University of Oslo

# Machine Learning for Wind Energy Forecasting

IN5410 – Assignment 2 Report

Neco Darian
Nikita Fedotovs
Victor Vaco Darian
Zélia Eduardo Sitoe

Supervisor: *Sabita Poudyal*

May 5, 2025

# Contents


**1 Introduction** **3**
1.1 Practical context . . . 3
1.2 Data foundation . . . 3
1.3 Objectives and tasks . . . 3

**2 Methodology** **3**
2.1 Time series and Forecasting . . . 3
2.2 Seasonal Decomposition and Stationarity Test . . . 4
2.3 Autocorrelation and Partial Autocorrelation . . . 4
2.4 Feature Engineering . . . 4
2.5 Feature Scaling . . . 5
2.6 Data Reshaping . . . 5
2.7 Algorithms . . . 5
2.7.1 Linear Regression (LR) . . . 5
2.7.2 Multiple Linear Regression (MLR) . . . 6
2.7.3 $k$-Nearest Neighbors (KNN) . . . 6
2.7.4 Support Vector Regression (SVR) . . . 7
2.8 Neural Networks Architecture . . . 7
2.9 Activation Function . . . 8
2.10 Optimization . . . 8
2.11 Regularization . . . 8

**3 Data Analysis** **10**
3.1 Variable Overview . . . 10
3.2 Frequency Distribution . . . 11
3.3 Feature engineering . . . 14
3.4 Power: target variable visualization . . . 14
3.4.1 Decomposition of Time series data for power . . . 17
3.4.2 Testing for Stationary . . . 17
3.4.3 Augmented Dickey-Fuller (ADF) test results . . . 17
3.4.4 KPSS Test Results for Stationarity . . . 18
3.4.5 ACF and PACF Graphs . . . 19
3.4.6 The weekly rolling mean of the wind Power . . . 21
3.5 Plotting U10, V10 and WS10 over time . . . 21
3.6 Visualising relationship between power, time and wind speed (WS10) . . . 22
3.7 Correlation Matrix . . . 23

**4 Task 1: Wind Power Forecasting Using Machine Learning** **24**
4.1 RMSE and SMAPE Comparison . . . 25
4.2 Forecast Plots . . . 25
4.3 $R^2$ Score Evaluation . . . 27

**5 Task 2: Multiple Linear Regression with Wind Speed & Direction** **28**

**6 Task 3: Wind Power Forecast Using Times Series** **31**
6.1 RMSE and SMAPE Comparison . . . 31
6.2 Forecast Plots . . . 32

**7 Task 4: Mathematically calculating a 2-2-1 Neural Network** **33**
7.1 Forward Propagation . . . 33
7.2 Backward Propagation . . . 34
7.3 Gradient and weight update for $\boldsymbol{w_6}$. . . . 35
7.4 Gradient and weight update for $\boldsymbol{w_5}$. . . . 36
7.5 Gradient for $\boldsymbol{w_4, w_3, w_2, w_1}$. . . . 37
7.6 Gradient and update for $\boldsymbol{w_4}$. . . . 37
7.7 Gradient and update for $\boldsymbol{w_3}$. . . . 38
7.8 Gradient and update for $\boldsymbol{w_2}$. . . . 39
7.9 Gradient and update for $\boldsymbol{w_1}$. . . . 40
7.10 Forward & backward propagation calculations . . . 41
7.11 Error Comparison Between Two Rounds . . . 43
7.12 Possible improvements to the Network . . . 43
7.13 Python implementation of forward and backward propagation . . . 44

# 1 Introduction

## 1.1 Practical context

Accurate day-to-day forecasts of wind-farm output underpin everything from hour-ahead grid balancing to long-term investment planning. Under-estimating generation forces operators to buy costly balancing power, while over-estimates can jeopardise system stability. Mastering the link between weather signals and delivered megawatts is therefore a matter of both economics and reliability.

## 1.2 Data foundation

The study is built on two years of hourly observations from an anonymised coastal wind farm in Australia (1.January 2012–31.October 2013). Each record couples measured, capacity-normalised power with European Centre for Medium-Range Weather Forecasts (ECMWF) wind predictions at 10m and 100m above ground, expressed through zonal(U) and meridional(V) components and the derived wind speed (WS). Forecast skill is assessed on an unseen test set covering 1–30 November 2013, for which only the meteorological outlook is provided in advance.

## 1.3 Objectives and tasks

The assignment specifies four progressively more demanding forecasting problems:

- **Wind-speed-only modelling** – learn the relationship between 10m wind speed and power using four algorithms (linear regression, k-nearest neighbours, support-vector regression and an artificial neural network) and compare their RMSE on November 2013.
- **Incorporating wind direction** – extend the baseline by adding direction (computed from U10 and V10) and evaluate the benefit of a multiple linear regression relative to speed-only regression.
- **Pure time-series forecasting** – simulate the loss of meteorological input by training linear regression, SVR, ANN and RNN models on historical power alone and benchmarking their performance on the same horizon.
- **Weight adaptation in a neural network** – demonstrate, through step-by-step forward and back propagation, how gradient descent reduces error in a simple two-input, single-hidden-layer network.

Together, these tasks test how progressively richer information and model sophistication translate into forecast accuracy.

# 2 Methodology

## 2.1 Time series and Forecasting

The definition of time series or temporal data is a sequence of observations that are collected over time intervals. Time series is used to capture changes over time. These changes can for example occur hourly, daily or annually. Time series data has become a valuable commodity in the age of data, helping governments and businesses to find valuable insights for better decision-making. Time series can either be discrete or continuous and can be divided into components which are described more in detail in the 'seasonal decomposition' section.

Forecasting, is a technique that uses historical data to predict what happens in the future. This decision-making tool help governments and businesses to deal with uncertainty of the future by analyzing the trends and past observations. Forecasting methods can be divided between qualitative and quantitative method. Qualitative method relies on an expert's personal decision and provides subjective results, while quantitative method is more of a mathematical process that provide more consistent and objective results.

## 2.2 Seasonal Decomposition and Stationarity Test

Seasonal decomposition can be described as a statistical method used to break a time series into different components:

- **Seasonality**: This component refers to the short-term cyclical patterns observed within a specific time.
- **Residual**: This component represents random variation in the time series after removing the previous components.
- **Trend**: The general direction of the time series either increases or decreases over time.

Two methods can be used to analyze the seasonality of the time series: Additive and multiplicative. In additive methods, which are used in this case, the variance of data does not change over different values in the series. The model is linear, and the trend line is straight. The seasonality has the same frequency and amplitude.

Stationarity refers to a dataset where the statistical properties (mean, variance, and autocorrelation) of the time series do not change over time and will remain constant. Stationarity is often described as the foundational concept in time series analysis. Common statistical tests for stationarity are Augmented Dickey-Fuller (ADF) and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests. The ADF is a unit root test that determines how great the time series can be defined by a trend. The null hypothesis is that there is a unit root in an autoregressive model with r = 1, time dependent, and that the process is therefore not stationary. The KPSS test follows the opposite direction by assuming stationarity around the mean or linear trend, while the alternative is that there is a unit root in the series, therefore not stationary.

Rolling statistics is another way to determine whether the time series is stationary. Time series are considered stationary if rolling mean or standard deviation remains constant with time. By plotting, one can see if lines are parallel to the x-axis.

## 2.3 Autocorrelation and Partial Autocorrelation

Correlation, in statistics refers to any statistical relationship between two variables. Correlation is used to measure the degree to which two variables are linearly related. For example, a power production company can decide how much electricity they want to produce based on the relationship between electricity demand and the weather. Pearson's correlation coefficient is one of the most used statistical methods for finding the correlation between variables. In this method, the correlation coefficient is a value between -1 and 1, with zero indicating no relationship.

In simple terms, autocorrelation is the presence of correlation connected to one or multiple lagged versions of a time series, which means that the past is connected to the future. The autocorrelation function (AFC) can be effectively used to identify the lags with significant correlations, understanding the properties and different patterns of the time series. In contrast, partial autocorrelation measures the relationship between an observation in a time series with data from earlier time steps with no correlation between the intervening observations. The partial autocorrelation function (PAFC) displays the relationship between two observations where the lags between them can not be explained. This function is mostly used to specify regression models with time series.

## 2.4 Feature Engineering

Feature Engineering can be described as the focus on creating additional feature based on the existing variables in the dataset. The aim is to better represent the underlying structure of the data. Additional features can highlight the trends in the data and also improve the overall performance of the model. Combining multiple variables that can result in being more informative or decomposition into two or more variables that each give better understanding and in-depth insight are the methods which can be applied.

For this project, we have created new features from the timestamp column to find patterns based on time. Features include hour, day, week, month, and year. Also to better understand repeating patterns, sine and cosine transformation are applied to hour, month, and day of the week. This allow the model to recognize the cyclical values.

## 2.5 Feature Scaling

Feature scaling is a technique used during the preprocessing stages, to enable values of variables in the dataset into a close and similar scale. This technique is necessary when we are working with datasets that contain variables with different measurements, orders or ranges. Feature scaling ensures that all variables can contribute equally to the model to avoid extreme effect of features that have large values, which can lead to biased performance of the model or other challenges during the training process.

In this case, StandardScaler is applied. The transformer centres each numerical feature at zero and scales it to unit variance by subtracting the training-set mean and dividing by its standard deviation. This technique preserves the relationship between values, and at the same time making sure all the features contribute equally to our model. This technique works effectively in algorithms that are sensitive to feature magnitude.

## 2.6 Data Reshaping

Neural Networks are designed to work with sequential data, and every unit in these networks require a specific input shape to process the data correctly. To match this specific shape, the input data needs to be reshaped. The networks are unable to correctly process the data and can cause incorrect results if the input data is not reshaped according to the specific architecture. This process ensures that the data can work seamlessly with the layers during the training and testing stages and increase the models performance. We have reshaped the selected training and testing features to a 2 and 3-dimensional format, based on the different models.

## 2.7 Algorithms

### 2.7.1 Linear Regression (LR)

Linear regression is a simple and widely used algorithm for modeling the relationship between a dependent variable (target) and an independent variable (predictor) by fitting a linear equation to observed data.

The linear model can be expressed as

$$y = \beta_0 + \beta_1 x + \epsilon,$$

where

- $y$ is the dependent variable,
- $x$ is the independent variable,
- $\beta_0$ is the intercept (bias term),
- $\beta_1$ is the slope (coefficient for $x$),
- $\epsilon$ is the error term (random noise).

The goal is to estimate $\beta_0$ and $\beta_1$ by minimizing the Ordinary Least Squares (OLS) loss function:

$$\min_{\beta_0,\,\beta_1} \sum_{i=1}^{n} \big(y_i - (\beta_0 + \beta_1 x_i)\big)^2.$$

**Key Characteristics**

- Assumes a linear relationship between the variables.
- Very simple and interpretable.
- Sensitive to outliers.

### 2.7.2 Multiple Linear Regression (MLR)

Multiple linear regression extends LR to handle multiple independent variables for predicting a dependent variable.

**Mathematical Formulation**

$$y \;=\; \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon,$$

where

- $x_1, x_2, \ldots, x_p$ are the predictor variables,
- $\beta_1, \beta_2, \ldots, \beta_p$ are the coefficients.

The optimization problem becomes

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \Big( y_i - \big[\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}\big] \Big)^2.$$

**Key Characteristics**

- Can model multi-factor relationships.
- Requires no multicollinearity among predictors.

### 2.7.3 $k$-Nearest Neighbors (KNN)

KNN is a non-parametric, instance-based learning algorithm that predicts values based on the local average of nearby data points.

**Mathematical Formulation**

For a new observation $x$, the algorithm

1. finds the $K$ closest training points (neighbors),
2. predicts $\hat{y}$ as the mean of their target values:

$$\hat{y} \;=\; \frac{1}{K} \sum_{i=1}^{K} y_i.$$

**Distance Metrics:** Euclidean distance (default):

$$d(x, x_i) = \sum_{j=1}^{p} \big(x_j - x_{ij}\big)^2.$$

**Key Characteristics**

- No assumptions about data distribution.
- Hyperparameters: $K$ (number of neighbors) and the distance metric (Euclidean, Manhattan, etc.).
- Computationally expensive for large datasets.

### 2.7.4 Support Vector Regression (SVR)

SVR is a regression adaptation of Support Vector Machines (SVM), aiming to fit the best possible hyperplane while allowing a tolerance margin ($\epsilon$-insensitive tube).

**Mathematical Formulation**

SVR solves

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\big(\xi_i + \xi_i^*\big)$$

subject to

$$\begin{cases} y_i - (w^\top \varphi(x_i) + b) \le \epsilon + \xi_i, \\ (w^\top \varphi(x_i) + b) - y_i \le \epsilon + \xi_i^*, \qquad\qquad i = 1, \dots, n, \\ \xi_i, \xi_i^* \ge 0, \end{cases}$$

where

- $w$ is the weight vector,
- $\varphi(x_i)$ maps $x_i$ to a higher-dimensional space (kernel trick),
- $C$ controls the trade-off between margin width and prediction error,
- $\xi_i, \xi_i^*$ are slack variables for deviations outside the $\epsilon$-tube.

**Kernel Functions**

- **Linear:** $\varphi(x_i) = x_i$,
- **RBF (Gaussian):** $\varphi(x_i, x_j) = \exp\big(-\gamma\|x_i - x_j\|^2\big)$.

**Key Characteristics**

- Effective for non-linear relationships.
- Robust to outliers due to the $\epsilon$-insensitive loss.
- Requires tuning of $C$, $\epsilon$, and kernel parameters.

## 2.8 Neural Networks Architecture

A Neural network can in simple term be describes as interconnection between artificial neurons that are able to solve complex problems. The concept stems from the structure and functions of the neurons in the human brain. It follows the same pattern of a human brain to capture different relationship between variables in a dataset.
The neural networks have three layers: **Input, hidden, and Output layer**. The input layer consists of nodes that represent the data variables. The hidden layers receive the input from the input layer or from the previous hidden layer, in case of multiple hidden layers. Every node in the hidden layer gets input from all nodes in the previous layer. The complex computations on the input data happen in these layers by

applying a weighted sum, adding a bias term, and then send the results through an activation function. The output layer displays the final result of all the data processed by the network. The number of nodes in this layer depends on the nature of the task. For regression tasks, the output has only one node, while in binary classification, there can be one node for every class.

A perceptron which is an artificial neuron of a network, consists of inputs, weights, bias and activation function. Every connection between two neurons has a weight to it and that represents how much impact an input has on the network. Bias term is used in the network to adjust the output with weighted sum of the input. Both weights and bias terms are learned duting the training.

## 2.9 Activation Function

The activation function is a function that is added to the neural network to help the network learn complex and deep patterns in the data. It happens when the nodes take in the output signal from the previous node and convert it into a form that can be accepted as input by the next node. Every node performs an activation function to the weighted sum of input and the bias term. The process where the output from the activation function to the next hidden layer is called forward propagation. A network's output after forward propagation and comparing that to the actual value is calculated using a loss function. Mean squared error (MSE) is a common loss function for regression tasks.

There are two types of activation functions: Linear and Non-Linear. In neural networks, the activation function's purpose is to bring non-linearity to the network to model complex relationships. The functions are divided based on the range and their curves.

The Sigmoid function, which is mostly used for binary classification, accepts a number as input and returns a number between 0 and 1 as output. The rise of vanishing gradients makes the Sigmoid function problematic when they are used in the hidden layers.

TanH function, known as hyperbolic tangent, is very similar to sigmoid function. The difference is in the output, where its range is from -1 to 1. Compared to the sigmoid function, the gradient of the tanh function is steeper and faces the same vanishing gradient problem. One advantage of tanh function is it being zero centered where the gradients are free to move in different direction.

The Rectified Linear unit (ReLU) activation function, one of the most popular used activation functions in deep learning, stands out from other activation functions by not activating all the neurons at the same time. This makes the relu function more efficient than tanh and sigmoid activation function, but the negative side is that some of the weights and biasis for neuron in the neural network do not get updated or activated ('dying ReLU' problem) during backpropagation, creating unusable neuron.

## 2.10 Optimization

Optimization algorithms are commonly used during training the networks and when modifying the parameters to minimize error. These algorithms help the models to reach better performance and accuracy. Adam (Adaptive Moment Estimation) optimizer is praised for its efficiency and adaptability. As an adaptive learning rate algorithm, the Adam optimizer adjusts the learning rate for every parameter dynamically based on its gradient history, improves training speed, and reaches convergence faster. Some of the benefits of using this optimizer are minimum hyperparameter tuning, robustness at handling situations where the features occur infrequently(sparse gradient), speed and efficiency, and the single adjustment to learning rates.

## 2.11 Regularization

Regularization is a technique that helps reducing the algorithms error over the test dataset. A great challenge in machine learning and deep learning is to make an algorithm having similar performance on both training and test dataset, and prevent a common problem which is overfitting. Overfitting is a result of a model

capturing different aspects of the data including noise and anomalies in the training data. The performance on the training data is good, but the model are not able to generalize to new data very well, causing overfitting. Generalization helps overfitting by setting constraints on the model during training.
Some of the regularization methods in deep learning are L1 and L2 regularization, early stopping, dropout etc. L2 regularization penalizes large weight and it limits the models capacity to memorize the training data, especially the noise and anomalies. It happens by adding a penalty term to the loss function which forces the weights to decay close to zero. L1 Regularization in the other hand penalizes the absolute value of the weights.

# 3 Data Analysis

We begin by examining the statistical summary of the dataset used for training and testing our wind power forecasting models. The main values that are being observed is 10-meter wind speed [WS10] and power generation [POWER], the data that was provided did not explicitly explain how they measured it therefore there are some assumptions. However looking at the ECMWF parameter database it is possible to conclude that the wind speed is being measured in m/s [**?**].

## 3.1 Variable Overview

The data includes the following variables:

- **TIMESTAMP**: The date and time at which each observation was recorded. The data is sampled at an hourly interval.
- **POWER**: The normalized power output at each time point (wheter it's from a single wind turbine or wind farm is not stated in the dataset). This value ranges from 0 (no power production) to 1 (maximum capacity). It serves as the target variable in our predictive modeling in task 1.
- **U10**: The zonal (west-to-east) component of wind velocity measured at 10 meters above the ground. A positive value indicates wind blowing from the west to the east, while a negative value indicates eastward winds.
- **V10**: The meridional (south-to-north) component of wind velocity at 10 meters height. A positive value represents wind blowing from the south to the north, and a negative value indicates northward winds.
- **WS10**: The total wind speed at 10 meters, presumably calculated using the U10 and V10 components via $\text{WS10} = \sqrt{\text{U10}^2 + \text{V10}^2}$.
- **U100**: The zonal (west-to-east) component of wind velocity at 100 meters above the ground.
- **V100**: The meridional (south-to-north) component of wind velocity at 100 meters height.
- **WS100**: The total wind speed at 100 meters, presumably calculated using the U100 and V100 components via $\text{WS100} = \sqrt{\text{U100}^2 + \text{V100}^2}$.

| **Metric** | **TIMESTAMP** | **POWER** | **U10** | **V10** | **WS10** | **U100** | **V100** | **WS100** |
|---|---|---|---|---|---|---|---|---|
| Count | 16080 | 16080 | 16080 | 16080 | 16080 | 16080 | 16080 | 16080 |
| Mean | — | 0.4317 | 0.8356 | 0.0656 | 4.2043 | 1.2221 | -0.0446 | 6.6983 |
| Min | — | 0.0000 | -8.0930 | -9.9704 | 0.0781 | -11.8815 | -16.5696 | 0.0776 |
| 25% | — | 0.1139 | -2.0172 | -1.9648 | 2.6958 | -3.1631 | -3.3756 | 4.7918 |
| 50% | — | 0.3794 | 0.7136 | 0.3297 | 3.8813 | 1.3001 | 0.5440 | 6.5019 |
| 75% | — | 0.7479 | 3.3137 | 2.1261 | 5.3441 | 5.1877 | 3.6114 | 8.2758 |
| Max | — | 1.0000 | 14.0190 | 8.7096 | 14.4750 | 19.6783 | 12.5256 | 20.4091 |
| Std | — | 0.3366 | 3.5373 | 2.9064 | 1.9962 | 5.4250 | 4.6977 | 2.8503 |

Table 1: Statistical Summary of Dataset Features

From this table, we observe the following:

- `POWER` (The power output) is normalized, ranging from 0 to 1, with a mean of 0.4317 and a standard deviation of 0.3366.
- `WS10` (Wind Speed at 10 meters above ground level) ranges from approximately 0.08 m/s to 14.47 m/s.

- `WS100` (Wind Speed at 100 meters above ground level) ranges from approximately 0.08 m/s to 20.41 m/s.
- `U10` (Zonal Wind Component at 10 meters) ranges from approximately −8.09 m/s to 14.02 m/s.
- `U100` (Zonal Wind Component at 100 meters) ranges from approximately −11.88 m/s to 19.68 m/s.
- `V10` (Meridional Wind Component at 10 meters) ranges from approximately −9.97 m/s to 8.71 m/s.
- `V100` (Meridional Wind Component at 100 meters) ranges from approximately −16.57 m/s to 12.53 m/s.
- The is no missing data as all the variables have an equal amount of count.
- `WS100`, `V100` and `U100` have a higher standard deviations than their counterparts `WS10`, `V10` and `U10`.

## 3.2 Frequency Distribution

Below are plots showcasing the distribution of the parameters.

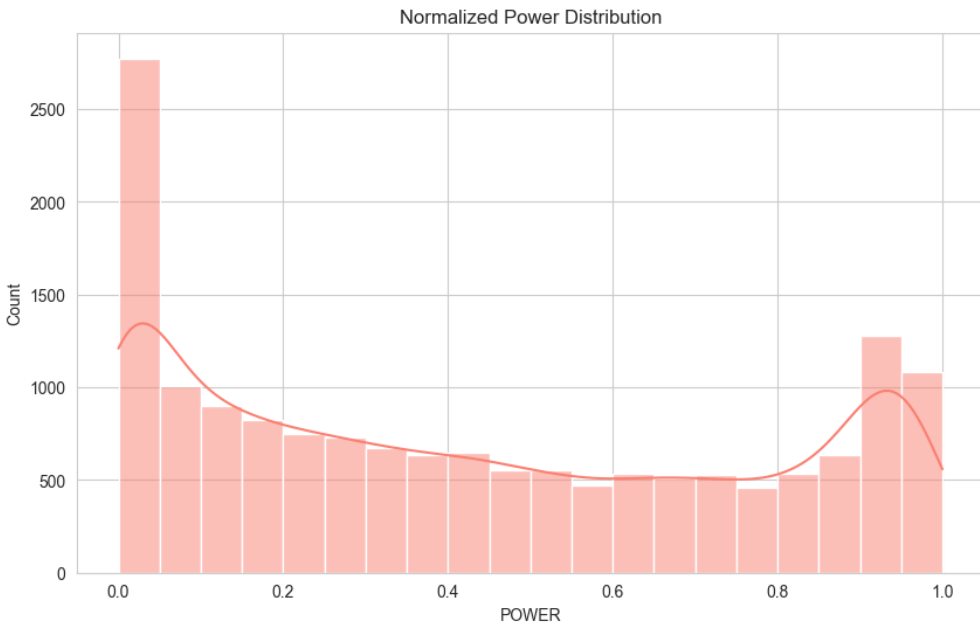


Figure 1: Distribution of Normalized Power Output (POWER)

An interesting observation on the above plots, is that power output distribution spikes on near 0 power and also increases on near 1. Meaning that most of the power that is measures is either 0 or 1 with some slight variation.

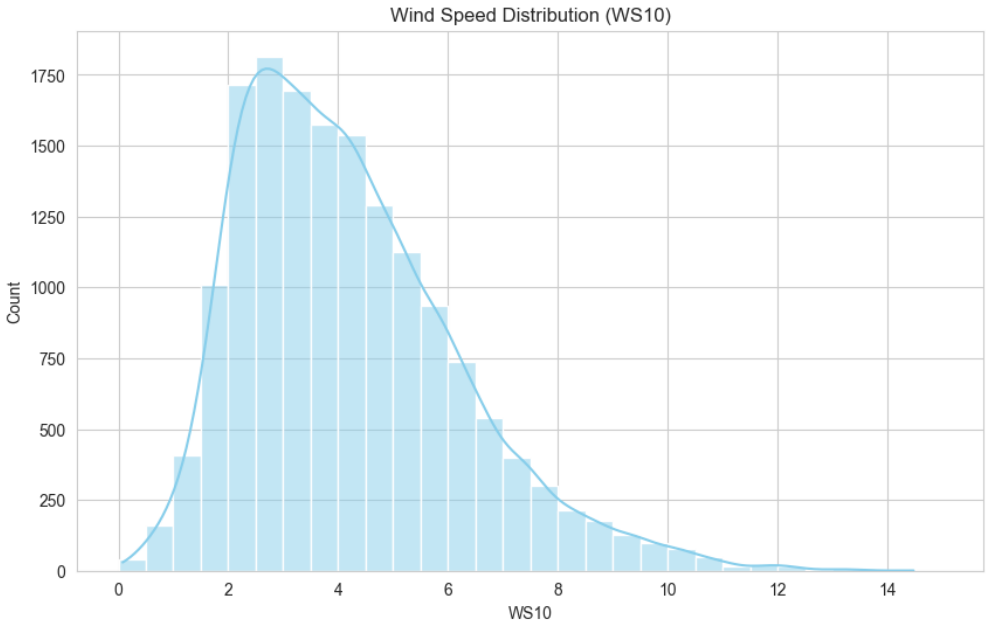


Figure 2: Distribution of Wind Speed at 10m (WS10)

The wind speed distribution looks fairly normal without any anomalies usually the most of the windspeed measured where about 4 m/s

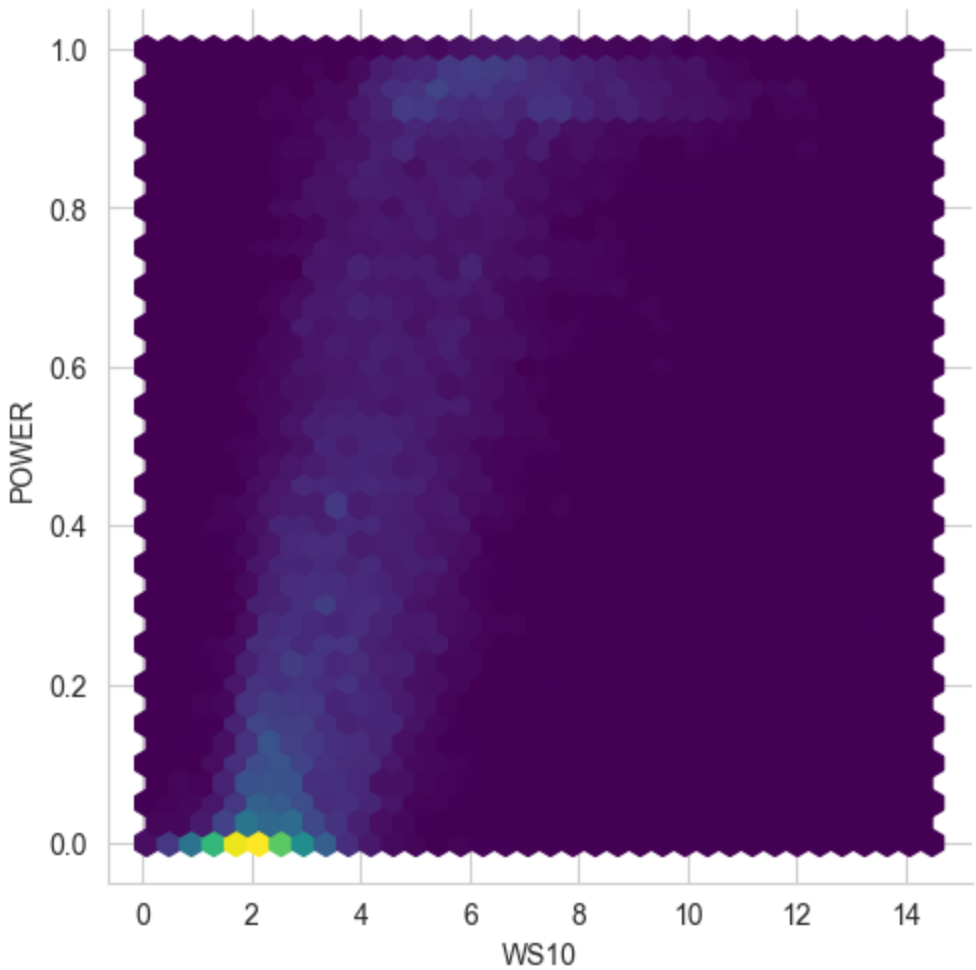


Figure 3: Distribution of power (POWER) in relative to wind speed (WS10)

In the above figure we can see that as wind speed reaches over 6 the power shifts towards 1. The biggest amount of data lies in wind speed 2 and power being around 0. This could impact our prediction models, because majority of the data lies in (0.0 power, 2 WS10). It also means that from all the observed data, the most prominent is being very low power when very low windspeed.

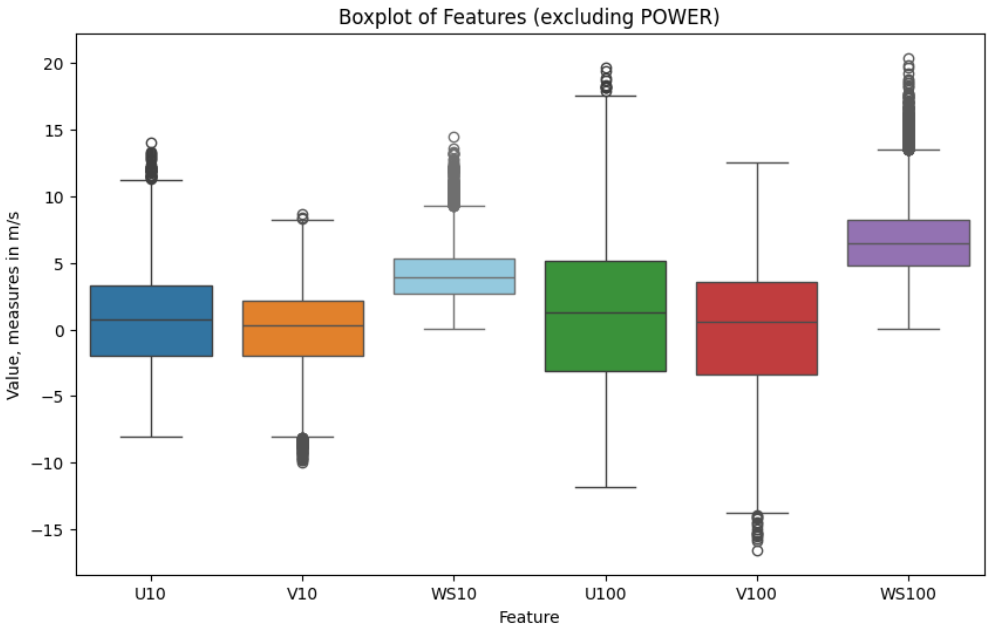


Figure 4: Distribution of each feature, excluding POWER

Since power is using it's own metric and is also normalized, it would be more beneficial to show the feature box-plots that only use m/s. In the figure above you can see that WS10 has a higher mean than U10 and V10, not surprisingly since it's calculated by U10 and V10.

## 3.3 Feature engineering

The `TIMESTAMP` data was extracted into the following features:

- `hour`: The hour of the day (ranging from 0 to 23).
- `day`: The day of the month.
- `day_of_week`: The day of the week (Monday = 0, Sunday = 6).
- `month`: The month of the year (ranging from 1 to 12).
- `year`: The year of the timestamp.
- `week`: The ISO week number.

This made it easier to visualize data based on specific timestamps.

## 3.4 Power: target variable visualization

Since our main target variable is power, we went over different possibilities on how we can visualize the data. Most of it is based off the extracted `TIMESTAMP` data, this leads into us visualizing time-series data for power.

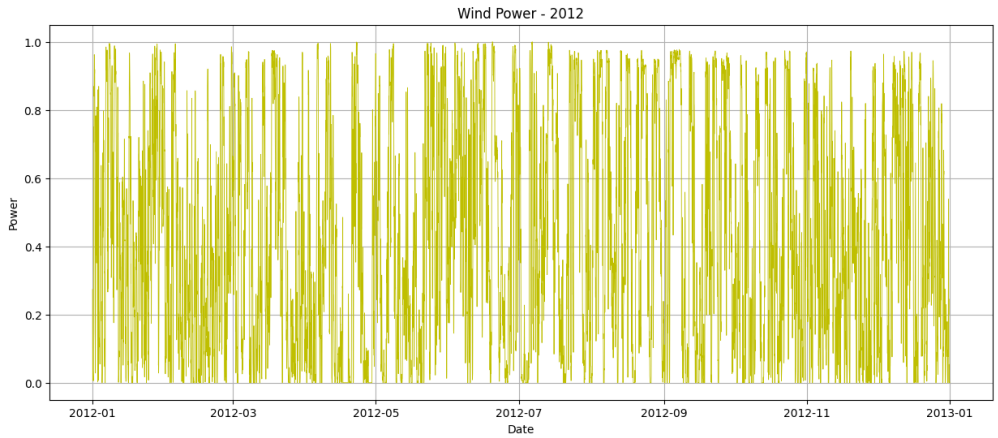


Figure 5: Power visualization - Year 2012

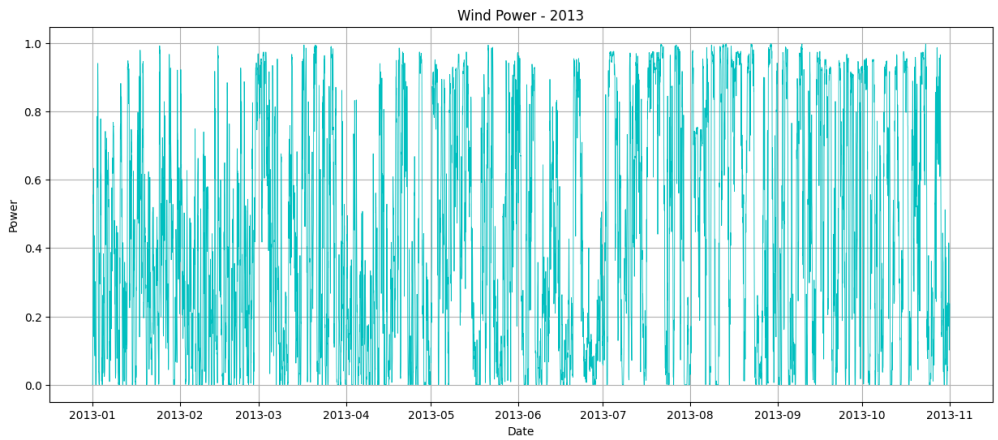


Figure 6: Power visualization - Year 2013

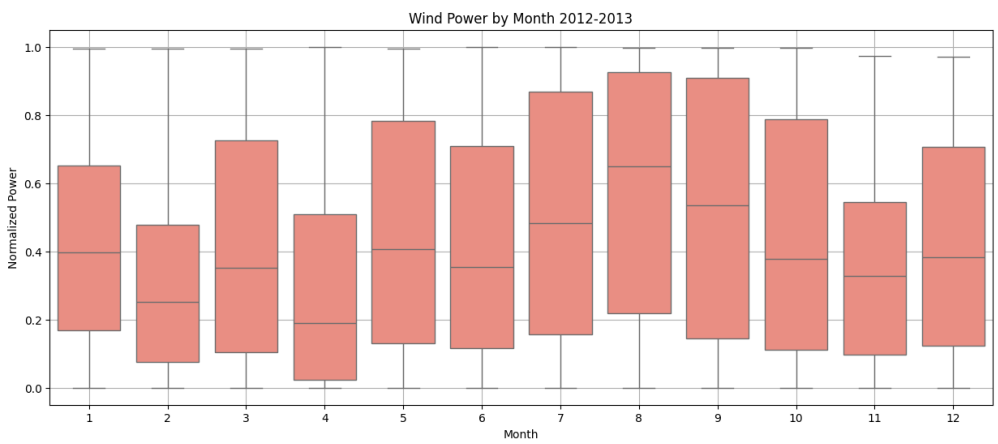


Figure 7: Generated power by month, 2012 - 2013

In the above figure we can see the spread of the power value, in the 4th month (april) we can see that the power had the lowest mean and also the interquartile range had it's lowest minimum amongst all other plots.

On the other hand in the 6th month (June) the mean was the biggest and it had the biggest maximum from it's IQR compared with all the other months.

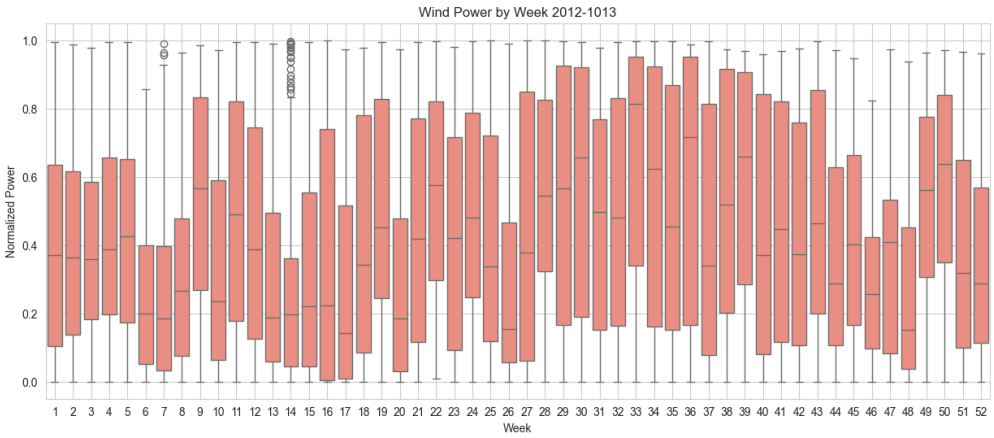


Figure 8: Generated power by week, 2012 - 2013

The above figure shows a very consistent mean and slight variation in the first 5 weeks, then the power drops on the mean and there is more variation between each week. Also on the weeks between 31 and 44 is where the power gets it's highest maximum IQR, meaning that during those weeks the power generation increases on average compared to the previous weeks where it's less or more varied.

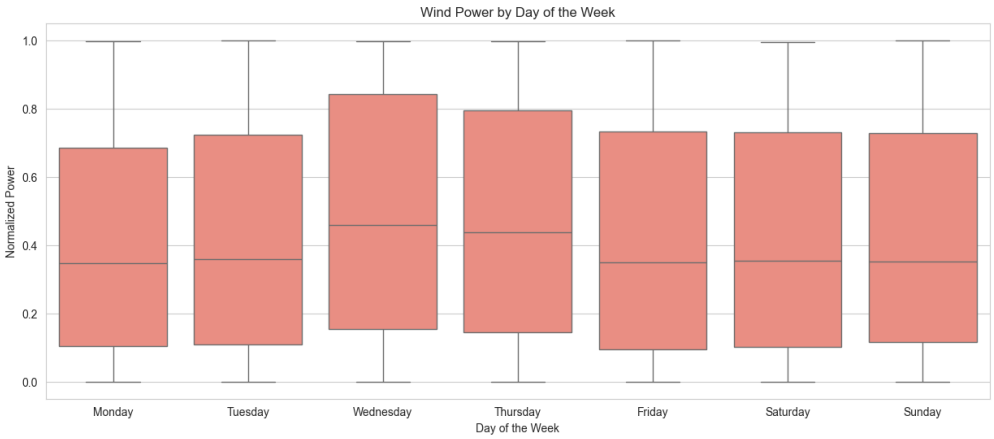


Figure 9: Generated power by day of the week

### 3.4.1 Decomposition of Time series data for power

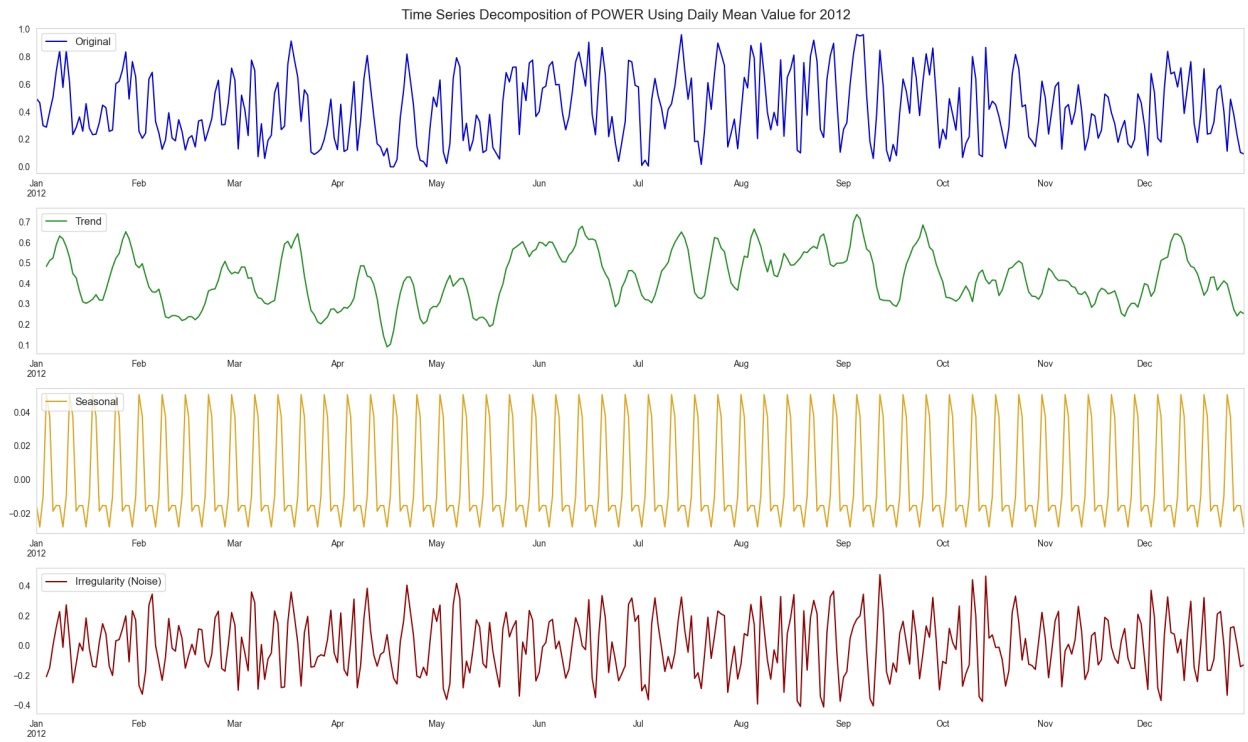


Figure 10: Time series decomposition of POWER using daily mean value

The trend shows the long-term increase or decrease in the data. Seasonality shows regular, repeating patterns or cycles. Irregularity or random noise, is just fluctuations that can not be predicted, it shows the residual error. The data was sampled each hour and changed into a daily mean for more readability and visual understanding of the data.

### 3.4.2 Testing for Stationary

To assess the stationarity of the time series, the Augmented Dickey-Fuller (ADF) and Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test was applied. The Augmented Dickey-Fuller statistical test examines the presence of a unit root in the series, which would indicate non-stationarity. The hypotheses for the ADF test are formulated as follows:

- **Null hypothesis** ($H_0$): The time series possesses a unit root (i.e., it is non-stationary).
- **Alternative hypothesis** ($H_1$): The time series is stationary.

A p-value less than the significance level (typically $\alpha = 0.05$) provides sufficient evidence to reject the null hypothesis, thereby suggesting that the time series is stationary. Conversely, a higher p-value indicates that the series may exhibit non-stationary behavior.

Meanwhile the KPSS null hypothesis ($H_0$) assumes stationarity, while the alternative hypothesis ($H_1$) suggests the presence of a unit root (non-stationarity). A p-value less than 0.05 indicates rejection of the null hypothesis.

### 3.4.3 Augmented Dickey-Fuller (ADF) test results

The ADF test yielded the following results:

| Metric | Value |
|---|---|
| ADF Test Statistic | -16.344 |
| p-value | $2.98 \times 10^{-29}$ |
| **Critical Values** | |
| 10% | -2.567 |
| 5% | -2.862 |
| 1% | -3.431 |

Table 2: Augmented Dickey-Fuller Test Results for POWER

The test statistic of **-16.344** is substantially lower than all of the critical values, and the p-value is significantly below the 0.05 significance threshold. These results provide strong evidence to reject the null hypothesis. Therefore, the power output time series can be considered **stationary**.

#### 3.4.4 KPSS Test Results for Stationarity

Although the Augmented Dickey-Fuller test yielded a highly significant result **ADF = -16.344**, strongly suggesting the series is stationary, we apply the KPSS test as a complementary diagnostic. The two tests have opposing null hypotheses, ADF tests for a unit root while KPSS tests for stationarity, so running both enhances our confidence in the inference. Confirmatory results from both would robustly establish the stationarity of the series. The KPSS test yielded the following results:

| Metric | Value |
|---|---|
| KPSS Statistic | 0.516 |
| p-value | 0.038 |
| Number of Lags | 68 |
| **Critical Values** | |
| 10% | 0.347 |
| 5% | 0.463 |
| 2.5% | 0.574 |
| 1% | 0.739 |

Table 3: KPSS test results on `POWER` time series data

Since the p-value is 0.038, which is less than 0.05, this implies rejection of the null hypothesis at the 5% level. Therefore we can reject the null hypothesis and also conclude with using the KPSS test that the `POWER` time series data is stationary.

### 3.4.5 ACF and PACF Graphs

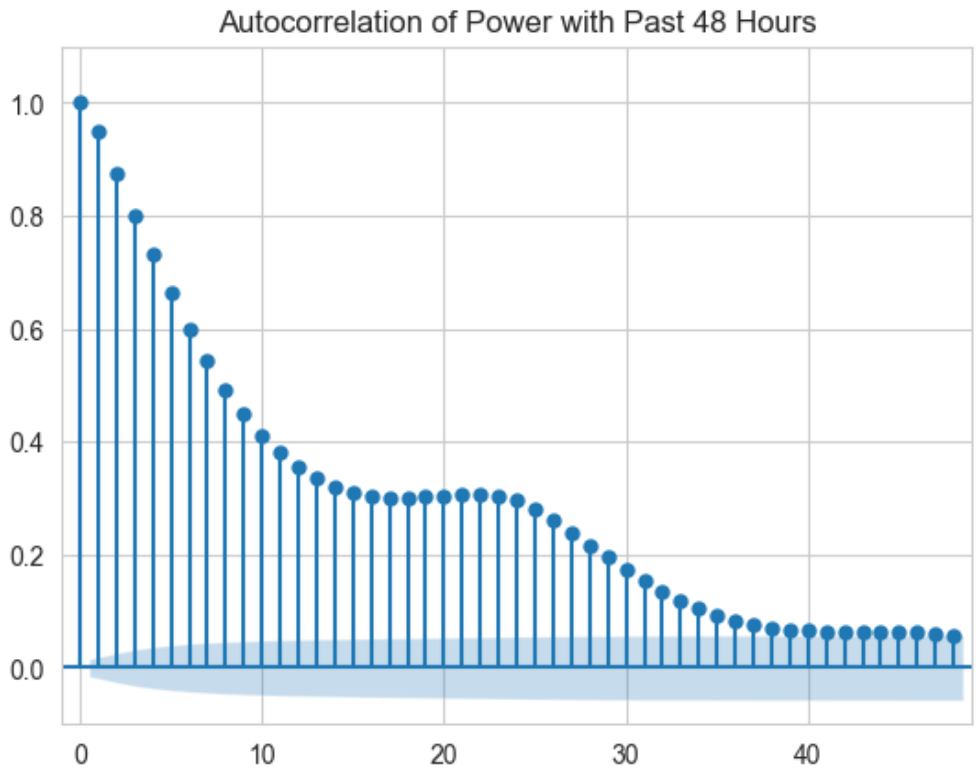


Figure 11: ACF graph that measures how POWER is correlated with it's past values

The above ACF graph shows how strong of a correlation Power has with it's past values, for example the first value corresponds how today's value relates to yesterdays, while the second bar shows the relationship with the value two days ago and so on. We can see by the points (or legs) being positive that power has a very high correlation with how it was in the past 48 hours, with a gradual decline and also a small little bump in the 24 hour mark. This indicated an auto-regressive process because there's a gradual decline or decay. The blue shaded area represents the confidence interval, bars outside this area indicates statistically significant correlations while bars within it are considered random noise. At leg 0 the correlation is always one since the power is compared to itself.

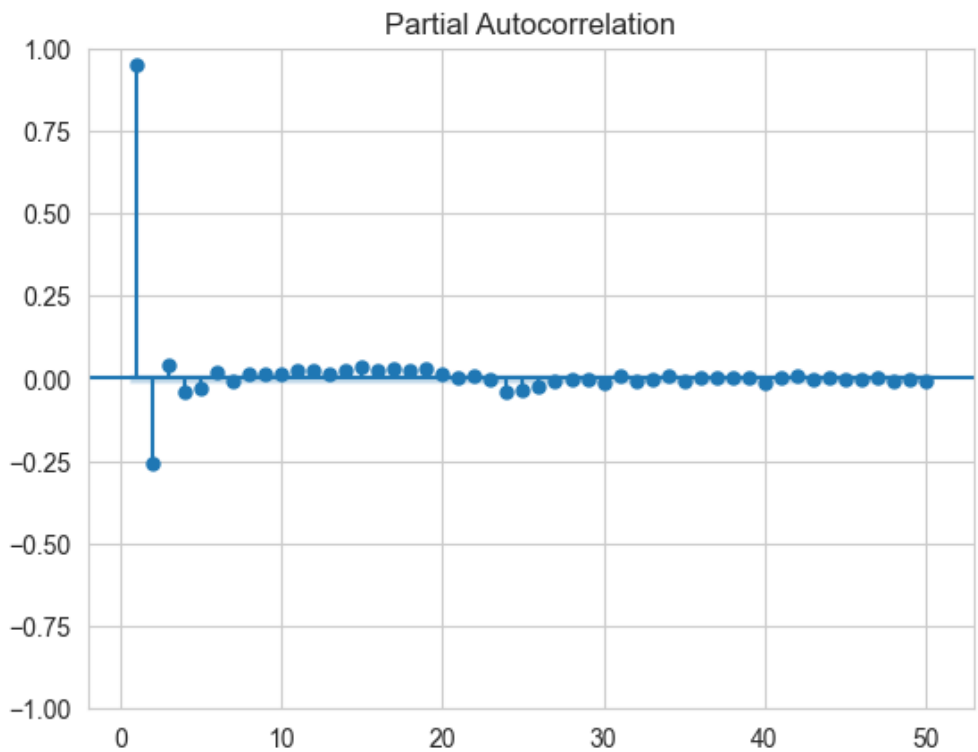


Figure 12: A PACF graph measuring the direct relationship between the time series and it's lagged values

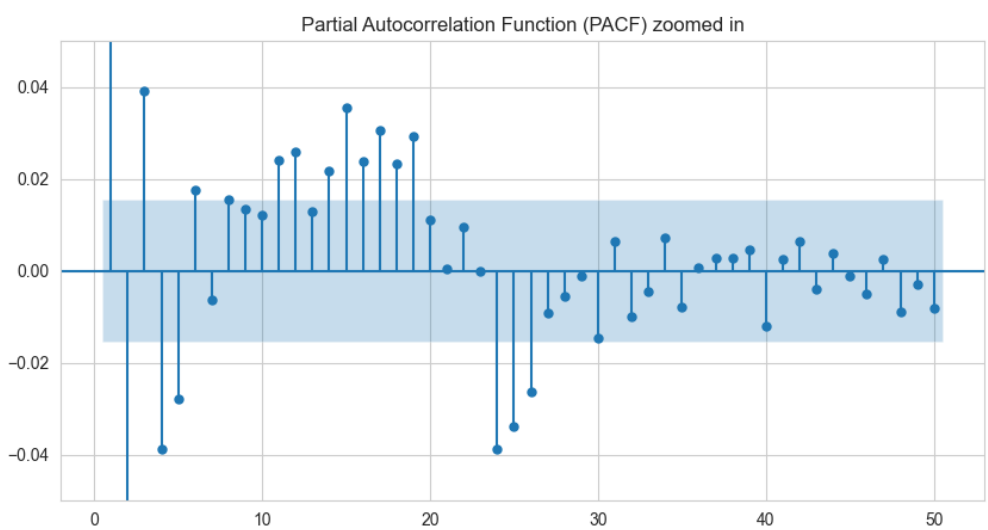


Figure 13: A zoomed in PACF graph measuring the direct relationship between the time series and it's lagged values

In the above figure we can see that majority of the legs are in the confidence interval meaning that they're not statistically significant correlations.

### 3.4.6 The weekly rolling mean of the wind Power

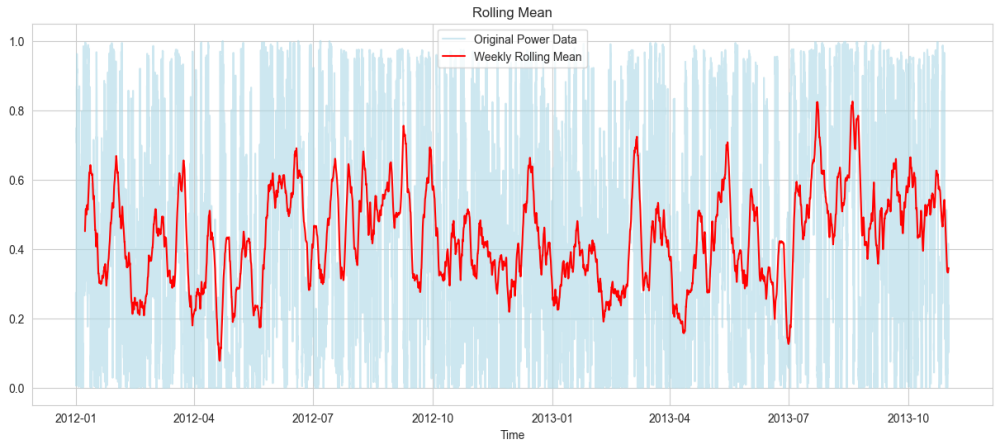


Figure 14: The weekly rolling mean of the wind Power

The above figure shows the fading blue as the original power data, the red lalong with the average weekly value (red curve). The powers average does show a repetable spike between 2012-10 to 2013-6.

## 3.5 Plotting U10, V10 and WS10 over time

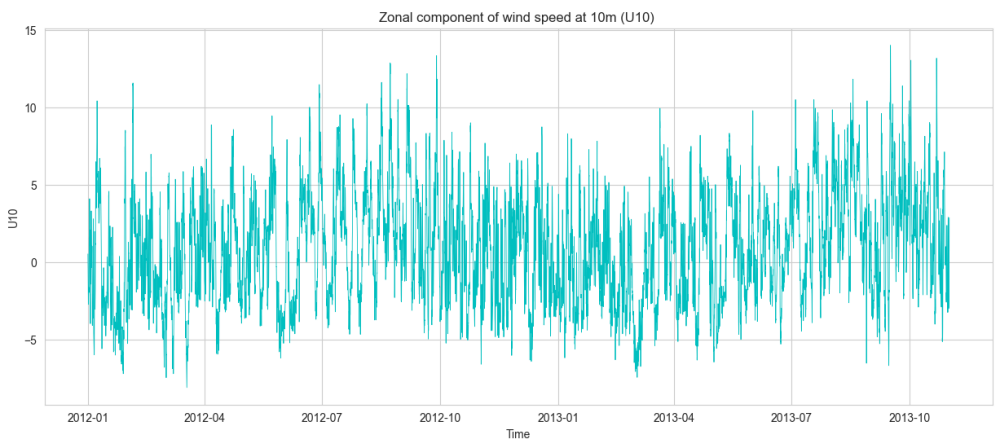


Figure 15: Zonal component of wind speed (U10) over time

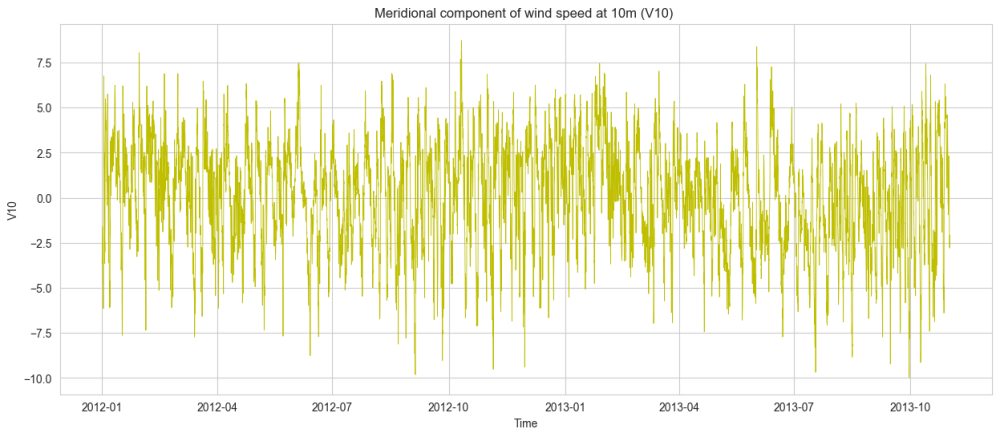


Figure 16: Meridional component of wind speed (V10) over time

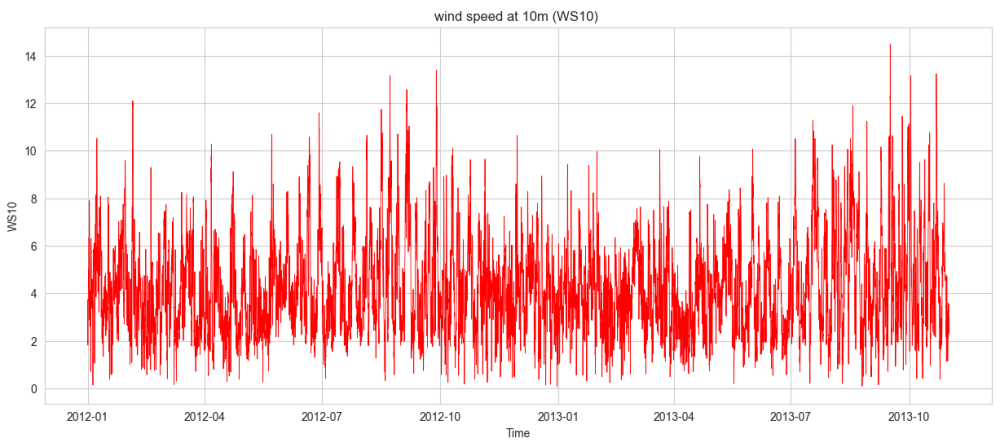


Figure 17: Windspeed (WS10) over time

## 3.6 Visualising relationship between power, time and wind speed (WS10)

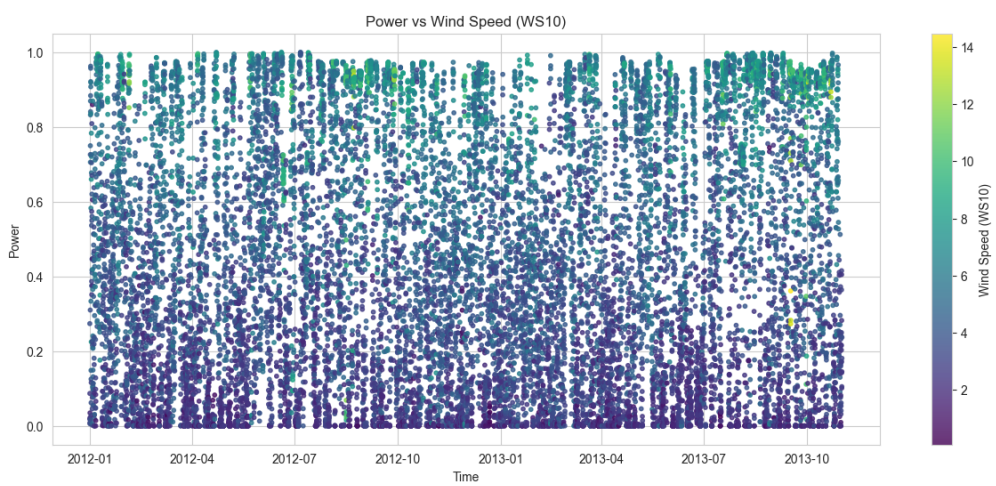


Figure 18: Relationship between power, time and wind speed (WS10)

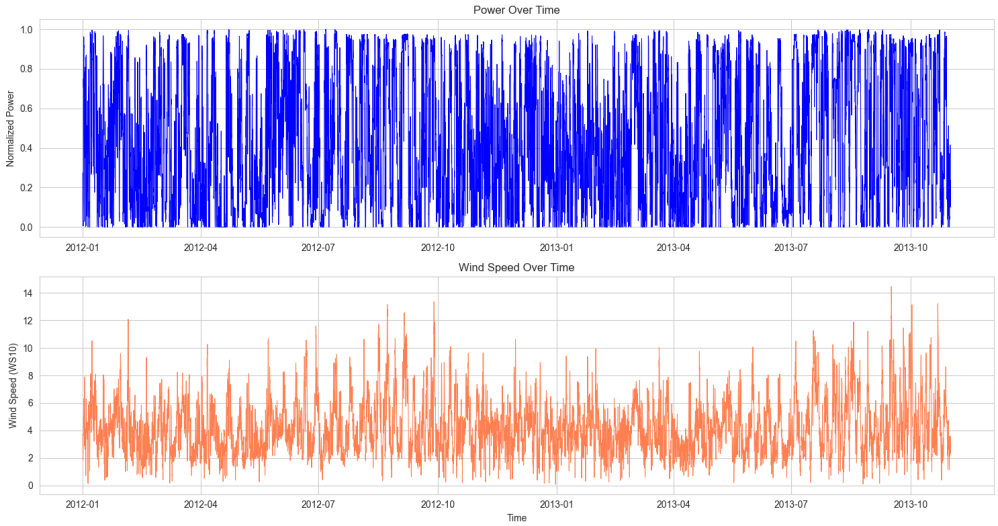


Figure 19: DistributionTwo curve graphs showing the relationship between power, time and wind speed (WS10)

## 3.7 Correlation Matrix

| | **POWER** | **WS10** | **U10** | **V10** |
|---|---|---|---|---|
| **POWER** | 1.000 | 0.727 | 0.298 | -0.178 |
| **WS10** | 0.727 | 1.000 | 0.512 | -0.044 |
| **U10** | 0.298 | 0.512 | 1.000 | 0.115 |
| **V10** | -0.178 | -0.044 | 0.115 | 1.000 |

Table 4: Correlation Matrix between POWER, WS10, U10, and V10

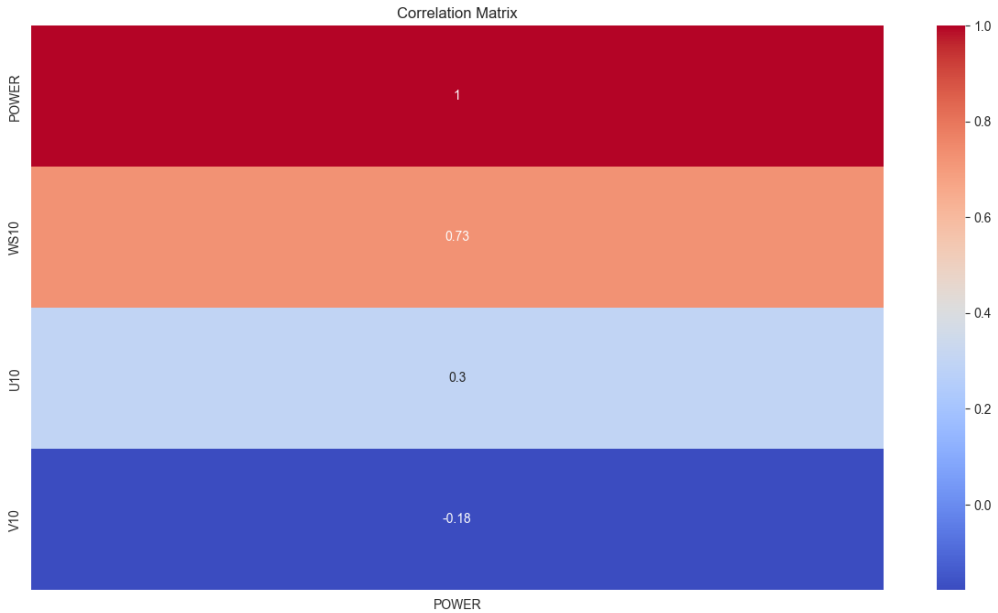


Figure 20: A table showcasing the correlation between power and all the variables

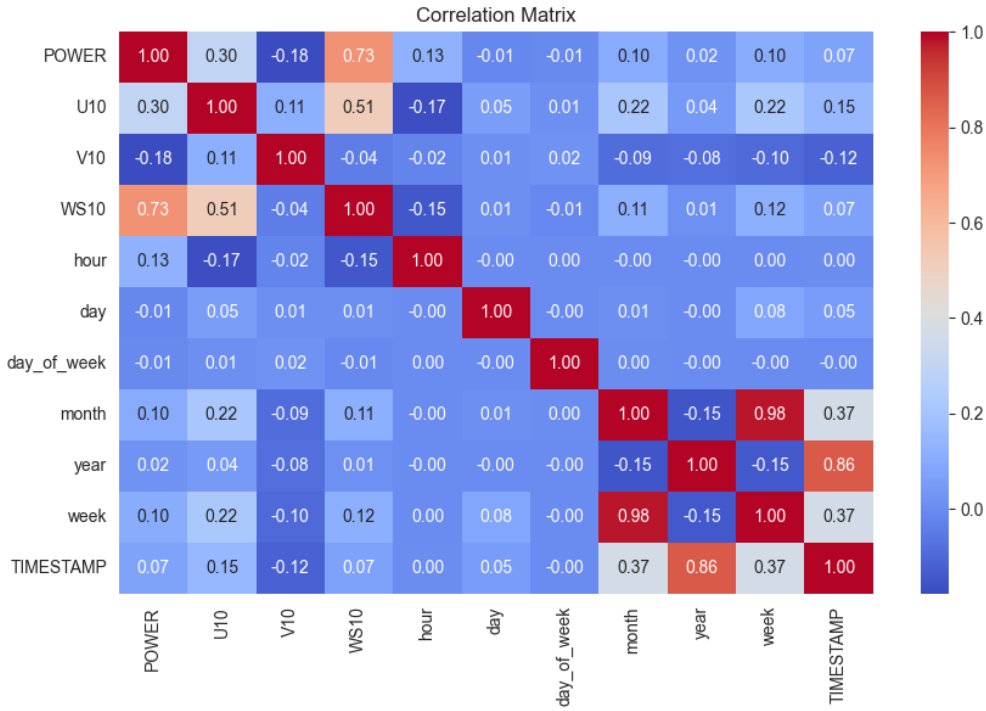


Figure 21: A table showcasing the correlation between all the variables

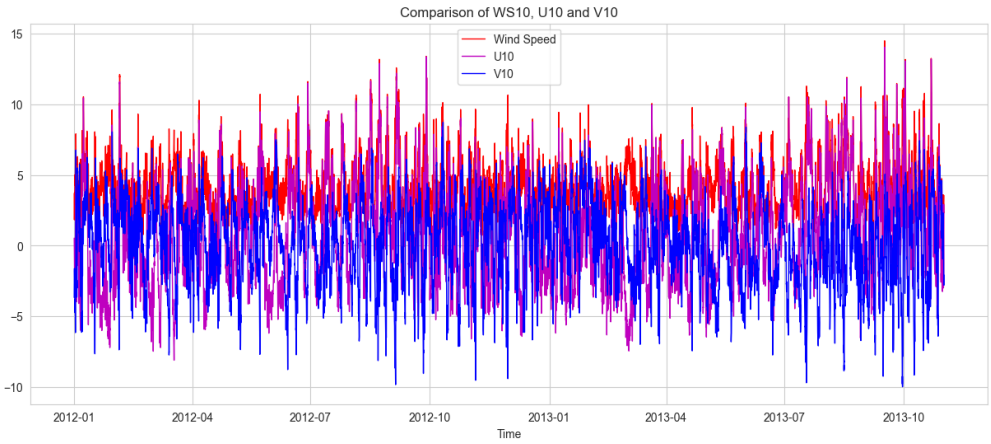


Figure 22: Comparison of WS10, U10 and V10

# 4 Task 1: Wind Power Forecasting Using Machine Learning

In this task we focused on the relationship between wind power generation and 10-meter wind speed using four machine learning models: Linear Regression (LR), K-Nearest Neighbors (kNN), Support Vector Regression, and Artificial Neural Networks (ANN). The models were trained on historical weather data (from 01.01.2012 to 01.11.2013) and then evaluated based on prediction (forecast) accuracy versus actual wind power measurements for November 2013.

## 4.1 RMSE and SMAPE Comparison

The table below summarizes the Root Mean Squared Error (RMSE), and symmetric mean absolute percentage error (SMAPE) for each model:

| Model | RMSE | Prediction Accuracy (SMAPE) |
|---|---|---|
| Linear Regression (LR) | 0.2164 | 33.13% |
| k-Nearest Neighbors (kNN) | 0.2349 | 31.71% |
| Support Vector Regression (SVR) | **0.2138** | 33.22% |
| Artificial Neural Network (NN) | 0.2154 | **33.66%** |

Table 5: RMSE and SMAPE Accuracy Comparison of Forecast Models

The SVR model achieved the lowest RMSE,
The ANN model showed slightly higher RMSE than SVR but achieved the best SMAPE accuracy.

## 4.2 Forecast Plots

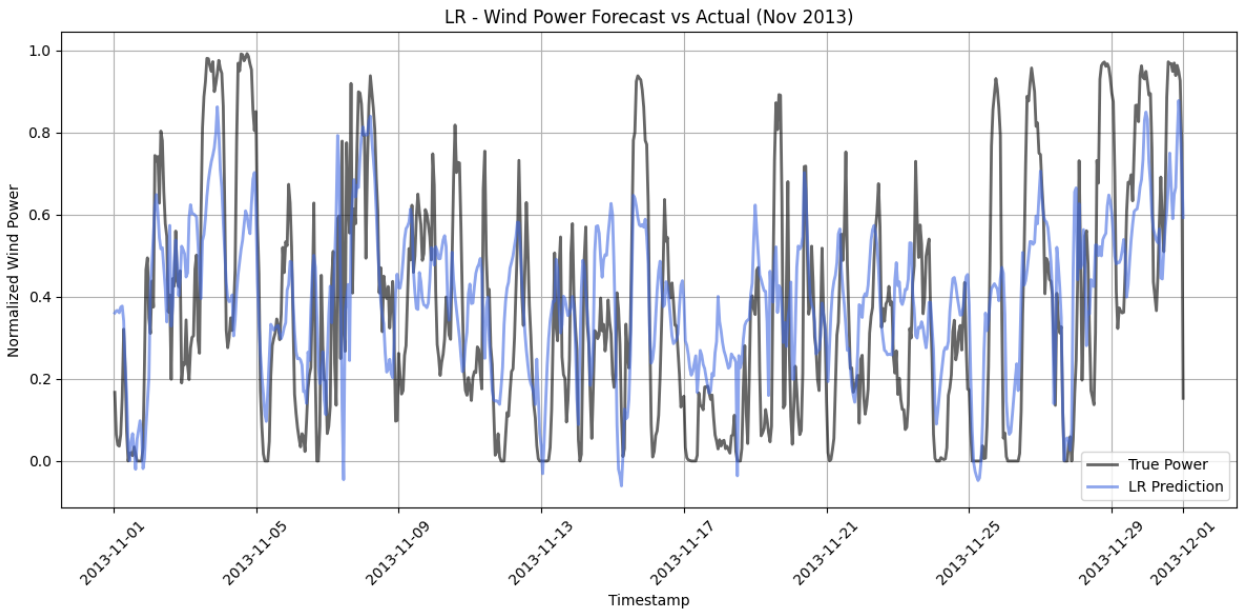


Figure 23: Linear Regression: Forecast vs. True Wind Power

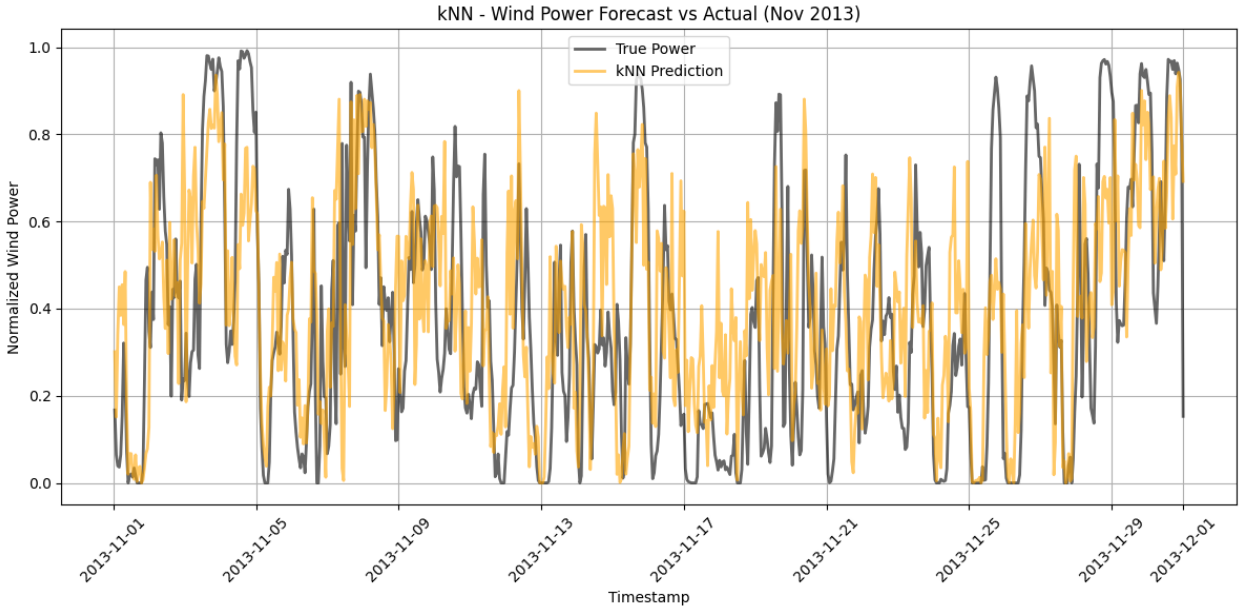


Figure 24: k-Nearest Neighbors: Forecast vs. True Wind Power

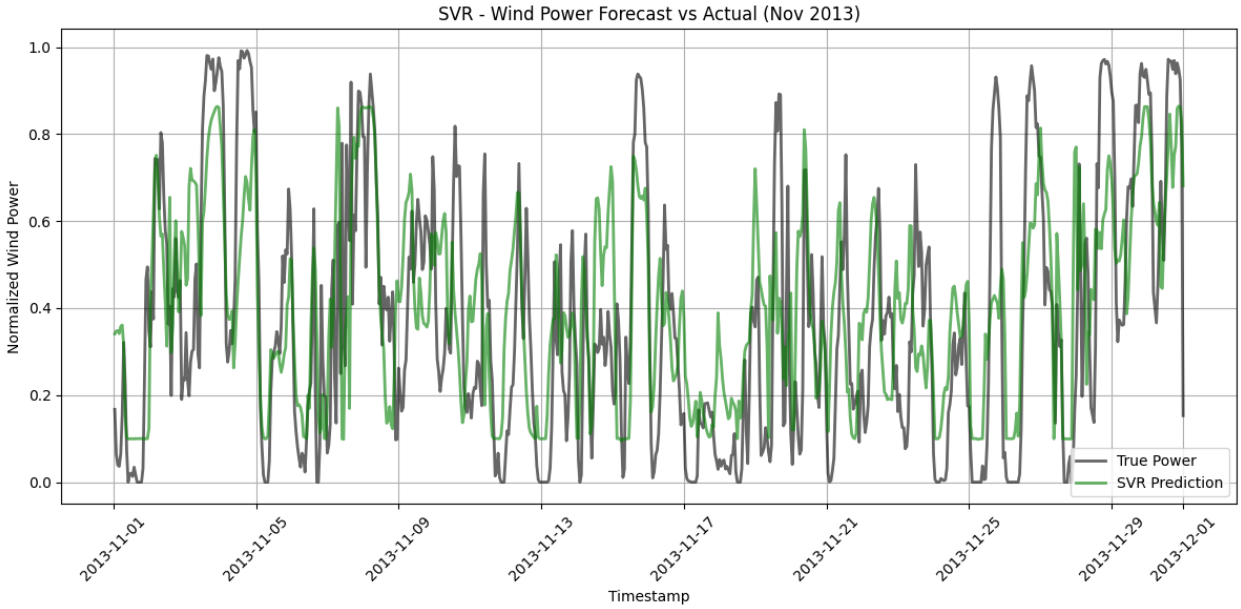


Figure 25: Support Vector Regression: Forecast vs. True Wind Power

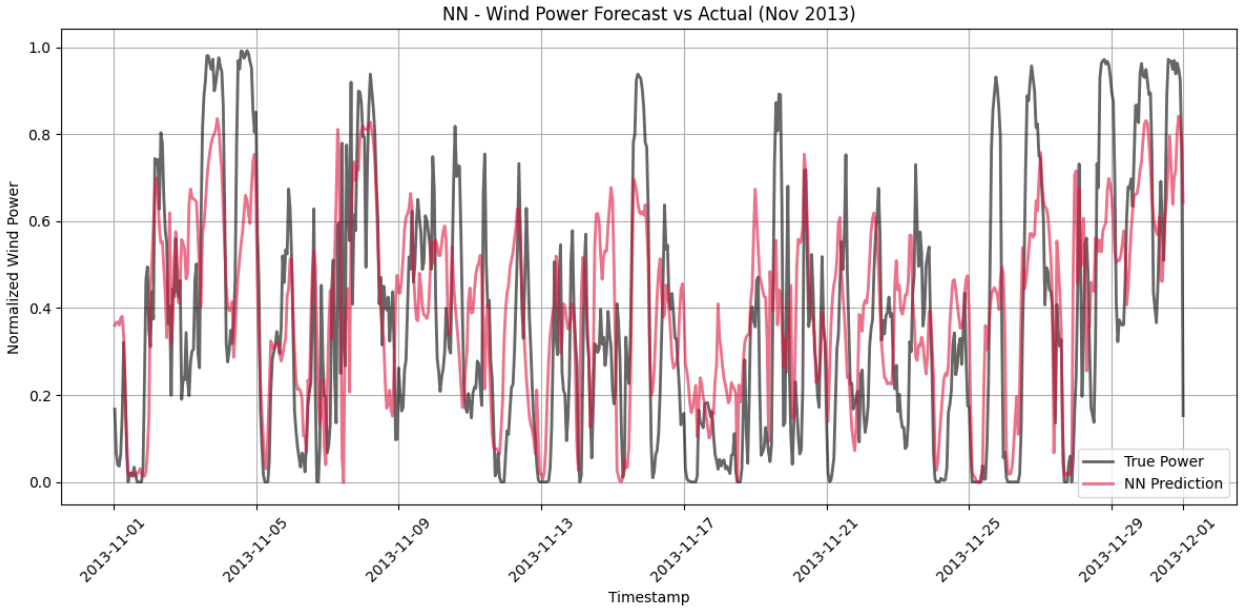


Figure 26: Neural Network: Forecast vs. True Wind Power

Each plot contains two curves: one for the true wind power measurements and one for the corresponding predictions from the respective model. The SVR while having the lowest RMSE does have some "bottom undercutting", meaning that the prediction does not reach below a specific threshold. In contrast to the ANN (artificial neural network), this model can predict the values in those areas unlike SVR.

## 4.3 $R^2$ Score Evaluation

To evaluate the goodness of fit for each machine learning model, we calculated the $R^2$ score, which indicates the proportion of variance in the dependent variable that is predictable from the independent variable(s). Below is a table showing the $R^2$ scores for both training and testing data sets:

| **Model** | **$R^2$ Score (Training)** | **$R^2$ Score (Testing)** |
|---|---|---|
| Linear Regression (LR) | 52.86% | 45.42% |
| k-Nearest Neighbors (kNN) | **65.43%** | 35.70% |
| Support Vector Regression (SVR) | 55.51% | **46.73%** |
| Artificial Neural Network (NN) | 56.17% | 45.90% |

Table 6: $R^2$ Score Comparison of Forecast Models (Goodness of Fit)

As shown in the table, Support Vector Regression (SVR) achieved the highest $R^2$ score on the test data (0.4673), indicating that it is the best performing model in terms of variance explanation on unseen data. However, this value still falls below the ideal range of 60% to 90%, suggesting that all the models have an under-fitted dataset. An $R^2$ score below 60% implies that the model does not capture enough variance, while a score above 90% could indicate overfitting. Another conclusion that could possibly arise from the $R^2$ scores is that the data might be noisy or random, meaning that we should consider environmental factors and also where the data location is used on, for example let's say you're collecting data from wind turbines in Australia but are trying to use this data to predict wind-power in Germany, there might be environmental factors that could play in which makes the data prediction different.

# 5 Task 2: Multiple Linear Regression with Wind Speed & Direction

We have for this task extended the original linear model (WS10) by computing wind-direction from zonal and meridional components (U10 and V10), and then training a multiple linear regression (MLR) model with both WS10 and wind direction as explanatory variables. The entire historical training set (Jan2012 - Oct2013) was used to maximize data volume before the model was run on the weather forecasts for November 2013. Figure (27) displays the predicted power versus actual data.

When visualizing the variables, plotting wind direction against power showeda wider spread of error than wind speed, informing that the direction on its own does not explaine much. To visualize how the variables interact, we used 3D scatter of Power versus WS10 and direction. The figures (28) show that power increases primarily on the wind speed axis, with direction having not having a noticable effect.

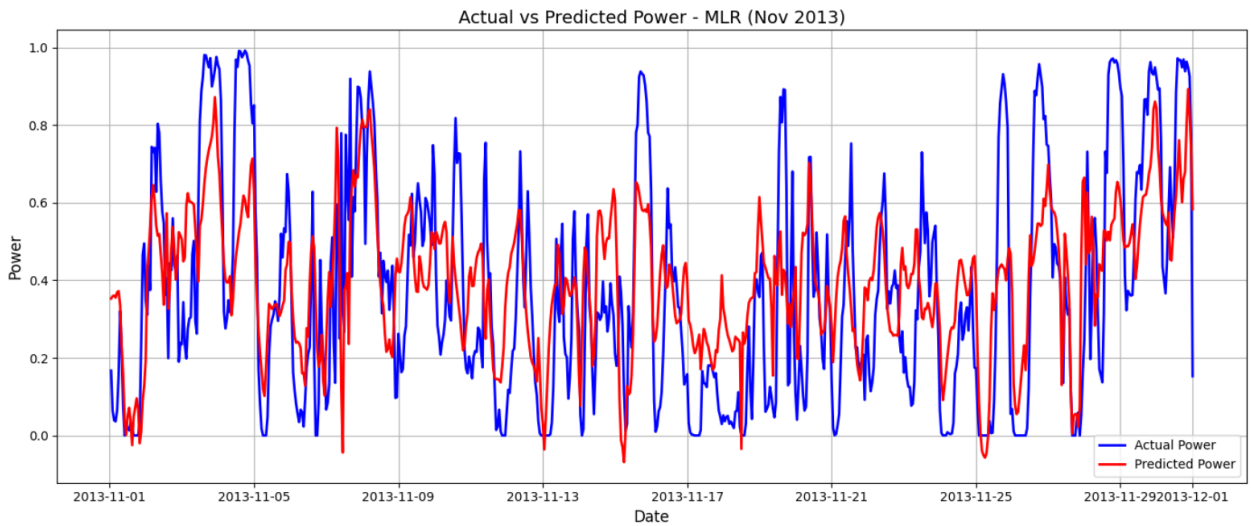


Figure 27: Actual vs Predicted Power - MLR (Nov 2013)

| **Model** | **RMSE** |
|---|---|
| Linear Regression (LR) | 0.21638 |
| Multiple Linear Regression (MLR) | 0.21494 |

Table 7: RMSE Accuracy Comparison of Forecast Models

| **Variable** | **Correlation** |
|---|---|
| WS10 | 0.7271 |
| Wind Direction | 0.1702 |

Table 8: Correlation table with respect to Power

The RMSE results in (Tables 7) show that the MLR model achieves a score of 0.21494 versus 0.21638 for the LR model, an improvment of only 0.0014 (0.65%). The correlation matrix (Table 8), explains that WS10 correlates strongly with power production (r=0.727), while wind direction contributes very poorly (r=0.170). This is also reflected in the model coefficients where wind speed recieves a slope of 0.1235, while the direction coefficient is near zero (Table 9). In addition, figure (29) shows no clear pattern between prediction and

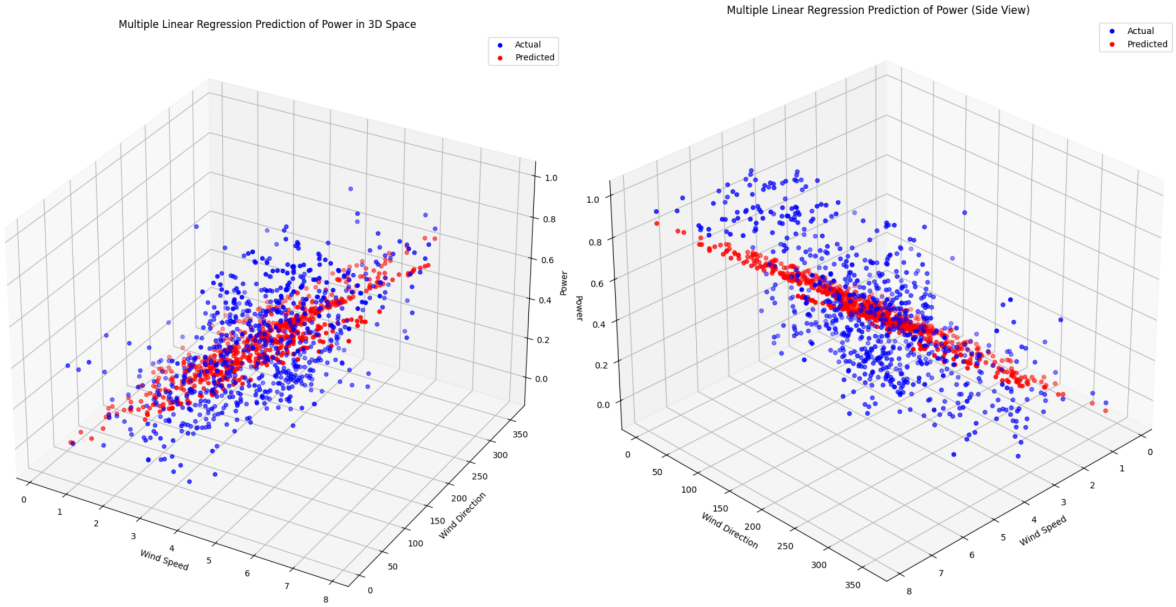


Figure 28: Actual Power vs Predicted Power in 3D Space

| **Parameter** | **Value** |
|---|---|
| Intercept | $-0.07463$ |
| WS10 | $0.12350$ |
| Wind Direction | $-0.00007$ |

Table 9: Coefficient Analysis

wind direction, while figure (30) shows the perfect linear relationship between power and wind speed when the direction is held constant at its mean.

Finally, in figure (31) we can see how closely the LR and MLR curves track each other throughout the November month. Both of the models miss primarily at sharp peaks and troughs. We can conclude that the wind direction does not add significant explanatory power for this park in November 2013.

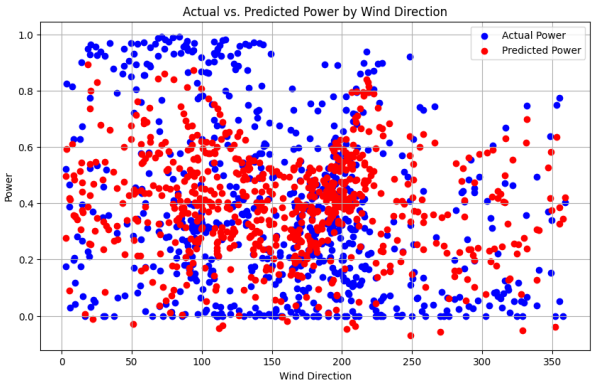


Figure 29: Actual vs Predicted Power

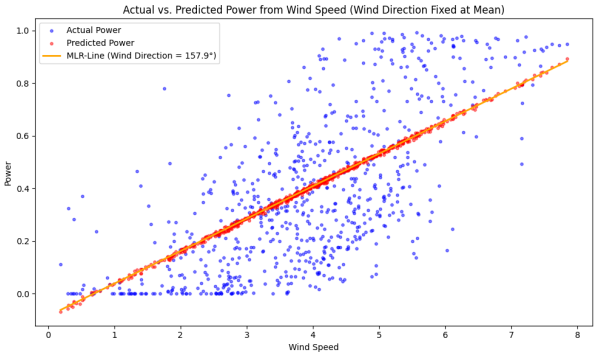


Figure 30: Actual vs Predicted power(Wind direction fixed at mean)

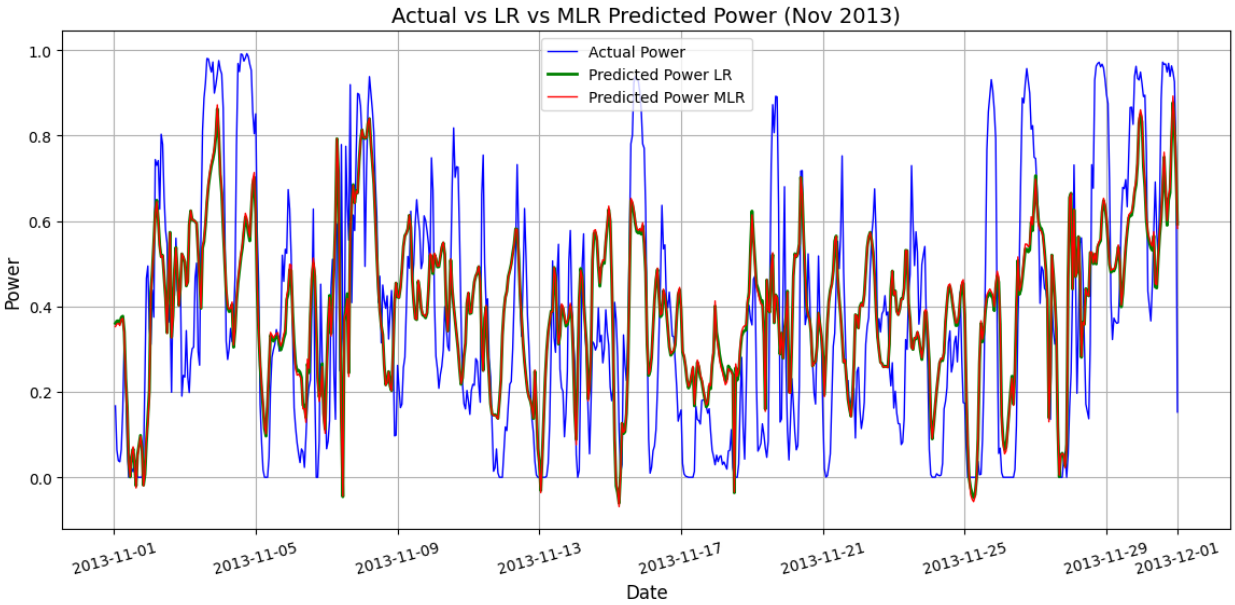


Figure 31: Actual vs LR vs MLR (Nov 2013)

# 6 Task 3: Wind Power Forecast Using Times Series

In this task, we focused on forecasting wind power production when only timestamp and historical power generation data were available, and no weather-related characteristics, such as wind speed, were provided, representing a time-series forecasting problem. To address this task, we applied four machine learning models: Linear Regression (LR), Support Vector Regression (SVR), Artificial Neural Network (ANN), and Recurrent Neural Network (RNN). These models were trained using historical power data and temporal features derived from timestamp. The goal was to predict wind power generation for November 2013 and evaluate the prediction accuracy against actual wind power measurements using the Root Mean Squared Error (RMSE).

## 6.1 RMSE and SMAPE Comparison

The table below summarizes the Root Mean Squared Error (RMSE), and Symetric Mean Absolute Percentage Error for each Model:

| **Model** | **RMSE** | **Prediction Accuracy (SMAPE)** |
|---|---|---|
| Linear Regression (LR) | 0.187384 | 41.39% |
| Support Vector Regression (SVR) | **0.184302** | **42.85%** |
| Artificial Neural Network (ANN) | 0.185688 | 40.09% |
| Recurrent Neural Network (RNN) | 0.188986 | 41.24% |

Table 10: RMSE and SMAPE Accuracy Comparison of Forecast Models

The relationship between RMSE and SMAPE in our results highlights that the models with lower absolute errors (RMSE) tend to also have higher prediction accuracy (100 - SMAPE), indicating a consistent ranking across metrics. Support Vector Regression (SVR), for instance, achieved both the lowest RMSE and the highest SMAPE-based accuracy, which aligns with the expectation that fewer absolute errors correspond to smaller percentage deviations. However, the differences between the models are small, suggesting that no single model significantly outperforms the others. Overall, all models show relatively low accuracy, with SMAPE-based scores around 40–43%, meaning that predictions deviate from actual values by approximately 57–60% on average. This indicates that the forecasting models are struggling, possibly due to noisy data, insufficient features, or the inherent variability of wind power.

## 6.2 Forecast Plots

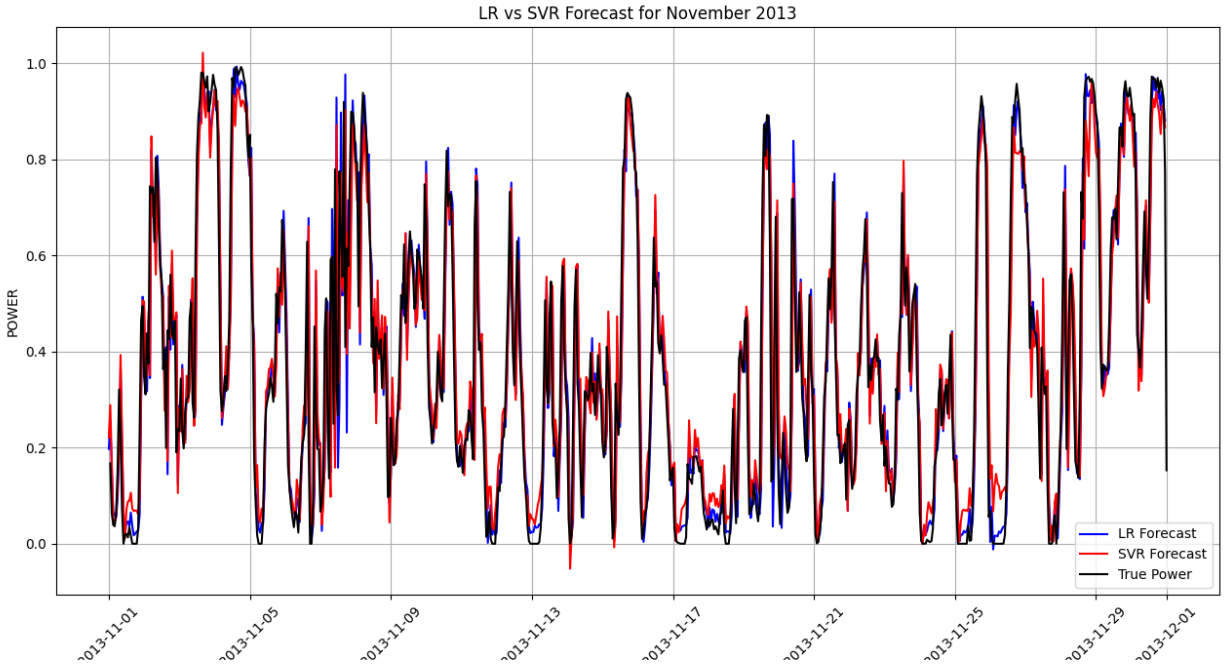


Figure 32: Times Series Forecast True Wind Power vs Linear Regression vs Support Vector Regression

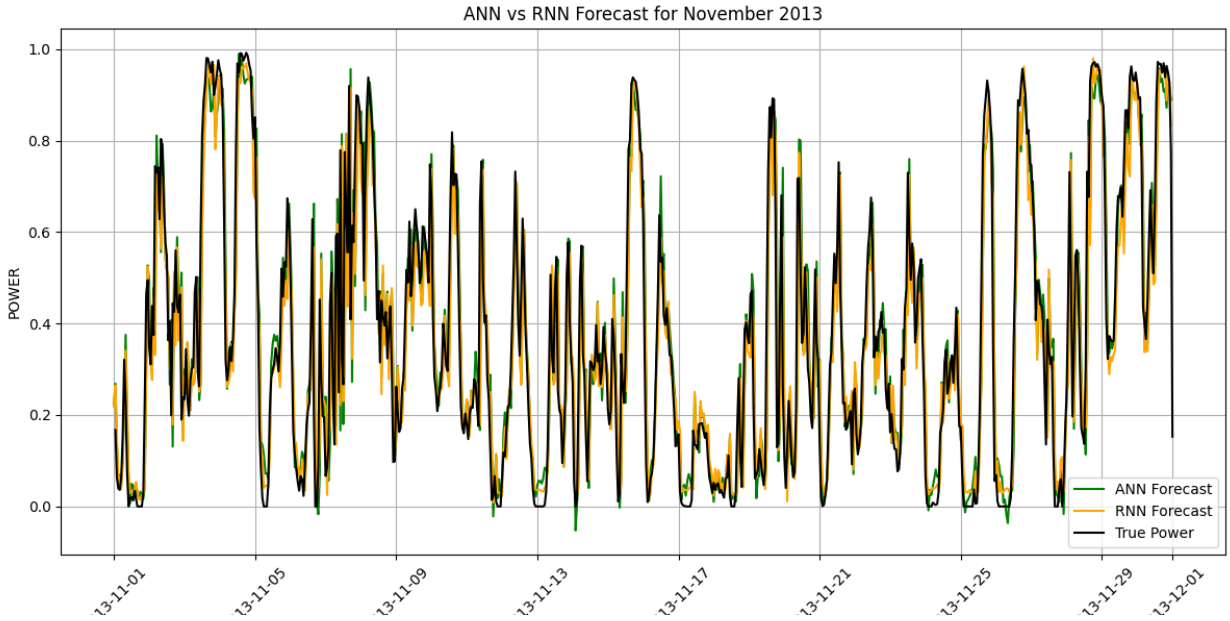


Figure 33: Times Series Forecast True Wind Power vs Artificial Neural Network vs Recurrent Neural Network

As evidenced by both the table and the forecast plots, Support Vector Regression (SVR) outperformed the other models, achieving the lowest RMSE (0.184302) and the highest SMAPE accuracy (42.85%). These results suggest that SVR is especially well-suited for univariate time-series forecasting scenarios where no external variables—such as weather data—are available. Its ability to capture patterns from historical power

generation and temporal features alone highlights its robustness and reliability in data-limited forecasting environments.

# 7 Task 4: Mathematically calculating a 2-2-1 Neural Network

## Problem Setup

- Input: $x_1 = 0.04,\ x_2 = 0.20$
- Desired output: $y_1 = 0.50$
- Initial random generated weights: $w_1 = 0.37,\ w_2 = 0.95,\ w_3 = 1.73,\ w_4 = 0.60,\ w_5 = 0.15,\ w_6 = 0.15$
- Learning rate: $\alpha = 0.4$

**Note.** *Throughout this document the symbol* sum *is* ***not*** *the summation operator but a plain variable name. We use the initital random generated weights w1=0.37, w2=0.95, w3=0.73, w4=0.6, w5=0.15, w6=0.15 for reproducibility and controlled calculations. In the code, every time we run the code, the weights will be different. For our case we need a controlled environment to reproduce the results as shown below.*

## 7.1 Forward Propagation

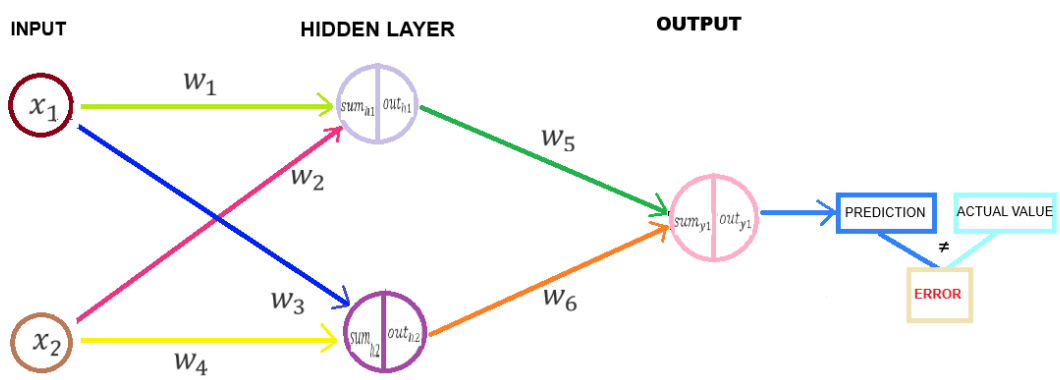


Figure 34: Network architecture and forward signal flow.

$$\text{sum}_{h_1} = x_1 w_1 + x_2 w_2, \qquad \text{out}_{h_1} = \sigma(\text{sum}_{h_1}), \tag{7.1}$$

$$\text{sum}_{h_2} = x_1 w_3 + x_2 w_4, \qquad \text{out}_{h_2} = \sigma(\text{sum}_{h_2}), \tag{7.2}$$

$$\text{sum}_{y_1} = \text{out}_{h_1} w_5 + \text{out}_{h_2} w_6, \qquad \text{out}_{y_1} = \sigma(\text{sum}_{y_1}). \tag{7.3}$$

**Activation function.**

$$\sigma(x) = \frac{1}{1+e^{-x}}, \qquad \sigma'(x) = \sigma(x)\big(1-\sigma(x)\big). \tag{7.4}$$

**Squared-error loss.**

$$E = \tfrac{1}{2}\big(y_1 - \text{out}_{y_1}\big)^2. \tag{7.5}$$

## 7.2 Backward Propagation

In the following, $\partial$ denotes a partial derivative and $E$ is the error function defined in 7.5.
In order to update the weights we use the first-order derivative of the error with respect to weight itself

$$w_i^{(new)} = w_i^{(old)} - \alpha \frac{\partial E}{\partial w_i^{(old)}}, \qquad \text{where } \alpha = 0.4,\ i = 1, \ldots, 6. \tag{7.6}$$

We need to find the expression for $\partial E / \partial w_i$.

## 7.3 Gradient and weight update for $w_6$.

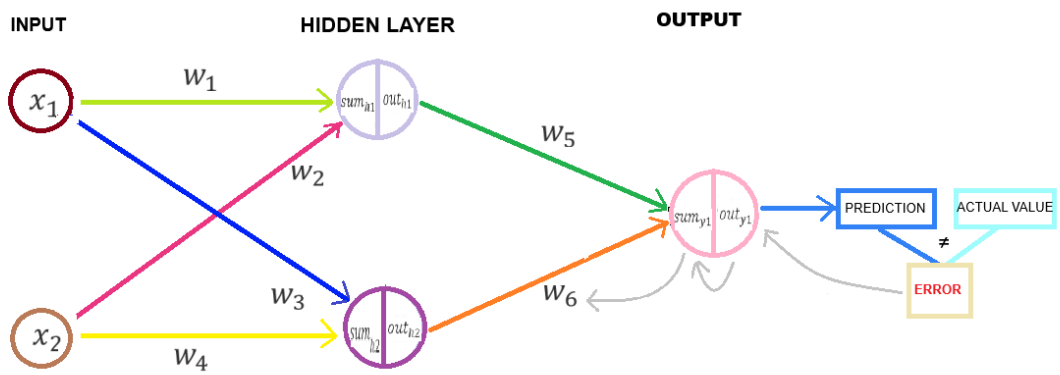


Figure 35: Error back-propagation path for $w_6$.

We use the chain-rule expansion to find the partial derivative:

$$\frac{\partial E}{\partial w_6} = \frac{\partial E}{\partial \text{out}_{y_1}} \cdot \frac{\partial \text{out}_{y_1}}{\partial \text{sum}_{y_1}} \cdot \frac{\partial \text{sum}_{y_1}}{\partial w_6},$$

where each intermediate derivative in the expression above is given by:

$$\begin{aligned}
\frac{\partial E}{\partial \text{out}_{y_1}} &= \frac{\partial}{\partial \text{out}_{y_1}} \left[\tfrac{1}{2}\left(y_1 - \text{out}_{y_1}\right)^2\right] \\
&= \tfrac{1}{2} \cdot 2\left(y_1 - \text{out}_{y_1}\right)^1 (-1) \\
&= -\left(y_1 - \text{out}_{y_1}\right) = \text{out}_{y_1} - y_1 \qquad (7.7a)
\end{aligned}$$

$$\frac{\partial \text{out}_{y_1}}{\partial \text{sum}_{y_1}} = \sigma(\text{sum}_{y_1})\left(1 - \sigma(\text{sum}_{y_1})\right) = \text{out}_{y_1}\left(1 - \text{out}_{y_1}\right) \qquad (7.7b)$$

$$\frac{\partial \text{sum}_{y_1}}{\partial w_6} = \frac{\partial}{\partial w_6}\left[\text{out}_{h_1} w_5 + \text{out}_{h_2} w_6\right] = \text{out}_{h_2} \qquad (7.7c)$$

Notice how the derivative of $\sigma$ appears as $\sigma(x)(1 - \sigma(x))$ defined in 7.4.

Combining these expressions, we obtain

$$\frac{\partial E}{\partial w_6} = (\text{out}_{y_1} - y_1)\,\text{out}_{y_1}(1 - \text{out}_{y_1})\,\text{out}_{h_2}. \tag{7.8}$$

Finally, the weight $w_6$ can be updated by:

$$w_6^{\text{new}} = w_6 - \alpha\,\frac{\partial E}{\partial w_6}.$$

## 7.4 Gradient and weight update for $w_5$.

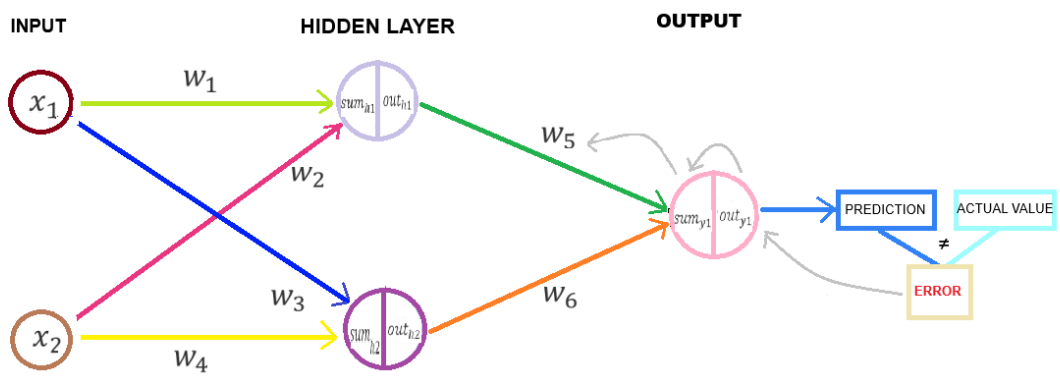


Figure 36: Error back-propagation path for $w_5$.

Using the chain-rule, we get the following expression for the partial derivative of the error with respect to $w_5$:

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial \text{out}_{y_1}} \cdot \frac{\partial \text{out}_{y_1}}{\partial \text{sum}_{y_1}} \cdot \frac{\partial \text{sum}_{y_1}}{\partial w_5}.$$

Following the same steps as for $w_6$ we obtain

$$\frac{\partial E}{\partial w_5} = \big(\text{out}_{y_1} - y_1\big)\text{out}_{y_1}\big(1 - \text{out}_{y_1}\big)\text{out}_{h_1}.$$

Finally, the weight $w_5$ can be updated by:

$$w_5^{\text{new}} = w_5 - \alpha\,\frac{\partial E}{\partial w_5}.$$

## 7.5 Gradient for $w_4, w_3, w_2, w_1$.

The expressions for $w_4, w_3, w_2, w_1$ are more involved, but we can use the chain rule to find each of them.

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \text{out}_{h_j}} \frac{\partial \text{out}_{h_j}}{\partial \text{sum}_{h_j}} \frac{\partial \text{sum}_{h_j}}{\partial w_i}, \qquad j \in \{1, 2\}. \tag{7.9}$$

where

$$\frac{\partial E}{\partial \text{out}_{h_j}} = \frac{\partial E}{\partial \text{out}_{y_1}} \frac{\partial \text{out}_{y_1}}{\partial \text{sum}_{y_1}} \frac{\partial \text{sum}_{y_1}}{\partial \text{sum}_{h_j}},$$

and the index $j$ refers to a specific hidden layer.

## 7.6 Gradient and update for $w_4$.

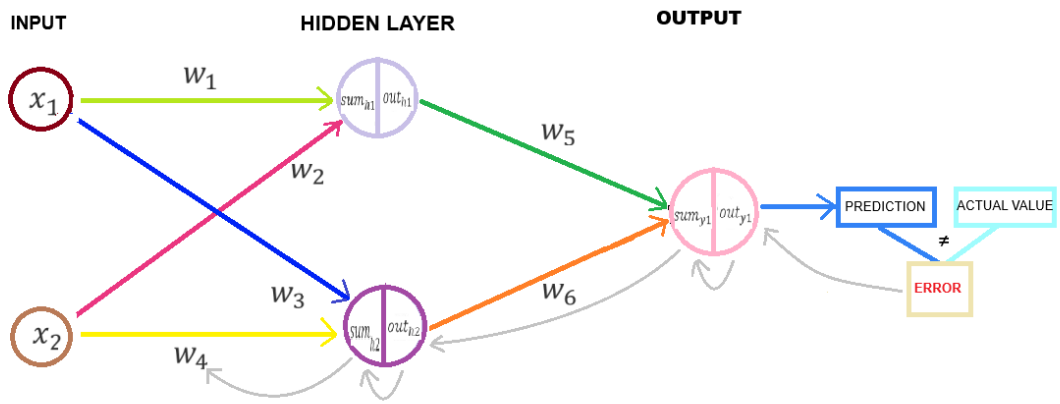


Figure 37: Error back-propagation path for $w_4$.

The weight $w_4$ is connected to the layer $h_2$, therefore filling in 7.9 for $w_4$, we obtain the following:

$$\frac{\partial E}{\partial w_4} = \frac{\partial E}{\partial \text{out}_{y_1}} \frac{\partial \text{out}_{y_1}}{\partial \text{sum}_{y_1}} \frac{\partial \text{sum}_{y_1}}{\partial \text{out}_{h_2}} \frac{\partial \text{out}_{h_2}}{\partial \text{sum}_{h_2}} \frac{\partial \text{sum}_{h_2}}{\partial w_4},$$

The first two factors in this equation have already been obtained in 7.7a and 7.7b. The last three factors are easily obtained and have the following expressions:

$$\frac{\partial \text{sum}_{y_1}}{\partial \text{out}_{h_2}} = w_6, \quad \frac{\partial \text{out}_{h_2}}{\partial \text{sum}_{h_2}} = \text{out}_{h_2}\big(1 - \text{out}_{h_2}\big), \quad \frac{\partial \text{sum}_{h_2}}{\partial w_4} = x_2.$$

Multiplying these factors, we finally end up with

$$\frac{\partial E}{\partial w_4} = (\mathrm{out}_{y_1} - y_1)\mathrm{out}_{y_1}\big(1 - \mathrm{out}_{y_1}\big) w_6\, \mathrm{out}_{h_2}\big(1 - \mathrm{out}_{h_2}\big) x_2. \tag{7.10}$$

Finally, the weight $w_4$ can be updated by:

$$w_4^{\mathrm{new}} = w_4 - \alpha\, \frac{\partial E}{\partial w_4}.$$

## 7.7 Gradient and update for $w_3$.

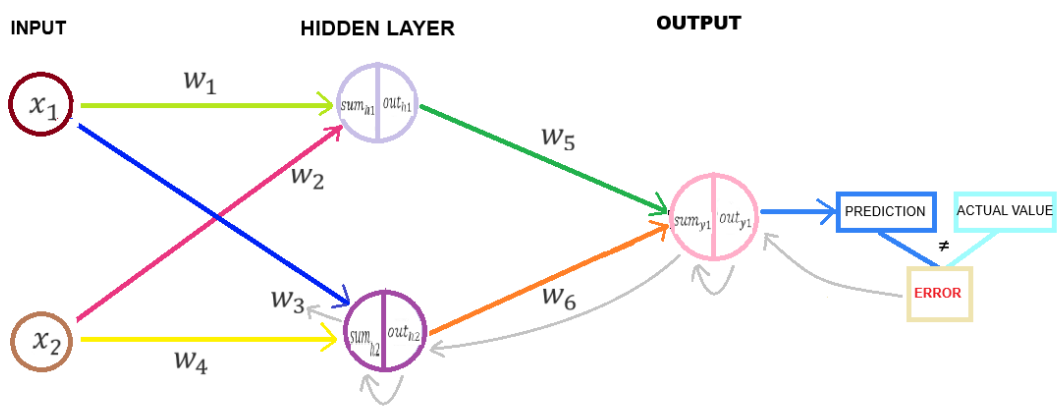


Figure 38: Error back-propagation path for $w_3$.

The weight $w_3$ is also connected to the hidden neuron $h_2$; substituting $w_3$ into Eq. (7.9) gives

$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial \mathrm{out}_{y_1}} \frac{\partial \mathrm{out}_{y_1}}{\partial \mathrm{sum}_{y_1}} \frac{\partial \mathrm{sum}_{y_1}}{\partial \mathrm{out}_{h_2}} \frac{\partial \mathrm{out}_{h_2}}{\partial \mathrm{sum}_{h_2}} \frac{\partial \mathrm{sum}_{h_2}}{\partial w_3}.$$

The first two factors are the same as in (7.7a)–(7.7b). The remaining three are

$$\frac{\partial \mathrm{sum}_{y_1}}{\partial \mathrm{out}_{h_2}} = w_6, \quad \frac{\partial \mathrm{out}_{h_2}}{\partial \mathrm{sum}_{h_2}} = \mathrm{out}_{h_2}\big(1 - \mathrm{out}_{h_2}\big), \quad \frac{\partial \mathrm{sum}_{h_2}}{\partial w_3} = x_1.$$

Multiplying everything together, we obtain

$$\frac{\partial E}{\partial w_3} = (\mathrm{out}_{y_1} - y_1)\,\mathrm{out}_{y_1}\big(1 - \mathrm{out}_{y_1}\big)\,w_6\,\mathrm{out}_{h_2}\big(1 - \mathrm{out}_{h_2}\big)\,x_1. \tag{7.11}$$

Finally, the weight $w_3$ can be updated by:

$$w_3^{\mathrm{new}} = w_3 - \alpha\,\frac{\partial E}{\partial w_3}.$$

## 7.8 Gradient and update for $w_2$.

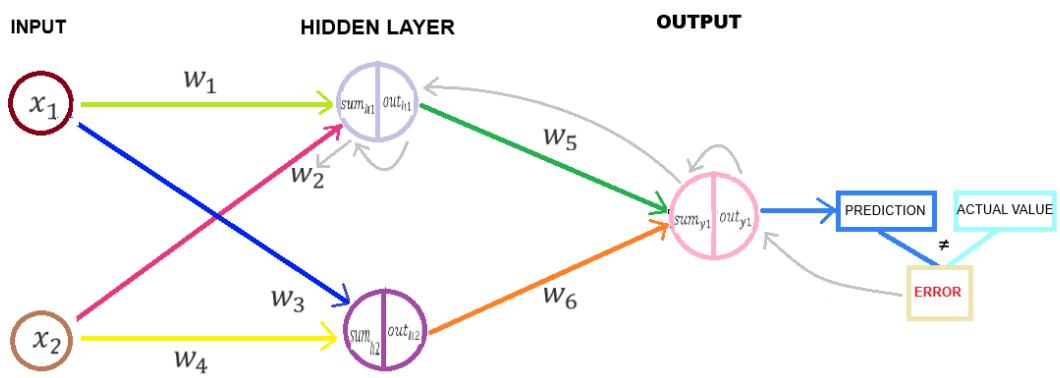


Figure 39: Error back-propagation path for $w_2$.

The weight $w_2$ is connected to the hidden neuron $h_1$; substituting $w_2$ into Eq. (7.9) yields

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial \mathrm{out}_{y_1}}\,\frac{\partial \mathrm{out}_{y_1}}{\partial \mathrm{sum}_{y_1}}\,\frac{\partial \mathrm{sum}_{y_1}}{\partial \mathrm{out}_{h_1}}\,\frac{\partial \mathrm{out}_{h_1}}{\partial \mathrm{sum}_{h_1}}\,\frac{\partial \mathrm{sum}_{h_1}}{\partial w_2}.$$

The first two factors are again given by (7.7a)–(7.7b). The remaining three factors are now

$$\frac{\partial \mathrm{sum}_{y_1}}{\partial \mathrm{out}_{h_1}} = w_5, \qquad \frac{\partial \mathrm{out}_{h_1}}{\partial \mathrm{sum}_{h_1}} = \mathrm{out}_{h_1}\big(1 - \mathrm{out}_{h_1}\big), \qquad \frac{\partial \mathrm{sum}_{h_1}}{\partial w_2} = x_2.$$

Combining everything, we obtain

$$\frac{\partial E}{\partial w_2} = (\mathrm{out}_{y_1} - y_1)\,\mathrm{out}_{y_1}\big(1 - \mathrm{out}_{y_1}\big)\,w_5\,\mathrm{out}_{h_1}\big(1 - \mathrm{out}_{h_1}\big)\,x_2. \tag{7.12}$$

Finally, the weight $w_2$ can be updated by:

$$w_2^{\mathrm{new}} = w_2 - \alpha\,\frac{\partial E}{\partial w_2}.$$

## 7.9 Gradient and update for $w_1$.

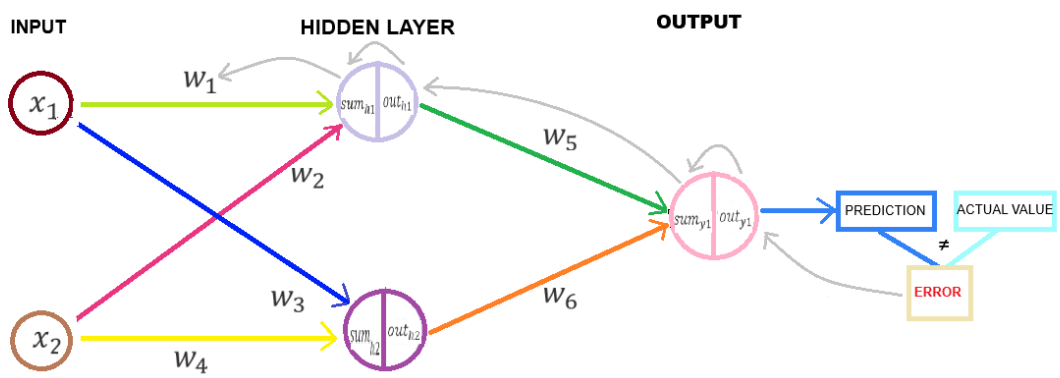


Figure 40: Error back-propagation path for $w_1$.

The weight $w_1$ is also connected to the hidden neuron $h_1$; substituting $w_1$ into Eq. (7.9) gives

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial \mathrm{out}_{y_1}}\,\frac{\partial \mathrm{out}_{y_1}}{\partial \mathrm{sum}_{y_1}}\,\frac{\partial \mathrm{sum}_{y_1}}{\partial \mathrm{out}_{h_1}}\,\frac{\partial \mathrm{out}_{h_1}}{\partial \mathrm{sum}_{h_1}}\,\frac{\partial \mathrm{sum}_{h_1}}{\partial w_1}.$$

The first two factors are already known from Eqs. (7.7a)–(7.7b). The remaining factors for the $h_1$ path are

$$\frac{\partial \mathrm{sum}_{y_1}}{\partial \mathrm{out}_{h_1}} = w_5, \qquad \frac{\partial \mathrm{out}_{h_1}}{\partial \mathrm{sum}_{h_1}} = \mathrm{out}_{h_1}\big(1 - \mathrm{out}_{h_1}\big), \qquad \frac{\partial \mathrm{sum}_{h_1}}{\partial w_1} = x_1.$$

Multiplying these factors, we find

$$\frac{\partial E}{\partial w_1} = (\mathrm{out}_{y_1} - y_1)\,\mathrm{out}_{y_1}\big(1 - \mathrm{out}_{y_1}\big)\,w_5\,\mathrm{out}_{h_1}\big(1 - \mathrm{out}_{h_1}\big)\,x_1. \tag{7.13}$$

Finally, the weight $w_1$ can be updated by:

$$w_1^{\mathrm{new}} = w_1 - \alpha\,\frac{\partial E}{\partial w_1}.$$

With the expressions $\partial E/\partial w_i$ we can now update the weights via Eq. (7.6).

## 7.10 Forward & backward propagation calculations

### Forward propagation calculation for the iteration 0

Computed activations and error using the initial weights.

$$\begin{aligned}
\mathrm{sum}_{h_1}^{(0)} &= 0.04 \cdot 0.37 + 0.20 \cdot 0.95 && = 0.204800 \\
\mathrm{out}_{h_1}^{(0)} &= \sigma(0.204800) && = 0.551022 \\
\mathrm{sum}_{h_2}^{(0)} &= 0.04 \cdot 1.73 + 0.20 \cdot 0.60 && = 0.149200 \\
\mathrm{out}_{h_2}^{(0)} &= \sigma(0.149200) && = 0.537231 \\
\mathrm{sum}_{y_1}^{(0)} &= 0.551022 \cdot 0.15 + 0.537231 \cdot 0.15 && = 0.163238 \\
\mathrm{out}_{y_1}^{(0)} &= \sigma(0.163238) && = 0.540719 \\
E^{(0)} &= \tfrac{1}{2}\big(0.50 - 0.540719\big)^2 && = 0.000829
\end{aligned} \tag{7.14}$$

### Backward propagation calculation for the iteration 0

We recall the expression for weight update in (7.6) and use it as shown below: Using the formulas derived earlier, we obtain:

$$\begin{aligned}
\partial E/\partial w_6^{(0)} &= 5.43 \times 10^{-3}, \quad w_6^{(1)} = 0.150000 - 0.4 \cdot 5.43 \times 10^{-3} = 0.145685, \\
\partial E/\partial w_5^{(0)} &= 5.57 \times 10^{-3}, \quad w_5^{(1)} = 0.150000 - 0.4 \cdot 5.57 \times 10^{-3} = 0.145574, \\
\partial E/\partial w_4^{(0)} &= 7.54 \times 10^{-5}, \quad w_4^{(1)} = 0.600000 - 0.4 \cdot 7.54 \times 10^{-5} = 0.599941, \\
\partial E/\partial w_3^{(0)} &= 1.51 \times 10^{-5}, \quad w_3^{(1)} = 1.730000 - 0.4 \cdot 1.51 \times 10^{-5} = 0.729988, \\
\partial E/\partial w_2^{(0)} &= 7.50 \times 10^{-5}, \quad w_2^{(1)} = 0.950000 - 0.4 \cdot 7.50 \times 10^{-5} = 0.949941, \\
\partial E/\partial w_1^{(0)} &= 1.51 \times 10^{-5}, \quad w_1^{(1)} = 0.370000 - 0.4 \cdot 1.51 \times 10^{-5} = 0.369988.
\end{aligned}$$

## Forward propagation calculation for iteration 1

Using the first–step weights $w^{(1)} = [0.369988,\ 0.949941,\ 0.729988,\ 0.599941,\ 0.145574,\ 0.145685]$, the forward pass yields

$$\begin{aligned}
\text{sum}_{h_1}^{(1)} &= 0.04 \cdot 0.369988 + 0.20 \cdot 0.949941 &&= 0.204794 \\
\text{out}_{h_1}^{(1)} &= \sigma(0.204794) &&= 0.551020 \\
\text{sum}_{h_2}^{(1)} &= 0.04 \cdot 0.729988 + 0.20 \cdot 0.599941 &&= 0.149194 \\
\text{out}_{h_2}^{(1)} &= \sigma(0.149194) &&= 0.537229 \\
\text{sum}_{y_1}^{(1)} &= 0.551020 \cdot 0.145685 + 0.537229 \cdot 0.145685 &&= 0.160842 \\
\text{out}_{y_1}^{(1)} &= \sigma(0.160842) &&= 0.540124 \\
E^{(1)} &= \tfrac{1}{2}\big(0.50 - 0.540124\big)^2 &&= 0.000805
\end{aligned} \tag{7.15}$$

A comparison of $E^{(1)}$ with $E^{(0)}$ confirms the expected decrease in the training error after the first weight update.

**Table 11.** Weights and squared error $E$ for the first ten gradient–descent steps and the final iteration.

| $i$ | $w_1$ | $w_2$ | $w_3$ | $w_4$ | $w_5$ | $w_6$ | $E^{(i)}$ |
|---|---|---|---|---|---|---|---|
| 0 | 0.370000 | 0.950000 | 0.730000 | 0.600000 | 0.150000 | 0.150000 | 0.000829 |
| 1 | 0.369988 | 0.949941 | 0.729988 | 0.599941 | 0.145574 | 0.145685 | 0.000805 |
| 2 | 0.369983 | 0.949913 | 0.729982 | 0.599912 | 0.143410 | 0.143574 | 0.000782 |
| 3 | 0.369977 | 0.949885 | 0.729977 | 0.599884 | 0.141276 | 0.141494 | 0.000759 |
| 4 | 0.369972 | 0.949858 | 0.729972 | 0.599858 | 0.139173 | 0.139444 | 0.000737 |
| 5 | 0.369966 | 0.949832 | 0.729966 | 0.599831 | 0.137101 | 0.137423 | 0.000715 |
| 6 | 0.369961 | 0.949807 | 0.729961 | 0.599806 | 0.135058 | 0.135432 | 0.000695 |
| 7 | 0.369957 | 0.949783 | 0.729956 | 0.599782 | 0.133045 | 0.133470 | 0.000674 |
| 8 | 0.369952 | 0.949759 | 0.729952 | 0.599758 | 0.131062 | 0.131536 | 0.000655 |
| 9 | 0.369947 | 0.949736 | 0.729947 | 0.599734 | 0.129107 | 0.129630 | 0.000636 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 2288 | 0.369796 | 0.948982 | 0.729790 | 0.598951 | -0.001877 | 0.001925 | 0.000000 |

**Table 12.** Forward values and error $E$ for the same ten steps.

| $i$ | $\text{sum}_{h_1}$ | $\text{out}_{h_1}$ | $\text{sum}_{h_2}$ | $\text{out}_{h_2}$ | $\text{sum}_{y_1}$ | $\text{out}_{y_1}$ | $E^{(i)}$ |
|---|---|---|---|---|---|---|---|
| 0 | 0.204800 | 0.551022 | 0.149200 | 0.537231 | 0.163238 | 0.540719 | 0.000829 |
| 1 | 0.204794 | 0.551020 | 0.149194 | 0.537229 | 0.160842 | 0.540124 | 0.000805 |
| 2 | 0.204788 | 0.551019 | 0.149188 | 0.537228 | 0.158480 | 0.539537 | 0.000782 |
| 3 | 0.204782 | 0.551017 | 0.149182 | 0.537226 | 0.156153 | 0.538959 | 0.000759 |
| 4 | 0.204776 | 0.551016 | 0.149176 | 0.537225 | 0.153859 | 0.538389 | 0.000737 |
| 5 | 0.204771 | 0.551015 | 0.149170 | 0.537224 | 0.151599 | 0.537827 | 0.000715 |
| 6 | 0.204765 | 0.551013 | 0.149165 | 0.537222 | 0.149371 | 0.537274 | 0.000695 |
| 7 | 0.204760 | 0.551012 | 0.149160 | 0.537221 | 0.147176 | 0.536728 | 0.000674 |
| 8 | 0.204755 | 0.551011 | 0.149155 | 0.537220 | 0.145012 | 0.536190 | 0.000655 |
| 9 | 0.204750 | 0.551009 | 0.149150 | 0.537218 | 0.142880 | 0.535659 | 0.000636 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 2288 | 0.204588 | 0.550969 | 0.148982 | 0.537177 | 0.000000 | 0.500000 | 0.000000 |

Table 12 lists every intermediate value that appears during forward propagation , shown in section 7.1, such as sums, activations, network output, and squared error—for the first ten training steps. The gradient formulas derived in Eqs. (7.7a)–(7.9) were evaluated at each step, and applying the update rule (7.6) with

$\alpha = 0.4$ produces the weight sequence in Table 11.

For each iteration the gradients are computed with the formulas in section 7.2. Substituting those gradients into (7.6) yields exactly the weight values in Table 11, completing the forward–backward cycle. The weight update from $i = 0$ to $i = 1$ in Table 11 matches the gradient step predicted by (7.6), confirming the correctness of both the analytic gradients and their implementation. Together, Tables 11 and 12 constitute the complete forward–backward loop for ten iterations.

## 7.11 Error Comparison Between Two Rounds

The error for the first round is

$$\mathrm{E}^{(0)} = \tfrac{1}{2}\big(y_1 - \mathrm{out}_{y_1}^{(0)}\big)^2,$$

giving:

$$\mathrm{E}^{(0)} = \tfrac{1}{2}\big(0.5 - 0.540719\big)^2 = 0.000829.$$

For the second round we have

$$\mathrm{E}^{(1)} = \tfrac{1}{2}\big(y_1 - \mathrm{out}_{y_1}^{(1)}\big)^2,$$

giving:

$$\mathrm{E}^{(1)} = \tfrac{1}{2}\big(0.5 - 0.540124\big)^2 = 0.000805$$

where $\mathrm{out}_{y_1}^{\mathrm{new}}$ is the output after the weights have been updated.

**Change in error.**

$$\Delta = \mathrm{E}^{(0)} - \mathrm{E}^{(1)} = \tfrac{1}{2}\Big[(y_1 - \mathrm{out}_{y_1}^{(0)})^2 - (y_1 - \mathrm{out}_{y_1}^{(0)})^2\Big]$$

$$\Delta = (\mathrm{out}_{y_1}^{(1)} - \mathrm{out}_{y_1}^{(0)})\,\Big[y_1 - \tfrac{1}{2}\big(\mathrm{out}_{y_1}^{(0)} + \mathrm{out}_{y_1}^{(1)}\big)\Big].$$

$$\Delta = E^{(0)} - E^{(1)} = 0.000829 - 0.000805 = 0.000024$$

**Did the error decrease?** If $\Delta > 0$ the error is decreasing; if $\Delta < 0$ it is increasing; if $\Delta = 0$ it is unchanged. Using the initial and final errors, hence the squared-error has *decreased* after the weights were updated and training proceeded to convergence.

## 7.12 Possible improvements to the Network

Our current neural network demonstrates the principles of forward and backward propagation, but we could further add enhancements that can improve its representational power, training dynamics, and applicability to real world tasks:

### Inclusion of Bias Terms.

The current architecture does not include bias parameters at both the hidden and output layers. Bias value allows each neuron to shift the activation function horizontally, which is especially critical when the optimal decision boundary does not pass through the origin. Including biases transforms the activation to

$$y = \sigma\left(\sum_i x_i w_i + b\right),$$

where $b$ is a trainable scalar bias. During backpropagation, the gradient with respect to the bias is simply:

$$\frac{\partial E}{\partial b} = \delta,$$

where $\delta$ is the local gradient of the neurons output with respect to its input.

### Deeper Architectures.

We have a single hidden layer with two neurons for this exercise, but more hidden layers or neurons can help with nonlinearities. This will be at the cost of increased training time and risk of overfitting.

### Activation Function Choices.

We are using the sigmoid activation function. This is useful for binary classification, but the downside is gradients that vanish for large inputs. Activations such as Rectified Linear unit (ReLU) or TanH function may offer faster convergence and better gradient flow for deeper networks.

### Regularization and Generalization.

Techniques like L2 regularization for weight decay, dropout, or batch normalization can also be used to improve generalization and for avoid overfitting.

### Weight Initialization Strategy.

The generated weights in this case are initialized randomly between 0 and 1. Initialization schemes, such as Xavier or He initialization, are preferred to maintain activation magnitudes and gradient stability throughout the network.

For our simple case studied here, these improvements are not strictly necessary, but they do become important as neural networks are scaled to handle complex, high-dimensional datasets.

## 7.13 Python implementation of forward and backward propagation

We implemented the complete forward and backward propagation steps in Python, in order to validate the mathematical derivations and for the iterative training. The implementation encapsulates each step of the neural network's training process in a structured and reusable way.

### Core Components

The neural network is implemented in a class called `NeuralNetwork`, with methods corresponding directly to the steps of the forward and backward propagation described in Sections 7.1 and 7.2.

#### Sigmoid activation function.

```
def sigmoid(x):
    return 1 / (1 + math.exp(-x))
```

Its derivative is computed using:

```
def sigmoid_derivative(x):
    s = sigmoid(x)
    return s * (1 - s)
```

#### The forward propagation includes:

- Computing net inputs for the hidden neurons (`compute_hidden_net`),
- Activating hidden neurons (`activate_hidden`),
- Computing the net input for the output neuron (`compute_output_net`),
- Final activation of the output (`activate_output`).

Once the prediction is made, the squared error is calculated:

```
def compute_error(y):
    return 0.5 * (y - self.out_y1) ** 2
```

**The backpropagation follows:**

- Output delta computed via chain rule (`compute_output_delta`),
- Hidden deltas propagated back (`compute_hidden_deltas`),
- Weight updates applied using gradient descent (`update_weights`).

### Training Iteration and Stopping Criterion

The method `iterate()` performs repeated training steps until the error falls below a user-defined threshold. Each iteration logs:

- Updated weights,
- Error value,
- Sums for all neurons,
- Activations,
- Derivatives for each weight.

### Usage and Output

A typical training run is initialized as:

```
nn = NeuralNetwork(
    w1=0.37, w2=0.95,
    w3=0.73, w4=0.6,
    w5=0.15, w6=0.15,
    alpha=0.4
)
x1, x2, target = 0.04, 0.20, 0.50
weights, error_dict, sums, outputs, derivatives = nn.iterate(x1, x2, target, delta_error=0)
```

The script prints:

- A table of weights and error per iteration,
- Forward sums and activations at each step,
- Gradient derivatives used for weight updates,
- A trend analysis of the error using the pandas.

### Verification

The calculations done for the first two iterations match Python output shown in the tables above:

- Initial and updated weight values (Table 11),
- Forward pass activations and error trend (Table 12),
- Gradient correctness verified via exact matches in $\partial E/\partial w_i$.

### Reference

The correct and full source code is included as a supplementary file named `neural_network.py`.

## References