UNIVERSITETET I AGDER

Intelligent estimation of the wake losses in wind farms

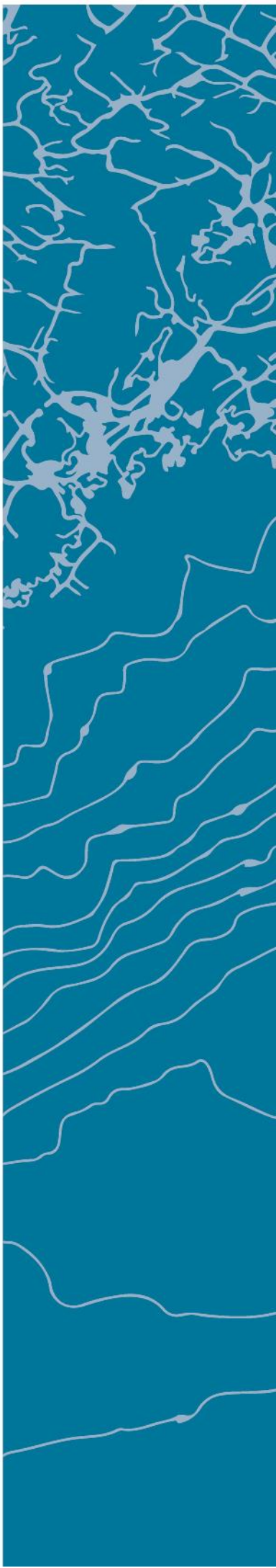
Artificial neural network estimation of the power of a wind farm considering the effect of wake losses

MARTÍ TARRAGÓ AYMERICH

SUPERVISORS

Prof. Dr. Sathyajith Mathew and Prof. Dr. Joao Leal

University of Agder, 2019
Faculty of Engineering and Science
Department of Engineering Sciences



Abstract

The transition from non-renewable to renewable energy production requires a detailed optimization and quantification of the generated power. The loss of power due to wake effect is a common problem for wind farms. The wake effect is the reduction of velocity and increase of turbulence in the wind flow downstream from a wind turbine. The wake effect is a complex multivariable phenomenon and its understanding is capital for appropriate estimations of the power of a wind field and its turbines.

This thesis builds an artificial neural network based on machine learning to model the performance of a single wind farm owned by WEICAN (Canada) taking into account the wake losses. Four different models have been considered. The first is not accounting for the wake losses; the second considers only the wake of the closest turbines; the third takes into account the wake in all the turbines; and the fourth provides all the data to the program in order to see what it can do on its own. The performance is evaluated using the mean absolute error, the root mean squared error and the normalized root mean square error.

The best results are obtained using the third model, hence showing that the wake loss is significant and must be considered in the model. It is proved that with the appropriate input variables, an artificial neural network can predict the power of a wind farm accounting for the wake losses. The best performance of the artificial neural network is obtained for wind speeds up to 14 m/s.

Acknowledgements

First of all, I would like to thank my supervisors, Dr. Sathyajith Mathew and Dr. Joao Leal, for entrusting me with this interesting and exciting project in an area which was new to me, and for their help during its elaboration.

I would also like to thank all the professors and workers at the University of Agder. I went to Norway as an exchange student, and found the University a really welcoming place, where I met extremely interesting people. A special thanks to all the personal involved in the Erasmus and other student organizations such as the Buddies and the Fadders. They made my arrival and stay in Norway very easy.

An Erasmus term is supposed to be an exciting period, but it turned out to be the best semester of my time in University. I am very grateful to have met many Erasmus students and that I get to call them my friends. To all those who share my passions and hobbies, including hiking, climbing, eating with friends, enjoying nature… And to those whom I shared the bonfires with, obviously, I send you all my love. This are times I will always remember.

The Grimstad Arendal Klatreklubb was like a second home to me. I want to thank all their members for making me feel a part of their family.

In Grimstad I had some of the best conversations in my whole life. To those who shared these conversations with me, you will be always in my heart.

To my friends in Mataró, because their support and love is always unconditional, thank you.

Finally, a wise man told me once that the only constant thing in life is change, but one thing has always remained constant for me: my family, the foundation of my life. You are always there for me and I will always be there for you too. You make my tough days easier the good ones better. We shared everything: burdens, medals, illnesses and prizes, and we will continue to do so forever. To my parents, because they were the beginning of everything, and their love makes us go always further. To my sister Mariona, who understand my feelings and shares my loads, because she always has the advice I need. And to my sister Roser, to keep me dreaming while we enjoy our amazing lives.

Table of content

Table of figures

List of tables

1. Introduction

1.1. World and energy

The access to energy is a primary need for current and near-future societies. The Total Primary Energy Supply (TPES), which represents the amount of energy produced in the world, is expected to grow around 30% until 2040. This increase will be caused by global population growth and the development of less developed countries in Asia – specially in India and the Middle East – and Africa (International Energy Agency (IEA), 2017). At the same time, the use of oil – which currently accounts for 31.9 % of the total production – will decrease due to reduced availability, rising prices and the implementation of more strict environmental policies. While the TPES is going to increase basically due to population growth and development of non-industrialized nations, the implementation of renewable energies has to be supported and developed by all governments, companies, universities and population. Hence, modern societies face an important challenge in the transition from non-renewable to renewable energies.

From the 1990's, the growth of renewable energies has increased an average of 2%, higher than the 1.7% growth of the TPES. However, the largest share of TPES produced from non-renewable energy sources is still 86.3% (International Energy Agency (IEA), 2018) Figure 1.

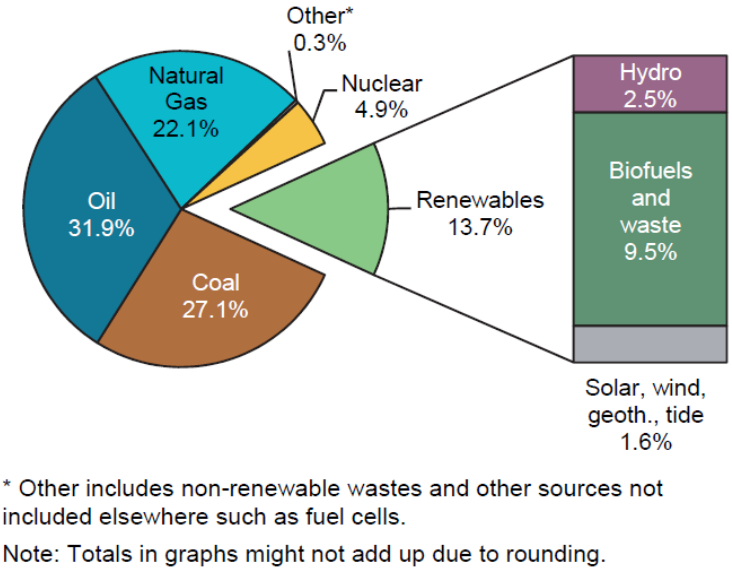


Figure 1. Fuel Shares in World Total Primary Energy Supply, 2016 (International Energy Agency (IEA), 2018)

According to the New Policies scenario, the generation of electricity using renewable sources is going to rise from 24% in 2016 to 40% in 2040 (International Energy Agency (IEA), 2017) as summarized in Figure 2Figure 1. Hydropower – currently the main renewable energy source – will continue to dominate but its share among the renewables will decrease from the current 67.6% to 39.48%. Wind and solar photovoltaic generation are going to increase substantially. Electricity generation from wind will increase from 981 TWh in 2016 to 4270 TWh in 2040. This means it is going to represent a share of 27.22% among the renewables, increasing around 18% annually (International Energy Agency (IEA), 2017). This information is represented in Figure 2.

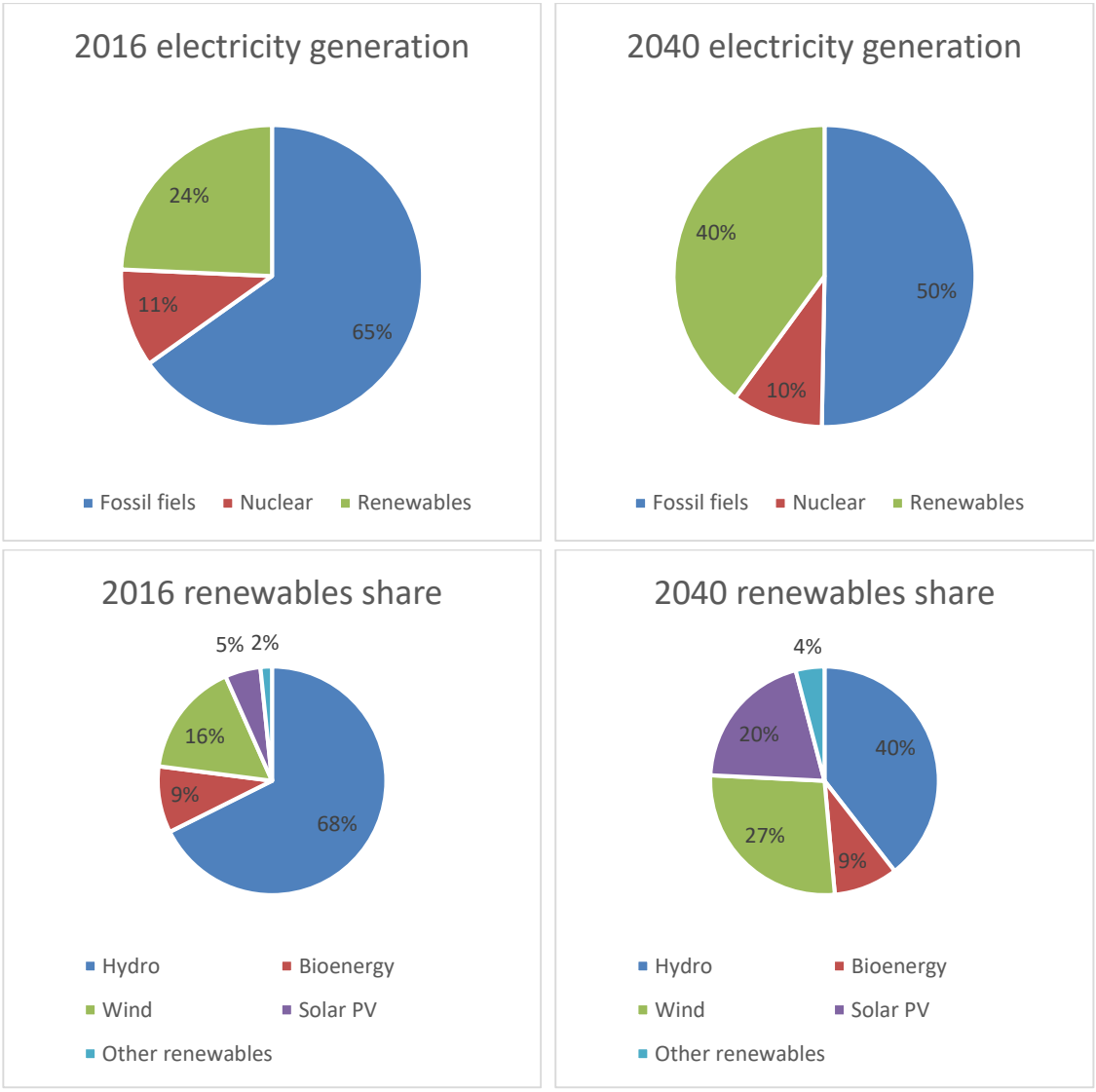


Figure 2. 2016 vs 2040 electricity situation. Data from (International Energy Agency (IEA), 2017)

Air pollution has been a concern in the European Union for decades due to its effects on human health. For instance, the concentration of Particulate Matter smaller than 2.5 µm ($PM_{2.5}$) led to 422.000 premature deaths in 41 European Countries in 2015 (European Environment Agency, 2018a). According to the World Health Organization (WHO), 96% of European citizens living in urban areas are exposed to health damaging air pollution levels (European Environment Agency, 2018b). The contributions to air pollution include energy generation, climate control, industry and agriculture. However, the most significant for human health is transport, due to the local contamination caused by cars in large cities (Niemenmaa et al., 2018). The possible solutions include switching from private cars to public transportation and replacing combustion with electric vehicles. These changes will only be effective if the electricity used to power the public transport and the electric cars is generated through renewable energies.

In summary, to preserve a healthy environment and the Earth as an inhabitable planet in the long term, human societies have to invest in environmentally respectful energy generation

procedures. The engineering field has to be upfront in this movement, providing the industry with new knowledge to be implemented in the future in the market.

1.2. Wind energy perspectives

Electricity generated through wind energy has been rising during the last decades, and it is expected to continue doing it at least until 2040. Figure 3 presents the installed wind energy capacity from 2001 until 2017, which has been growing during the last 10 years (GWEC, 2018) in spite of a certain stagnation between 2009 and 2013 attributable to the global economic crisis. The improvements in wind energy generation during the last decade made it evolve from a subsidized to a fully competitive energy source, able to fight in the market with the rest energy sources in price (GWEC, 2018). Wind energy is now able to evolve not only due to environmental motives, but also due to economical ones. Optimizing the production and power of wind-generated energy requires a deep understanding of their oscillations. Hence, it is essential to understand and create models to control and predict the energy that we are going to receive from wind.

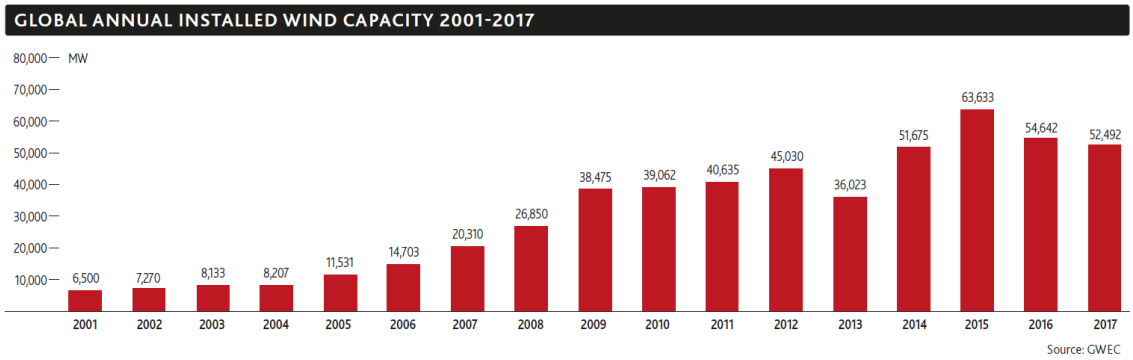


Figure 3. Global anual installed wind capacity 2001-2017. From (GWEC, 2018)

1.3. Introduction to the wake effect

The wake effect is the loss of velocity and appearance of turbulence in the wind flow downstream of the wind turbine due to the absorption of energy in the turbine. Additionally, the wake effect can generate unbalanced mechanical loads to the wind turbines, leading to fatigue in some components (Thomsen et al., 2007), (Frandsen, 2007). This disturbance makes the wind velocity lower and has to be taken into account when designing wind farms. However, it is difficult to know the power of a wind farm with important wake, due to the complexity of the calculations needed to know the wind velocity and profile downstream, which affects the wind that the turbines receive downstream.

The first approach to the wake effect included using physical models. A full chapter of this thesis is devoted to explaining the different models and in which conditions they may be applied. However, although the physical models are found to give good results in specific cases, they provide neither the desired accuracy nor the flexibility required to face real-life problems. The use of physical models evolved into Computational Fluid Dynamics (CFD) programs. Some of

these programs are driven by complex equations that numerically solve the Navier-Stokes equations for the full wind flow in the wind farm. Theoretically, they provide better accuracy than the physical models due to their breadth and larger calculation range. In the other hand, they are found to use a lot of resources, requiring really expensive computers, energy and time to calculate. Another approach to the wake effect is through machine learning (Brusca et al., 2017; Japar et al., 2014; Kusiak et al., 2009; Răzuşi and Eremia, 2011). Machine learning is a computer science that uses statistical techniques in order to give a computer the ability to learn and improve its performance on a specific task. These methods can deliver quite good results using significantly less resources than CFD.

The objective of this study is to define a low resource model that takes into account the wake effect to estimate the power generated in a wind farm.

2. Theoretical Background

This chapter summarizes the main theoretical topics treated on this thesis, focusing on the wake models.

2.1. Wake effect

Wind turbine wake is a volume of fluid downstream a turbine whose properties have been modified by the interaction between the wind and the turbine blades. The blades are designed as an airfoil, and when the wind moves through them they undergo lift and drag forces. These forces are converted to a rotational movement, which is driven through a different set of elements to finish in a motor that transforms this movement into electricity. In consequence, the wind leaving the turbine has lower energy than the wind upstream; its velocity will be reduced and it will have higher turbulence. As the distance from the disturbance increases, the flow spreads and it begins to return to free stream conditions.

The optimization of wind farming requires maximizing the generated energy using the minimum number of turbines. The location of wind farms is limited by geographical restrictions and policy issues, resulting in turbines being generally close to each other. In conditions of wind directions where the wake produced by one turbine affects another one, because it is located downstream of the first one, it is said that the second wind turbine is in the shadow of the first one.

Wake losses are a sophisticated, difficult to predict phenomenon because they are non-linear events of a complex nature. For instance, the wind hits the blades of a turbine making them rotate. The speed of the wind depends on uncountable variables, such as the weather or the aerodynamics of the place. The energy that the turbine is able to extracts from the wind also depends on many factors, such as wind speed, wind shear, turbulence, aerodynamic properties of the airfoils or pitch angle of the blades. Then, the rotation of the wind turbine's blades is transited to a gearbox, which drives this energy to a generator. This generator transforms the mechanical energy to electrical energy in another non-linear event. In summary, the wake effect is the result of a wide series of events, and each of these events is difficult to calculate, making the wake effect a really complex phenomenon.

Figure 4 shows a graphic representation of the wake effect, where some of the wind turbines are in the shadow zone.

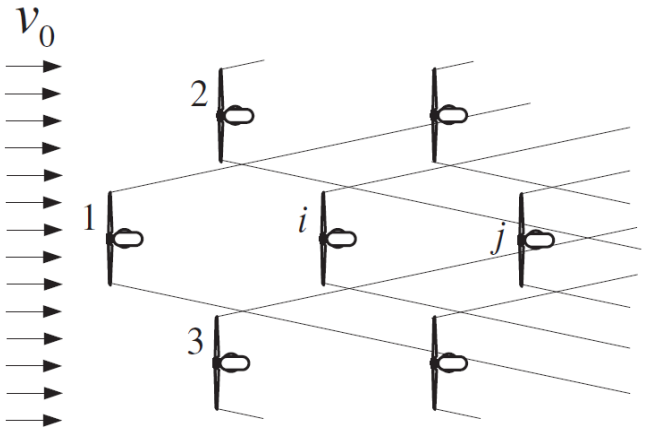


Figure 4. Wind turbines in the wake shadow zone. From (González-Longatt et al., 2012)

The wake effect can severely affect the total power output in certain turbine distributions when the wind is in a determinate direction. In a typical wind farm, the wake effect can cause differences around 10-20% between the power of the undisturbed turbines and the disturbed ones (Sørensen et al., 2006), (Barthelmie et al., 2009). In conditions where the wind is parallel to a column of wind turbines the power loss may even around 70% (Archer et al., 2018).

Another problem related to wind wake is the appearance of turbulences. The wake effect increases the turbulence in comparison with the non-disrupted layer. This effect creates mechanical loads affecting the turbine structure and the blades. This loads can diminish the length of the life cycle of the turbine (Thomsen et al., 2007), (Frandsen, 2007). Figure 5 shows a graphic representation of the turbulence in front of a wind turbine, which generates deflections and material stress.

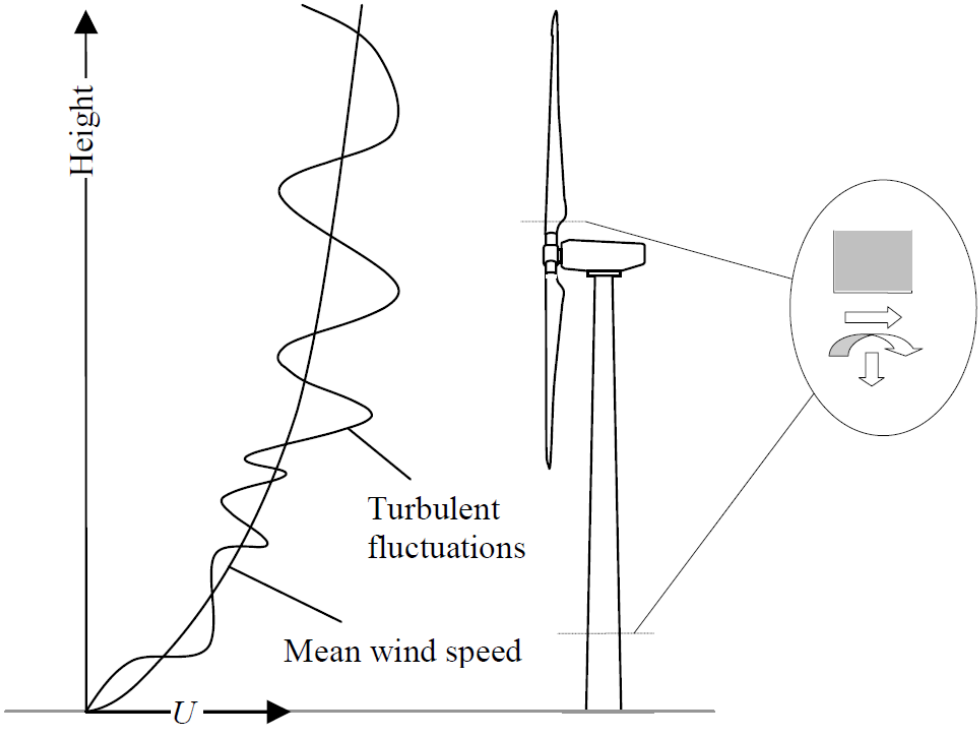


Figure 5. Wind loading of a wind turbine structure. From (Frandsen, 2007)

2.2. Physical models

The physical wake loss models are divided into two different groups on the approach to the phenomenon. On one hand, the analytical or kinematic models are characterized by looking for an analytical solution to the wind speed deficit in the turbines downstream. They are based in the conservation of mass and empirical equations of wake decay. On the other hand, the CFD models are based in the applications of Navier-Stokes equations to define the entire flow field in the wind farm (Archer et al., 2018).

The first physical model used to describe wind turbines wake, known as kinematic model, was proposed by Lissaman (S. Lissaman, 1979). It uses the momentum equation to find the velocity deficit downstream. The model itself does not take into account the change in turbulence. Hence, it is not useful to calculate loads or the turbulence intensity if it is not combined with another model.

2.2.1. Jensen's model

The most used wake model is likely the Jensen's model (Jensen, 1983). It was later improved by Katic (I.KATIC et al., 1987), and thereafter known as Park model. It is quite simple, and although it is one of the oldest models it provides good results. The model neglects the disturbed field near the turbine, making possible to treat the fluid as turbulent wake. Then, it proposes to study the wake as linearly expanding, where the velocity deficit only depends on the downwind distance "d" shown in Figure 6.

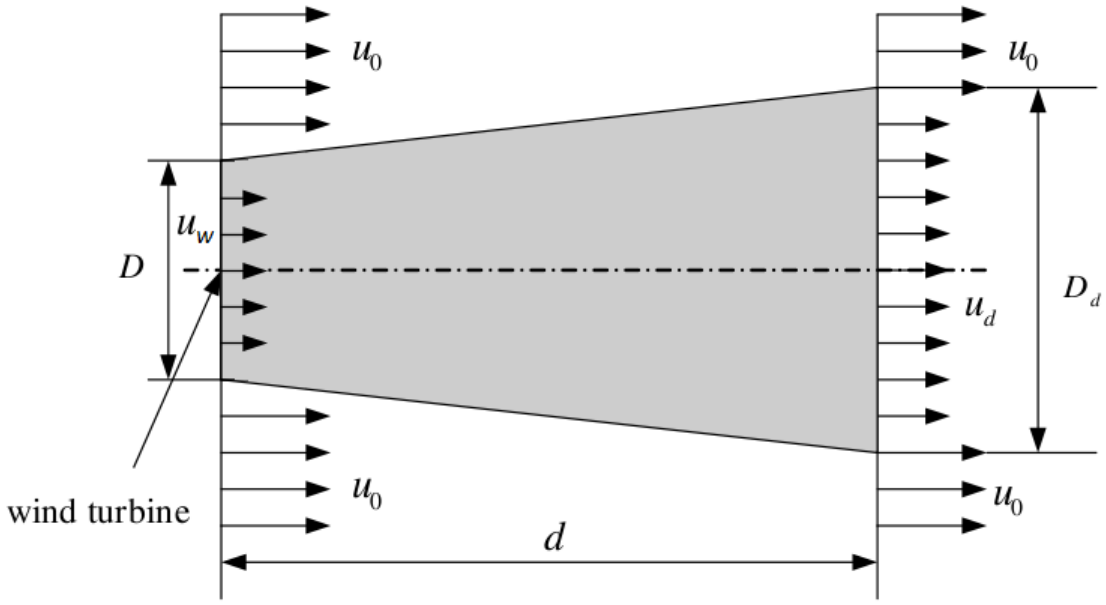


Figure 6. Scheme of Jensen's wake model. From: (Bonanni et al., 2012)

The equations governing the wind wake in Jensen's single wake model develop as follows:

$$r = \frac{D}{2};\ r_w = \frac{D_d}{2} \tag{1}$$

Conservation of momentum:

$$M_{with\ wake} = M_{ideal} - M_{losses} \rightarrow \pi \cdot r_w^2 \cdot u_d = \pi \cdot r_w^2 \cdot u_0 - \pi \cdot r^2 \cdot (u_0 - u_w) \tag{2}$$

Simplifying:

$$\pi \cdot r^2 \cdot u_w + \pi \cdot (r_w^2 - r^2) \cdot u_0 = \pi \cdot r_w^2 \cdot u_d \rightarrow$$
$$u_d = \frac{\pi \cdot r^2 \cdot u_w}{\pi \cdot r_w^2} + \frac{\pi \cdot r_w^2 \cdot u_0}{\pi \cdot r_w^2} - \frac{\pi \cdot r^2 \cdot u_0}{\pi \cdot r_w^2} \rightarrow u_d = u_0 + \frac{r^2}{r_w^2}(u_w - u_0) \tag{3}$$

Applying Betz theory (Ammara et al., 2002):

$$u_w = (1 - 2a) \cdot u_0 \tag{4}$$

Replacing the equation (4) in the equation (3):

$$u_d = u_0 + \frac{r^2}{r_w^2} \cdot \big((1 - 2a) \cdot u_0 - u_0\big) = u_0 + u_0 \cdot \frac{r^2}{r_w^2} \cdot (-2a) \tag{5}$$

Where a is the axial flow induction coefficient, also expressed as follows:

$$a = \frac{1 - \sqrt{1 - C_t}}{2} \tag{6}$$

Replacing (6) in (5):

$$u_d = u_0 + u_0 \cdot \frac{r^2}{r_w^2} \cdot \big(\sqrt{1 - C_t} - 1\big) \tag{7}$$

Where C_t is the thrust coefficient of the turbine.

The radius of the wake cone, r_w can be defined by the equation (8):

$$r_w = d \cdot (1 + 2\alpha \cdot d)/2 \tag{8}$$

Where α is the decay constant, without dimension. It can be defined by the equation (9):

$$\alpha = 0{,}5/\ln\left(\frac{z}{z_0}\right) \tag{9}$$

The constants z and z_0 correspond the wind turbine's hub height and the surface roughness respectively.

The fully developed equation of the velocity of the wind can be represented as follows:

$$u_d = u_0 \cdot \left[1 - \frac{1 - \sqrt{1 - C_t}}{(1 + 2\alpha \cdot s)^2}\right] \cdot \pi \cdot r_w^2 \tag{10}$$

And:

$$s = d/2r_w \tag{11}$$

It is important to note that the equation (10) is only meant to represent the wake effect in far wake regions, around 3-5 D on-shore or 6-8 D off-shore.

The update of the Jensen model, the Park model, was implemented in WAsP (Mortensen et al., 2001) software.

2.2.2. Larsen's model

Another model used to represent the wake effect is the Larsen's model (Larsen, 1988). The model is based on the turbulent boundary equations, and provides closed form solutions for the width and the mean wind profile of the wake. It assumes a self-similar velocity based in Prandl's boundary layer equations profile and an incompressible, stationary and axisymmetric flow.

In the first order approximation, the simplified form of the expression to be solved is as follows:

$$U_\infty \cdot \frac{\partial u_x}{\partial x} = \frac{1}{r} \cdot \frac{\partial}{\partial r}\left[l^2 \cdot r \cdot \left(\frac{\partial u_x}{\partial r}\right)^2 \right] \tag{12}$$

Where r is the radial direction, u_x the inflow wake perturbation and x the symmetry axis. To solve this equation two boundary conditions are needed. The first one is $u_x = 0$ on the boundary of the wake. The second one, $U_\infty \gg u_x$, which comes from a momentum balance, according to the hypothesis that inflow velocity is much higher than the axial wake perturbations.

Then, Larsen developed the first order equation for the boundary layer as shown below:

$$r_w(x,r) = \left(\frac{35}{2\pi}\right)^{\frac{1}{5}} \cdot (3 \cdot c_1^2)^{\frac{1}{5}} \cdot \left(C_t \cdot A(x)\right)^{\frac{1}{3}} \tag{13}$$

$$u_x(x,r) = -\frac{U_\infty}{9}(C_t A x^{-2})^{\frac{1}{3}} \left[r^{\frac{3}{2}} (3 \cdot c_1^2 \cdot C_t \cdot A \cdot x)^{-\frac{1}{2}} - \left(\frac{35}{2\pi}\right)^{\frac{3}{10}} \cdot (3 \cdot c_1^2)^{-\frac{1}{5}} \right]^2 \tag{14}$$

$$u_r(x,r) = -\frac{U_\infty}{3}(C_t A)^{\frac{1}{3}} \cdot x^{-\frac{5}{3}} \cdot r \left[r^{\frac{3}{2}} (3c_1^2 \cdot C_t \cdot A \cdot x)^{-\frac{1}{2}} - \left(\frac{35}{2\pi}\right)^{\frac{3}{10}} \cdot (3 \cdot c_1^2)^{-\frac{1}{5}} \right]^2 \tag{15}$$

Where r_w is the radius of the wake, A is the rotor swept area, C_t is the thrust coefficient of the wind turbine and c_1 is the non-dimensional mixing length, defined by equation (16).

$$c_1 = l \cdot \left(C_t \cdot A(x)\right)^{\frac{1}{3}} \tag{16}$$

The solution for the second order equation was found to be negligible in real engineering applications (Larsen, 1988).

The Larsen's model is integrated in the software WindPRO (International A/S, 2010).

2.2.3. RANS models

Other models intended to predict the wake or its effects. Depending on the conditions, some provide better results than others. Some of them, are based on Reynolds-averaged Navier Stokes (RANS) equations. FUGA (Ott et al., 2011) is a very robust CFD program that uses the cartesian form of the RANS equations. Most of these programs run computational fluid dynamics code. This kind of programs need a lot of resources to compute, and anyway normally they provide good results in a specific range of study. Some reviews of the models, programs and the theoretical way to understand them can be found in various articles, like the one from the Technical University of Denmark (Göçmen et al., 2016).

3. Artificial intelligence

The story of artificial intelligence officially started in 1956, during a conference in Dartmouth College in Hanover, NH (USA). In this first conference, John McCarthy defined artificial intelligence as “the science and engineering of making intelligent machines”. However, philosophers and scientists have been playing with this idea for many centuries. For example, Descartes stated that animals are nothing more than complex machines. To this date, there’s no absolute consensus on the definition of artificial intelligence. Some, like Kaplan and Haenlein define AI as “a system’s ability to correctly interpret external data, to learn from such data, and to use those learnings to achieve specific goals and tasks through flexible adaptation” (Kaplan and Haenlein, 2018).

Artificial intelligence can also be defined as the ability of a machine to think or behave in a specific way. For example, we can classify artificial intelligence into four categories, Thinking Humanly, Thinking Rationally, Acting Humanly and Acting Rationally (Russell and Norvig, 2009). These categories can disagree with each other, so it’s clear that there’s not a perfect way to define artificial intelligence.

Another conflict is raised by the AI effect: as machines become able to do a task, this task is normally considered a non-intelligent task, because an intelligent human is not needed to perform it. Hence, as far as we develop machines to be intelligent, the goal of an intelligent machine is always going to be a little bit further. In fact, Tesler’s Theorem even defines intelligence as “whatever machines haven’t done yet” (Boosman, 2017).

3.1. Learning

Learning programs are those able to improve their performance doing a task due to the incorporation of data and feedback. The program is able to learn due to its capability of constructing itself. This raises an interesting question: why do we need a program that builds itself? Why do we need it to be able to learn, don’t we have the information to make it “perfect” in the first place? According to Russel and Norvig, there are three main reasons (Russell and Norvig, 2009). First of all, the designers of the program can not anticipate all the possible situations and relations that the program could find by himself. In the second place, if a program is designed and closed, meaning that no further change on the program is going to be made, the program is not going to be able to adapt to changes. In a self-built program, when the circumstances change the program is able to adapt its variables in order to provide better results, if it is designed to do so. In the third place, which corresponds to the case of this thesis, sometimes the problems are much too complex to find a solution. In these cases, to design a fix program to do this task can be too complicated. But when applying learning techniques, the problem changes to a “black box” point of view. There’s no need to deeply understand the problem, because the program will build itself just with input and output data.

3.1.1. Feedback

When a designer is building an artificial intelligence program, there are three ways to make the program learn.

The first one is unsupervised learning. In this case, the learning agent learns using only the input information, without any feedback. This means that the program will extract its own patterns or "conclusions" from the given information. In a photo recognizing program for example, it can make groups of similar images and conclude that the image it sees represents the same thing.

The second one is reinforcement learning. This kind of programs learns from feedback in forms of reward or punishment. At the end of a task, the learning agent receives a good or bad qualification, and it has to decide what is responsible for this mark, trying to improve it if it was a bad qualification.

The third kind of feedback is the supervised learning. In this case, every time the learning agent receives input data it gets also the correct answer for this particular example. For example, in the same image recognition example, when the program gets learning data it receives also the answer or label - Figure 7 - which could be the object it represents (book, chair, cat, dog, …).

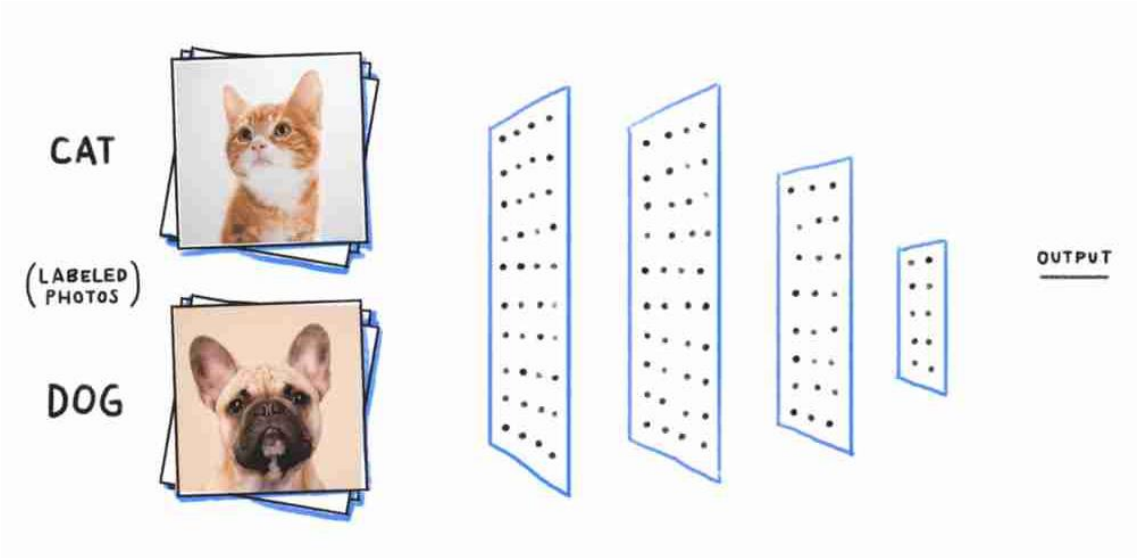


Figure 7. Supervised learning, labeled photos of cats and dogs. From (mc.ai, 2018)

In the case of this master's thesis, the program learns through a supervised learning process. It gets a set of input data, like the angle and velocity of the wind, and it has to predict the output power of wind turbines. But when it is learning, the program also receives the power of the wind turbine, so the feedback is the answer it has to get.

3.2. Neural networks basics

Neural networks are artificial intelligence programs intended to work like human neurons. The idea of neural networks was proposed by the neurophysiologist Warren McCulloch and the

mathematician Walter Pitts in an article, explaining the functioning of neurons work, which modeled a simple neural network with electric circuits, Figure 8 (McCulloch and Pitts, 1943).

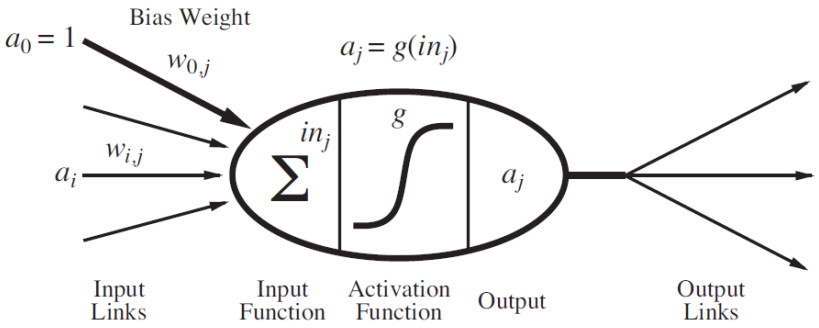


Figure 8. Simple mathematical model for a neuron, developed by McCulloch and Pitts (1943). From: (Russell and Norvig, 2009)

In the 1950s Nathaniel Rochester, a computer scientist at IBM, made the first unsuccessful attempts at the simulation of neural networks. The first successful neural network was developed by Bernard Widrow and Marcian Hoff of Stanford, in 1959. The two models they developed where called “Adaline” and “Madaline”. “Adaline” was developed to recognize binary patterns, and “Madaline” was designed to eliminate echoes on phone lines. “Madaline” was in fact the first neural network applied to a real world problem (Winter and Widrow, 1988).

3.2.1. Functioning of a neuron

The building and operation of artificial neural networks is based on the functioning of neurons and how they communicate with each other creating a network in human brains, providing with the ability to think. A neuron is a cell that has the purpose of transmitting information. Anatomically, its shape is similar to a tree, Figure 9. Neurons use electrical and chemical signals to transmit information through the brain and nervous system. Each neuron has three parts, the cell body and two extensions, the axon and the dendrite. The cell body contains the nucleus, which is responsible for the control of the activity of the cell and contains the genetic information. The axon is like a long tail, responsible for the transmission of messages to other neurons. The dendrites are like branches of a tree, coming from the cell body, and are responsible of receiving the messages from neurons close by (National Institute of Neurological Disorders and Stroke, 2018). The human brain is a really complex and huge grid including about 86,000 million neurons (Herculano-Houzel, 2009), each one of which can be connected through its receiving and transmitting ends to 10,000 other neurons.

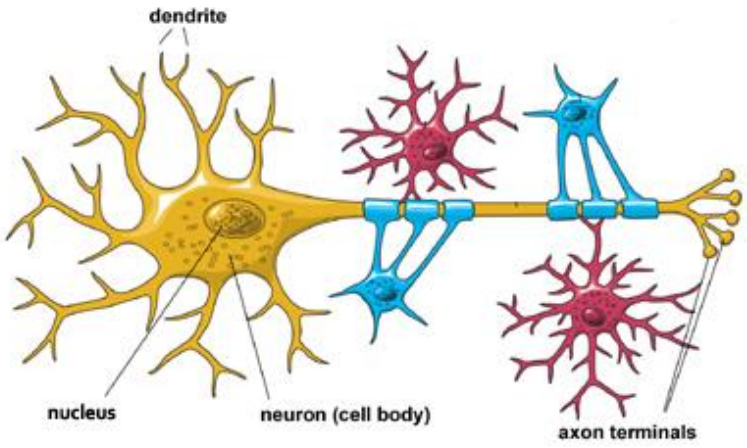


Figure 9. Anatomy of a neuron. By (National Institute of Neurological Disorders and Stroke, 2018)

3.2.2. Artificial neurons

Artificial neural networks try to copy the structure of our brain in a simplified way. In a neural network the neuron is represented by a node, a "mathematical neuron".

A simple kind of artificial neuron is the "perceptron". The perceptrons where developed by Frank Rosenblatt (Rosenblatt, 1957), Figure 10.

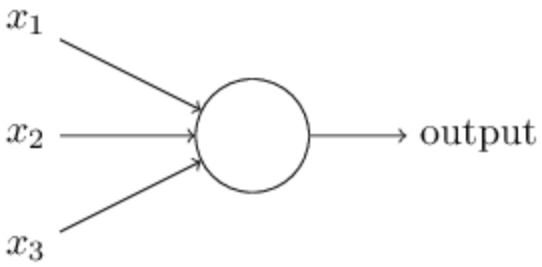


Figure 10. Perceptron. From (Nielsen, 2015)

A perceptron has multiple binary inputs: x_1, x_2, x_3 in the case of the example perceptron seen in Figure 9. To generate an output, Rosenblatt introduced the weights, real numbers which intention is to express the importance of each input to the output, corresponding in the example case to w_1, w_2, w_3. The output of the neuron is binary, depending on whether the sum of the weights multiplied with each input variable is higher or lower than a threshold. The equation is mathematically written as follows:

$$output = \begin{cases} 0 \; if \; \sum_j w_j \cdot x_j \leq threshold \\ 1 \; if \; \sum_j w_j \cdot x_j \geq threshold \end{cases} \quad (17)$$

Simplifying this equation to have all the variables in the same part of the equation, the result is as follows:

$$output = \begin{cases} 0 \; if \; w \cdot x + b \leq 0 \\ 1 \; if \; w \cdot x + b \geq 0 \end{cases} \tag{18}$$

Where:

$$\sum_j w_j \cdot x_j \equiv w \cdot x \tag{19}$$

The variable b is called bias, and it represents the difficulty of a perceptron to be true.

3.2.3. Artificial neural networks structure

Figure 11 shows an example of a neural network considering 6 variables. Inside the network, the input layer represents the data given to the program. The hidden layer(s) contain the neurons. The output layer gives the result of the network after a specific input.

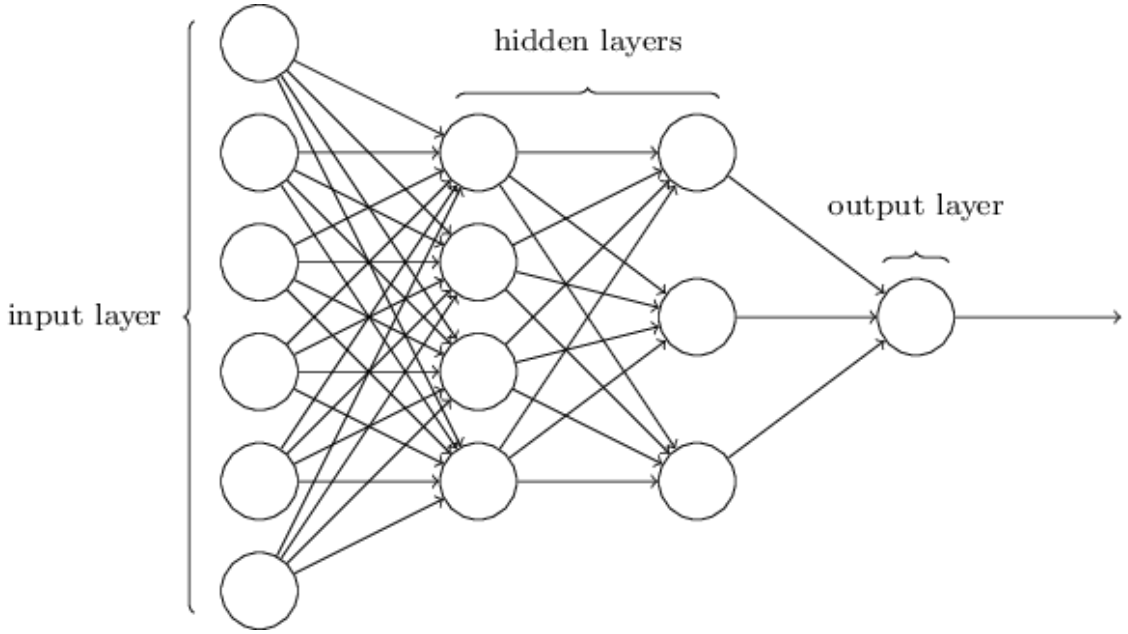


Figure 11. Example of the representation of a neural network. From (Nielsen, 2015)

Each represented neuron in the first hidden layer has a weight for every input. In the example network, the 4 first-level hidden neurons have 6 weights corresponding to each input variable, a total of 24 weights. At the same time, each node of this layer has a bias, giving a total of 4 biases. The second hidden layer has 12 weights and 3 biases, and the last one has 3 weights and 1 bias. This results in a total of 47 variables that can be combined in complex ways by the network.

The model has to learn from the given data, using one of the previously explained learning methods – unsupervised, reinforcement, supervised. In the case of this thesis the feedback is provided to the designed neural network using supervised learning. Both the input variables and the expected output corresponding to the actual measured values for speed and power are

provided and compared to the output calculated by the neural network model. Then a correction is made to the model to try to minimize the cost function, which represents the difference between them.

Perceptrons are simplified neurons that can have a true or false output. The problem with perceptrons is that when a change is done in the model, a perceptron can change its behavior from 0 to 1, creating big differences in the output. The desired behavior is however that when a change is done in a weight it causes only a small change in the output. Then, with this small change the program can make the network perform better.

3.2.4. Sigmoid neurons

The solution to the problem found in the previous section is to introduce a different kind of neuron to introduce non-linearity, for example an activation function called sigmoid. In this kind of neurons, a small change in the weights or the bias causes only a small change in the output. The inputs can be any value between 0 and 1, and the output is now defined as follows:

$$\sigma(w \cdot x + b) \tag{20}$$

Where the sigmoid function, σ, is defined by:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \tag{21}$$

Or in the example case:

$$output = \frac{1}{1 + exp^{(-(w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3) - b)}} \tag{22}$$

In the case of the perceptrons, the shape of the activation function is a step and it can have either 0 or 1 as an output. The sigmoid function has the shape represented in Figure 12.

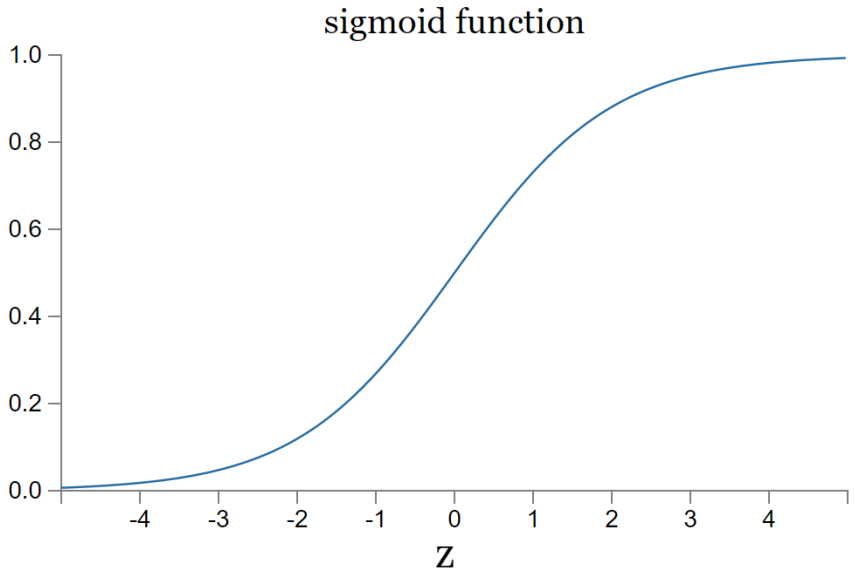


Figure 12. Shape of sigmoid function. From (Nielsen, 2015)

3.2.5. Loss function

Loss minimization or empirical risk minimization is a tool that determines whether the model performs adequately or not and whether a correction is required. A loss function is characterized for penalizing the model predictions when they differ from the assigned label. A simple commonly used loss function is the squared loss, named mean squared error (MSE) when applied to a set of data. It represents the averaged squared loss over the whole dataset, and can be calculated using the following equation:

$$MSE = \frac{1}{N}\sum_{(x,y)\in D}\big(y - prediction(x)\big)^2 \tag{23}$$

Where (x, y) is an example in which x represents the input variables, y the label, $prediction(x)$ is a function of weights and bias with the input variables, D is a data set containing many labeled examples and N is the number of examples in D. Although other loss functions can be defined, and MSE is not the best loss function to use, all the loss functions have the same purpose and a similar way to work.

The first step is to reduce the loss using an iterative approach. At the start of the model all weights and bias are set up randomly. Then, the program gets data, in form of features and their corresponding labels. The model makes its prediction, and this prediction is compared to the label using the loss function. Afterwards, the program uses other tools to compute new parameters (weights and bias) and updates the model, returning to a fresh start with new features and labels as represented in Figure 13.

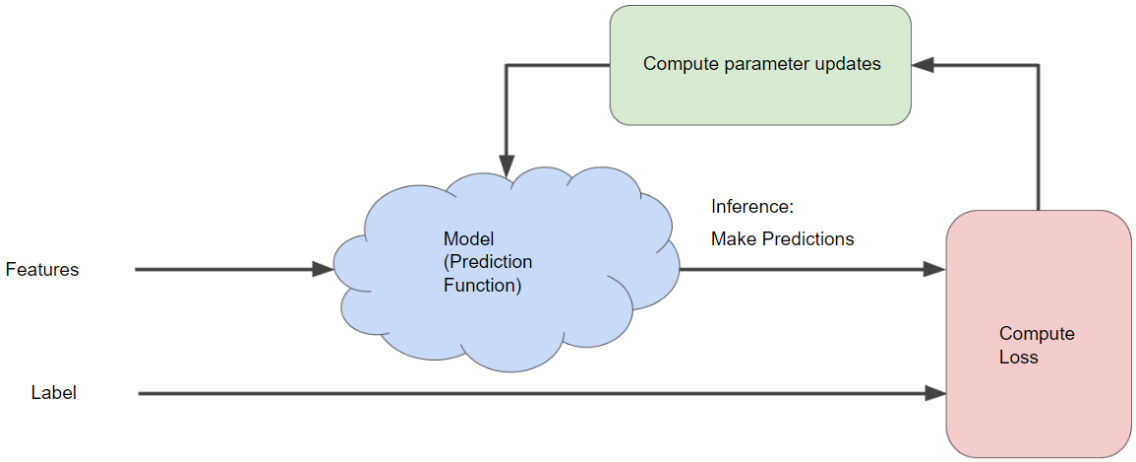


Figure 13. Iterative approach procedure. From (Developers, 2018)

3.2.6. Computing parameter updates

The learning process is completed by defining how to compute new weights and bias. In this study a variant of the technique called gradient descent will be used. At first the cost function that will be minimized is defined. A two-dimension graphic representation of a cost function might for instance have the shape shown in Figure 14.

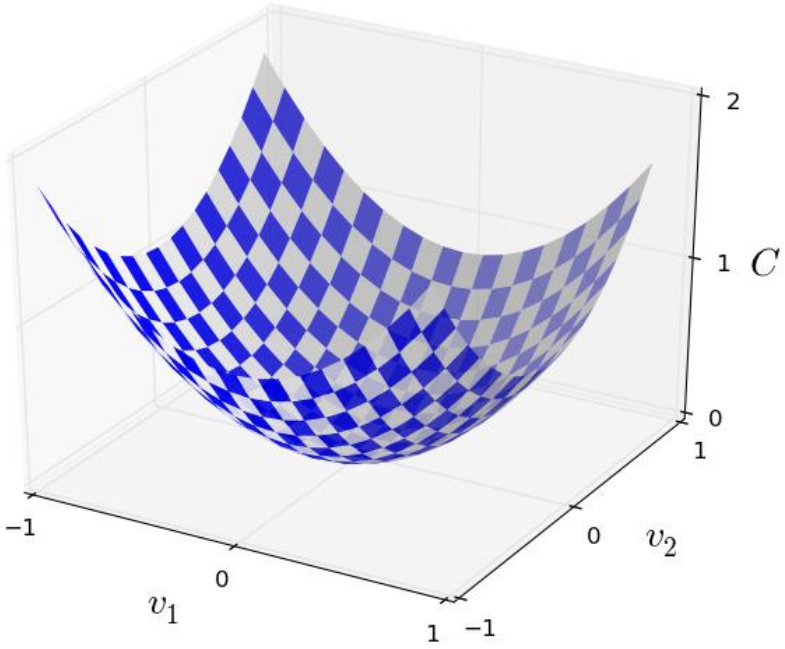


Figure 14. 2D Cost function representation. From (Nielsen, 2015)

The goal of the technique has to be to find the minimum as fast as possible, but not all the cost functions are as simple as the shown in Figure 14. Mathematically, the easier way to find the minimum is through derivatives. In a complex problem considering more than 2 dimensions the derivation is performed using the gradient descent using the equations presented below to find a function to indicate the direction and magnitude in which the variables have to change in order to reduce the loss. In the case of a 2-variable problem:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 \tag{24}$$

Where v_n are the function variables and the increment of the cost function ΔC has to be negative. To do so, the gradient vector is defined:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T \tag{25}$$

Expressing also the velocities as a vector, the equation (24) becomes:

$$\Delta C \approx \nabla C \cdot \Delta v \tag{26}$$

The remaining parameter to add to this function is the learning rate η. The learning rate is a parameter meant to control the "velocity" (more accurately the size of the steps) that the program introduces in its parameters to reduce the cost.

$$\Delta C \approx -\eta \cdot \nabla C \cdot \Delta v \tag{27}$$

In fact, to compute the change in the function variables, the equation should be:

$$\Delta v = -\eta \cdot \Delta C \tag{28}$$

Using this method, the program should keep improving its performance, eventually finding a minimum to converge.

3.2.7. Summary

An artificial neural network is a function that contains a set of neurons with assigned weights and a bias. The output signal sent by the neuron that depends on these weights, the bias, the input variables and the kind of activation function it uses – the sigmoid function for example. The network receives some data as input variables and an output (label). Then it generates an output, and it is compared with the actual value using a loss function. Afterwards, the gradient descent is calculated to know in which way the parameters have to change. Once they are updsted, the iterative process starts again with another set of data, until the problem converges, or all the programed iterations are done.

4. Wake losses estimation in a neural network

The purpose of this thesis is to estimate the effect of the wake in a wind farm, or said in another way, to estimate the power of the wind farm having into account the wake. In this chapter the designed neural network will be presented, and also the experiments performed with this network and the available data.

4.1. Presentation of the data

The data used in this thesis belongs to a wind farm located in the Prince Edward Island, in Canada. The wind farm is a power generation plant that has 5 DeWind D9.2 turbines, with an individual power of 2 MW each and a total of 10 MW. The owner of the wind farm is a not-for-profit entity formed in 1981 called Wind Energy Institute of Canada, focused in research, testing, innovation and collaboration in the field of wind energy. Figure 15 shows a view of the wind farm.

Figure 15. WEICAN 10 MW wind farm. From (Canada, 2018)

The received information consisted in ten minutes averaged performance data for turbines one to five. The data includes the wind direction at the farm, its velocity and power for each of the turbines. The satellite view in Figure 16 signals the location of each turbine. Due to problems during the harvesting of the information, a considerable amount of data from the fifth wind turbine is missing. However, this fifth turbine is located further away from the other turbines and it is not at the coast line. In the study cases the information of this turbine is not used as it is not relevant, but conclusions of the other ones as an independent wind farm can be extracted.

In Figure 16 the turbines are localized with an identification. This identification is used to identify the wind turbines in the next sections.

Figure 16. WEICAN 10 MW Wind Farm Satellite view. From: (Google Earth Pro, 2018). (31/10/2018), Prince Edward Island, Canada. 47°02'07"N 64°00'52"S Eye Altitude 3.15 km [November 23, 2018]

4.1.1. Data cleaning

The data has to be cleaned before the artificial programming tests, meaning all the missing data points which could make the program perform inappropriately have to be removed. This step is undertaken using a code written in Microsoft Excel with Visual Basic. It can be found in Annex D

Data cleaning program. Once processed, the remaining data is considered available data. Figure 17 shows the amount of available data over the total provided data. In the individual turbines cleaning, data is available when the turbine data point is not missing. In case of the T1-T4 and T1-T5 data, it is only considered valid when all the turbines have a power recorded in the same ten minutes interval.

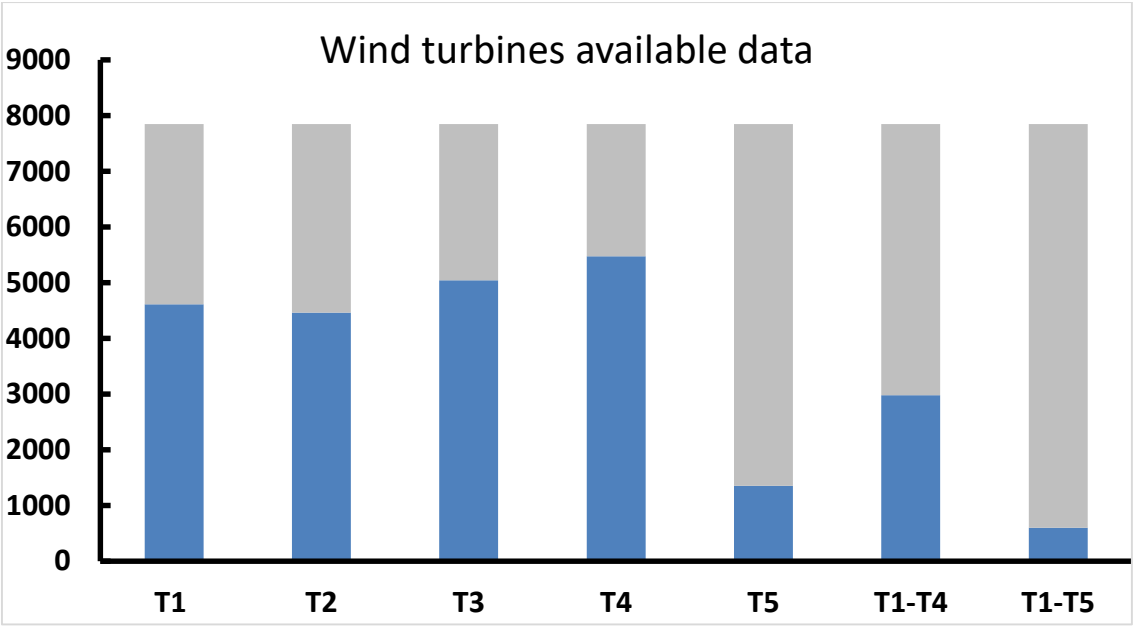


Figure 17. Data available for each of the wind turbines

Also, when cleaning the data, it is appreciable that the turbines present highest performance when the wind is between 13 and 14 m/s. When the wind exceeds 14 m/s, it does not generate more power. To reduce the noise on the data and facilitate the work to the neural network, the wind values exceeding 14 m/s are reduced to 14 m/s.

4.2. Summary of the clean data

Table 1 presents a summary of the powers in the turbines that are going to be used in the experimental part. It is possible to appreciate that the performance of the different wind turbines is similar.

	Power 1	Power 2	Power 3	Power 4
count	2980	2980	2980	2980
mean	1437.7	1379.4	1365.4	1469.7
std	656.7	660	659.2	644.8
min	105	102	101	109
25%	814.8	759.8	745	849.8
50%	1676.5	1530	1487.5	1834
75%	2043	2029	2018	2032
max	2365	2302	2290	2254

Table 1. Summary of the clean data

4.3. Neural network design

In this section the designed neural network is described, starting from the basics, which are the software used to develop the program, followed by the buildup of the code and its functioning.

4.3.1. Software

The chosen software to write the neural network code is Python to take advantage of the previously developed applications and libraries (deposits of functions) in artificial intelligence, including Machine Learning and Neural Networks. The program selected to run Python in is PyCharm Community Edition 2018.2.4 from JetBrains developers. Anaconda is used to simplify the management of packages and complements needed to develop Neural Networks in PyCharm. Anaconda is a free and open-source software used in lots of computing fields like data science or machine learning, and it simplifies the management of all the packages and complements, allowing the development of applications in a virtual environment. Apart from PyCharm and Anaconda – the basic software used to build the Neural Network – two basic libraries, created specifically to develop artificial intelligence programs, have been used: TensorFlow and Keras. TensorFlow is an open-source machine learning library widely used in research. Together with Keras, another Python deep learning library, they provide a high-level API (Application Program Interface), which makes deep learning and neural networks designs simpler.

4.3.2. Programming the neural network

This section explains the development of the code. The full program with some comments can be found in Annex A Neural network building program, and to fully understand it is recommended to read the chapter and the code at the same time.

Once all the required libraries are imported, the first thing to do is to define a function to let the program know which variables or features it is going to work with. The same has to be done with the desired target, the output of the neural network. Both features and target can be defined as regular variables, available in the provided data, or as synthetic ones (combination of other variables). Then, the columns that TensorFlow is going to use as features have to be built using another function. These columns will contain only the variables used in each experiment.

The next step is to define another function that will provide the batches of data to the training function. The inputs of this function are the features and the targets. Additionally, it needs some parameters such as the batch size, the shuffle parameter and the number of epochs. The batch size is the number of samples used to train (update) the neural network. The shuffle parameter defines whether to mix the data randomly to remove the time-related dependencies. The number of epochs defines the amount of times that a program can go through all the data to train the model, but it can also be restricted using the steps.

The most important function of the program is the training function. This is the loop that will improve the results – decreasing the cost – iteration after iteration. The input variables of the function are the optimizer which will be used, and the steps, which are the number of times that the program will go through a batch calibrating the system every time a step is finished. It also needs the hidden layers and the number of neurons in each layer. Finally, it needs the training and validation data. The result of this set of steps is the optimized model of the neural network.

Inside the training function, first of all, the periods are defined. The periods are a way to divide the training of the model in order to follow the process while the program is running. Then, there's a line to clip the gradient in case it is too large to avoid overflow. The next step is to define the deep neural network regressor, which is the estimator to train the neural network. Next, the input functions are defined, calling the function explained in the previous step for the training data and the training and validation predictions. Then, the actual training loop starts and it will be repeated the number of periods defined before the neural network is completed. First, the model starts training from a random point or the last one if the previous is not the initial one. The training and validation losses are computed and printed to give the user some information on the construction of the networks. The progress data is saved in a list and the loop is finished. The last part of the training function is to provide the final performance of the neural network, the Root Mean Squared Error (RMSE), the Normalized Root Mean Squared Error (NRMSE) and the Mean Absolute Error (MAE).

The last part of the program is the main program, which will call a series of functions when it is started. Initially the data is read from a csv file. Then, it is randomized and sorted into two

groups, training and validation data. A summary of the data is shown on the programmer screen to be verified. Finally, the remaining code is the one defining the parameters of the neural network, including the optimizer, the number of steps, the learning rate, the batch size and the architecture of the network (number of neurons per layer and number of hidden layers). These parameters have to be chosen by the programmer through tests.

The tests show that the best optimizer is Adam (Kingma and Ba, 2014). Its functioning is similar to the Gradient Descent optimizer, but it is superior in some aspects like in its adaptable learning rate, which changes from variable to variable in the neural network. The learning rate is chosen through tests that observes the evolution of the performance of the network. The activation function used in the neurons is called ReLu. Its purpose and functioning is similar to the explained Sigmoid function, but it is proved to show better results in artificial neural networks (Glorot et al., 2011). The architecture of the neural network consists in two hidden layers with ten neurons each and full connection between layers. The model has provided the best performance in the undertaken tests.

Figure 18 is a representation of the defined neural networks, where two fully connected hidden layers are represented: an input layer with one variable and an output layers with also one output variable. Depending on the experiment, more input variables are defined.

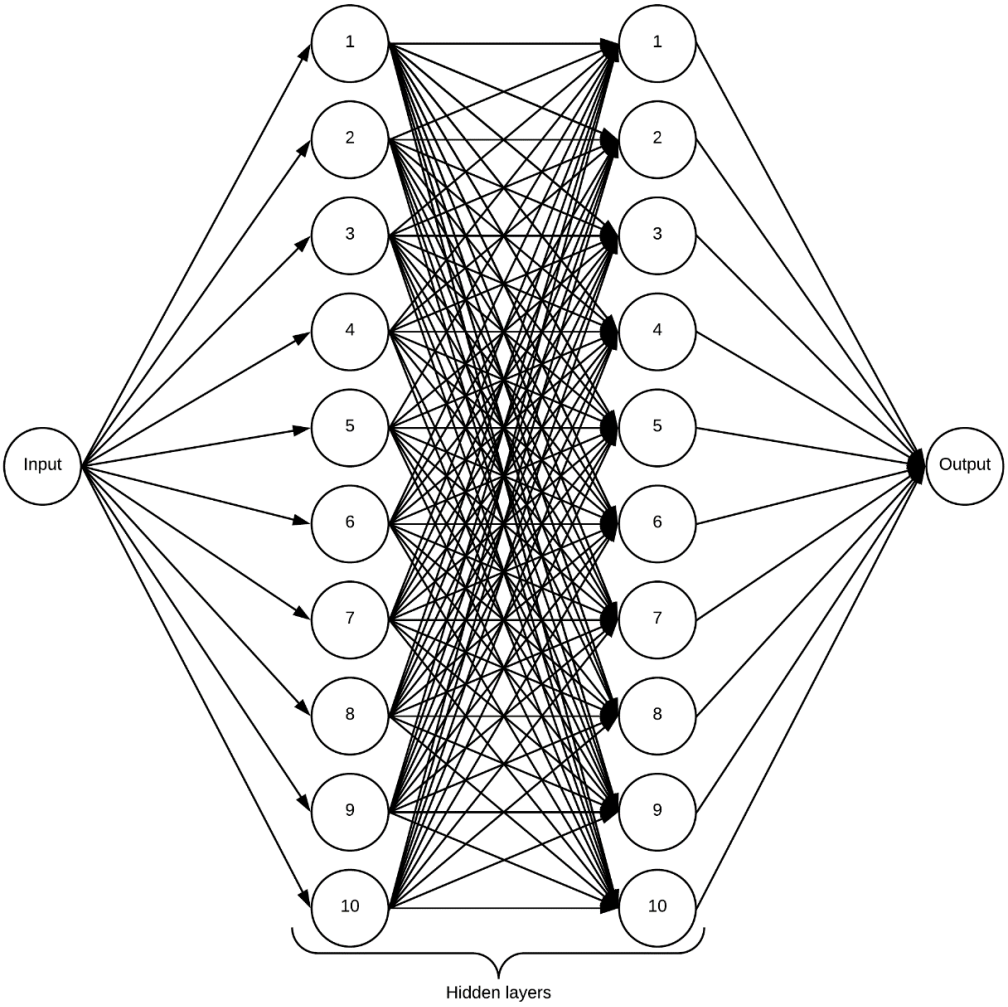


Figure 18. Schematic representation of the defined artificial neural network

In the represented neural network, the model has 10 weights and 10 biases in the first hidden layer, 100 weights and 10 biases in the second hidden layer and 10 weights and 1 bias in the output layer, a total of 141 variables in the model.

4.4. Designed experiments

The experiments designed in this thesis have the purpose to prove if a model can be designed to have into account the wake created by wind turbines in other wind turbines. The experiments are presented in increasing order of complexity and compared using quality indications such as RMSE, NRMSE, MAE and the coefficient of determination (R^2). The parameters of the network (hidden layers, learning rate, …) are kept constant throughout the different experiments. The dataset includes the points available from turbines T1-T4, represented in Figure 17. The data is separated into two groups: 2300 points will be used to train the model and 680 to validate it.

4.4.1. Experiment 1

In this experiment, only the wind speed recorded at a certain turbine is used to predict its power. This is the stand-alone case, in which none of the other turbines are taken into consideration. The program cannot take into account the wake of another turbine affecting the selected one, and as a consequence its power prediction will be less accurate. This approach should define the velocity-power characteristics of the turbine. The artificial neural network resulting from this model will have a total of 141 variables, like it has been explained with the Figure 18.

4.4.2. Experiment 2

In the second experiment, the power of each turbine is predicted using the velocity of the wind in the selected wind turbine and its closest neighbors. A relationship matrix tells the program in which conditions a wind turbine influences another. This matrix multiplies by one the wind speed of a turbine when it has an effect in the studied wind turbine, and by zero when it has no effect. For example, when the wind turbine 1 is being modeled, the matrix is only 1 when the wind comes from the direction of wind turbine 2. In this experiment only the closest turbines are considered. In this case, the artificial neural network resulting from the model will have a total of 151 or 161 weights and biases. Annex E Matrix program includes the code of the program used in the experiments 2 and 3 to build the matrix, developed in Excel VBA.

4.4.3. Experiment 3

This third experiment is similar to the second but all the wind turbines are considered when the wind comes from their direction. For example, the power of the first turbine may be influenced by all the others when the wind is blowing at certain angles. The performance and accuracy of this model are then expected to be higher than the second experiment if the faraway wake effect is important. The resulting neural network built using this experiment will have a total of 171 variables.

4.4.4. Experiment 4

In this fourth experiment the model is fed information on all the wind speeds and angle of the wind at the wind farm without a direct input on when the wake is relevant. Since the program is expected to learn on its own all the information given by the programmer in experiment 3, it's an interesting way to check if it can find these patterns automatically. The artificial neural network resulting from this experiment will have 181 variables.

4.5. Evaluation of the models

The experimental design includes a model built for each particular case. For example, in the first experiment, there's a model for the wind turbine 1, another for the wind turbine 2, the 3rd and the 4th. The same happens with the other experiments, up to a total of 16 models. This setup implies that a model representing the first wind turbine will be independent of the second one. Each model has been run with all the available data to present their performance. Then, the results are plotted against the labeled power (real power measured at the wind farm). If the model was perfect, this graph should be a straight line. R^2 gives statistical information about how well a model follows linearity. Also, a line representing the model is plotted in the same graph. Other statistical indicators such as the RMSE, the NRMSE and the MAE are used to discuss the accuracy of the models. The equations to calculate these statistical indicators are presented below.

Mean absolute error (MAE):

$$MAE = \frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|}{n} \tag{29}$$

Where n is the number of experiments, $\hat{y}$ the power predicted by the model for a data point i and y the labeled power for the same i data point.

Root mean squared error (RMSE):

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}{n}} \tag{30}$$

And the normalized root mean squared error (NRMSE):

$$NRMSE = \frac{RMSE}{y_{max} - y_{min}} \tag{31}$$

Where y_{max} and y_{min} are respectively the maximum and minimum of the labeled powers.

The evaluation includes the comparison of the performance of each wind turbine independently. This way, it is possible to compare the outcome of the different experiments for all the different wind turbines.

5. Results

This section includes the predictions made by the neural network. First, the results are presented divided into experiments, representing the results through graphs. Then, the results are grouped into turbines to find which model explains the behavior of each turbine better.

5.1. Results divided into experiments

5.1.1. Experiment 1

Figure 19, Figure 20, Figure 21 and Figure 22 represent the predictions of the first experiment for turbines one to four. All the models show a linear correlation between the measured power and the one predicted by the artificial neural network. The coefficient of determination R^2 has the highest value in turbine four, meaning it is the best model following linearity.

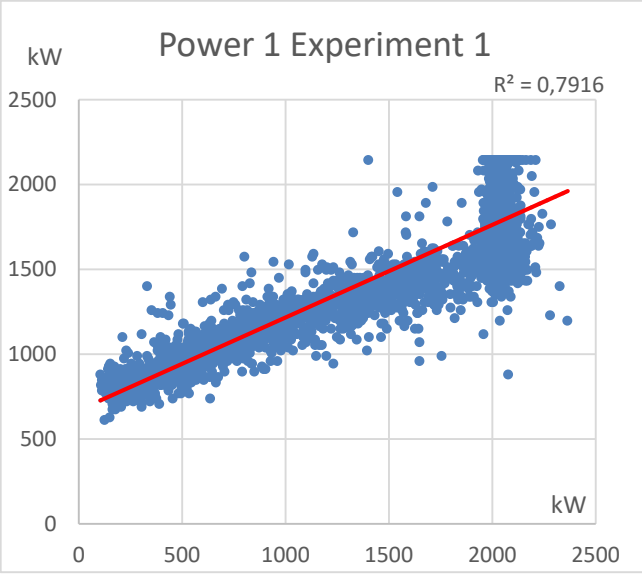


Figure 19. Power at wind turbine 1 in experiment 1

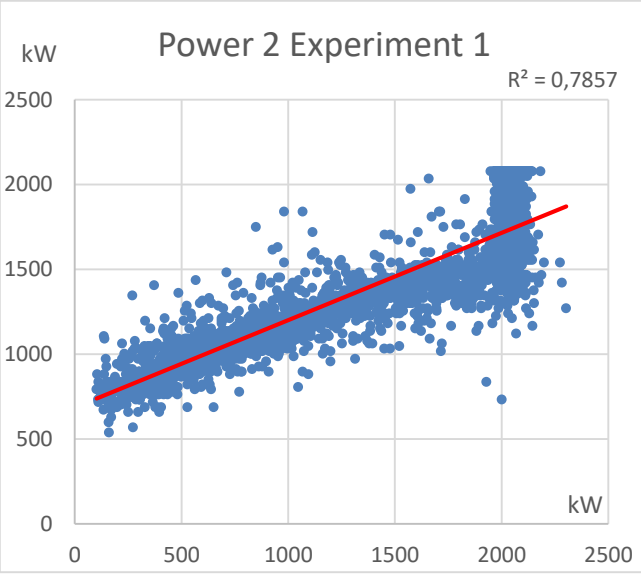


Figure 20. Power at wind turbine 2 in experiment 1

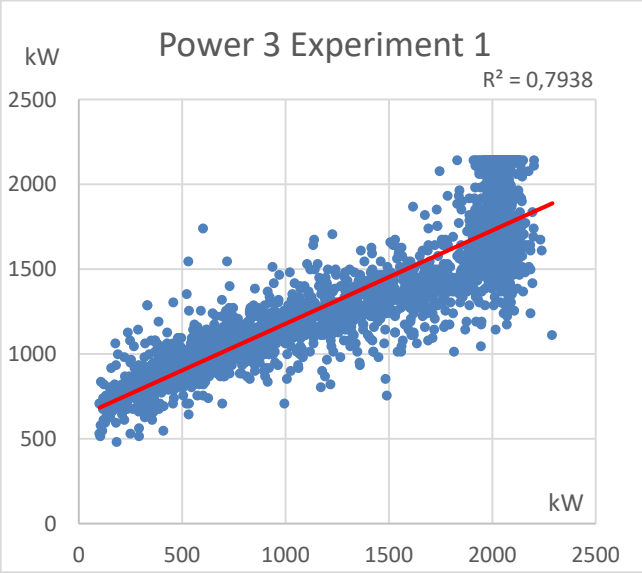


Figure 21. Power at wind turbine 3 in experiment 1

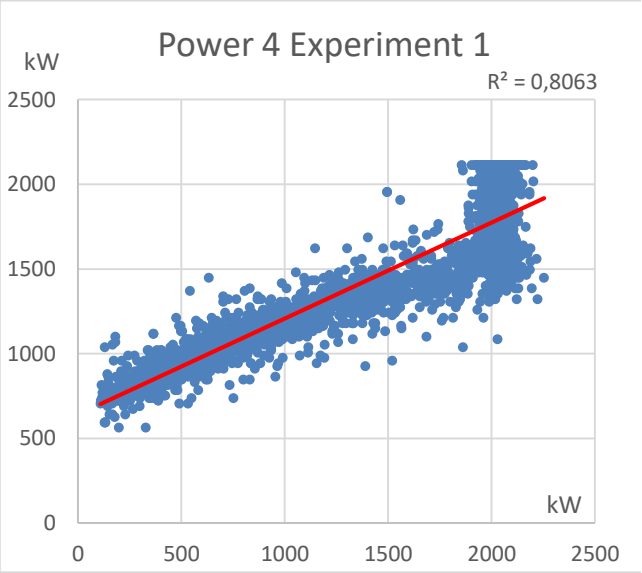


Figure 22. Power at wind turbine 4 in experiment 1

The plot also shows that when the maximum wind speed is reduced to 14 m/s to limit the noise, following the procedure explained in the data cleaning section, the representation has the shape of a line in the top part. Setting a maximum of wind velocity results in a better fit than the using same model without the filter. Table 2 shows the calculated error indicators for all the turbines in the experiment 1. The best performance of the neural network corresponds to turbine 4 as it has both the highest R^2 and the lowest MAE, RMSE, and NRMSE.

E1	T1	T2	T3	T4
R2	0.79116	0.7857	0.7938	0.8063
RMSE	350.7625	366.3888	349.656	332.0038
MAE	296.1015	308.1293	297.6755	276.917
NRMSE	0.1552	0.1665	0.1597	0.1548

Table 2. Experiment 1 indicators

5.1.2. Experiment 2

Figure 23, Figure 24, Figure 25 and Figure 26 represent the predictions of the second experiment for the turbines one to four. All the models show a linear correlation between the measured power and the one predicted by the artificial neural network. In this experiment, the turbine with the highest R^2 value is the first one, followed by the fourth

.

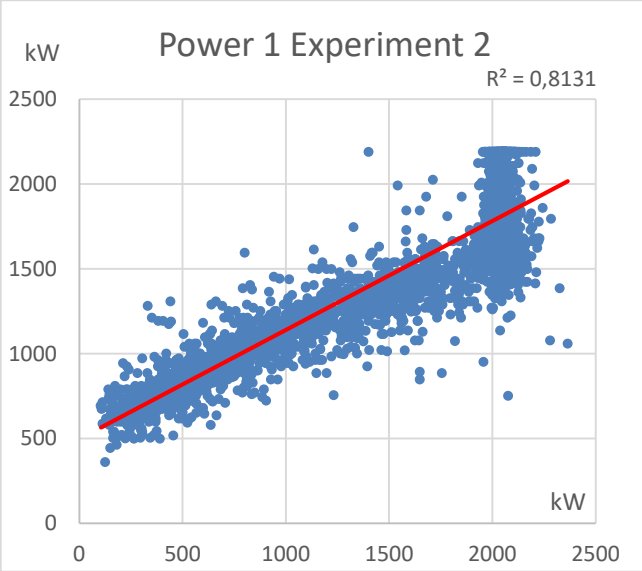


Figure 23. Power at wind turbine 1 in experiment 2

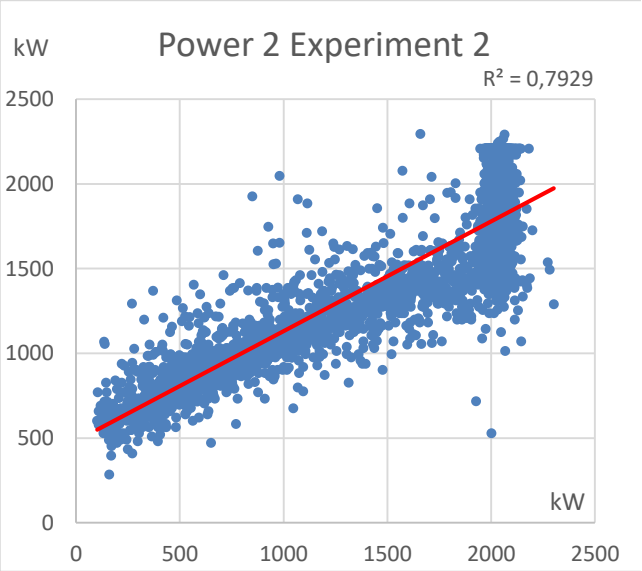


Figure 24. Power at wind turbine 2 in experiment 2

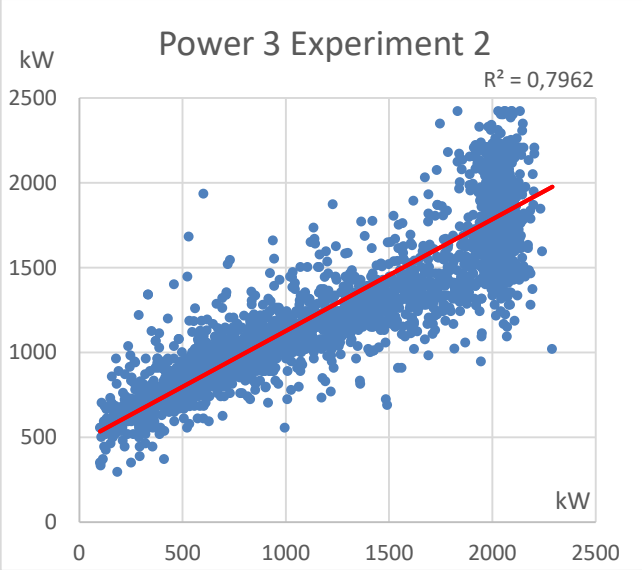


Figure 25. Power at wind turbine 3 in experiment 2

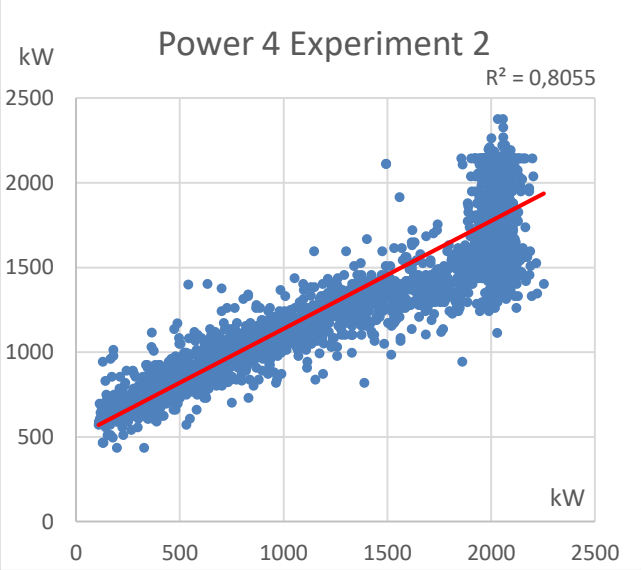


Figure 26. Power at wind turbine 4 in experiment 2

Table 3 presents the calculated indicators for all the turbines in the experiment 2. In this case, the best indicators also correspond to turbine 4 apart from the NRMSE, which is slightly lower in turbine 1.

E2	T1	T2	T3	T4
R2	0.8131	0.7929	0.7962	0.8055
RMSE	310.5114	318.9844	314.413	310.4788
MAE	260.2317	264.24826	261.2116	257.0302
NRMSE	0.1374	0.145	0.1436	0.1447

Table 3. Experiment 2 indicators

5.1.3. Experiment 3

Figure 27, Figure 28, Figure 29 and Figure 30 represent the predictions of the third experiment for the turbines one to four. All the models show a linear correlation between the measured power and the one predicted by the artificial neural network. In this experiment turbines one, two and four present really close R^2 values, but the highest value is the one of the second wind turbine.

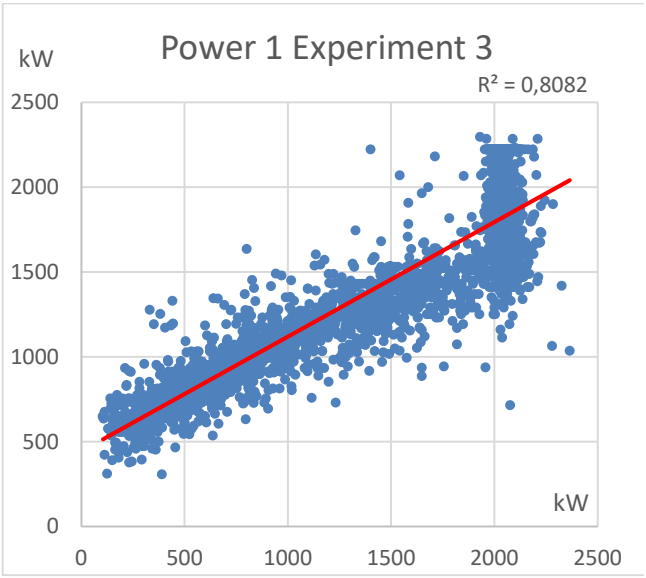


Figure 27. Power at wind turbine 1 in experiment 3

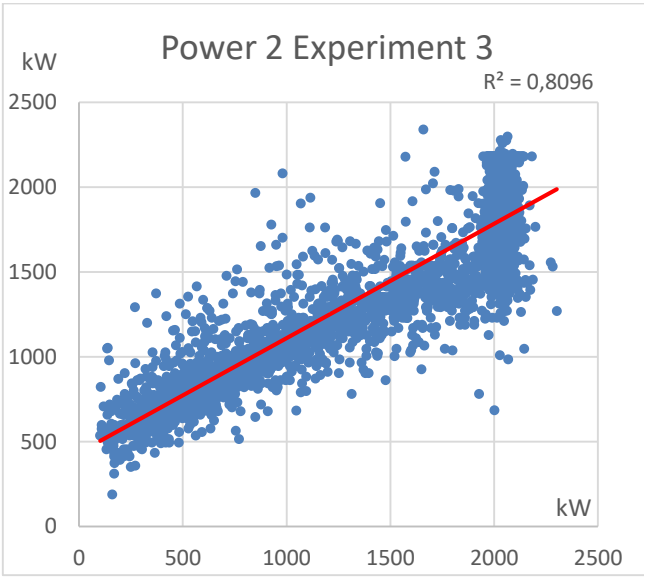


Figure 28. Power at wind turbine 2 in experiment 3

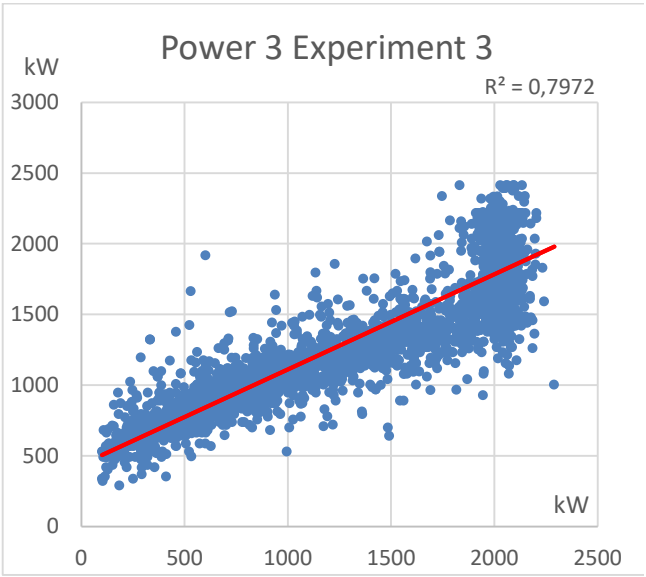


Figure 29. Power at wind turbine 3 in experiment 3

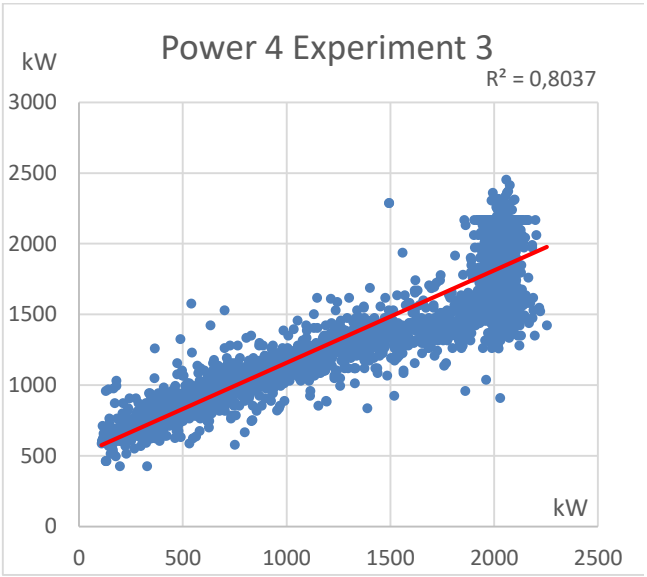


Figure 30. Power at wind turbine 4 in experiment 3

Table 4 shows the indicators calculated for the four turbines in this experiment. In this experiment both the first and second wind turbines perform similarly good according to the indicators. The wind turbine four is also close to the best results.

E3	T1	T2	T3	T4
R2	0.8082	0.8096	0.7972	0.8037
RMSE	304.3329	304.8149	310.6873	305.3367
MAE	253.8752	247.871	256.8317	252.8263
NRMSE	0.1347	0.1386	0.1419	0.1423

Table 4. Experiment 3 indicators

5.1.4. Experiment 4

Figure 31, Figure 32, Figure 33 and Figure 34 represent the predictions of the fourth experiment for the turbines one to four. All the models show a linear correlation between the measured power and the one predicted by the artificial neural network. The coefficient of determination R^2 has the highest value in turbine one, meaning fits linearity better.

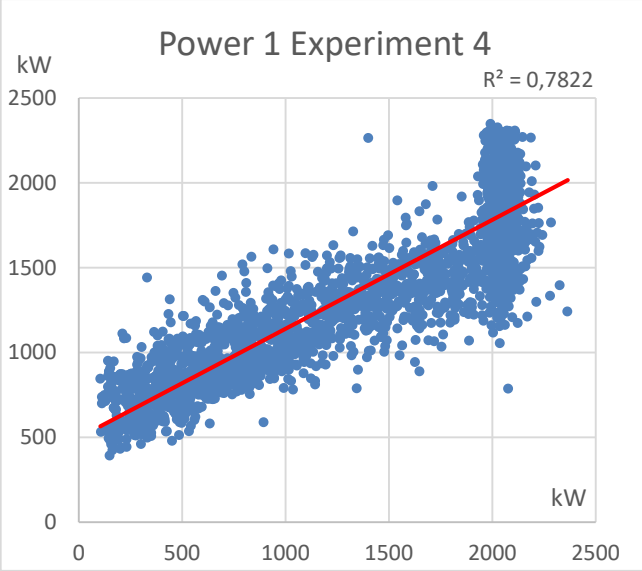


Figure 31. Power at wind turbine 1 in experiment 4

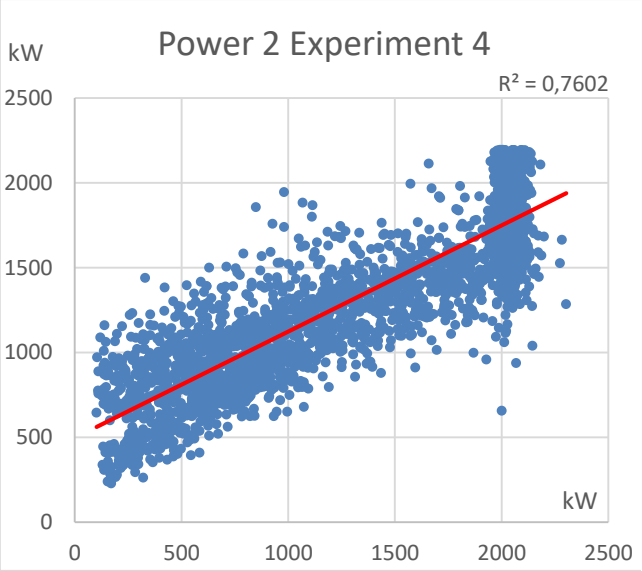


Figure 32. Power at wind turbine 2 in experiment 4

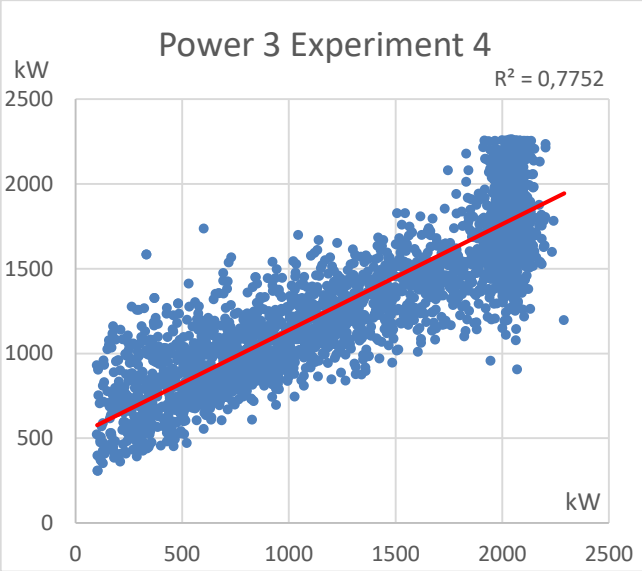


Figure 33. Power at wind turbine 3 in experiment 4

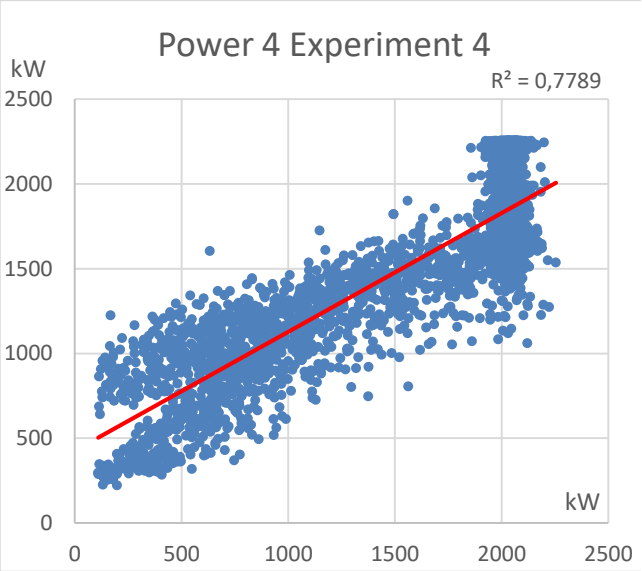


Figure 34. Power at wind turbine 4 in experiment 4

Table 5 shows the calculated indicators for all the turbines in the experiment 4. In this case, the best models according to the indicators are the ones corresponding to the first and fourth wind turbines.

E4	T1	T2	T3	T4
R2	0.7822	0.7602	0.7752	0.7789
RMSE	324.0669	339.1047	332.5054	308.8229
MAE	258.0322	267.9857	265.926	247.0056
NRMSE	0.1434	0.1541	0.1519	0.144

Table 5. Experiment 4 indicators

5.2. Results divided into wind turbines

5.2.1. Wind turbine 1

Table 6 presents the indicators for the first wind turbine in the experiments one to four.

T1	E1	E2	E3	E4
RMSE	350.7625	310.5114	304.3329	324.0669
MAE	296.1015	260.2317	253.8752	258.0322
NRMSE	0.1552	0.1374	0.1347	0.1434

Table 6. Wind turbine 1 indicators

The lowest value for all the indicators is found in the experiment 3, which creates an artificial neural network providing as data the wind of the first turbine and the winds of the other three turbines as well as a matrix to indicate when the wake of this turbines should influence the first wind turbine. The second best performing model is the one built in the second experiment. Then, the one from the experiment four and the last is the experiment one.

5.2.2. Wind turbine 2

Table 7 presents the indicators for the second wind turbine in the experiments one to four.

T2	E1	E2	E3	E4
RMSE	366.3888	318.9844	304.8149	339.1047
MAE	308.1293	264.2483	247.871	267.9857
NRMSE	0.1665	0.145	0.1386	0.1541

Table 7. Wind turbine 2 indicators

The same way that happened with the first wind turbine, for the second one the lowest value for all the indicators belongs to the experiment three. The second best model is the one from the second experiment; then come experiments four and one.

5.2.3. Wind turbine 3

Table 8 presents the indicators for the third wind turbine in the experiments one to four.

T3	E1	E2	E3	E4
RMSE	349.656	314.413	310.6873	332.5054
MAE	297.6755	261.2116	256.8317	265.926
NRMSE	0.1597	0.1436	0.1419	0.1519

Table 8. Wind turbine 3 indicators

As well as in the wind turbines one and two, the best model to represent the behavior of wind turbine three is the one belonging to the experiment three. It presents the lowest value for all the indicators. Also like in the two previously studied turbines, order of the other models from best to worse is model two, four and the worst is the first model one.

5.2.4. Wind turbine 4

Table 9 presents the indicators for the fourth wind turbine in the experiments one to four.

T4	E1	E2	E3	E4
RMSE	332.0038	310.4788	305.3367	308.8229
MAE	276.917	257.0302	252.8263	247.0056
NRMSE	0.1548	0.1447	0.1423	0.144

Table 9. Wind turbine 4 indicators

The wind turbine four presents also the best indicators when calculated from the model three. In this case though, the lower MAE was found in the fourth model, which is the second best model according to the indicators. Then the second and the last, like in all the other wind turbines, is the model built in the experiment one.

5.3. Summary of results

Table 10 presents a summary of the times that, according to the calculated indicators, a model performs the best, 2nd, 3rd or 4th. The best model to represent the behavior of all the turbines is always the third, where the program has all well-organized data in order to take into account the effect of the wake. Also, the worst model according to the indicators is number one, where no actual measurements were provided to the program in order to make the artificial neural network learn whether or not the wake was influencing the obtained power.

	Model 1	Model 2	Model 3	Model 4
No. times best	0	0	4	0
No. times 2nd	0	3	0	1
No. times 3rd	0	1	0	3
No. times 3th	4	0	0	0

Table 10. Number of times that a model is the best, 2nd, 3rd or 4th according to the indicators

Table 11 presents the reduction (%) of the Mean Absolute Error between models three and one. This decrease exceeds 10 %.

	T1	T2	T3	T4	Mean
MAE reduction E1 vs E3 (%)	14.2608	13.0282	13.7209	8.6996	12.4274

Table 11. MAE reduction between models 3 and 1

Table 12 shows the relative error represented using the MAE indicator for model three and its correlation to the actual measured power in all the studied turbines.

	Wind turbine 1	Wind turbine 2	Wind turbine 3	Wind turbine 4
Mean power (kW)	1437.7	1379.4	1365.4	1469.7
MAE model 3 (kW)	253.8752	247.871	256.8317	252.8263
MAE/Mean power (%)	17.66	17.97	18.81	17.20

Table 12. Error percentatge MAE model 3/Mean power

Annex C Example of weights and bias includes a presentation of all the weights and values for one of the modeled artificial neural networks, specifically for the wind turbine two in the experiment three.

6. Discussion

The aim of this master thesis was to build a model based on artificial neural networks able to estimate the power of wind turbines in a certain wind farm considering the wake effect. In the experimental part, four models were presented in order to provide different inputs to the system and determinate which kind of inputs could make the model build the best artificial neural network. The best model is the one corresponding to the third experiment. In the experimental part for each wind turbine 4 indicators are calculated, giving a total of 16 indicators. From this indicators, 13 are best when calculated from the estimations done by the artificial neural network built using the experiment three. The particularity of this experiment is that it provides the program with the required information to take into account the wake losses to estimate the power of the wind turbines. Then, this model is the best at explaining the behavior of the wind turbines and the wake losses in a maximum of situations possible.

From the plots in the previous section (Figure 19 to Figure 34) it is possible to appreciate that the model represents the performance of the wind farms in most of the cases. However, it is possible to recognize a strange behavior of the models in all the experiments when the power is high, or more precisely, when the wind is high. It is a matter to study further why the models are not able to learn good enough the fact that at a certain wind speed, the increase of this wind speed is not transported in an increase of the power. To minimize this problem, a filter in the data cleaning was set.

Table 12 presents the error percentage that the MAE represents over the real power mean. The values are around 17%. Although the model clearly represents the behavior of the wind turbines, the accuracy of the model is not high. A possible explanation for this fact is the case commented in the previous paragraph. Also, it is possible that the amount of available data used to build the model is not enough for the program to learn some particularities. In future research it would be interesting to gather more data from the same wind farm and build the models again. Theoretically, more data point should make the neural network learn more patterns from it, like the fact that when the winds are higher than 14 m/s the power is not getting higher.

7. Conclusions

Wind energy plays an important role in the electric generation worldwide, and the share of power generated through it is going to increase in the next twenty years. Hence, it is important to further research in the field, in order to provide better wind farms in the future.

One of the actual problems in wind energy is to know the power that a wind farm is going to generate in certain conditions. This is difficult due to the complexity of the physics involved in all the variables that influence the result. One of the factors to take into account is the wake effect. It is the loss of velocity and appearance of turbulence in the wind downstream a turbine, due to the disturbance created the same. This effect reduces the energy that the wind turbines can generate downstream, and it is then interesting to study and take into account when predicting wind farms power and designing new wind farms.

Between the different approaches used to calculate the wake effect, or the power of a wind farm including the wake effect, a relatively new is through artificial intelligent problems. This thesis applies machine learning through artificial neural networks, in order to find out if a model can be built with the available resources (a regular computer) to calculate the power of a certain wind farm taking into account the losses due to the wake effect. A limitation is presented by the inability of the neural network to learn with the available data that a shift in the pattern such as that from certain wind speed the power will not increase any further.

A total of 16 models have been built in the experimental part, four representing each wind turbine in all the wind turbines. According to all the indicators, the best performing models have always been the ones where information was provided to the program in order to build a neural network to have into account the wake losses to calculate the power of the turbines. In the best model information for the program to learn about far and close wake effect was provided, in the second best only to consider wake from close wind turbines. In the second worse, the information was provided to the model to learn from far and close wind turbines wake, but additional information about how to do it. The worst model was the one where no way to learn from wake was provided. In summary, more refined organization of the data provided to the neural network provides the best results. Given the presented results, it is possible to predict the power of wind turbines taking into account the wake losses by using an artificial neural network that improves the performance of an artificial neural network that is not considering wake losses.

8. References

Ammara, I., Leclerc, C., Masson, C., 2002. A Viscous Three-Dimensional Differential/Actuator-Disk Method for the Aerodynamic Analysis of Wind Farms. J. Sol. Energy Eng. 124, 345. https://doi.org/10.1115/1.1510870

Archer, C.L., Vasel-Be-Hagh, A., Yan, C., Wu, S., Pan, Y., Brodie, J.F., Maguire, A.E., 2018. Review and evaluation of wake loss models for wind energy applications. Appl. Energy 226, 1187–1207. https://doi.org/10.1016/j.apenergy.2018.05.085

Barthelmie, R.J., Hansen, K., Frandsen, S.T., Rathmann, O., Schepers, J.G., Schlez, W., Phillips, J., Rados, K., Zervos, A., Politis, E.S., Chaviaropoulos, P.K., 2009. Modelling and measuring flow and wind turbine wakes in large wind farms offshore. Wind Energy 12, 431–444. https://doi.org/10.1002/we.348

Bonanni, A., Banyai, T., Conan, B., VanBeeck, J., Deconinck, H., Lacor, C., 2012. Wind Farm Optimization Based on CFD Model of Single Wind Turbine Wake. Eur. Wind Energy Conf. Exhib. 2012 1–10. https://doi.org/10.1016/j.ejim.2013.03.005

Boosman, F., 2017. "Whatever Machines Haven't Done Yet" [WWW Document]. December 2017. URL http://www.udu.co/blog/whatever-machines-havent-done-yet/

Brusca, S., Capizzi, G., Lo Sciuto, G., Susi, G., 2017. A new design methodology to predict wind farm energy production by means of a spiking neural network-based system. Int. J. Numer. Model. Electron. Networks, Devices Fields 1–14. https://doi.org/10.1002/jnm.2267

Canada, N.R., 2018. Wind Energy R&D Park and Storage System for Innovation in Grid Integration [WWW Document]. URL https://www.nrcan.gc.ca/energy/funding/current-funding-programs/cef/4979

Developers, G., 2018. Machine Learning Crash Course [WWW Document]. URL https://developers.google.com/machine-learning/crash-course/

European Environment Agency, 2018a. Air pollution still too high across Europe.

European Environment Agency, 2018b. Outdoor air quality in urban areas.

Frandsen, S., 2007. Turbulence and turbulence-generated structural loading in wind turbine clusters.

Glorot, X., Bordes, A., Bengio, Y., 2011. Deep Sparse Rectifier Neual Networks. Proc. Fourteenth Int. Conf. Artif. Intell. Stat. 15, 315–323. https://doi.org/10.1.1.208.6449

Göçmen, T., Laan, P. Van Der, Réthoré, P.E., Diaz, A.P., Larsen, G.C., Ott, S., 2016. Wind turbine wake models developed at the technical university of Denmark: A review. Renew. Sustain. Energy Rev. 60, 752–769. https://doi.org/10.1016/j.rser.2016.01.113

González-Longatt, F., Wall, P.P., Terzija, V., 2012. Wake effect in wind farm performance: Steady-state and dynamic behavior. Renew. Energy 39, 329–338. https://doi.org/10.1016/j.renene.2011.08.053

Google Earth Pro, 2018. Google Earth Pro.

GWEC, 2018. Global Wind Energy Report: Annual Market Update 2017.

Herculano-Houzel, S., 2009. The human brain in numbers: a linearly scaled-up primate brain. Front. Hum. Neurosci. 3, 1–11. https://doi.org/10.3389/neuro.09.031.2009

I.KATIC, N.O.JENSEN, J.HØJSTRUP., 1987. A Simple Model for Cluster Efficiency.

International A/S, E., 2010. WindPRO 2.7 User Guide 3.

International Energy Agency (IEA), 2018. Renewables information: overview. https://doi.org/10.1787/re_mar-2018-en

International Energy Agency (IEA), 2017. World energy outlook 2017. https://doi.org/10.1787/weo-2017-en

Japar, F., Mathew, S., Narayanaswamy, B., Ming Lim, C., Hazra, J., 2014. Estimating the Wake Losses in Large Wind Farms : A machine learning approach. Isgt 2014 1–5. https://doi.org/10.1109/ISGT.2014.6816427

Jensen, N., 1983. A note on wind turbine interaction, Riso-M-2411, Risoe National Laboratory, Roskilde, ….

Kaplan, A., Haenlein, M., 2018. Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence. Bus. Horiz. 62, 15–25. https://doi.org/10.1016/j.bushor.2018.08.004

Kingma, D.P., Ba, J.L., 2014. Adam: A method for tochastic optimization. ICLR 2015 1631, 58–62. https://doi.org/10.1063/1.4902458

Kusiak, A., Zheng, H., Song, Z., 2009. Wind farm power prediction: a data-mining approach. Wind Energy 12, 275–293. https://doi.org/10.1002/we.295

Larsen, G.C., 1988. A Simple Wake Calculation Procedure, Risø-M.

mc.ai, 2018. Cats And Dogs Classifier | Convolutional Neural Network with Python and Tensorflow (9 steps of… [WWW Document]. URL https://mc.ai/cats-and-dogs-classifier-convolutional-neural-network-with-python-and-tensorflow-9-steps-of/

McCulloch, W.S., Pitts, W.H., 1943. A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. 5, 115–133. https://doi.org/10.1016/j.joca.2006.06.009

Mortensen, N.G., Landberg, L., Troen, I., Petersen, E.L., Rathmann, O., Nielsen, M., 2001. Wind Atlas Analysis and Application Program (WAsP). Vol. 3: Utility Programs.

National Institute of Neurological Disorders and Stroke, 2018. Brain Basics: The Life and Death of a Neuron [WWW Document]. URL https://www.ninds.nih.gov/Disorders/Patient-Caregiver-Education/Life-and-Death-Neuron

Nielsen, M.A., 2015. Neural Networks and Deep Learning.

Niemenmaa, V., Happach, B., Kubat, J., Otto, J., Pirelli, L., Simeonova, R., Zalega, A., Wisniewska-Danek, K., Radecka-Moroz, K., Wojciechowski, J., Soblet, F., Friel, C., Coelho, J., 2018. Air pollution: Our health still insufficiently protected. https://doi.org/10.2865/363524

Ott, S., Berg, J., Nielsen, M., 2011. Linearised CFD Models for Wakes. Risø Natl. Baeredygtig Energi. 1772, 41.

Răzuşi, P.C., Eremia, M., 2011. Prediction of wind power by artificial intelligence techniques.

2011 16th Int. Conf. Intell. Syst. Appl. to Power Syst. ISAP 2011 1–6. https://doi.org/10.1109/ISAP.2011.6082239

Rosenblatt, F., 1957. The perceptron, a perceiving and recognizing automaton.

Russell, S., Norvig, P., 2009. Artificial Intelligence: A Modern Approach. https://doi.org/10.1017/S0269888900007724

S. Lissaman, P.B., 1979. Energy Effectiveness of Arbitrary Arrays of Wind Turbines. J. Energy 3, 323–328. https://doi.org/10.2514/3.62441

Sørensen, T., Nielsen, P., Thøgersen, M.L., 2006. Recalibrating Wind Turbine Wake Model Parameters - Validating the Wake Model Performance for Large Offshore Wind Farms. Eur. Wind Energy Conf. Exhib. EWEA. https://doi.org/10.1080/01904167.2012.737885

Thomsen, K., Madsen, H.A., Larsen, G.C., Larsen, T.J., 2007. Comparison of methods for load simulation for wind turbines operating in wake. J. Phys. Conf. Ser. 75. https://doi.org/10.1088/1742-6596/75/1/012072

Winter, R., Widrow, B., 1988. Madaline rule II: A training algorithm for neural networks. Neural Networks 1, 148. https://doi.org/10.1016/0893-6080(88)90187-6

Annex

Annex A Neural network building program

```
# Start of the code importing the libraries
from __future__ import absolute_import, division, print_function
import math
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn import metrics
from tensorflow.python.data import Dataset

# Function to define the inputs. Different depending on the model and turbine
def preprocess_features(wind_farm_dataframe):
  selected_features = wind_farm_dataframe[
    ["WSpeed_1"]]
  processed_features = selected_features.copy()
  return processed_features


def preprocess_targets(wind_farm_dataframe):  # Function to define the target of the neural
network. “Power_1” changes depending on the wind turbine
  selected_targets = wind_farm_dataframe[
    ["Power_1"]]
  output_targets = selected_targets.copy()
  return output_targets

# Function used to construct the columns used by the program with the data
def construct_feature_columns(input_features):
  return set([tf.feature_column.numeric_column(my_feature) for my_feature in
input_features])

# Function in charge to provide the batches of data to the training function
def my_input_fn(features, targets, batch_size=1, shuffle=True, num_epochs=10):
  features = {key: np.array(value) for key, value in dict(features).items()}
  ds = Dataset.from_tensor_slices((features, targets))
  ds = ds.batch(batch_size).repeat(num_epochs)

  if shuffle:
    ds = ds.shuffle(10000)

  features, labels = ds.make_one_shot_iterator().get_next()
  return features, labels


def train_nn_regression_model(my_optimizer, steps, batch_size, hidden_units,
training_examples, training_targets,
                              validation_examples, validation_targets):  # Training function
  periods = 10  # Definition of the number of periods in which to separate the training
  steps_per_period = steps / periods
```

```
    my_optimizer = tf.contrib.estimator.clip_gradients_by_norm(my_optimizer, 5.0)  # Clipping of the gradient, to avoid overflow

    # Definition of the regressor (estimator) used to train the neural network. The model is saved with the name specified by model_dir, which changes with every wind turbine and experiment.
    dnn_regressor = tf.estimator.DNNRegressor(feature_columns=construct_feature_columns(training_examples), hidden_units=hidden_units, optimizer=my_optimizer, model_dir='Program_1_T1')

    # Creation of the input data columns which the network will use. “Power_1” changes depending on the modeled wind turbine
    def training_input_fn(): return my_input_fn(training_examples, training_targets["Power_1"], batch_size=batch_size)

    def predict_training_input_fn(): return my_input_fn(training_examples, training_targets["Power_1"], shuffle=False, num_epochs=1)

    def predict_validation_input_fn(): return my_input_fn(validation_examples, validation_targets["Power_1"], shuffle=False, num_epochs=1)

    print("Training model...")  # Information for the developer about the program track
    print("RMSE (on training data):")
    # Creation of two variables vectors, which will store the training and validation rmse to monitor the progress
    training_rmse = []
    validation_rmse = []

    # Start of the training. Inside a loop, to provide updates to the developer
    for period in range(0, periods):
        dnn_regressor.train(input_fn=training_input_fn, steps=steps_per_period)  # Training of the ANN

        # Compute and manage predictions on the training data
        training_predictions = dnn_regressor.predict(input_fn=predict_training_input_fn)
        training_predictions = np.array([item['predictions'][0] for item in training_predictions])

        # Compute and manage predictions on the validation data
        validation_predictions = dnn_regressor.predict(input_fn=predict_validation_input_fn)
        validation_predictions = np.array([item['predictions'][0] for item in validation_predictions])

        # Compute training and validation root mean squared error
        training_root_mean_squared_error = math.sqrt(metrics.mean_squared_error(training_predictions, training_targets))
        validation_root_mean_squared_error = math.sqrt(metrics.mean_squared_error(validation_predictions, validation_targets))

        # Print the training root mean squared error as information for the developer
        print(" period %02d : %0.2f" % (period, training_root_mean_squared_error))

        # Adds training and validation RMSE to the vector generated before
```

```
        training_rmse.append(training_root_mean_squared_error)
        validation_rmse.append(validation_root_mean_squared_error)

    # End of the training loop. Printing of some vectors.
    print("Model training finished.")
    print(training_rmse)
    print(validation_rmse)

    # Prints the final training and validation RMSE
    print("Final RMSE (on training data):   %0.2f" % training_root_mean_squared_error)
    print("Final RMSE (on validation data): %0.2f" % validation_root_mean_squared_error)

    return dnn_regressor, training_rmse, validation_rmse  # End of the training function


# First executed lines on the code. Lecture of the data file
wind_farm_dataframe = pd.read_csv('4T_data_z.csv', sep=";")
# Randomization of the data
wind_farm_dataframe =
wind_farm_dataframe.reindex(np.random.permutation(wind_farm_dataframe.index))

# Separation of the data into training and validation
training_dataframe = wind_farm_dataframe.head(2300)
validation_dataframe = wind_farm_dataframe.tail(680)

# Definition of the training data input variables and targets, calling the preprocess function
training_examples = preprocess_features(training_dataframe)
training_targets = preprocess_targets(training_dataframe)

# Definition of the validation data input variables and targets, calling the preprocess function
validation_examples = preprocess_features(validation_dataframe)
validation_targets = preprocess_targets(validation_dataframe)


# Description of the data
print('Training examples summary:')
print(training_examples.describe())
print('Validation examples summary:')
print(validation_examples.describe())

print('Training targets summary:')
print(training_targets.describe())

print('Validation targets summary:')
print(validation_targets.describe())


# Definition of the variables of the network. It also calls the training function, starting the
training process
_, adam_training_losses, adam_validation_losses = train_nn_regression_model(
    my_optimizer=tf.train.AdamOptimizer(learning_rate=0.003),
```

```
    steps=2000,
    batch_size=120,
    hidden_units=[10, 10],
    training_examples=training_examples,
    training_targets=training_targets,
    validation_examples=validation_examples,
    validation_targets=validation_targets)
# End of the program
```

The function preprocess_targets changes depending on the experiment and wind turbine. The code shown above corresponds to the first experiment and first wind turbine.

Second experiment second wind turbine:

```
def preprocess_targets(wind_farm_dataframe):  # Function to define the target of the neural network, changes depending the model
    selected_features = wind_farm_dataframe[
      ["WSpeed_2"]]
    processed_features = selected_features.copy()
    # Create a synthetic feature. Only turibines 1 and 3 are close to turbine 2
    processed_features["WW12"] = (
          wind_farm_dataframe["WSpeed_1"] *
          wind_farm_dataframe["T1_T2"])
    processed_features["WW32"] = (
          wind_farm_dataframe["WSpeed_3"] *
          wind_farm_dataframe["T3_T2"])
    return processed_features
```

Third experiment second wind turbine:

```
def preprocess_features(wind_farm_dataframe): # Function to define the target of the neural network, changes depending the model
    selected_features = wind_farm_dataframe[
      ["WSpeed_2"]]
    processed_features = selected_features.copy()
    # Create a synthetic feature. All the turbines are considered
    processed_features["WW12"] = (
          wind_farm_dataframe["WSpeed_1"] *
          wind_farm_dataframe["T1_T2"])
    processed_features["WW32"] = (
          wind_farm_dataframe["WSpeed_3"] *
          wind_farm_dataframe["T3_T2"])
    processed_features["WW42"] = (
          wind_farm_dataframe["WSpeed_4"] *
          wind_farm_dataframe["T4_T2"])
    return processed_features
```

Fourth experiment any wind turbine:

```
def preprocess_features(wind_farm_dataframe): # Function to define the target of the neural network, changes depending the model
  selected_features = wind_farm_dataframe[
    ["Direction",
     "WSpeed_1",
     "WSpeed_2",
     "WSpeed_3",
     "WSpeed_4"]]
  processed_features = selected_features.copy()
  return processed_features
```

Annex B Prediction program

```
# Start of the code importing the libraries
from __future__ import absolute_import, division, print_function
import math
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn import metrics
from tensorflow.python.data import Dataset


# Function to define the inputs. Different depending on the model and turbine
def preprocess_features(wind_farm_dataframe):
  selected_features = wind_farm_dataframe[
    ["WSpeed_1"]]
  processed_features = selected_features.copy()
  return processed_features

# Function to define the target of the neural network. "Powe_1" changes depending on the
modeled wind turbine
def preprocess_targets(wind_farm_dataframe):
  selected_targets = wind_farm_dataframe[
    ["Power_1"]]
  output_targets = selected_targets.copy()
  return output_targets

# Function used to construct the columns used by the program with the data
def construct_feature_columns(input_features):
  return set([tf.feature_column.numeric_column(my_feature) for my_feature in
input_features])

# Function in charge to provide the data to the prediction function
def my_input_fn(features, targets, batch_size=120, shuffle=False, num_epochs=10):
  features = {key: np.array(value) for key, value in dict(features).items()}
  ds = Dataset.from_tensor_slices((features, targets))
  ds = ds.batch(batch_size).repeat(num_epochs)
  if shuffle:
    ds = ds.shuffle(10000)
  features, labels = ds.make_one_shot_iterator().get_next()
  return features, labels


# Function to make the predictions. Changes depending on the modeled wind turbine
def make_predictions(data_examples, data_targets):
  # Input data into columns
  def predict_data_input_fn(): return my_input_fn(data_examples, data_targets["Power_1"],
shuffle=False, num_epochs=1)

  # Definitions of some parameters of the network. They have to be the same than the built
model
```

```
    print("Computing predictions")
    data_rmse = []
    my_optimizer = tf.train.AdamOptimizer(learning_rate=0.003)
    dnn_regressor =
tf.estimator.DNNRegressor(feature_columns=construct_feature_columns(training_examples),
hidden_units=[10,10], optimizer=my_optimizer, model_dir='Program_1_T1')  # model_dir
specifies the names of the built neural network to use. Changes with every model and turbine
    # Computes the predictions for the data
    data_predictions = dnn_regressor.predict(input_fn=predict_data_input_fn)
    data_predictions = np.array([item['predictions'][0] for item in data_predictions])

    # Calculate and print the data root mean squared error
    print("RMSE on data data:")
    data_root_mean_squared_error = math.sqrt(
        metrics.mean_squared_error(data_predictions, data_targets))
    print(data_root_mean_squared_error)

    # Calculate and print the data mean absolute error
    print("MAE on data data:")
    data_mean_absolute_error = metrics.mean_absolute_error(data_predictions, data_targets)
    print(data_mean_absolute_error)

    # Calculate and print the data normalized root mean squared error
    print("NRMSE on data:")
    dt_max = data_targets.max()
    dt_min = data_targets.min()
    dt_range = dt_max - dt_min
    nrmse = data_root_mean_squared_error / dt_range
    print("%.4f" % nrmse)
    nothing = 0

    # Save the predictions in a csv file. Changes for every wind turbine and model
    np.savetxt("E1_T1_predictions.csv", data_predictions, fmt='%i', delimiter=".")

    return dnn_regressor, data_rmse, nothing


# First executed lines on the code. Lecture of the data file
wind_farm_dataframe = pd.read_csv('4T_data_z.csv', sep=";")

# Division into training (0 data points) and prediction points (all data)
training_dataframe = wind_farm_dataframe.head(0)
data_dataframe = wind_farm_dataframe.tail(2980)

# Definition of the training data input variables and targets, calling the preprocess function
training_examples = preprocess_features(training_dataframe)
training_targets = preprocess_targets(training_dataframe)

# Definition of the validation data input variables and targets, calling the preprocess function
data_examples = preprocess_features(data_dataframe)
data_targets = preprocess_targets(data_dataframe)
```

```
make_predictions(
   data_examples,
   data_targets,)
# End of the program
```

The same way it happened with the neural network building program, the function preprocess_targets changes depending on the experiment and wind turbine. The code shown above corresponds to the first experiment and first wind turbine.

Second experiment third wind turbine:

```
def preprocess_targets(wind_farm_dataframe):  # Function to define the target of the neural network, changes depending the model
   selected_features = wind_farm_dataframe[
      ["WSpeed_3"]]
   processed_features = selected_features.copy()
   # Create a synthetic feature. Only turivines 1 and 3 are close to turbine 2
   processed_features["WW23"] = (
         wind_farm_dataframe["WSpeed_2"] *
         wind_farm_dataframe["T2_T3"])
   processed_features["WW43"] = (
         wind_farm_dataframe["WSpeed_4"] *
         wind_farm_dataframe["T4_T3"])
   return processed_features
```

Third experiment third wind turbine:

```
def preprocess_features(wind_farm_dataframe): # Function to define the target of the neural network, changes depending the model
   selected_features = wind_farm_dataframe[
      ["WSpeed_3"]]
   processed_features = selected_features.copy()
   # Create a synthetic feature. All the turbines are considered
   processed_features["WW13"] = (
         wind_farm_dataframe["WSpeed_1"] *
         wind_farm_dataframe["T1_T3"])
   processed_features["WW23"] = (
         wind_farm_dataframe["WSpeed_2"] *
         wind_farm_dataframe["T2_T3"])
   processed_features["WW43"] = (
         wind_farm_dataframe["WSpeed_4"] *
         wind_farm_dataframe["T4_T3"])
   return processed_features
```

Fourth experiment any wind turbine:

```
def preprocess_features(wind_farm_dataframe): # Function to define the target of the neural
network, changes depending the model
   selected_features = wind_farm_dataframe[
      ["Direction",
       "WSpeed_1",
       "WSpeed_2",
       "WSpeed_3",
       "WSpeed_4"]]
   processed_features = selected_features.copy()
   return processed_features
```

Annex C Example of weights and bias

The variables of the artificial neural network built for experiment three on turbine two are presented in this section.

Hidden layer 0 biases (10):

[-2.2629993 0.0 -2.3334885 -2.1038103 -2.3230484 -1.5351306 5.3352494 -1.9405178 -1.613827 -1.9217118]

Hidden layer 0 weights (40):

[1.5577463 -0.5589123 1.2662009 1.4083465 1.6274635 1.3164632 -0.5147033 2.0170853 1.4034286 1.4338775] [0.10441468 -0.21803194 -0.1592556 0.10318922 0.70210016 0.52267194 -0.14389883 0.48518068 -0.22579837 -0.06501166] [-0.3916983 -0.60563266 -0.4928379 0.45800066 -0.14909334 0.33005875 0.516707 -0.376141 0.53988814 0.2710582] [0.06120335 0.18874496 -0.44205 0.65646493 -0.12867501 0.8255637 -0.34948352 -0.27778846 0.0477577 -0.29761925]

Hidden layer 1 biases (10):

[-1.7156622 0. 0. -1.2858016 -1.4235102 -0.08341152 -0.15444517 -1.5596482 -1.5470536 -0.13749859]

Hidden layer 1 weights (100):

[1.02061856e+00 -5.42722940e-01 -4.12156314e-01 8.65389824e-01 1.43676686e+00 -3.90585005e-01 -6.18425965e-01 1.59634686e+00 9.21121001e-01 -2.31957704e-01] [1.28054380e-01 -1.37600303e-01 3.32804561e-01 2.01771498e-01 7.06251264e-02 -3.15542102e-01 3.69530559e-01 -2.19855100e-01 -2.60879219e-01 4.43746150e-01] [1.40822208e+00 -4.19518173e-01 -1.81456327e-01 7.83065915e-01 1.17729330e+00 -4.96812791e-01 1.53294697e-01 6.34682178e-01 1.14038789e+00 2.80405313e-01] [1.70690370e+00 4.00354266e-02 -4.82270807e-01 1.42761171e+00 7.64858663e-01 -3.95639926e-01 1.78135887e-01 1.37728548e+00 1.71924436e+00 1.43486083e-01] [1.22178602e+00 -4.73049402e-01 3.20699692e-01 1.56479418e+00 7.42276192e-01 -8.17710906e-02 1.14480406e-01 1.13044453e+00 1.34552848e+00 3.40805292e-01] [1.49686325e+00 -5.19893110e-01 -3.31989974e-01 1.64493823e+00 1.70003986e+00 1.39515817e-01 -1.36673048e-01 1.06868792e+00 1.02461112e+00 -3.43844771e-01] [-5.78605461e+00 1.39009714e-01 -1.48350120e-01 -5.82846165e+00 -6.01868296e+00 -4.21262681e-01 2.99277276e-01 -5.51626205e+00 -5.14218473e+00 8.90611708e-02] [1.71358025e+00 -3.57750058e-02 -1.99437797e-01 6.33343935e-01 1.41652727e+00 2.04169840e-01 1.48590684e-01 1.25519311e+00 1.36558366e+00 -2.32458100e-01] [1.90962303e+00 -3.58574033e-01 -1.80594325e-01 1.25331187e+00 1.22728205e+00 -3.51428092e-01 -3.56823415e-01 2.18730259e+00 2.13092041e+00 1.61754116e-01] [

1.77002585e+00 -5.39994240e-03 -3.57782006e-01 1.36180329e+00 1.90675390e+00 2.14951783e-02 2.78257817e-01 1.76161599e+00 1.25287867e+00 -5.53097665e-01]

Output layer bias (1):

[-0.2766465]

Output layer weights (10):

[1.9593222] [0.6672378] [0.26278406] [2.3600223] [1.917514] [0.264897] [-0.47745773] [1.8691419] [2.6341004] [-0.2943368]

Total number of parameters: 171

Annex D Data cleaning program

Data cleaning program, developed in VBA for Microsoft Excel:

```
Sub cleaning()

'Declare excel files
Dim wb_cur As Workbook              'Declaration of the file which is being used
Dim sh_cur As Worksheet             'Declaration of the working sheet

'Declaration of variables
Dim P1 As Double
Dim P2 As Double
Dim P3 As Double
Dim P4 As Double
Dim P5 As Double
Dim Erased As Double
Dim initial_data As Double
Dim i As Integer
Dim cont As Integer

Set wb_cur = ActiveWorkbook  'Assignation of wb_cur to the current open file
Set sh_cur = wb_cur.Sheets(1)  'The same with the shit (number)

''''''''''''''''''''''''''''''''Body of the program''''''''''''''''''''''''''''''''
initial_data = sh_cur.Range(Cells(3, 1), Cells(3, 1).End(xlDown)).Rows.Count + 2 'Counting the
number of data
Erased = 0                                  'Initiation of variables
cont = 1
i = 3
While cont < 7823                           'Cleaning loop
        P1 = sh_cur.Cells(i, 3)             'Powers lecture
        P2 = sh_cur.Cells(i, 5)
        P3 = sh_cur.Cells(i, 7)
        P4 = sh_cur.Cells(i, 9)
        If P1 <= 100 Or P2 <= 100 Or P3 <= 100 Or P4 <= 100 Then 'Condition to clean the row
                Cells(i, 1).Select
                Range(Selection, Selection.End(xlToRight)).Select   'Cleaning the row
                Selection.Delete Shift:=xlUp
                Erased = Erased + 1
                i = i - 1
        End If
        i = i + 1
        cont = cont + 1
Wend                                        'End of the cleaning loop
End Sub                                     'End of the program
```

Annex E Matrix program

Code used to build the matrix for the experiments 2 and 3, developed in VBA for Microsoft Excel:

```
Sub matrix()

'Declare excel files
Dim wb_cur As Workbook              'Declaration of the file which is being used
Dim sh_cur As Worksheet             'Declaration of the working sheet

'Declaration of variables
Dim initial_data As Double
Dim i As Integer

'Defineixo arxius excel
Set wb_cur = ActiveWorkbook         'Assignation of wb_cur to the current open file
Set sh_cur = wb_cur.Sheets(1)       'The same with the shit (number)

''''''''''''''''''''''''''''''''''''''''Body of the program''''''''''''''''''''''''''''''''''''''''

initial_data = sh_cur.Range(Cells(2, 1), Cells(2, 1).End(xlDown)).Rows.Count 'Counting the
number of data
i = 2
While i < initial_data + 2          'Loop used to build the matrix
        'T1_T2
        If sh_cur.Cells(i, 1) >= 5 And sh_cur.Cells(i, 1) <= 35 Then
                sh_cur.Cells(i, 10) = 1
        Else: sh_cur.Cells(i, 10) = 0
        End If
        'T2_T3
        If sh_cur.Cells(i, 1) >= 17 And sh_cur.Cells(i, 1) <= 47 Then
                sh_cur.Cells(i, 11) = 1
        Else: sh_cur.Cells(i, 11) = 0
        End If
        'T3_T4
        If sh_cur.Cells(i, 1) >= 26 And sh_cur.Cells(i, 1) <= 56 Then
                sh_cur.Cells(i, 12) = 1
        Else: sh_cur.Cells(i, 12) = 0
        End If
        'T4_T3
        If sh_cur.Cells(i, 1) >= 206 And sh_cur.Cells(i, 1) <= 236 Then
                sh_cur.Cells(i, 13) = 1
        Else: sh_cur.Cells(i, 13) = 0
        End If
        'T3-T2
        If sh_cur.Cells(i, 1) >= 197 And sh_cur.Cells(i, 1) <= 227 Then
                sh_cur.Cells(i, 14) = 1
        Else: sh_cur.Cells(i, 14) = 0
        End If
        'T2_T1
```

```
        If sh_cur.Cells(i, 1) >= 185 And sh_cur.Cells(i, 1) <= 215 Then
                sh_cur.Cells(i, 15) = 1
        Else: sh_cur.Cells(i, 15) = 0
        End If
        'T1-T3
        If sh_cur.Cells(i, 1) >= 10 And sh_cur.Cells(i, 1) <= 40 Then
                sh_cur.Cells(i, 16) = 1
        Else: sh_cur.Cells(i, 16) = 0
        End If
        'T1-T4
        If sh_cur.Cells(i, 1) >= 15 And sh_cur.Cells(i, 1) <= 45 Then
                sh_cur.Cells(i, 17) = 1
        Else: sh_cur.Cells(i, 17) = 0
        End If
        'T2-T4
        If sh_cur.Cells(i, 1) >= 22 And sh_cur.Cells(i, 1) <= 52 Then
                sh_cur.Cells(i, 18) = 1
        Else: sh_cur.Cells(i, 18) = 0
        End If
        'T4-T2
        If sh_cur.Cells(i, 1) >= 202 And sh_cur.Cells(i, 1) <= 232 Then
                sh_cur.Cells(i, 19) = 1
        Else: sh_cur.Cells(i, 19) = 0
        End If
        'T4-T1
        If sh_cur.Cells(i, 1) >= 195 And sh_cur.Cells(i, 1) <= 225 Then
                sh_cur.Cells(i, 20) = 1
        Else: sh_cur.Cells(i, 20) = 0
        End If
        'T3-T1
        If sh_cur.Cells(i, 1) >= 190 And sh_cur.Cells(i, 1) <= 220 Then
                sh_cur.Cells(i, 21) = 1
        Else: sh_cur.Cells(i, 21) = 0
        End If

        i = i + 1

Wend                    'End of the matrix loop
End Sub                 'End of the program
```