.8509

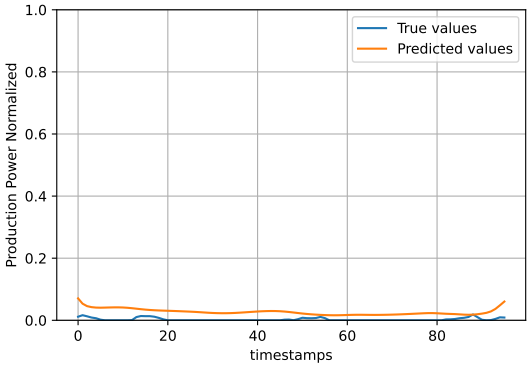

Figure 6.9: Best on test set w.r.t. the RMSE metric with RMSE 0.0260, MAE 0.0242 and SMAPE 1.6949

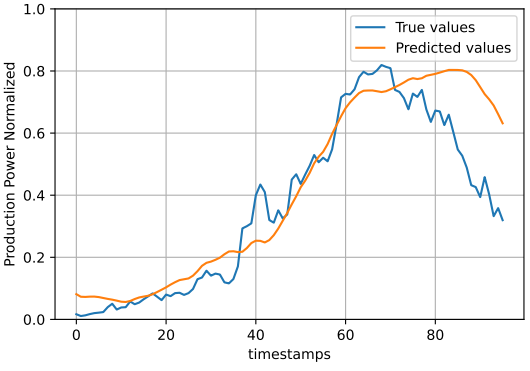

Figure 6.10: Sample 6 on valid set with RMSE 0.1252, MAE 0.0848 and SMAPE 0.3175

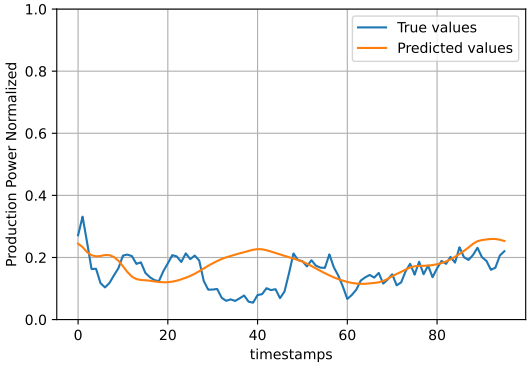

Figure 6.11: Sample 6 on test set with RMSE 0.0699, MAE 0.0533 and SMAPE 0.3419

Figure 6.12: History Forecast Context RNN On different samples

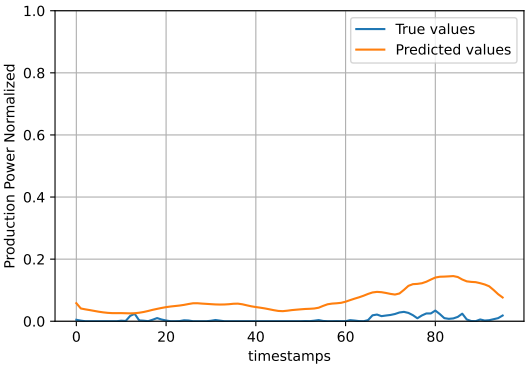

Figure 6.13: worst on valid set w.r.t. the SMAPE metric with RMSE 0.0693, MAE 0.0616 and SMAPE 1.7531

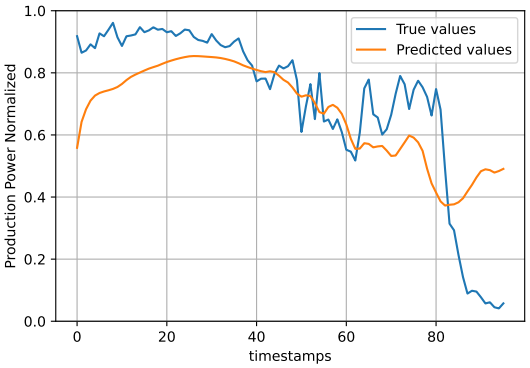

Figure 6.14: best on valid set w.r.t. the SMAPE metric with RMSE 0.1772, MAE 0.1374 and SMAPE 0.2904

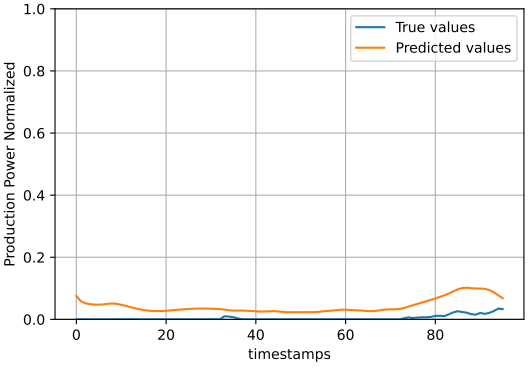

Figure 6.15: worst on test set w.r.t. the SMAPE metric with RMSE 0.0425, MAE 0.0390 and SMAPE 1.8113

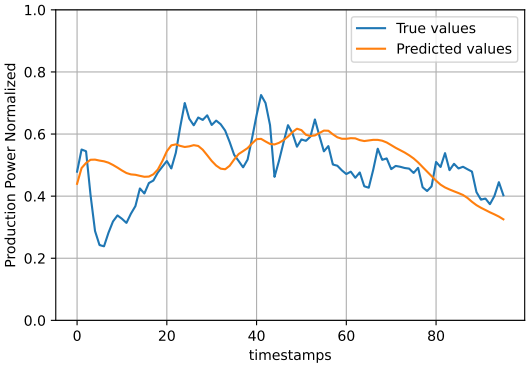

Figure 6.16: best on test set w.r.t. the SMAPE metric with RMSE 0.0970, MAE 0.0797 and SMAPE 0.1658

Figure 6.17: History Forecast Context RNN On different samples

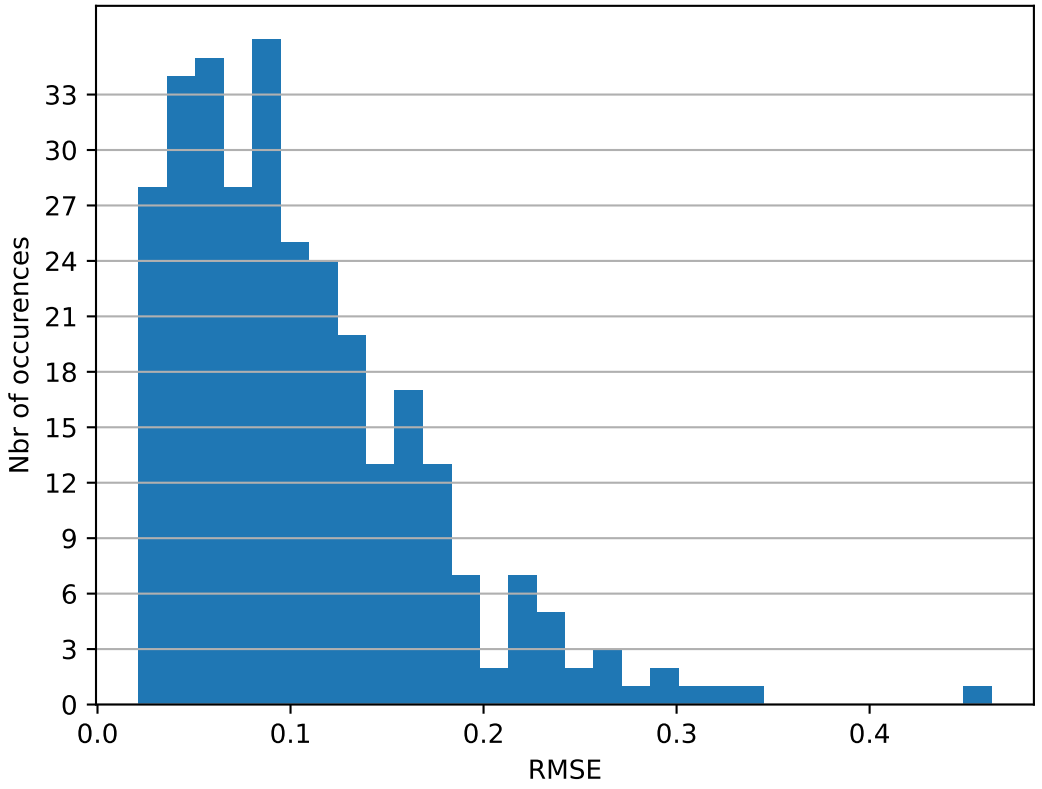

Figure 6.18: Histogram on the Train set

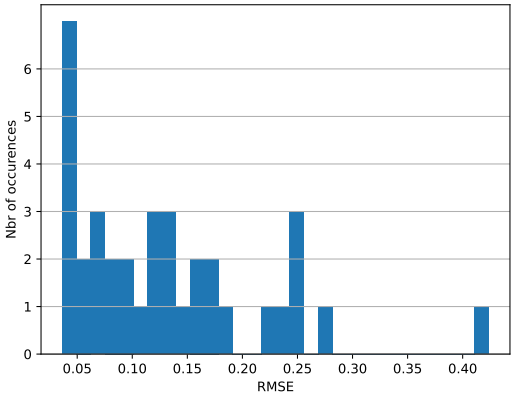

Figure 6.19: Histogram on the Valid set

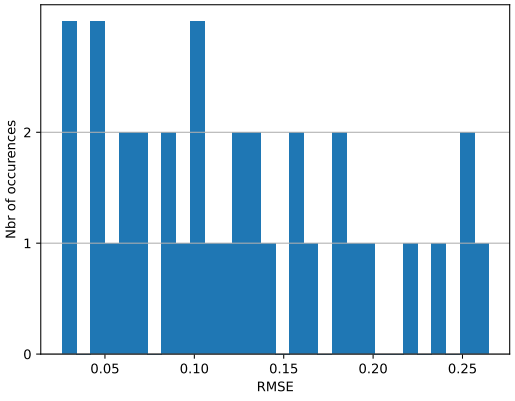

Figure 6.20: Histogram on the Test set

Figure 6.21: Histograms of the History Forecast Context RNN on the different sets.

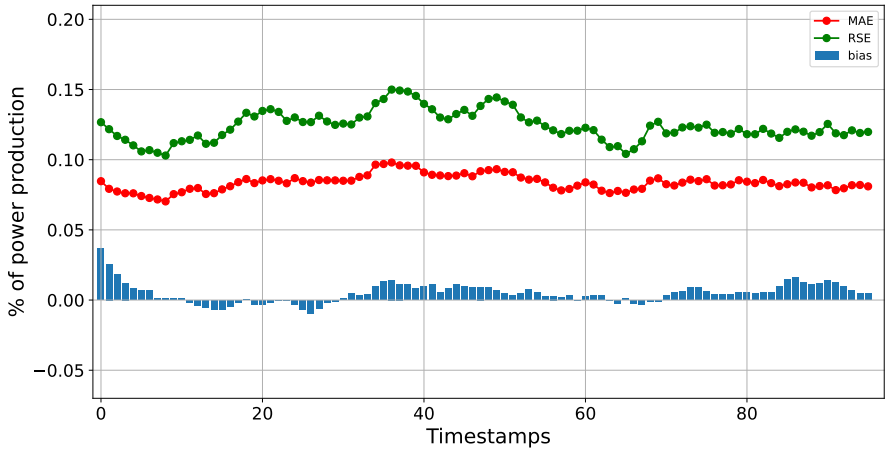

Figure 6.22: Timestamp analysis on the train set.

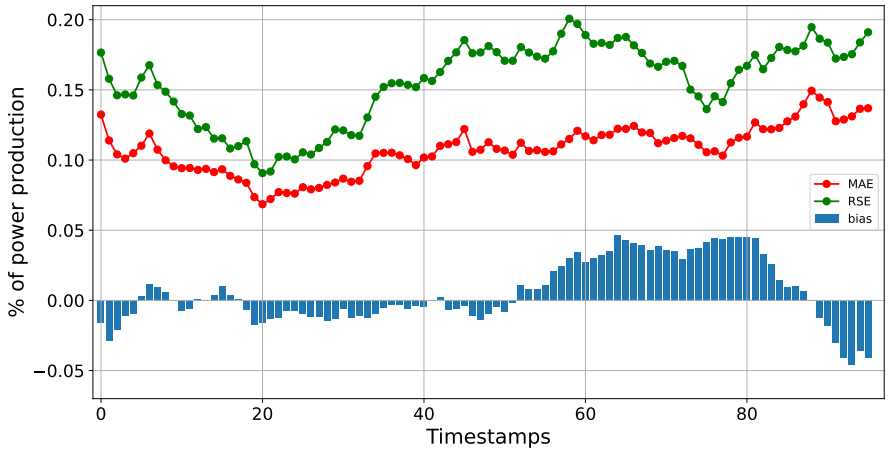

Figure 6.23: Timestamp analysis on the valid set.

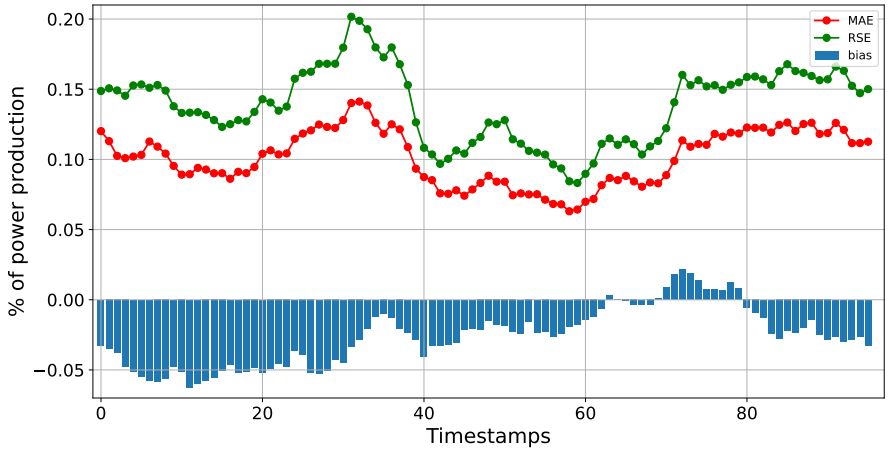

Figure 6.24: Timestamp analysis on the test set.

Figure 6.25: History Forecast Context RNN Timestamp analysis.

Chapter 7

Experiments on Gefcom Dataset

In this chapter, we will conduct several experiments on the Gefcom dataset. First, in section 7.1 we will analyse the results of the baselines models. Second, in section 7.2 we use the classical `sklearn` models. Third, in section 7.3, we will compare the different RNN architectures. In section 7.4, we will confront the two Transformers architecture developed. Then , in section 7.5, we will look at some qualitative results and perform some further analysis. Lastly, in section 7.6, we will compare the results obtained with the Gefcom dataset on the light of the results with the ORES dataset.

7.1 Baselines

In Table 7.1, we can see baselines performance on the Gefcom dataset. The Climatology model performs the best on each set. It achieves surprising good results on the test set with a RMSE of 0.26 which is better than on the train set. Nevertheless, let us remind the fact that the validation and test sets are composed of a few samples only.

Model	train	valid	test
Persistence	0.2859 ± 0.1855	0.3744 ± 0.1912	0.3113 ± 0.1813
Climatology	$\mathbf{0.2745 \pm 0.0843}$	$\mathbf{0.3455 \pm 0.0987}$	$\mathbf{0.2600 \pm 0.0917}$

Table 7.1: NRMSE ($\mu \pm \sigma$) on Gefcom train, validation and test set

7.2 Sklearn

In Table 7.2, we can see the results of our two `sklearn` models. The random forest model is slightly better on each set. Again, we can see the same phenomena as with the baselines where the test set performance is better than on the validation set.

Model	train	valid	test
ExtraTree	0.0616 ± 0.0299	0.2126 ± 0.0647	0.1750 ± 0.0750
Random Forest	$\mathbf{0.0612 \pm 0.0295}$	$\mathbf{0.2019 \pm 0.0697}$	$\mathbf{0.1748 \pm 0.0701}$

Table 7.2: NRMSE ($\mu \pm \sigma$) on Gefcom train, validation and test set

7.3 RNN

In Table 7.3, the results for the three architectures of RNN trained with a MSE loss for three different *hidden_size*. The simple RNN architecture with an *hidden_size* of 1024 performs better than the other models. In Figure 7.1, we clearly see that the simple RNN is better on the validation set than the other architectures. Moreover, the convergence behavior is different. Indeed, we can see that the simple RNN converges slowly but steadily while the other architectures converge quicker to their best performance but cannot achieve as good performance.

Model - Cell - Loss - Hidden Size	train	valid	test
simple RNN GRU MSE 256	0.1631 ± 0.0817	0.2060 ± 0.0846	0.1723 ± 0.0735
history forecast GRU MSE 256	0.1515 ± 0.0710	0.2007 ± 0.0674	0.1728 ± 0.0740
architecture GRU MSE 256	0.1599 ± 0.0774	0.2107 ± 0.0720	0.1787 ± 0.0776
simple RNN GRU MSE 512	0.1505 ± 0.0732	0.1952 ± 0.0799	0.1709 ± 0.0754
history forecast GRU MSE 512	0.1507 ± 0.0729	0.1979 ± 0.0704	0.1752 ± 0.0792
architecture GRU MSE 512	0.1580 ± 0.0756	0.2084 ± 0.0714	0.1775 ± 0.0780
simple RNN GRU MSE 1024	$\mathbf{0.1371 \pm 0.0679}$	$\mathbf{0.1776 \pm 0.0799}$	$\mathbf{0.1633 \pm 0.0777}$
history forecast GRU MSE 1024	0.1443 ± 0.0705	0.1954 ± 0.0673	0.1702 ± 0.0716
architecture GRU MSE 1024	0.1561 ± 0.0805	0.2043 ± 0.0751	0.1748 ± 0.0824

Table 7.3: NRMSE ($\mu \pm \sigma$) on Gefcom train, validation and test set

7.4 Transformer

In this section, we will describe the results that we get with the Transformer only composed of an Encoder part and then with the transformer composed of both an Encoder and Decoder.

7.4.1 Transformer Encoder

In Table 7.4, we can see all the Transformers' performance with an Encoder part tested with different hyper parameters. We analyse the influence of the number of stacked Encoders and the size of the feedforward Neural Networks. We can observe that the more stacked Encoders we have, the less good the performance is. The best performance is obtained with the Transformer

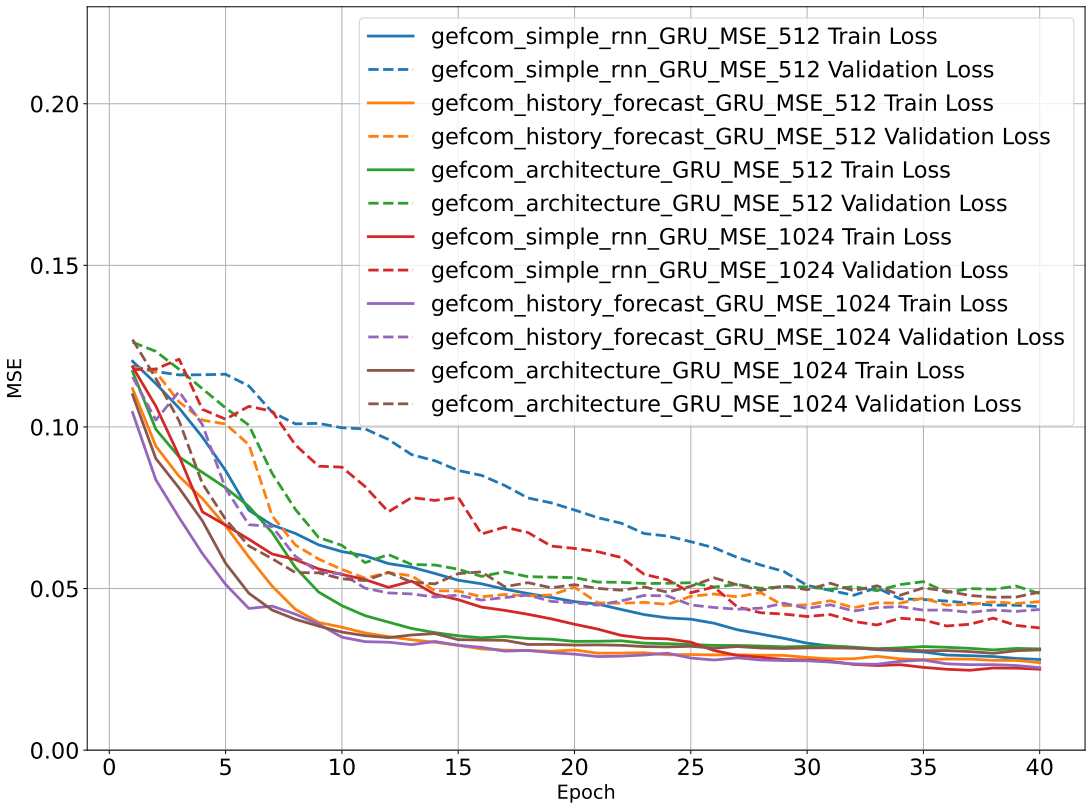

Figure 7.1: Curve loss of the RNNs on the gefcom dataset: the simple RNN composed of 1024 units has the best performance on the validation set.

only composed of one Encoder and with a feedforward size of 1024 units. This performance is close to the best RNN which performs with 0.1776 ± 0.0799 but a little bit less good with 0.01883 ± 0.0670. In the Figure 7.2, we can see the training losses of the transformers composed with 1 or 2 stacked Encoder and feedforward size equal to 512 and 1024. We clearly see that the validation loss of the Transformer with 1 stacked Encoder and 1024 as feedforward size is under the others.

Model - N - Loss - ff_dim	train	valid	test
Transformer 1 MSE 128	0.1613 ± 0.0683	0.2018 ± 0.0668	0.1755 ± 0.0679
Transformer 1 MSE 256	0.1648 ± 0.0661	0.2109 ± 0.0602	0.1765 ± 0.0620
Transformer 1 MSE 512	0.1594 ± 0.0641	0.2000 ± 0.0613	0.1768 ± 0.0643
Transformer 1 MSE 1024	$\mathbf{0.1504 \pm 0.0651}$	$\mathbf{0.1883 \pm 0.0670}$	$\mathbf{0.1681 \pm 0.0615}$
Transformer 2 MSE 512	0.1817 ± 0.0744	0.2213 ± 0.0656	0.1954 ± 0.0689
Transformer 2 MSE 1024	0.1781 ± 0.0700	0.2189 ± 0.0622	0.1876 ± 0.0612
Transformer 4 MSE 512	0.2561 ± 0.1093	0.2852 ± 0.0706	0.2513 ± 0.0974
Transformer 4 MSE 1024	0.2308 ± 0.0628	0.2642 ± 0.0581	0.2288 ± 0.0557
Transformer 6 MSE 512	0.3207 ± 0.1433	0.3631 ± 0.1079	0.3045 ± 0.1262
Transformer 6 MSE 1024	0.2634 ± 0.0697	0.2887 ± 0.0664	0.2566 ± 0.0692
Transformer 8 MSE 512	0.3448 ± 0.1478	0.3793 ± 0.1208	0.3185 ± 0.1312
Transformer 8 MSE 1024	0.2901 ± 0.0782	0.3254 ± 0.0762	0.2719 ± 0.0725

Table 7.4: NRMSE ($\mu \pm \sigma$) on Gefcom train, validation and test set

7.4.2 Transformer Encoder Decoder

In Figure 7.3, we can see the curve losses of the transformer composed with Encoder and Decoder. We can see very noisy curve losses for the validation set. By looking at Table 7.5, we can see that the Transformer Encoder on the validation set is composed of 1 stack of Encoder and Decoder and 1024 of feedforward size. Nevertheless, it is not the best model on the test set. Indeed, it is beaten by the same model but with a feedforward size of 512 but with a very close performance on the validation set.

We would like to emphasize the fact that this architecture is beaten by the `sklearn` models. Thereby, it would be interesting to look after new architectures for the Transformer composed of encoder and Decoder. However, it can be caused to the propency of Transformer to overfit the data. Again, the dataset is rather small.

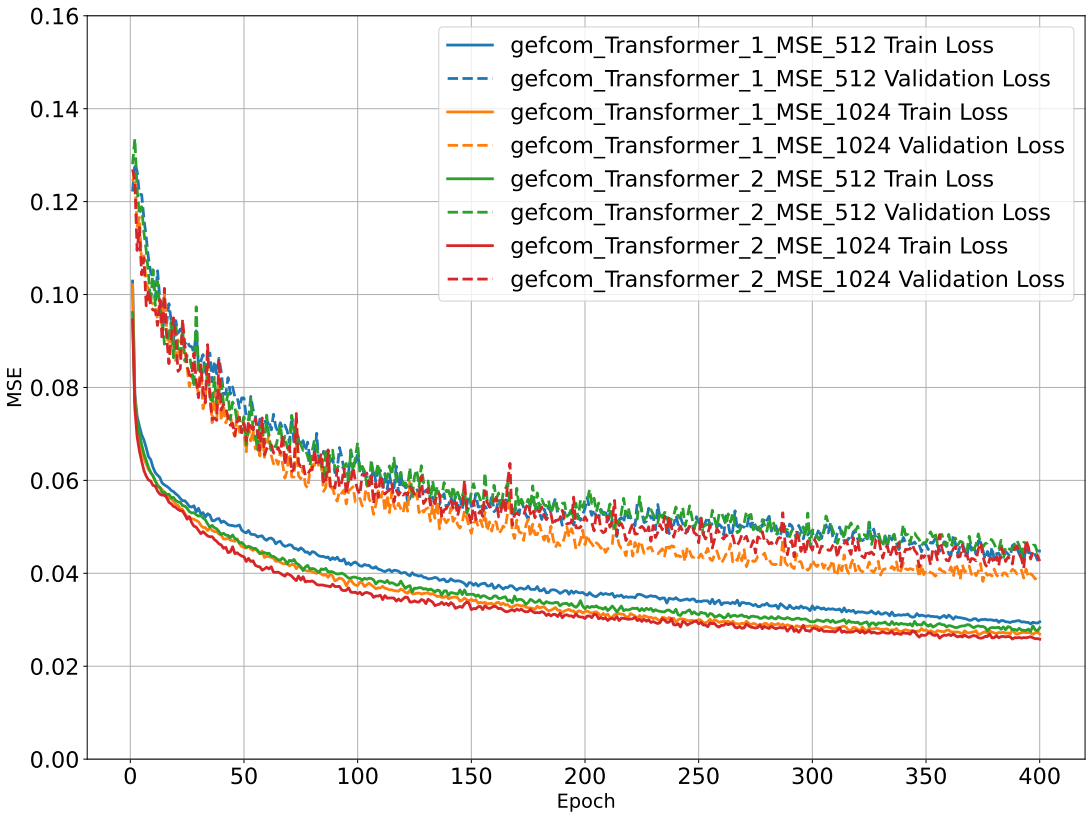

Figure 7.2: Curve losses of the Transformers: the one composed of 1 Encoder and Decoder and a feed forward size of 1024 performs the best on the validation set.

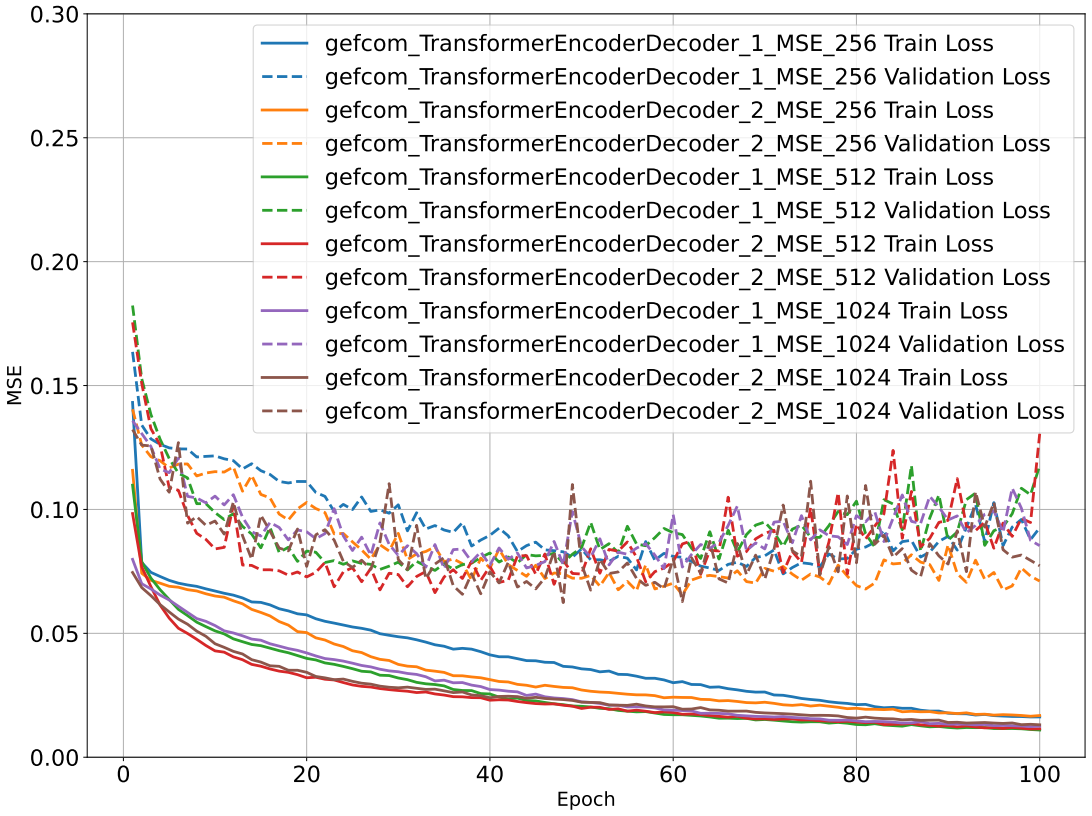

Figure 7.3: Curve loss of the Transformers Encoder Decoder on the gefcom dataset: difficult to draw a conclusion.

Model - N - Loss - ff_dim	train	valid	test
TransformerEncDec 1 MSE 256	0.2257 ± 0.0926	0.3173 ± 0.0890	0.2314 ± 0.1069
TransformerEncDec 2 MSE 256	0.2928 ± 0.1456	0.4292 ± 0.1864	0.3038 ± 0.1533
TransformerEncDec 1 MSE 512	$\mathbf{0.2030 \pm 0.0816}$	0.2653 ± 0.0848	$\mathbf{0.2186 \pm 0.0825}$
TransformerEncDec 2 MSE 512	0.2638 ± 0.1395	0.3700 ± 0.1641	0.2478 ± 0.1343
TransformerEncDec 1 MSE 1024	0.2054 ± 0.0784	$\mathbf{0.2605 \pm 0.0696}$	0.2271 ± 0.0880
TransformerEncDec 2 MSE 1024	0.3201 ± 0.1096	0.3411 ± 0.1112	0.3499 ± 0.1184
TransformerEncDec 4 MSE 256	0.3723 ± 0.2083	0.5844 ± 0.2607	0.3695 ± 0.2100
TransformerEncDec 6 MSE 256	0.4077 ± 0.2432	0.6541 ± 0.2918	0.4019 ± 0.2378
TransformerEncDec 8 MSE 256	0.4240 ± 0.2556	0.6878 ± 0.3053	0.4393 ± 0.2501
TransformerEncDec 4 MSE 512	0.2997 ± 0.1670	0.4168 ± 0.1975	0.3032 ± 0.1704
TransformerEncDec 6 MSE 512	0.3826 ± 0.1797	0.5098 ± 0.2183	0.3789 ± 0.1780
TransformerEncDec 8 MSE 512	0.3797 ± 0.1931	0.5342 ± 0.2431	0.3735 ± 0.1942
TransformerEncDec 4 MSE 1024	0.4559 ± 0.1693	0.5041 ± 0.1976	0.5245 ± 0.1825
TransformerEncDec 6 MSE 1024	0.4609 ± 0.1936	0.5079 ± 0.1961	0.5534 ± 0.2034
TransformerEncDec 8 MSE 1024	0.4534 ± 0.2109	0.5193 ± 0.2139	0.5641 ± 0.2144

Table 7.5: NRMSE ($\mu \pm \sigma$) on Gefcom train, validation and test set

7.5 Qualitative Results

In this section, we will have a look at the results obtained with the simple RNN with a *hidden_size* of 1024. In Figure 7.10, we can see the worst and best results according to the RMSE and the sample 6 on the validation and test set. As in section 6.7, the worst results are disapointing. It is why we conduct the same analysis with the histograms of the RMSE that we display in Figure 7.14. We can see that the distribution is more homogeneous than in section 6.7 with a clear outlier on the validation set. It seems that in case of high change in the production the model tends to have difficulties to predict them. Indeed, in each qualitative results the model gives more the general trend of the production but is not very precise in terms of drastic changes.

In Figure 7.18, we can see the timestamp analysis of the simple RNN. As for the ORES dataset, we do not notice any interesting patterns, except some alternation of over and under estimation on the valid and test sets while the error on the train set is rather constant.

7.6 Discussion comparison ORES and Gefcom Datasets

In this section, we will discuss the results obtained over the ORES and Gefcom dataset. First of all, we would like to highlight the fact that the models were developed and validated on the ORES dataset and only then, they were tested on the Gefcom dataset. The objective was to see their capacity to handle a new data distribution in the same practical setting of the day-ahead forecast. It is also important to notice the difference in temporal resolution, the ORES dataset is at the quarter resolution while the gecfom is at the hour resolution. This has a big influence on the size of the time series to handle. Indeed, the ORES dataset is composed of samples with an historic window, a gap window and a forecast window of sizes respectively equal to 96, 48 and 96 while it is only equal to 24, 12 and 24 for the Gefcom dataset.

It is interesting to notice that the best model on the ORES dataset (*i.e.* the history forecast context rnn) is not the same as the best one on the Gefcom dataset (*i.e.* the simple RNN). Moreover, the simple RNN is the model which is unable to capture good patterns on the ORES dataset but outperforms the other architectures on the Gefcom dataset. This fact shows the importance of testing different architectures with different data distribution and resolution. Indeed, one hypothesis to understand the cause of the ineffectiveness of the context RNN could be the fact that it is specially designed to handle separately data distribution: the forecast features and the historical features.

In the qualitative results, the models applied on the ORES dataset section 6.7 are more able to follow changes in the power production while the qualitative results on Gefcom section 7.5 are not. They show a general trends more than an high resolution forecast.

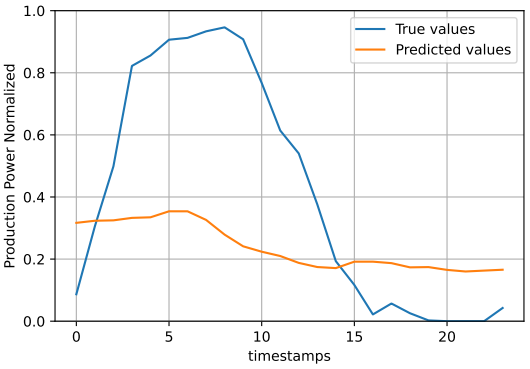

Figure 7.4: Worst on valid set w.r.t. the RMSE metric with RMSE 0.3704, MAE 0.3048 and SMAPE 1.0822

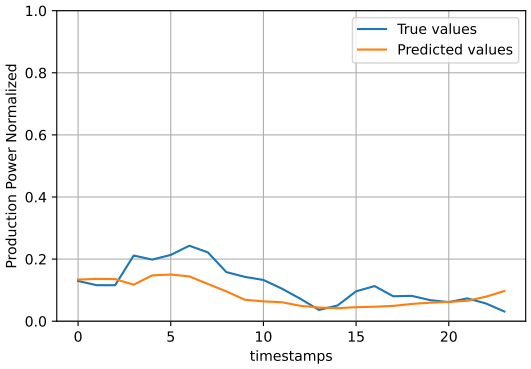

Figure 7.5: Best on valid set w.r.t. the RMSE metric with RMSE 0.0526, MAE 0.0425 and SMAPE 0.4101

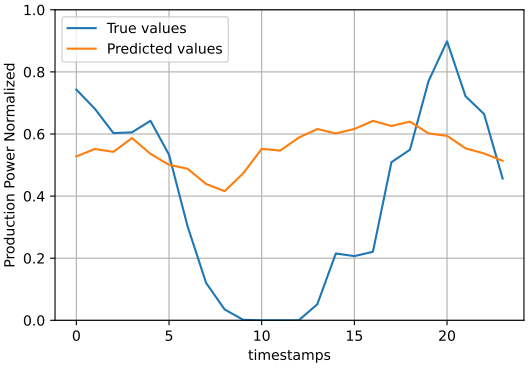

Figure 7.6: Worst on test set w.r.t. the RMSE metric with RMSE 0.3251, MAE 0.2674 and SMAPE 0.7676

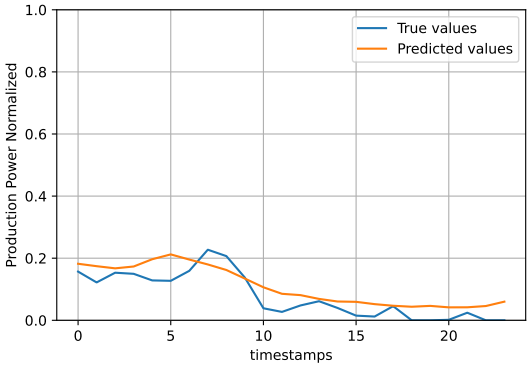

Figure 7.7: Best on test set w.r.t. the RMSE metric with RMSE 0.0440, MAE 0.0386 and SMAPE 0.7592

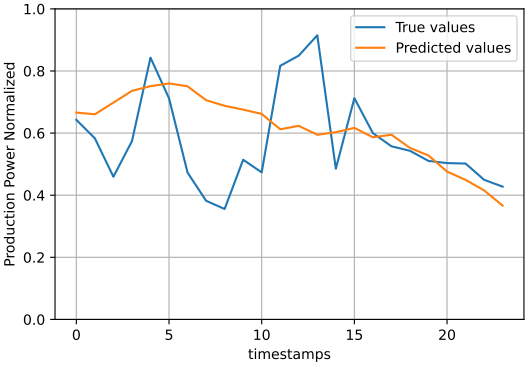

Figure 7.8: Sample 6 on valid set with RMSE 0.1687, MAE 0.1309 and SMAPE 0.2168

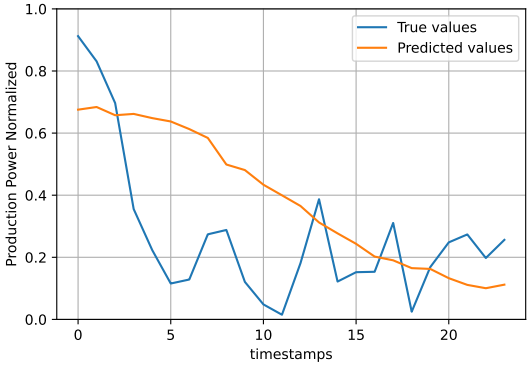

Figure 7.9: Sample 6 on test set w.r.t. the RMSE metric with RMSE 0.2587, MAE 0.2147 and SMAPE 0.7506

Figure 7.10: Simple RNN On different samples

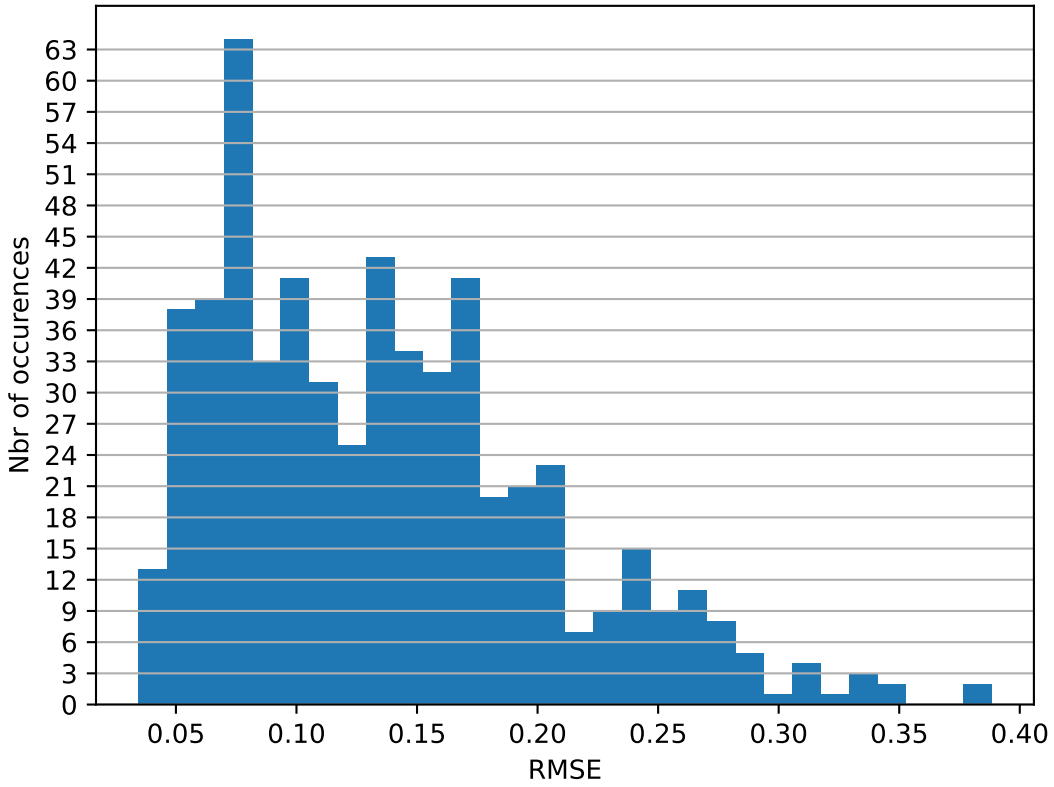

Figure 7.11: Histogram on the Train set.

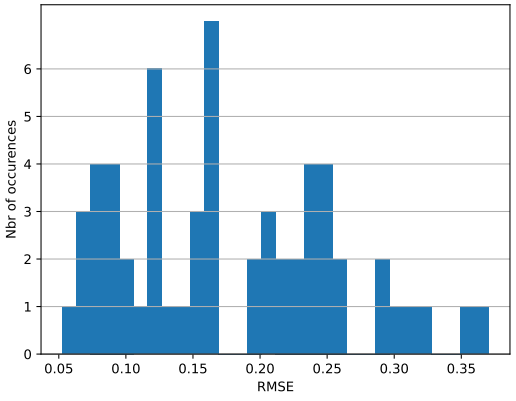

Figure 7.12: Histogram on the Valid set.

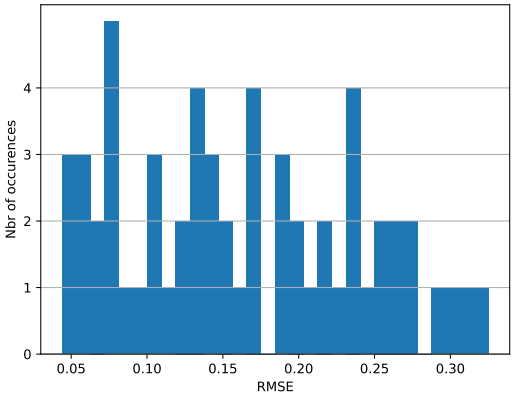

Figure 7.13: Histogram on the Test set.

Figure 7.14: Histograms of the simple RNN on the different sets on the gefcom dataset.

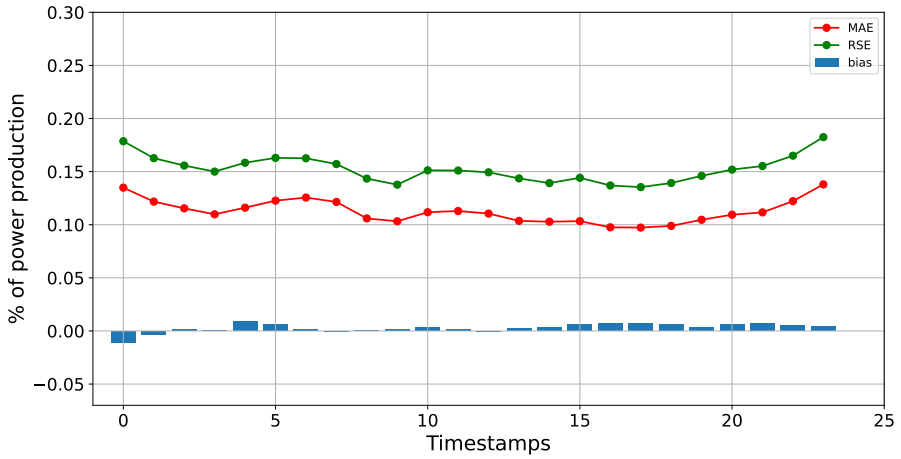

Figure 7.15: Timestamp analysis on the train set.

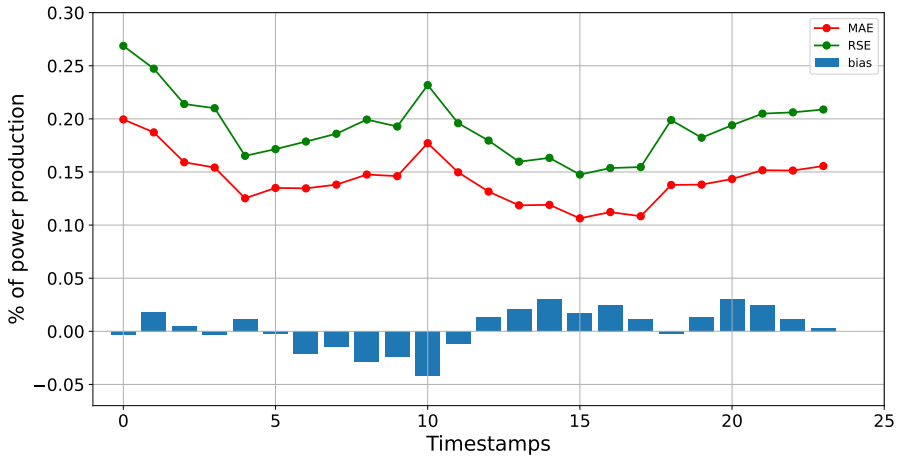

Figure 7.16: Timestamp analysis on the valid set.

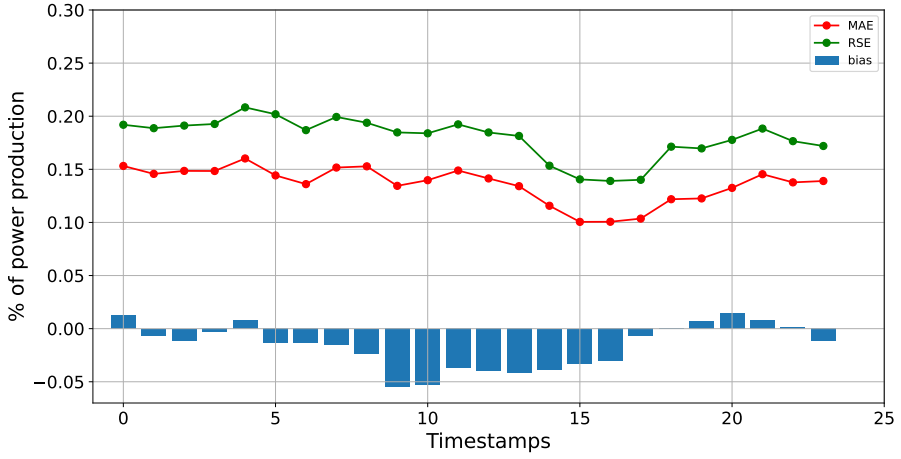

Figure 7.17: Timestamp analysis on the test set.

Figure 7.18: Simple RNN Timestamp analysis on Gefcom dataset.

Chapter 8

Conclusion

In this chapter, we will first discuss some future work. Then, we will conclude this work by summarizing what has been done in this work and answering the questions mentioned in chapter 1.

8.1 Future Work

This work is an open door towards the field of wind power forecasting. The originality of the work lies more into the architectures used than in an exhaustive comparison of the methods existing. There are many aspects to explore deeper.

First, exploring different data cleansing techniques may be interesting. Indeed, in this work there is no data cleansing applied except the averaging to get the time resolution expected. Moreover, wind turbine data are really noisy. Some techniques are developed (*e.g.* [27]) in order to improve the quality of the features in the datasets. Furthermore, new features could be added as mentioned in the chapter 3 *e.g.* wavelet decomposition algorithm is often used to extract frequency features of the wind signal.

Second, getting more data would be of great help to better estimate the performance of our models but also to train them better. We already discussed the over-fitting problem with the transformers' architectures but this could be alleviate by more training data.

Third, new architectures could be developed. We already mentioned some alternative architectures for the transformer models but new RNN architectures could be interesting to investigate as well as convolutional neural network models not used in this work. Besides, the teacher/professor forcing techniques [28] could be implemented too. These techniques could be helpful for convergence purposes but also to get better performance. We could also explore the extension to multiple layers in the RNN and to the bidirectional RNN which are not explored in this work. Moreover, an ensemble model using all the best models developed throughout this work could be used in order to maybe get better performance.

Fourth, exploring transfer learning. In this work, we got access to three wind farms in Belgium. It would be interesting to see if our algorithms generalize well on the other wind farms without being trained on them. In chapter 3, it was mentioned that most of the time the models generalize pretty badly over wind farms on which the model is not trained. Thus, it

could be interesting to try transfer learning techniques to see if we need less data to converge and understand patterns on other wind farm with one model already trained on another one. Transfer learning could help in cases where only a few data is available.

Fifth, to extend our model architectures towards probabilistic forecast as described in section A. Indeed, probabilistic forecast is more and more developed because it gives more information about the uncertainty associated with the forecast of our model. This is an interesting topic in order to dive into a more wide class of problems like automatic control of production power plants in order to balance the network optimally and to minimize the demand of technical staff maintaining the network day to day.

8.2 Conclusion

In conclusion, we tried in this work to forecast wind power thanks to meteorological data obtained with the model MAR developed at the University of Liège and ORES power production records over Belgium. We created ourselves the dataset quarter per quarter of hour with both sources of data. In a second time, we also used the gefcom dataset at an hour resolution. For both datasets, we work into the practical setting of the day ahead spot market. In this setting, at midday we should forecast the data of interest for the entire next day. This leads to the different matrices we described across the problem statement in section 5.1: the historical, gap and forecast matrices. They contain either features available or forecast features.

To make these forecasts, we developed several statistical models from naive methods towards classical machine learning models and advanced deep learning methods. Indeed, we tested some baselines useful in times series forecasting in order to assess the performance of our newly developed models. Other less naive baselines were also done with Random Forests and Extra Tree models implemented thanks to [29]. Then, we implemented three architectures of recurrent neural networks (RNN). Lastly, we developed two architectures of transformers, a very recent architecture which displayed great results on natural language processing tasks. Thus, we modify the architecture to fit the needs of our regression task. These architectures presented different performance depending on the dataset.

First, on the ORES dataset, the RNN architectures all beat the naive baselines but only the two last beat the sklearn baselines. Moreover, we compared with the best RNN *i.e.* the history forecast context RNN the influence of newly discovered RNN cells like BRC, nBRC and Hybdrid cells against the more classic GRU. Even if the GRU stayed the best cell, it was interesting to notice the high performance achieved by the BRC cell while decreasing a lot the size of the model. Another analysis comparing the training losses was performed with this last RNN showing that a combining loss with MSE and SMAPE improved the performance. Then, the transformers did not show as great results as we get with the RNN. Nevertheless, in subsection 6.6.3 we discuss potential improvements of these architectures.

After analyzing the results of all experiments, we made a deep analysis of the results we get with our selected best model based on its performance in terms of RMSE on the validation set. This best model is the history forecast context RNN an original architecture from this work described in subsection 5.5.3. So it was trained with a loss combining the MSE and the SMAPE metric. It has an hidden size of 512 units. The analysis of the results was done by

looking at some qualitative results with the best and worst forecasts done on the valid set and test set. We also look at the histograms of the losses on the training, validation and testing sets in order to better understand the distributions of the results. Lastly, we made an analysis per timestamp hoping to find some relevant patterns but no one was found. The analysis of the results showed that the metrics we used were really important to assess our results. Finally, the forecasts we produce do not perfectly follow the power production. Sometimes they miss some peaks or changes in the production. However, they often give a good idea of what is going to be produced on the next day. The forecast of renewable energy is a hot topic and still needs some progress. However, thanks to AI we already get some good models in order to get some insights at the resolution of the quarter for the next day when predicting at midday.

Second, on the gefcom dataset we get different results than on the previous dataset. It is also an RNN which performs the best but the one which was the worst on the other dataset: the simple RNN. For this best RNN, the same analysis of the qualitative results was done with an histogram and timestamp analysis. The timestamp analysis again do not show interesting patterns. Besides, it seems that in contrary with the models applied on the ORES dataset, the output on the gefcom were more general trends than a close follow of the production. As far as Transformers are concerned, they displayed interesting results. Indeed, the Transformer only composed of an encoder part was close to the best RNN but the Encoder Decoder Transformer had bad performance. Even if it stays better than the naive baselines, it is worse than the `sklearn` baselines.

Finally, we could conclude that advanced AI techniques beat naive baselines and give some better insights on the forecasts of renewable energy. The new developed cells introduced in [3] did not beat the classic GRU but were close in performance and the BRC cell displayed an interesting light weight property. The RNNs stay the best algorithms on this task in comparison with Transformers, Random Forest, Extra Trees and naive baselines.

Glossary

AI Artificial Intelligence.

ANN Artificial Neural Network.

ARCH Auto Regressive Conditional Heteroscedasticity.

ARIMA Auto Regressive Integrated Moving Average.

ARMA Auto Regressive Moving Average.

BRC Bistable Recurrent Cell.

GPU Graphical Process Unit.

GRU Gated Recurrent Unit.

KDE Kernel Density Estimation.

LSTM Long Short Term Memory.

Lube Lower and upper bound estimation.

MAE Mean Absolute Error.

MAPE Mean Absolute Percentage Error.

MAR Modèle Atmosphérique Régional.

MLP Multi Layer Perceptron.

MSE Mean Square Error.

nBRC Neuromodulated Bistable Recurrent Cell.

NLP Natural Language Processing.

NRMSE Normalized Root Mean Square Error.

NWP Numerical Weather Prediction.

QR Quantile Regression.

RMSE Root Mean Square Error.

RNN Recurrent Neural Network.

SMAPE Symmetrical Mean Absolute Percentage Error.

TSO Transmission System Operator.

WD Wavelet Decomposition.

Bibliography

[1] Xavier Fettweis, Jason E. Box, Cécile Agosta, Charles Amory, Christoph Kittel, Charlotte Lang, Dirk van As, Horst Machguth, and Hubert Gallée. Reconstructions of the 1900–2015 greenland ice sheet surface mass balance using the regional climate mar model. *The Cryosphere*, 11:1015–1033, 2016.

[2] Tao Hong, Pierre Pinson, Shu Fan, Hamidreza Zareipour, Alberto Troccoli, and Rob J. Hyndman. Probabilistic energy forecasting: Global energy forecasting competition 2014 and beyond. *International Journal of Forecasting*, 32:896–913, 2016.

[3] Nicolas Vecoven, Damien Ernst, and Guillaume Drion. A bio-inspired bistable recurrent cell allows for long-lasting memory. *PLOS ONE*, 16(6):e0252676, Jun 2021.

[4] David J. C. MacKay. *Sustainable Energy — Without the Hot Air*. UIT, 2008.

[5] Elise Dupont, Rembrandt Koppelaar, and Hervé Jeanmart. Global available wind energy with physical and energy return on investment constraints. *Applied Energy*, 209, 10 2017.

[6] Wikipedia Community. Load Factor (Electrical). 2021. [Online; accessed 31-March-2022].

[7] Prof. Gilles Louppe. Deep learning. 2021.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[9] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. 2014.

[10] Schmidhuber J. Hochreiter S. Long short-term memory. *Neural Comput.*, Nov 1997.

[11] Zhuyi Rao and Yunxiang Zhang. Transformer-based power system energy prediction model. pages 913–917, 06 2020.

[12] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[13] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[15] Wikipedia Community. Pearson Correlation Coefficient. [Online; accessed 18-March-2022].

[16] Wikipedia Community. Spearman's Rank Coefficient. [Online; accessed 18-March-2022].

[17] Ioannis K. Bazionis and Pavlos S. Georgilakis. Review of deterministic and probabilistic wind power forecasting: Models, methods, and future research. *Electricity*, 2(1):13–47, 2021.

[18] Zhongda Tian. A state-of-the-art review on wind power deterministic prediction. *Wind Engineering*, 45(5):1374–1392, 2021.

[19] Erasmo Cadenas and Wilfrido Rivera. Short term wind speed forecasting in la venta, oaxaca, méxico, using artificial neural networks. *Renewable Energy*, 34:274–278, 2009.

[20] João P. S. Catalão, Hugo M. I. Pousinho, and Victor M. F. Mendes. Short-term wind power forecasting in portugal by neural networks and wavelet transform. *Renewable Energy*, 36:1245–1251, 2011.

[21] Zexian Sun and Mingyu Zhao. Short-term wind power forecasting based on vmd decomposition, convlstm networks and error analysis. *IEEE Access*, 8:134422–134434, 2020.

[22] Zacharie De Grève, Jérémie Bottieau, David Vangulick, Aurélien Wautier, Pierre-David Dapoz, Adriano Arrigo, Jean-François Toubeau, and François Vallée. Machine learning techniques for improving self-consumption in renewable energy communities. *Energies*, 13(18), 2020.

[23] NOAA's Pacific Marine Environmental Laboratory. Software ferret.

[24] Scikit-learn. Random forest regressor. [Online; accessed 02-May-2022].

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

[26] Juan M Morales and al. Integrating renewables in electricity markets: operational problems. *Springer Science and Business Media*, pages 15–56, 2014.

[27] Xiaojun Shen, Xuejiao Fu, and Chongcheng Zhou. A combined algorithm for cleaning abnormal data of wind turbine power curve based on change point grouping algorithm and quartile algorithm. *IEEE Transactions on Sustainable Energy*, 10(1):46–54, 2019.

[28] Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. 2016.

[29] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

Appendix

A Pinson formalism

A.1 Renewable energy generation as a stochastic process

In [26, Chapter 2], they stated the generation of renewable energy as a stochastic process. Indeed, in chapter 4 we will see the stochasticity of the process with point clouds which will be further detailed. The stochastic nature of renewable energy is due to the shadowing effects on wind farms, dust, turbulence effects, etc. As defined in [26],

$$\{Y_{r,s,t}, r = r_1, \ldots, r_m, s = s_1, \ldots, s = s_n, t = 1, \ldots, T\} \tag{1}$$

is a multivariate stochastic process in space (different locations $s = s_1, \ldots, s = s_n$) and time ($t = 1, \ldots, T$). There exist potentially different sources $r = r_1, \ldots, r_m$. The corresponding realization of that stochastic process are denoted by

$$\{y_{r,s,t}, r = r_1, \ldots, r_m, s = s_1, \ldots, s = s_n, t = 1, \ldots, T\}. \tag{2}$$

In our case all the sources $r = r_1, \ldots, r_m$ will be wind farms. Thus, in the following we will omit the subscript r. In the same spirit, subscripts r and s may be omitted if we are only focusing on the time dimension. We will also assume that

$$Y_{r,s,t} \in [0, 1], \forall r, s, t \tag{3}$$

Indeed, all the power plants may be normalized by their nominal capacity installed.

A.2 Model based forecasting

Forecasting can be seen as making an estimate $\hat{\cdot}_{t+k|t}$ of a particular characteristic of the stochastic process Equation 1. Given a model f_θ with the parameters $\theta \in \mathcal{H}$ in a chosen hypothesis space $\mathcal{H}$. Notice that k denotes the forecasting horizon and the notation "$|t$" is coming from the probability theory "*given* t" to express that we have an information set Ω_t containing information up to time t.

A.3 Deterministic Forecast

In the case of deterministic forecast (also know as point forecast), we focus ourselves on the conditional expectation of the energy production. More precisely,

$$\hat{y}_{t+k|t} = \mathbb{E}[Y_{t+k}|f, \Omega_t, \mathcal{H}] \tag{4}$$

is a point forecast at time $t + k$.

A.4 Probabilistic Forecasts

In the case of probabilistic forecasting, we will focus on the prediction of the probabilistic density function of the variable Y. There exists several methods:

Quantile Forecasts

A quantile forecast $\hat{q}^{(\alpha)}_{t+k|t}$ with

$$P\left[Y_{t+k} \leq \hat{q}^{(\alpha)}_{t+k|t} \mid f, \Omega_t, \mathcal{H}\right] = \alpha \tag{5}$$

The forecaster gives the information that at lead time k there is a probability α that the the production is less than $\hat{q}$.

Prediction intervals

A prediction interval is defined as

$$\mathrm{P}\left[Y_{t+k} \in \hat{I}^{(\beta)}_{t+k|t} \mid f, \Omega_t, \mathcal{H}\right] = 1 - \beta \tag{6}$$

$$\hat{I}^{(\beta)}_{t+k|t} = \left[\hat{q}^{(\alpha)}_{t+k|t}, \hat{q}^{(\bar{\alpha})}_{t+k|t}\right] \tag{7}$$

where the production is included between the two bounds $\hat{q}^{(\alpha)}_{t+k|t}$ and $\hat{q}^{(\bar{\alpha})}_{t+k|t}$ at a given level probability $1 - \beta$.

Density Forecasts

Density forecast $\hat{f}_{t+k|t}$ output at time t for time t+k would be a complete description of the pdf given a model $f \in \mathcal{H}$ and an information set Ω_t.

B Supplementary Results

In this appendix, we display several qualitative results of other models on the ORES dataset. In Figure 3 the naive persistence method is displayed with its best performances in RMSE. In Figure 6, the best and worst samples for the naive method climatology are displayed.

In Figure 9 and Figure 12, we can see the best and worst results of the Extra Tress and Random Forest model on the validation set.

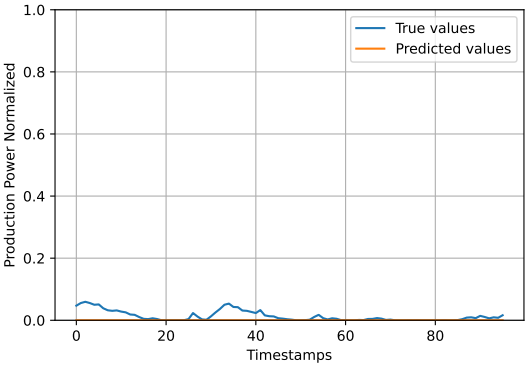

Figure 1: Best

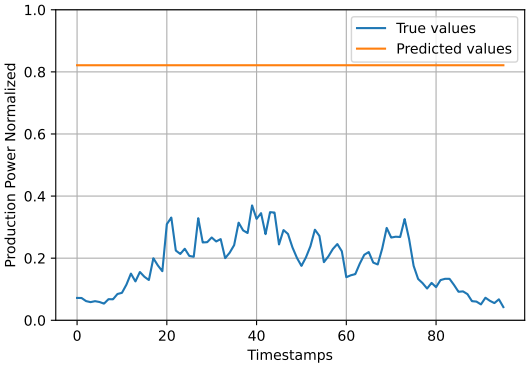

Figure 2: Worst

Figure 3: Persistance method on validation set samples

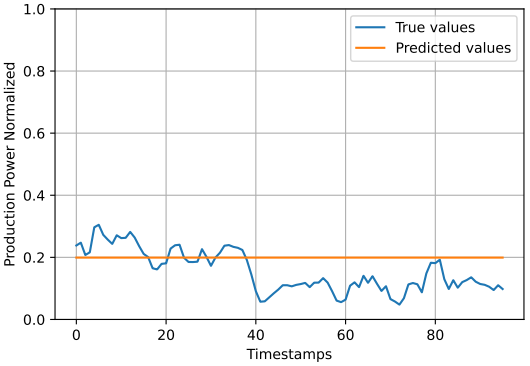

Figure 4: Best

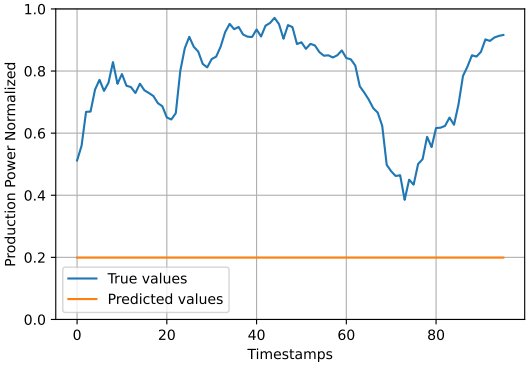

Figure 5: Worst

Figure 6: Climatology method on validation set samples

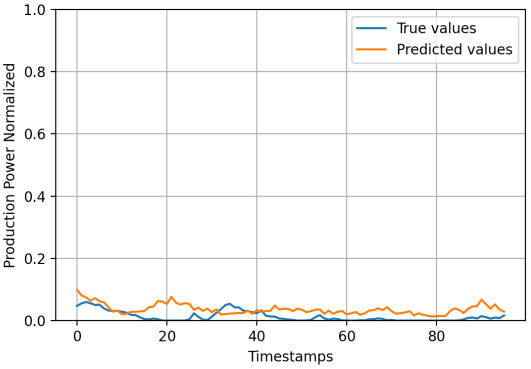

Figure 7: Best

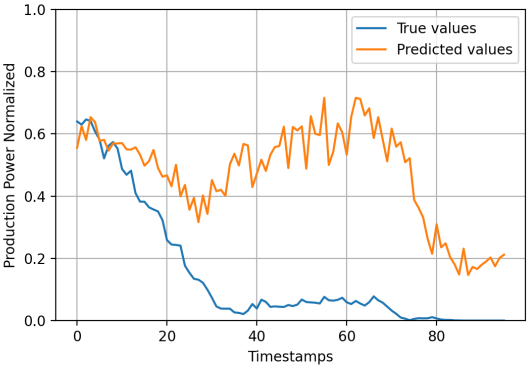

Figure 8: Worst

Figure 9: Extra Trees on validation set samples

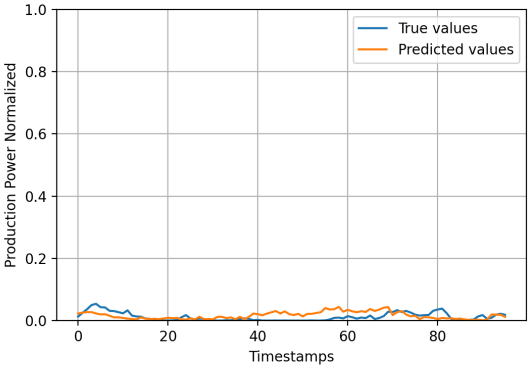

Figure 10: Best

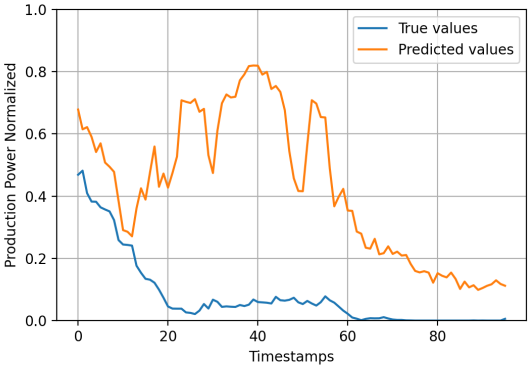

Figure 11: Worst

Figure 12: Random Forest method on validation set samples