uc3m | Universidad **Carlos III** de Madrid

Master in Statistics for Data Science
2024-2025

Master Thesis

"Optimizing Wind Turbine Placement in Wind Parks via Mixed Integer Optimization using Neural Network based Constraint Learning."

Simon Schmetz

Carlos Ruiz Mora
2nd Tutor complete name

Madrid the 29. of January

Acknowledgments

TODO Acknowledgments

ABSTRACT

This thesis combines Linear Optimization with Constraint Learning to optimize wind turbine placement for maximum performance in a predefined area under randomly distributed wind. A Neural Network is trained on simulated data to model the impact of turbine positioning on power output. This model is integrated as a constraint in a linear optimization problem, and the problem evaluated for a current state-of-the-art wind farm configuration.

CONTENTS

1 INTRODUCTION

With the clean energy transition currently taking place in europe with ambitious targets for 2030 and beyond [7] , wind energy is playing a central role in that transition, with wind energy expected rise to 50 % in the EU energy mix. [4] With wind energy thus expected to become the main contributer to the EU's energy production and large potentials identified for both onshore and offshore parks [1] attempts to optimize all parameters of windparks with even minor power efficeny improvement can be expected to yield significant returns in absolute power due to the scale of future wind energy production.

As a contribution to increasing power efficeny on future wind farms, this thesis is dedicated to a new approach for optimizing the placement of a fixed number of wind turbines in a predefined area (typically a square). To solve this optimization problem, a extension to the pyomo python library is used, that allows the introduction of Neural Networks to the optimization problem as constraints. [3] This extension allows for introducing a Neural Neural Network to model the effects of wind turbine placement relative to each other on power production for the respective windturbines. Introducing this model to the optimization problem defined in pyomo then allows for the optimization of overall power productions across all wind turbines in the wind park. The optimization problem in its simplest form can be defined as

$$\max_{\mathbf{x},\mathbf{y}} \sum_{i=1}^{n} f_{Power,\mathrm{NN}}(x_i, y_i; \mathbf{x}, \mathbf{y}) \tag{1}$$

$$\text{s.t.} \quad X_{\min} \leq x_i \leq X_{\max}, \quad \forall i \in \{1, \dots, n\} \tag{2}$$

$$Y_{\min} \leq y_i \leq Y_{\max}, \quad \forall i \in \{1, \dots, n\} \tag{3}$$

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq d_{\min}, \quad \forall i \neq j \tag{4}$$

where:

- (x_i, y_i) are the coordinates of turbine i out of n total turbines,
- $f_{\mathrm{NN}}(x_i, y_i; \mathbf{x}, \mathbf{y})$ is a neural network approximating the power output for each turbine i ,
- $X_{\min}, X_{\max}, Y_{\min}, Y_{\max}$ define the boundaries of the rectangular placement
- $d_{\min}$ is the minimum distance between any two turbines.

To create a model optimally fit to the needs of the optimization problem, the model is trained on data specifically generated with the FLORIS wind farm simulation tool for optimal coverage of the parameter space of the optimization.

To simplify the problem, the surface below the turbines is assumed to be perfectly flat and equal wind speed is assumed along the entire hight of the turbines.

Data Generation and Neural Network:

(...)

Optimization Problem:

(...)

This thesis is structued according to the two main steps required to solve the optimization problem as presented above.

2 STATE OF THE ART

TODO: Run Perplextiy Research
nrel optimization

3 FARM POWER MODEL

The first central component to the optimization of the wind farm layout is to generate a data drive surrogate Model that can be introduced into the optimization problem and be solved by a solver. As detailed in the introduction, more specifically the aim is to use the distCL extension to the pyomo python package, requiring a small Neural Network as surrogate model. The following chapter documents the steps taken to generate such a model. To train such a Neural Network, data is required that covers the parameter space of the optimization to prevent extrapolation by the model. Therefore, the chapter start by explaining how the open source wind farm simulation tool FLORIS® was used to generate a dataset and what the simulations behind this dataset are. Then, the fundamentals of Neural Network arcitecture and training are briefly introduced, before the model of the interactions for two turbines is trained and evaluated.

3.1 DATA SOURCE

The power output of wundturbines and wind farms as a whole, is fundamentally connected to the aerodynamic conditions in the airflow every wind turbine experiences, with wind speed as the biggest factor relevant to how much power a wind turbine can generate [SOURCE] . The main effect that reduces the power generated by a wind turbine, is be positionend in the wake downwind of another turbine, with the reduction in windspeed in the wake together with the increased turbolence in the wake airflow, being the two maincontributors to reduced power output (see 1) [11]. Moving downstream of a wind turbine, the wake gradually mixes with the outer airflow and thus again increases windspeed until the entire airflow reaches a new homogenious air speed [13]. Thus, even if a wind turbine is positionend in the wake of another wind turbine, the greater the distance between the two, the less the second wind turbine is affected, while ideally wind turbines are not positioned in the wake of other wind turbines at all.

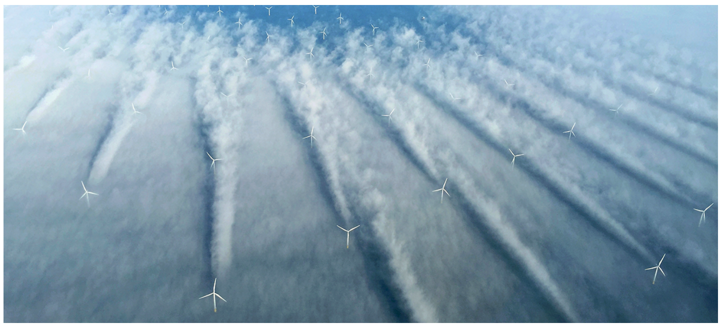

Figure 1: Visible wind turbine wakes due to contrail generation [19]

With the overall goal of creating a model that models how the interaction between wind turbines (induced by the wake) is related to the wind turbines relative positioning to each other and the wind conditions as input to a optimization model, a tailor made dataset from simulation results is the easiest to handle to ensure that the data fits the use case well. Using self made simulation data, allows to set the parameter space of the model as required for the different variants of the optimization problems explored in the second part of this thesis. After investigating two python based open source wind farm simulation tools FLORIS (FLORIS official website) and PyWake (PyWake Documentation), FLORIS was chosen after a brief comparison between the two due to an apparent ease of use. The FLORIS is a wind farm simulation tool developed by the National Renewable Energy Laboratory (NREL) as one of multiple tools containing models of varying complexity available for different use cases. FLORIS is a control-systems oriented toolbox and therefore contains a lower complexity model yielding fast simulation results. This allows the automatic generation of large quanteties of data in a relatively short time. As the data is coming from simulation results, the data is deterministic even though there is an underlying uncertainty attributed which can be quantified by FLORIS if required.

For this thesis, depending on the optimization problem, FLORIS is given a grid of the following parameters:

- Position of Wind Turbines
- Wind Direction
- Wind Speed
- Wind turbolence

defining the parameter space limits with evenly spaced data points to then generate the corresponding simulation data, with the simulation outputs being the generated power by the individual wind turbines as well as by the farm as a whole (e.g. the sum of individual power outputs). The configuation of all of these parameters as well as the number of wind turbines can be modified at will in this set up and the result is a dataset with the rows corresponding to a specific combination of parameters and the resulting power outputs of the turbines. As reference turbine, an IEA 10-MW was used due to its repeated use as baseline turbine in other scientific works like [12] and [10] , a more exact technical specification can be found in [6]. The configurations of the specific dataset generated for the individual optimization problem formulations are documented in the corresponding Sections of Chapter 4.

3.2 MODELLING

To model the relationship between the attributes of a incoming airflow to a wind turbine and the output generated by the same wind turbine many surrogate models could be chosen. As the model generated in this for this thesis is created with the goal of introducing it into the pyomo extension referenced in Chapter 1, the model has to be compatible with said extension. That is why the model chosen is a simple neural network with limited number of hidden layers and nodes, with the size of the Neural Network being limited for said extension.

3.2.1 *Introduction into Neural Networks*

In the following section, a biref introduction is given into how Neural Networks work, are trained including some regularization teqniques.

Arcitecture of a Neural Network

Fundemanetally, Neural Networks represents Graph Networks consisting out of Function blocks as the nodes, in the case of Neural Networks called perceptron (or Neurons as more general terms) and arcs which correspond to the inputs/outputs of a given perceptron. In simple words, the inputs to the Neurons are summed up and introduced as argument into a function $f()$ which yields a output y, representing the output of the given perceptron. These Neurons are organized in layers, which correspond to the row type structure Neural Networks are usually represented in and each perceptron of one layer is connected to every other perceptron of the following layer. The following layer in this case means in the direction of flow, in visualizations usually from left to right. The arcs thus in simple terms correspond to a single number and the Neurons to the action of summing up all inputs and then applying the unspecified function $f()$ to generate a output. This process is repeated for all Neurons in the network, with the first layer of the network taking as input the values of the values of the given data features (the input values to the model) and the last layer producing outputs corresponding to the output value(s) of the model. The number of Neurons in this first and last layer correspond to the task the Neural Network is supposed to perform. If the goal is to identify handwritten numbers 1-9 from 50x50 pixel images, the imput layer might have 50x50 Neurons for the value of each pixel, and the output layer 9 layers with the output value of each perceptron representing how much the networks thinks the given image shows the corresponding numbers 1-9. A schematic of this architecture is given in Figure 2.

As shown in Figure 3 the output of a Neuron is slighlty more complicated as described befoe, as the output y is generated by summing up the inputs x multiplied by a corresponding weight w together with a bias b and introducing this summation as argument into a activation function $f()$. In this process, the weights w represent a weigh to give importance to the individual inputs and the bias b serves to set a minimum output value that will always be reached, regardless of the inputs.

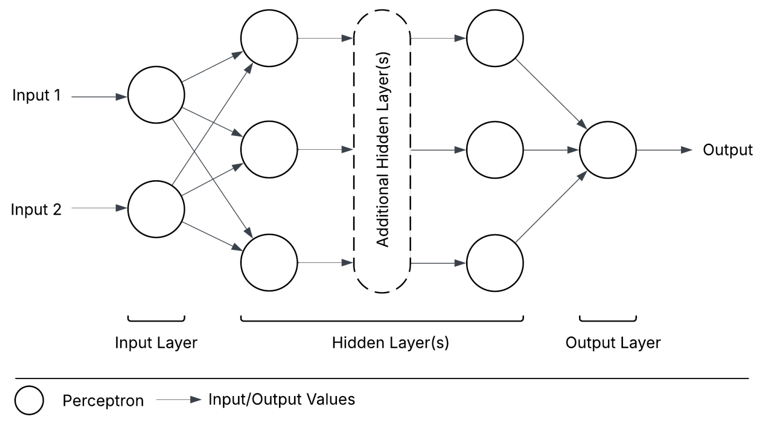

Figure 2: Schematic of Neural Network Architecture

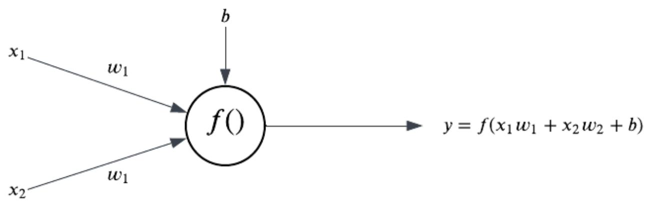

Figure 3: The output of a neuron is generated by applying the activation function to the sum as the weighted inputs $w_i x_i$ and the bias of the neuron b

The activation function $f()$ is called activation function, as in its most simplest form it represents a step function that decides if a neuron activates or not, e.g. takes the binary values $0, 1$ for a given threshhold. Contrary to the human brain where neurons are indeed binary, most Neural Networks resort to a activation function whose outputs are not binary but deliver continious values between 0 and 1 to avoid the boundary issues that occour with binary threshholds. The most common function used instead of a step function is the sigmoid function, which roughly corresponds to a continuous version of the step function, with $\sigma(x) \approx 1$ for $x \to \infty$ and $\sigma(x) \approx 0$ for $x \to -\infty$.

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

This relationship also becomes aparent when plotting both of those functions over each other, as shown in 4

TODO: ReLU Function in text and plot

[15]

Training a Neural Network

With the general structure set up, and assuming that a layout has been chosen for the neural network (e.g. number of layers and number of their coirresponding neurons), the training of the neural network corresponds to adjusting the weights w and the biases b of each neuron

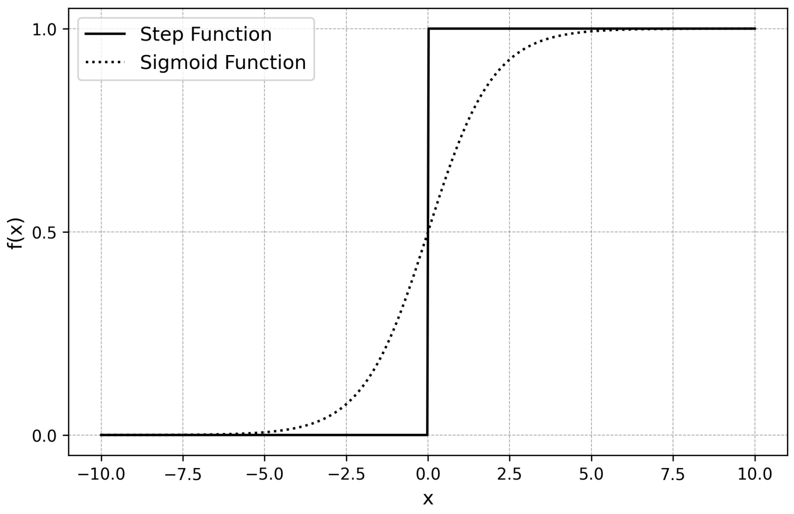

Figure 4: The Sigmoid Function being a close to the Step Function without being as sensetive to slight changes in x due to its continuity

in a way, that allows the model to perform the task it is given well. What is means for the model to perform well is defined by a *Loss Function*, that defines a relationship between the output of the model and the correct output (defined by training data) and gives a Loss as output, which is some sort of delta between model prediction and truth. As what it means for a model to perform well heavily depends on the task (regression, binary classification, multiclass classification etc.), but regardless of the task, the goal is always to minimize the Loss Function. A well known Loss Function in regression is the Mean Squared Error (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The training of the Neural Network thus corresponds to adjusting the weights and biases in a way that minimizes the chosen loss function. The algorithm most commonly used for this is called *Backpropagation*. [15]

Before diving into how the algorithms works exactly, it makes sense to first define specifically what parameters have to be optimized, e.g. all weights and biases of the network to be optimized. A common notation of the individual weights is as w^l_{jk} with l being the layer of nodes into which w^l_{jk} are the weights of its inputs (weighing the values from the outputs of the $(l-1)^{th}$ layer) and j as the neuron from the l^{th} layer as well as k the neuron of the $(l-1)^{th}$. Similarly, the biases are written as b^l_j with l as the layer and j the neuron in that layer as as can be seen for both the weights and bias in an example in Figure 5.

Using this notation, the output of a given neuron a^l_j can be calculated as

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right),$$

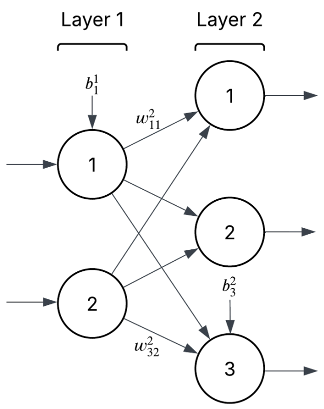

Figure 5: Notation of Neural Network weights and biases with weight w_{jk}^l and l being the input layer, j the neuron from that input layer and k the neuron of the $(l-1)^{th}$ output layer. Similarly, the bias as b_j^l with l as the layer and j the neuron.

this expression can be further simplified,by writing weights and biases in matrix form, using for each layer, a weight matrix w^l for all weights input into the given layer l and a bias vector b^l. By doing so, the output vector a^l of a the entire layer l can be calculated as.

$$a^l = \sigma(w^l a^{l-1} + b^l)$$

With the notation for weights and biases established, the goal becomes tuning them in such a way, to minimize a chosen Loss function as discussed above. This is done by using training data with a known correct output and changing weights and Biases in such a way, that moves the Neural Network towards outputting the true known value. The algorithm to do so essentially corresponds to:

1. Introduce Training Data into the Neural Network

2. Evaluate the Loss Function for the Training Points (with small random values for weights and biases for the first iteration) and evaluate the gradients of the Loss function for the gradients of the individual weights and biases ($\frac{\Delta Loss}{\Delta w_{jk}^l}$, $\frac{\Delta Loss}{\Delta b_j^l}$) using backpropagation [1]

3. Update the weights and biases according to a learning rate ρ weighted by the specific weight or biases gradient like $w_{jk}^l = w_{jk}^l - \rho \frac{\delta Loss}{\Delta w_{jk}^l}$ and $b_j^l = b_j^l - \rho \frac{\Delta Loss}{\Delta b_j^l}$

4. Repeat process n Epochs, with one Epoch being the algorithm passing through all training points/batches of points once

This algorithm represents a gradient descent algorithm with learning rate ρ corresponding to the step length, while evaluating the par-

[1] The Gradient can either be evaluated for each training point individually or for batches of training points

tial derivatives $\frac{\Delta Loss}{\Delta w^l_{jk}}$, $\frac{\Delta Loss}{\Delta b^l_j}$ is the critical step. Here the packpropagation algorithm gives an efficient method of finding these partial derivatives by evaluating the individual error contributions of each Neuron and how errors accumulate as the inputs move through the Network. A more extensive explonation of how backpropagation works can be found in [16]. As common for gradient descent, finding the global optimum for the weights and biases is not guaranteed and even improbable if many local minima exist as they do for the many parameters of a Neural Network. None the less, using gardient descent it can be hoped to find a good local minimum. [9] [16]

Regularization

Like for most machine learning models, Neural Networks run at risk of overfitting. Neural Networks are especially prone to overfitting due to their many degrees of freedoms represented by the many biases and weights, potentially exceeding the number of training samples.

To prevent overfitting, regularization tecniques like Cross-Validation can be applied in training like for any other model. One very specific regularization method for Neural Networks is *Dropout Learning*, which corresponds to randomly removing Neurons from the network (by effectively setting their activation function and thus their output to 0) for each training observation. By other nodes having to "stand-in" for the dropped-out nodes, nodes are prevented from developing overspecialization. [9]

Notes - many Degrees of Freedom, many local maxima - Adaptive Moment Estimation (ADAM) optimization

Estimating Distributions using Neural Networks

Neural Networks can also be used to estimate distributions of a random variable. Two possible approaches are to so called Quantile Neural Networks and Distributional Neural Networks. While their Architecture is that of a normal Neural Network as discussed in the previous section, they differentiate in the output layer, where the Quantile Neural Network has outputs that correspond to the probability for a specific quantile while the distributional Neural Network outputs the distribution, like in Figure 6 for a normal distribution.

The different outputs from both neural Networks require appropriate Loss functions. For Quantile Neural Networks, the Pinnball Loss (Quantile loss, for more see [17]) used while in the case of Distributional Neural Networks, the Likelihood can be used as commonly used in classical statistic for parameter estimation.

[2] [14]

Distributional Neural Networks Quantile Neural Networks

Loss for Parametric Normal: Gaussian Negative Log Likelihood

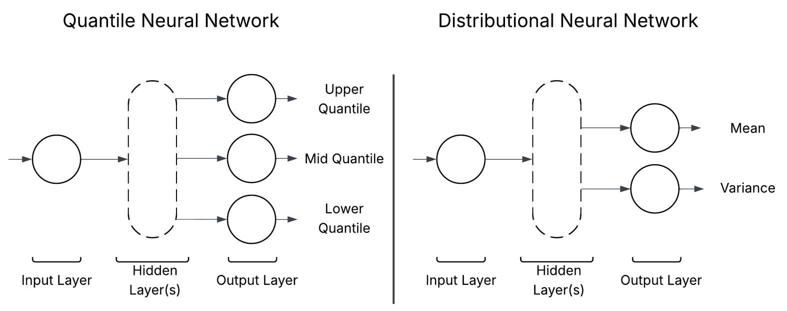

Figure 6: Neural Networks to estimate a distribution by estimating specific quantiles (left) or the parameters of a distribution (right)

3.3 GENERATING NEURAL NETWORK MODEL FOR TWO WINDTURBINES

Generate Data

Train Model

Validate Model

4 OPTIMIZATION

4.1 OPTIMIZATION UNDER UNCERTAINTY

While some things in life are certain, most are not and this is as true for optimization as for anything else. That means, that while optimizing anything from next years crop to tomorrows energy pricing might be done by assuming fixed values for the parameters of a optimization problem, in reality most of these parameters will contain uncertainty. The field of Stochasitc Programming contains is occupied with finding methods that allow for introducing these uncertainties into optimization problems.

One way to approach these uncertainties is to split the problem in scenarios. A bakery for example has to decide on how many Baguettes to bake for the next day with the objective of maximizing its profit. Baking too little Baguettes will lead to missing out on potential sales while baking too many baguettes will mean that the demand is fullfilled but the money invested in the excess number of baguettes is lost. Assuming the mean number of baguettes bought every day is 1000, the price to buy a baguette is 1 and the cost to produce an baguette is $0,2$ the classical approach to solving such a problem would be by the following formulation [1]:

$$\max_{x} \quad (1.00 \cdot \min(x, 1000) - 0.20 \cdot x)$$

To introduce the two scenarios additional scenarios of the demand being 10% lower (900 Baguettes) and 10% higher (1100 Baguettes) can be added. Assuming that the mean demand having a 50% probability of occouring and the 10% demand increase a 20% probability and the decrease a 10% probability, the problem can be modified to maxmize the expected profit across these three scenarios by the formulation

$$\begin{aligned} \max_{x} \quad & 0.5 \cdot (1.00 \cdot \min(x, 1000) - 0.20 \cdot x) \\ & + 0.3 \cdot (1.00 \cdot \min(x, 900) - 0.20 \cdot x) \\ & + 0.2 \cdot (1.00 \cdot \min(x, 1100) - 0.20 \cdot x) \end{aligned}$$

The result from this optimization problem would be the Expected profit and how many Baguettes are baked the optimal number of Baguettes to yield the maximum Expected profit across all scenarios. This would be optimal assuming there exists no more information about the next days demand, meaning that by using this approach the total profit over a long time would be maximal. In case there

1 x of course non negative

is more information regarding the next days demand the probabilities that give the weights in this optimization might shift, with one scenario potentially reaching probability 100% if there were to be absolute certainty that the next days demand would be for example 1100 baguettes. As having such exact information is however very rare, the best solution will be in most cases to maximize the profit Expectation.

The obvious connection to conventional statistical analysis is that the demand is a random variable that can take multiple values, in this case we assumed it to be a discrete random variable Y with support $900, 1000, 1100$ even though in real world applications the demand of baguettes will move somewhere between $[0, \infty]$. Finding the Expectation for such a discrete random variable can be done as

$$\mathbb{E}[X] = \sum_i x_i \cdot \mathbb{P}(X = x_i) = \sum_i x_i p_i$$

or for continious random variables

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x \cdot f_X(x)\, dx$$

Using these two expressions, optimization problems can thus be formulated to optimize the Expectation of objective functions containing random variables. [5]

4.2 CONSTRAINT LEARNING

Constraint learning refers to introducing a model that has learned relationships between certain variables from data into a optimization problem. In the case of constraint learning, the model gets more specifically introduced into a optimization problem as part of a constraint. As many real real life relationships struggle to be represented by explicit function to be defined as objective function or constraint, introducing machine learning models to optimization problems opens up many new possibilities [8].

In the case of Neural Networks, one way of introducing a Network as constraints into a optimization problem, is by recognizing that when using the Rectifier Linear Unit (ReLu) Function is used as activation function with the (linear) sum of neuron bias and weighted inputs $\tilde{v}_i^\ell$ being the function argument

$$v_i^\ell = \max(0, \tilde{v}_i^\ell) = \max(0, b_i^\ell + \sum_j w_{ij}^\ell v_j^{\ell-1}) \tag{5}$$

the function can be rewritten as the following constraints

$$v_i^\ell \geq \tilde{v}_i^\ell \tag{6}$$
$$v_i^\ell \leq \tilde{v}_i^\ell - M^{\text{low}}(1 - j_i) \tag{7}$$
$$v_i^\ell \leq M^{\text{up}} j_i \tag{8}$$

with $j_i \in \{0, 1\}$ a integer variable such that

$$j_i = \begin{cases} 0 & \text{if } \tilde{v}_i^\ell < 0 \\ 1 & \text{if } \tilde{v}_i^\ell > 0 \end{cases} \tag{9}$$

This decomposition allows to for introducing a Neural Network of limited size into a optimization problem by decomposing it into a set of linear constraints of the form shown above. [3]

In practice, this decomposition
https://www.sciencedirect.com/science/article/pii/S0957417423013970?via
https://www.sciencedirect.com/science/article/pii/S0377221724005186

4.3 THE TWO TURBINE PROBLEM

Optimizing the positioning of two wind turbines can be expressed as optimizing the relative position of a second wind turbine T_2 to a fixed first turbine T_2, defined by the relative distances Δx and Δy. Both Δx and Δy are constrained by minimum distance Δ_{min} to T_2 and maximum distances $\Delta x_{max}/\Delta y_{max}$ to make the problem bounded. This problem can be visualized as shown in Figure 7.

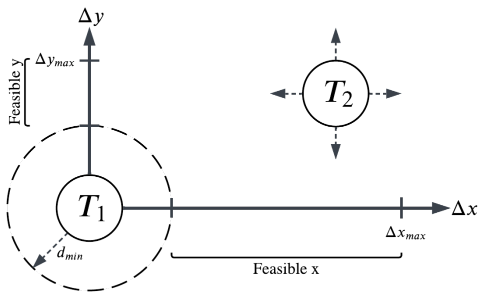

Figure 7: Optimizing the relative position $\Delta x/\Delta y$ of a second wind turbine T_2 to a fixed first turbine T_2, constrained by minimum distance d_{min} to T_1 and maximum distances $\Delta x_{max}/\Delta y_{max}$

The objective function to be optimized is the total power generation, e.g. the sum of power generated by both turbines. This objective is a function both of the position of the wind turbine as well as of the wind conditions like wind direction and wind speed.

$$f_{totalPower}(x, y, \text{windspeed}, \text{wind direction}, (...))$$

Differently from the geographic coordinates, th wind condition parameters like windspeed are inherently not deterministic and follow distributions like the normal distribution as shown in Figure 8.

The two main challenges of the optimization of the shown two turbine problem are thus:

1. Introduce the complex relationship between turbine position and wind conditions and power output into the optimization problem

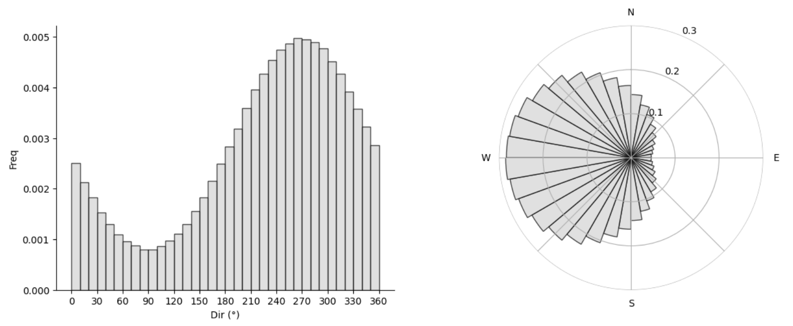

Figure 8: Histogram and Polar Plot across radiants for a normally distributed wind direction probability density function with mean West

2. Introduce the non-deterministic nature of wind conditions into the optimization problem

The first of these challenges is tackled by applying the Neural Network models discussed in Section 3.2 and introduce them into the optimization problem via constraint learning as described in Section 4.2. For the second problem, multiple approaches are now explored in the following subsection.

4.3.1 *Deterministic Optimization : Optimization for main Wind direction and Speed*

To begin with solving the problem, the most simplest approach is taken by assuming the wind conditions to be discrete. In application, this might be analouge to getting the Expectation of the joined probability distribution all wind condition parameters. Taking these parameters as constant and homogenious across the entire parameter space, the result is a objective function that is effectively only dependent on the relative positions of the turbine with the previously discussed geometrical constraints.

$$\max_{\mathbf{x},\mathbf{y}} f_{Power,\text{NN}}(\Delta x, \Delta y) \tag{10}$$

$$\text{s.t.} \quad \Delta x \leq X_{\max} \tag{11}$$

$$\Delta y \leq Y_{\max} \tag{12}$$

$$\sqrt{(\Delta x)^2 + (\Delta y)^2} \geq d_{\min} \tag{13}$$

where:

- $(\Delta x, \Delta y)$ are the relative distances of the two turbines,
- $f_{Power,\text{NN}}(\Delta x, \Delta y)$ is a neural network (deterministic) approximating the total power output ,
- $X_{\max}, Y_{\max}$ define the maximal distance the two turbines can be placed apart
- $d_{\min}$ is the minimum distance between the two turbines.

When solving this problem as defined using Gurobi, a major challange becomes apparent as the result shows that there is an infinite amount of equally optimal solutions (e.g. degeneracy of the problem, see [18] for exact definition of degeneracy) everywhere outside the wake of turbine 1. This phenomenon is very intuitive when generating a heatmap of the power generation for all possible positions of turbine 2, as shown in Figure **??**

TODO : Insert Figure

This phenomenom whill have to be kept in mind going forwards, while introducing a distribution for wind condition parameters might reduce the number of global optima.

4.3.2 *Stochastic Optimization: Power weighted by probability*

As the deterministic optimization does not take into accound the distribution of wind condition parameters and having found that the deterministic approach leaves a range of infinite global optima outside of the wake of turbine 1, the next step is proceeding with the introduction of uncertainty in the form of probability distributions of wind condition parameters into the problem. This step primarily changes the objective function, which is set up such that the neural network takes in both position and wind condition parameters. the output of this neural network is then weighed by the probability of the specific wind condition occouring and the Expected power is calculated by integrating (discretized as sum) across all combinations of wind condition parameters.

$$\max_{\mathbf{x},\mathbf{y}} \sum_{i=1}^{n} f_{Power,\mathrm{NN}}(\Delta x, \Delta y, \text{wind condition}) \cdot p_{n,\text{wind condtion combination}} \tag{14}$$

$$\text{s.t.} \quad \Delta x \leq X_{\max} \tag{15}$$

$$\Delta y \leq Y_{\max} \tag{16}$$

$$\sqrt{(\Delta x)^2 + (\Delta y)^2} \geq d_{\min} \tag{17}$$

where:

- $(\Delta x, \Delta y)$ are the relative distances of the two turbines,
- $f_{Power,\mathrm{NN}}(\Delta x, \Delta y)$ is a deterministic neural network approximating the total power output for turbine positions and wind conditions
- $X_{\max}, Y_{\max}$ define the maximal distance the two turbines can be placed apart
- $d_{\min}$ is the minimum distance between the two turbines
- n is index of the discretized possible combinations of wind conditions

  In practice, this approach come with challenges as it requires the constraints representing the neural network to be realized n

times for each wind condition combination. With one instance of the neural network already adds a significant amount of constraints to the optimization model, realizing those constraints *n* times may lead to an overflow of constraints, making the model to dificult to solve.

4.3.3 *Stochastic Optimization: Power function as Quantile Neural Network*

An alternative approach to the one shown in the previous section, is to implant the uncertainties of the wind condition into the power model, with the output as quantiles from a quantile neural network.

4.4 THE 3 TURBINE PROBLEM

4.5 THE N TURBINE PROBLEM

Thoughts:

IDEA: generalize inputs in a way that allow for power calculation of each turbine individually instead of farm power

IDEA : derive basic rules from 2 turbine problem and set up new optimization problem without NN

IDEA : analogue to forward/backward selection with 2 turbine neural network

FINIDINGS:

problem degenerate if not one wind turbine fixed

5 CONCLUSION

Once a proof of concept for the approach taken in this thesis has ben successfully performed, the data source may be eventually switched for experimental or real world data.

6 ADDENDA

7 ABBREVIATIONS

MSE	Mean Squared Error
ReLu	Rectifier Linear Unit
IEA	International Energy Agency
MW	Mega Watt

LIST OF FIGURES

LIST OF TABLES

BIBLIOGRAPHY

[1] European Environment Agency. Europe's onshore and offshore wind energy potential. Technical Report Technical report 6/2009, European Environment Agency, 2009. Accessed: 2025-04-03.

[2] Inimfon Akpabio and Serap Savari. On the construction of distribution-free prediction intervals for an image regression problem in semiconductor manufacturing. 03 2022.

[3] Antonio Alcántara and Carlos Ruiz. A neural network-based distributional constraint learning methodology for mixed-integer stochastic optimization. *Expert Systems with Applications*, 232:120895, 2023.

[4] Analysis and General Secretariat of the Council of the European Union Research Team. Harnessing wind power: Navigating the eu energy transition and its challenges. https://www.consilium.europa.eu/media/1kyk0wjm/2024_685_art_windpower_web.pdf, September 2024. Accessed: 2025-04-03.

[5] John R. Birge and François Louveaux. *Introduction to Stochastic Programming*. Springer Series in Operations Research and Financial Engineering. Springer, July 1997.

[6] Pietro Bortolotti, Helena Canet Tarrés, Katherine Dykes, Karl Merz, Latha Sethuraman, David Verelst, and Frederik Zahle. Iea wind task 37 on systems engineering in wind energy – wp2.1 reference wind turbines. Technical report, National Renewable Energy Laboratory, 2019.

[7] European Commission. Renewable energy targets. https://energy.ec.europa.eu/topics/renewable-energy/renewable-energy-directive-targets-and-rules/renewable-energy-targets_en, 2023. Accessed: 2025-04-03.

[8] Adejuyigbe O. Fajemisin, Donato Maragno, and Dick den Hertog. Optimization with constraint learning: A framework and survey. *European Journal of Operational Research*, 314(1):1–14, 2024.

[9] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, and Jonathan Taylor. *An Introduction to Statistical Learning: with Applications in Python*. Springer Texts in Statistics. Springer, 2023.

[10] Samuel Kainz, Julian Quick, Mauricio Souza de Alencar, Sebastian Sanchez Perez Moreno, Katherine Dykes, Christopher Bay, Michiel Zaaijer, and Pietro Bortolotti. Iea wind tcp task 55: The iea wind 740-10-mw reference offshore wind plants. Technical Report NREL/TP-5000-87923, National Renewable Energy Laboratory, 2024. Technical Report.

[11] C.T. Kiranoudis and Z.B. Maroulis. Effective short-cut modelling of wind park efficiency. *Renewable Energy*, 11(4):439–457, 1997.

[12] Mads Madsen, Frederik Zahle, Sergio Gonzalez Horcas, Thanasis Barlas, and Niels Sørensen. Cfd-based curved tip shape design for wind turbine blades. *Wind Energy Science*, 7:1471–1501, 07 2022.

[13] M. Magnusson and A.-S. Smedman. Air flow behind wind turbines. *Journal of Wind Engineering and Industrial Aerodynamics*, 80(1):169–189, 1999.

[14] Grzegorz Marcjasz, Michał Narajewski, Rafał Weron, and Florian Ziel. Distributional neural networks for electricity price forecasting. *Energy Economics*, 125:106843, September 2023.

[15] Michael Nielsen. Neural networks and deep learning - chapter 1, 2015. Accessed: 2025-04-10.

[16] Michael Nielsen. Neural networks and deep learning - chapter 2: How the backpropagation algorithm works, 2015. Accessed: 2025-05-01.

[17] Ingo Steinwart and Andreas Christmann. Estimating conditional quantiles with the help of the pinball loss. *Bernoulli*, 17(1), February 2011.

[18] Robert J. Vanderbei. *Linear Programming*, chapter 3. Springer, New York, NY, 5 edition, 2020.

[19] Windpower Monthly. Offshore wind clusters to lower energy production – study. *Windpower Monthly*, 2019. Accessed: 2025-04-29.