UNIVERSITÄT
LIECHTENSTEIN

WIND SPEED FORECASTING USING NEURAL NETWORKS

Tomas Grosup

[Enrolment number]

Master's Thesis

University of Liechtenstein

This Thesis is submitted for the Master's degree in Data Science

November 2018

Supervisor: Dr. Johannes Schneider

ABSTRACT

The work is aimed at short-term wind speed prediction based on wind direction and wind speed using deep learning, in particular artificial neural networks. It is aimed only at wind speed prediction and describes different approaches towards forecasting weather with neural networks. Basics of neural networks are briefly described in the work as well as the current state of the art knowledge in weather prediction using artificial neural networks.

The experimental part of the work is aimed at benchmarking two simple topologies and one ensemble topology against each other with the aim to get a better performing model for a single spot forecast. The first topology introduces is a simple LSTM network learned on collected measurements, the second experimental topology is a Multilayer Perceptron network learned on WRF9km forecast outputs and real measurements. The third experimental topology introduced in this work is an ensemble neural network which uses both previous models as input.

ACKNOWLEDGMENTS

Thank you to my supervisor Dr. Johannes Schneider for support with writing this thesis and help with understanding neural networks, machine learning and time-series analysis. And to Pavel Hudeček for learning me the basics of signal processing in analog and digital electronics.

CONTENTS

LIST OF ACRONYMS **1**
1 INTRODUCTION **2**
2 RELATED WORK **3**
3 PROBLEM FORMULATION **6**
4 NEURAL NETWORKS **7**
4.1 BIOLOGICAL NEURON 7
4.2 MODEL OF ARTIFICIAL NEURON 8
4.3 ARTIFICIAL NEURAL NETWORK 11
5 TOPOLOGIES OF ARTIFICIAL NEURAL NETWORKS **12**
5.1 PERCEPTRON 12
5.2 MULTILAYERED FEED FORWARD NEURAL NETWORK 14
5.3 RADIAL BASIS NEURAL NETWORK 15
5.4 RECURRENT NEURAL NETWORK 16
5.5 LONG / SHORT TERM MEMORY 17
6 LEARNING A NEURAL NETWORK **20**
6.1 BACKPROPAGATION 21
6.2 TRAINING RNN WITH BACKPROPAGATION 22
6.3 BATCHING 23
6.4 OVERFITTING 24
6.5 DROPOUT 25
6.6 INITIALIZATION 26
6.7 MODEL COMPLEXITY 26
6.8 COMBINATION OF DIFFERENT MODELS 27
7 OPERATION OF ELECTRONICS IN ANEMOMETERS **29**
7.1 VOLTAGE DIVIDER 29
7.2 OPERATIONAL AMPLIFIER AS COMPARATOR 30
7.3 A/D CONVERTER 31
7.4 OPTICAL ROTATION SENSOR 32
7.5 MAGNETIC ROTARY SENSOR 33
8 NUMERICAL WEATHER PREDICTION **34**
8.1 GLOBAL MODELS 35
8.2 LOCAL MODELS 35
8.3 THERMAL CONVECTION 35
8.4 INACCURACY IN MODELS 36
9 WIND DATA SPECIFICS **36**
9.1 MEASUREMENTS 37
9.2 WRF 9KM FORECAST 37
9.3 COMPLEX TERRAIN EXAMPLE – LAKE GARDA 38
9.4 FLAT LANDSCAPE EXAMPLE - ZLONCICE 41
9.5 ERROR CAUSED BY ANEMOMETERS WITH GENERATORS 44
9.6 ERROR CAUSED BY A/D CONVERTER IN THE MEASUREMENT HARDWARE 44
9.7 ERRORS CAUSED BY LOW SAMPLING RATE IN MICROCONTROLLERS 45

9.8 Errors caused by downsampling in optical and magnetic sensors 46
9.9 Other sources of error 46
9.10 Interference of resolutions 46
9.11 Weibull distribution 47
9.12 Error in downsampling mathematics 48
10 FORECAST WITH FULLY CONNECTED LSTM 49
10.1 Network topology 50
10.2 Training 52
10.3 Results 53
11 MODIFICATION OF NWP WITH MLP 54
11.1 Network topology 54
11.2 Training 61
11.3 Results 62
12 ENSEMBLE MLP NETWORK 62
12.1 Network topology 63
12.2 Training 64
12.3 Results 65
12.4 Reliability of forecast for different time horizonts 65
13 IMPLEMENTATION INTO SERVER WINDGURU.COM 66
13.1 Windsurfing forecast windguru. com 66
13.2 Reliability of forecasts 66
13.3 User interface 66
13.4 Proposed architecture 67
14 POSSIBLE IMPROVEMENTS 68
15 FURTHER RESEARCH 69
16 CONCLUSION 71
17 REFERENCES 74
A. SOURCE CODE 78
A.1 Neural network 78
A.2 Data pre-processing 80

LIST OF ACRONYMS

NWP	Numerical Weather Prediction
ANN	Artificial neural network
RMSE	Root mean square error
NRMSE	Normalized root mean square error
MAE	Mean absolute error
NMAE	Normalized mean absolute error
MLP	Multilayer perceptron
A/D	Analog to digital
LSTM	Long Short-Term Memory

1 INTRODUCTION

Precise short-term wind speed forecasting is a complex problem that needs solution which could be used across variety of different industries spanning from renewable energy across air transport all the way to sport and competition activities like sailing, windsurfing or kite surfing.

There are many different approaches towards weather prediction using artificial neural networks. Some researchers try feeding feedforward networks with input vectors containing historical wind speed values and single or multivariate forecast outputs. Other approaches include applications of LSTM networks as time series predictions or convolutional networks for post processing NWP forecasts.

A most common approach towards wind speed forecasting comes from numerical weather prediction (NWP). NWP models are usually mesoscale to globalscale meteorological models computed using a combination of computational fluid dynamics and probabilistic equations. Post-processing of NWP forecast outputs using different approaches or using neural networks is an interesting way of using neural networks in weather prediction.

The major drawback of numerical weather prediction lies in the fact that forecasts do not correspond to reality from various different reasons. A forecast for a particular geographical location is very imprecise using the unmodified outputs from globalscale or mesoscale numerical weather prediction models. Nevertheless, outputs from NWP forecasts are valuable basis in making a local prediction precise, bebause they include probabilistic and deterministic calculations of phenomena and events which are affected by surrounding larger scale meteorological measured events. Outputs of global scale models like GFS or ECMWF are possible to access and process due to their wide industry usage. Numerical weather prediction forecasts, even though, very rough in the sense of its outputs and the reliability, correlate well with atmospheric trends that affect local weather conditions.

The main goal of this work is to show that numerical weather prediction models can be recalculated to reflect wind speed course in very specific geographical locations using neural networks. For windsurfing, sailing, kiting and all other kind of wind powered water sports specific places exist that exhibit stable wind conditions on the sea as well as on lakes. Especially in the middle of Europe, the wind powered water sports are dependent for its existence on lake spots with working thermal winds that bring the sports to life not only on the coastline but also inland.

The question stated before writing this research paper was whether thermal winds prediction can be based on forecasts from numerical weather prediction models that don’t include the features causing the thermal winds to occur based on predictors from numerical weather prediction models. Since numerical weather prediction models are based on the measurements of different atmospheric variables around the globe, it should be possible to recalculate those rough predictions of numerical weather prediction models to higher resolution reaching better precision. In this paper, an approach was chosen that includes machine learning models, specifically deep learning and neural networks.

Different theoretical approaches are presented in the area of deep learning. The basic function of an artificial neuron is explained as well as different neural network topologies including perceptrons, feedforward networks, recurrent networks radial basis networks, and special gated units namely LSTM units and fully connected stacked LSTM networks.

Further principles of measuring of meteorological variables are examined for the perspective of electrical sensors and signal processing of information produced by these sensors. A brief insight into electronics used for post-processing of signals from measurement sensors and devices is introduced especially in the area of digital cup anemometers. Digital cup anemometers are devices capable of measuring wind speed. Measurement of wind speed using cup anemometers brings a couple of problems into the area of signal processing. Cheap electronic parts and software mistakes cause anemometers to be imprecise in measurements. Resolution and preciseness of anemometers is an important precondition towards further signal processing like processing in machine learning for obtaining predictions.

In the experimental part of this paper, three different neural network architectures are examined. First is simple LSTM network used for prediction based on past measurement values. Second is a simple MLP network that transforms WRF forecast outputs into more precise predictions. A third neural network is an ensemble network that makes better predictions based on the two previous ensembles.

Results of the experiments are conducted on four different datasets which represent four different locations. Very different locations with very different wind characteristics and wind speed distributions were chosen to benchmark the proposed three different architectures. The four different datasets include measurements and WRF forecasts outputs from windsurfing spots where local thermal winds strongly affect the measurement. Measurements in places suitable for windsurfing correlate very badly with the forecasts. Other datasets where WRF forecast predictors correlate very well with measurements were chosen as well to compare results of the experiments. Proposed architectures were tested on them in order to show the capabilities on neural networks in weather prediction.

2 RELATED WORK

Contemporary research concludes, especially in the area of wind forecasting different models together perform better than one particular model, either using ensemble approach or hybrid architecture. Current research presents different models of wind speed prediction or power output prediction which seems to report probabilistic dependency of variables which are presumed to have deterministic

dependency. The contemporary research uses mostly RMSE or MAE metrics to compare the model performance of different ANN architectures or approaches. Only a very narrow spectrum of authors point out the specifics of wind forecasting in the context of learning algorithms or algorithm performance measures. The most interesting approach found in this literature review is the combined LSTM and convolutional nets approach, which enables to keep extracted features from geospatial weather information across time steps in the LSTM memory cell inside a recurrent neural network for processing.

Only a few pieces of research mention or speculate, where the source of error in prediction of wind speed exactly comes from. In fact, the character of the prediction problem disallows the experiments to be well comparable while different sources of error enter the measurements.

The work of Cyril Voyantab, Marc Musellia, Christophe Paolia and Marie-Laure Niveta introduces a hybrid ARMA ANN model for radiation prediction based on statistical method ARMA and transformation of numerical weather prediction outputs from ALADIN model using MLP. From the experiments conducted on four different time series from four different geospatial points the combined ARMA/ANN models performed the best measured by RMSE. (Voyant, 2012)

Xingjian SHI, Zhourong Chen, Hao Wang and Dit-Yan Yeung in their work on precipitation forecasting suggest the use of modified LSTM a convLSTM network that enables a spatiotemporal sequence forecasting using the same principle a fully connected LSTM enables to remember and predict based on historical data but the input sequence fed into the convLSTM network as a 2D vector enables to improve the forecasting by leveraging the information contained in geospatial data inputs. (Shi, 2015)

Work of Stephan Rasp and Sebastian Lerch on post-processing ensemble weather forecasts showed that using neural networks for distributional regression post-processing of ensemble weather forecast is not only possible but also more computationally effective than other state-of-the-art techniques for post-processing ensemble weather forecasts. Results of the experiments conducted indicate that in order to provide a forecast that performs well, encoding of local information is vital especially in temperature forecast. Researches as well showed that with a systematic analysis of the proposed correcting systems based on ANN better insights into how the corrections systems should work. (Rasp, 2018)

Jing Zhao, Yanling Guo, Xia Xiao, Jianzhou Wang, Dezhong Chid and Zhenhai Guo report that the use of forecast mechanism based on optimized association method to forecast wind speed and power output from wind-powered generators based on single value WRF input show strong model uncertainties and larger dispersion of forecasting error. Interestingly the experiments showed that a lower forecasting error for wind speed does not always lead to lower forecasting error for power prediction, suggesting that conversion of wind speed to power includes a degree of uncertainty. (Zhao, 2017)

In a research on spatiotemporal forecasting using LSTM networks Amir Ghaderi, Borhan M. Sanandaji and Faezeh Ghaderi used a RNN with LSTM blocks which predict wind speed on multiple locations from which windmill measurements were collected. Experiments were conducted on 57 windmills spread on a terrain with low terrain roughness where the mutual correlations were relatively high. (Ghaderi, 2017)

A interesting research using convolutional networks was introduced by Pablo Rozas Larraondo, Inaki Inza and Jose A. Lozano to predict precipitation using forecast outputs from a numerical weather

prediction model namely ERA-interim dataset. The research showed that it's possible to learn a convolutional network to make corrections to a numerical weather prediction outputs based on geospatial inputs. Researchers further showed using class activation mapping, that the activations that predictions were based upon are from surrounding area near to the to the forecasted point of interest in. But the accuracy of the suggested convolutional model did not significantly outperform simple heuristics in the sense of precision, but showed that the most relevant informations to correct numerical weather prediction outputs originate in the data from the gospatial surrounding of a particular spot which is being forecasted. (Larraondo P. I., 2017)

A collective of authors Abinet Tesfaye Eseye, Jianhua Zhang, DehuaZheng, Han Li and Gan Jingfu showed in the research paper revolving around a hybrid PSO-ANN model, used for short-term wind power prediction, that combination of particle swarm optimization and ANN wind power forecasting post-processing brings improvements into predictions of wind power generation as well as to wind speed prediction. The research further shows a significant difference in prediction precision for different yearly seasons using the hybrid PSO-ANN approach. (Eseye, 2017)

Matthias Fetzner in his work concentrated the research efforts not only to wind power or wind speed prediction but also to network load prediction. He showed, that a multivariate approach is preferable to iterative approach because of the error accumulation. Interestingly he managed in his work to reach relatively low RMSE for ARX model as well as for ANN model. (Fetzner, 2015)

Honglu Zhu, Xu Li, Qiao Sun, Ling Nie and Jianxi Yao and Gang Zhao concentrated their research to exploring wavelet decomposition of wind speed time series for the use in processing by an artificial neural network. Experiments with wavelets were carried out on power output series from windmills as well as on wind speed measurements obtained by using anemometers. The experiments have shown that the use of wavelet decomposition in combination with ANN processing improves the forecasting performance. (Zhu, 2015)

Another study in post-processing a WRF model forecasts was conducted by Philippe Lauret, Hadja Maïmouna Diagne and Mathieu David. The study proposes a MOS correction technique based on ANN applied to 24 hours ahead WRF forecast geospatial mean on a square of multiple WRF output points to remove a bias factor in the WRF forecast but the study showed that a single point approach, as well as the square space average over an area, produced similar results. The study suggests tracking the source of error in ANN systems in order to identify the source of the error from different input variables. (Lauret, 2014)

A collective of authors Athraa Ali Kadhem, Noor Izzri Abdul Wahab, Ishak Aris, Jasronita Jasni and Ahmed N. Abdalla in study aimed at forecasting based on combination of ANN and Weibull distribution derived a conclusion out of conducted experiments that using Weibull distribution to randomly predict weather data based on different parameterizations for different yearly season and later post-processed using ANN to better match measurements is more successful than using Weibull distributions only. The study also proposes that parameterization of wind speed data distribution changes based on the seasonality. (Kadhem, 2017)

Ümmühan Başaran Filik and Tansu Filik used an ANN to feed a time frame of historical data into the inputs of ANN trained on measurements to create a one-point prediction on two different locations. The study suggests that including a historical data of three different variables wind speed, temperature and pressure performs better than sub-selection of only two or one of those. Performance

of the models in the experiments was measured by MAE and RMSE as standard metrics for model performance. (Filik, 2017)

V. Ranganayaki and S. N. Deepa constructed and measured the performance of ensemble neural network that creates forecast based on four different sub inputs consisted from four ANN with applied different learning algorithms. The ensemble sub inputs are four different neural networks either learned using different algorithm or initialized differently. The conducted study showed that using the ensemble neural network approach contributes to minimization of the error. (Ranganayaki, 2016)

Francesco Castellania, Massimiliano Burlandob, Samad Taghizadehc, Davide Astolfia, Emanuele Piccionia in a research focused on experimenting with a hybrid model consisting of ANN and equations based on computational fluid dynamics conducted that using SCADA data from 8 turbines under investigation in a relatively complex topographic terrain does not lead to any significant improvements in modeling the turbulent effect in a complex terrain. Concluded was that the joined model does not perform significantly better than a simple ANN. (Castellania, 2014)

3 PROBLEM FORMULATION

Precise wind speed prediction is difficult while the core of the problem is relatively complex in its nature. Numerical weather prediction models using computational fluid dynamics and probabilistic equations are relatively good at the prediction of wind speed for large areas but because of their relative computational complexity the models are calculated in relatively low resolution and initialized only few times a day. For the purpose of precise wind speed prediction in places where local features are not entering the calculation in the numerical weather prediction models, like specific mountain terrain or a presence of industry creating turbulence in the natural wind flow a more precise way of forecasting is needed.

Wind speed prediction helps to better forecast how much power can be generated by generators in the windmills and helps to optimize the power possible to produce in the future, which plays in the operation of wind power plant as a significant economic factor. This work though is aimed strictly at wind speed prediction and does not consider power output forecasting at all.

Online wind forecasting services available for wind-based sports like windsurfing, kiting or sailing mostly origin from predictions of numerical weather prediction models like GFS, WRF of more precise models like COSMO and other. Specifically in locations where certain local factors like topology multiplicates the effects of global winds or local topological features cause periodical thermal wind occurrences the low-resolution numerical weather prediction models significantly

underperform which harms the local tourism, while the users of weather forecasting services often tend to think that weather forecast services display accurate data for wind sports spots.

The availability of cheap wind sensors and hardware to process archive and share the data from sensors enabled networks of weather measurement stations to get developed.

There are many available works on prediction of wind from either historical data as simple time series prediction with and without the use of artificial neural networks and some available on correction of outputs of numerical weather prediction using neural networks, but there are not many available on using both approaches to increase the precision of weather prediction using both of above mentioned.

4 NEURAL NETWORKS

The theory of neural networks origins from neurobiophysics and tries to explain the functions and principles of information processing in neural cells. Work of Warren McCulloch and Walter Pitts from 1943 is considered as origins of neural networks discipline. In this chapter the principles of neurons and neural networks are introduced.

Artificial neuron adopts the biological neuron principles, therefore it's important to shortly explain the functionality of biological neuron.

4.1 Biological neuron

Neurons are considered being the building elements of neural system. They are autonomous specialized cells, which are developed to transfer, process and store information, which are vital to carry out the functions of living organisms. The structure of biological neuron is shown on fig. 1.1

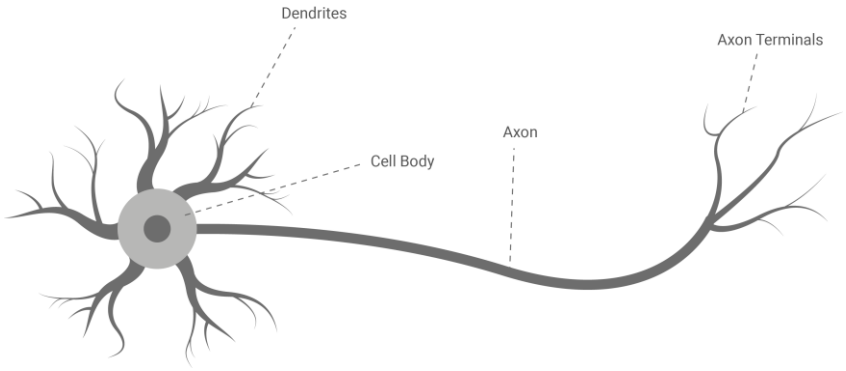


Figure 1. 1: Biological neuron (Jiang, 2016)

The typical neuron consists of a body from which number of shorter spurs (dendrites) and one longer spur (axon) protrude. The size of the cell body is approximately in units of micrometers. Dendrites are in the units of millimeters and axons can be from few centimeters up to one meter long. Axon of every neuron is ended by a synapse which connects to other neurons. A neuron receives impulses from other neurons through synapses, in case a whole membrane potential crosses a certain threshold, the neuron is activated through its synapse and starts to affect other connected neurons.

Typically axons from one neuron are in contact with dendrites from another neuron and form a structure which is called synapse.

Hinton uses interesting approach towards describing biological neurons functionality, he mentions that a spike of activity traveling along the axon causes charge to be injected into a postsynaptic neuron at a synapse. A neuron generates spike when it receives enough charge in its dendrites to depolarize a part of neuron body called the axon hillock. And when the axon hillock gets depolarized the neuron sends a spike along its axon. Synapses contain little vesicles of transmitter chemical which are able to migrate to the surface of a synapse when the axon receives a charge. The transmitter molecules diffuse across the synapse and bind to receptor molecules in the membrane of the post-synaptic neuron and create holes in the membrane which allows only specific ions to flow in our out of the neuron body and thus polarize or depolarize it. (Hiton, 2016)

4.2 Model of artificial neuron

The model of artificial neuron represents an abstraction of the mechanism how neural cells process information. It's not possible to reproduce a precise function of real neuron, while biological neurons are rather complicated objects and so far not all the mechanisms are yet fully known. Models of artificial neurons, which are currently most often used, implement a basic function of the artificial neuron.

Mathematically neuron's basic function is the following mathematical formula:

$$f(q) = f(\sum_{i=1}^{n} x_i w_i)$$

Where:

n – is number of inputs
x_i – is i-th input signal of a neuron
w_i – is i-th input weight
f – is the activation function of a neuron
q – is the neurons potential

A formal neuron has n real inputs, which determine the inputs vector $x = (x_1, \ldots, x_n)$. These inputs are rated by their synaptic weights, which consist of a vector $w = (w_1, \ldots, w_n)$. On fig. 1. 2 an artificial neuron model with a bias on the output is shown. Mathematical formula for a neuron with bias is:

$$f(\sum_{i=1}^{n} w_i x_i + b)$$

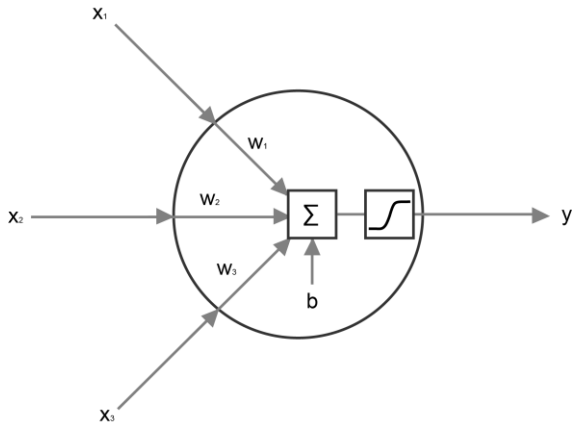
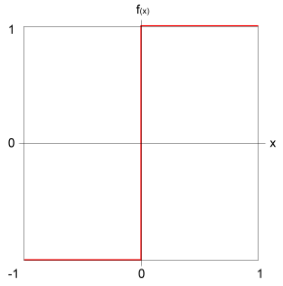


Fig 1. 2: Formal neuron with inputs (x) weights (w) sigmoid output function and output (y)

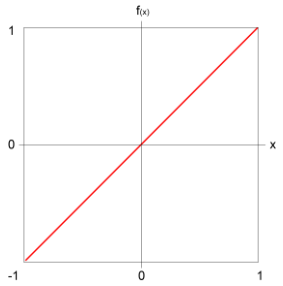
To get an output from a neuron it's necessary to activate the activation function f. A number of activation functions are used for artificial neurons, the most frequently used are listed below:

- Binary functions: $f_{(x)} = \text{sign}(x)$ shown on fig. 1. 3 A)
- Linear functions: $f_{(x)} = x$ shown on fig. 1. 3 B)
- Sigmoid functions, for example binary sigmoid $f_{(x)} = 1 / (1+e^{-x})$ shown on fig 1. 3 C)
- Gaussian functions $f_{(x)} = e^{-x2}$ shown on fig 1. 3 D)

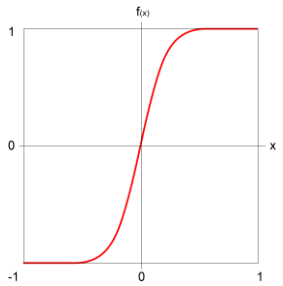
For a discontinuous activation function a fix threshold is set, upon which the containing value of a neuron is calculated (in this case 0 or 1). In case of linear activation function, the weighted inputs are transferred on the output without alternation. Sigmoid activation functions are asymptotical (to 0 and 1 in this case) and enable to accomplish sensitivity for lower signals but on higher signals, their sensitivity decreases.



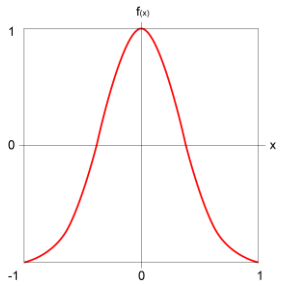
Binary threshold A)



Linear B)



Sigmoid C)



Gaussian D)

Figure 1. 3: Different types of activation functions

4.3 Artificial neural network

Every artificial neural network consists of neurons interconnected in such a way that the output of one neuron is an input of another neuron. The number of neurons and their mutual interconnection determines the topology of an artificial neural network. A neural network is characterized by different properties.

The most basic neural network is feedforward network. All the input signals in feedforward network proceed from the input layer to the output layer without reverse feedback connections. By recurrent neural networks the input of every neuron is dependent on the output of a previous cycle. An example of a neuron with a recurrent topology is illustrated on fig. 1. 4.

Artificial neurons are organized in layers, where layers are filled with neurons with common characteristics. An artificial neural network consists minimally of two layers, input layer, and an output layer. Layers in between input and output layers are called hidden layers.

The purpose of the input layer is to distribute input signals to other layers. Neurons in input layers have only one input, which is also an input to the neural network and their input function is linear, which means that input layer neurons transfer the input signal on their output without any change.

Output layer determines the output of a neural network. Layers between the input and output layer are called hidden because their inputs and outputs don't directly interfere with the outside environment.

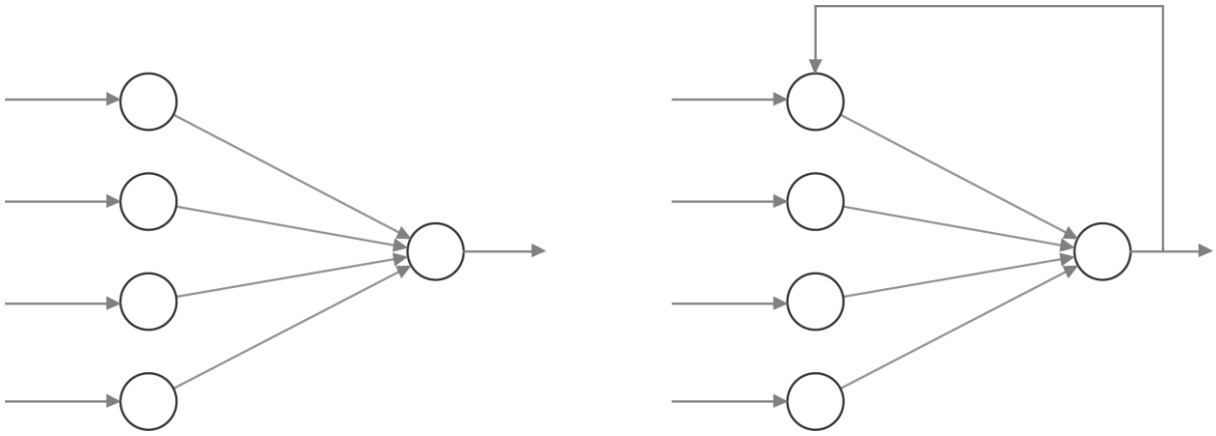


Figure 1. 4 Example of a network without and with feedback topology

An activity of neural network can be perceived as a transformation of an input signal to the output signal and it's possible to define by three different features of a network:

- Activation functions of neurons
- Set of connections between neurons
- Input weights

The activation function is chosen in such a way that the neural network is capable to map as much input and output relations as possible. We can speak about a stable neural network when all the weights in the learning process converge to an equilibrium in which the model approximates the actual function of a process. In case neural network diverges it's considered unstable.

5 TOPOLOGIES OF ARTIFICIAL NEURAL NETWORKS

5.1 Perceptron

The simplest topology of an artificial neural network is a perceptron. It was originally designed as a model of visual system by Frank Rosenblatt. A typical perception consists of n inputs ($x_1, x_2, \ldots, x_n$) and one output neuron which is connected to all its inputs.

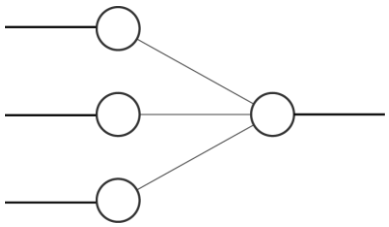


Figure 1. 5: Simple perceptron with three input cells and one output cell

Inputs contain binary values of ether 0 or 1. Every connection between input and output neuron carries a weight ($w_1, w_2, \ldots, w_n$). The formula to calculate inputs to outputs is the following:

$$in(t) = \sum_{i=1}^{n} w_i(t)x_i(t) - \Theta$$

For learning a perceptron following algorithm is used: (Lagandula, 2018)

- Setting the weights to random values and setting the threshold
- Activation of an input vector to the network
- Calculation of the real output value
- Comparison of the output value to the expected value
- Adaptation of weights and calculation of the error function as a difference between expected and the calculated value
- If the input vector is evaluated correctly, the weights are not altered and another input vector is processed
- If the input is 1, but should be 0, weights on active inputs are decremented
- If the input is 0, but should be 1, weights on active inputs are incremented
- Alteration of the threshold by adding error function

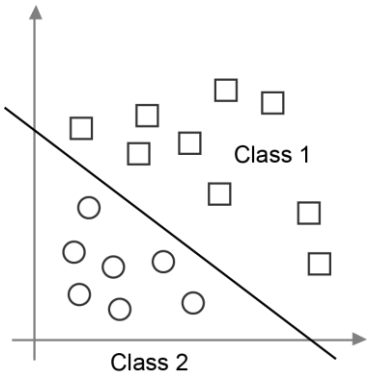


Figure 1. 6 Example of linearly separated set

Perceptrons with only one layer are only capable of learning linearly separable patterns. Example of a linearly separated set is illustrated on fig. 1.6. More complicated functions like XOR, which are not possible to solve using a perceptron is solvable by extending the basic model of perceptron to a multilayer feed forward neural network or multilayer perceptron.

Perceptrons were made famous by Frank Rosenblat who published a book called "Principles of Neurodynamics". The piece of the book, which was most important, was the learning algorithm. After the work of Rosenblat was published, it was said that they could differentiate between pictures of tanks and pictures of trucks out of which both were partially obscured in a forest. Later it turned out that not all of the claims earlier mentioned about the capabilities of perceptons met their expectations. As the experiments were conducted on a dataset of pictures prepared specially for the experiments it later turned out that perceptons could tell the difference between tanks and trucks based on different attributes in the picture than shape or look of a truck of tank because in the experimental dataset all pictures of the tanks were taken on a sunny day and pictures of the trucks were taken on a cloudy day. (Kaufman, 2015)

Everything the percepton was actually doing was measuring the overall intensity of pixels in the picture simply because a perceptron can learn something like adding up intensity of pixels in an image. (Kosinov, 2017)

It turned out that perceptons are limited in what they can do. In a paper published in 1969, two researchers mentioned what the limitation of percepton is. Limitation of what computations it can do or what problems it can solve. After this paper being published the general opinion on neural networks was rather negative in the manner that many people misjudged, that results of the research on limitation of perceptons, are valid for all neural networks. It was thought that neural networks, in general, could not learn difficult things. But in fact, the work of Minsky and Papert from 1969 showed that perceptron and its learning algorithm can't learn difficult problem. (Wikipedia, 2018)

Hinton mentions an interesting analogy of how the process of creation of computer programs resembles a learning algorithm of perceptons and neural networks in general. A standard way how to recognize patterns exists, that corresponds to a way how neural networks function and how the process of learning reflects the creation of a pattern recognition of which a computer program is a final output:

Feature activations that correspond to part of the system that affects the output of the algorithm judged based on common sense are produced and coded into a computer program. At that point there is no learning involved, the initialization of feature activations is done by hand based on knowledge of the principles using a simple common sense process. The system is observed and the decision which features should be included in the computer program and how they should be processed is made based on the observations. During the process that corresponds to learning in neural networks new alternative features are included in the system depending on whether they work or not new features are included in the system until a final result is reached. The system ends up with some features that allow it to solve the problem. What corresponds to the learning process is the problem of how to weight each feature activation in order to get the right scalar output. So that the weights of features are actually an evidence of the recognized pattern. And when the sum of the weighted features crosses some threshold it's claimed that the total evidence is in favor of our hypothesis. So if the sum is above a certain threshold it is then obvious that the sample lies in the class of patterns being recognized. Percepton in its sense is a pattern recognition system. A perceptron contains similar characteristics like pattern recognition system in the sense that it consists of inputs which are converted to feature activities using a system that resembles neuron without its learning algorithm part. So the part where perceptron resembles a statistical pattern recognition system is the part where inputs are simply converted into features. As long as the activities of features are set, the weights setting comes into play. Once the weights are learned feature activities times weight of the inputs are summed up and it is decided whether a sample lies in a class that is of interest. (Hiton, 2016)

5.2 Multilayered Feed Forward neural network

Multilayered Feed Forward neural network was introduced in the year 1986 and the authors are G. E. Hinton, E. Rumelhart a R. J. Williams The goal of this neural network was to solve the problem of the limited computational capacity of perceptrons. In feed-forward networks signals are running in the network only forward from a layer to layer and those networks are consisted of minimally three layers (input layer, hidden layer/s, output layer).

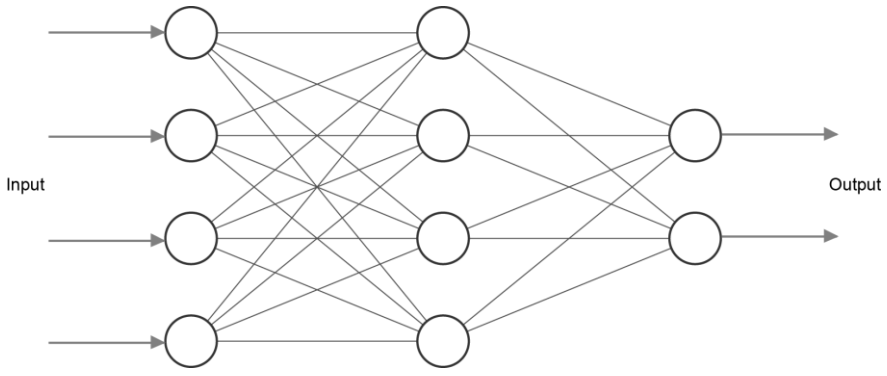


Figure 1. 7 Multilayered Feed Forward neural network

Connections in between single neurons are done in such a way that every neuron in the hidden layer is connected to all the neurons from a previous layer and to all the neurons in the next layer. An example of this type of neural network is illustrated on fig. 1.7

For learning multilayered neural networks the most commonly used algorithm is backpropagation. In backpropagation, the error is propagated in altering the neuron weights. This method was published in the 1980s and its author is E. Rumelhart. Backpropagation is based on altering the synaptic weights calculated from the error of the output.

For learning a neural network two sets of vectors are needed. The first vector characterizes the values that form the input and the second vector their respective output values. On the start, input vector is supplied on the input to a neural network and after passing through the neural network, the output is compared to the expected output. In case these two values are not the same an error is calculated backward to previous layers, which causes the alteration of the synaptic weights. The whole process has an iterative manner in which a each time a new input vector is supplied to the network and the whole upside down process repeats itself. (Mazur, 2015)

5.3 Radial Basis neural network

Radial basis neural networks are networks with three layers of neurons and feedforward topology. A special feature of these networks is the characteristics of the hidden neurons, which is radially symmetric. In applications are Radial Basis networks a serious competitor to Multilayer Perceptron networks. Historically they appeared later and come up with a series of advantages mainly in learning speed. (Devaraja, 2002)

Radial basis function network with the illustrated neurons with radial activations is illustrated in fig. 1.8.

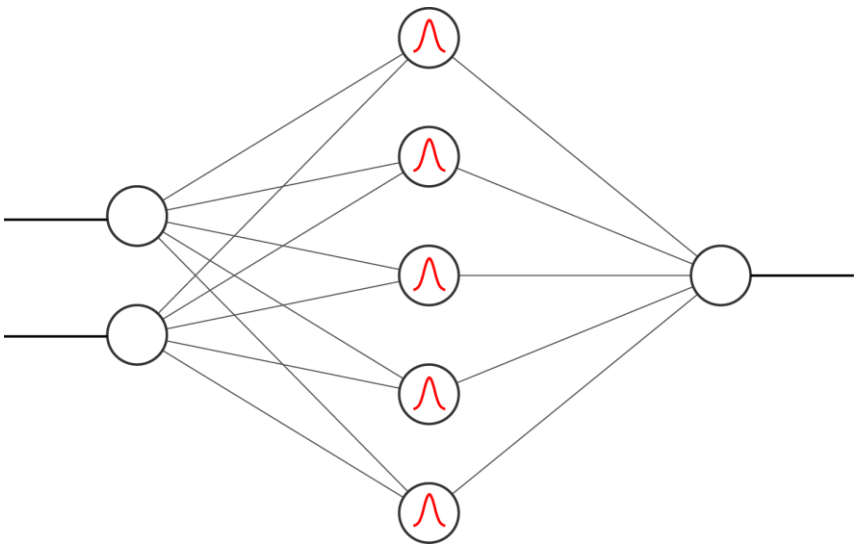


Figure 1.8 Radial basis network with an input layer and a hidden layer with radial activations and output layer

The three layers of Radial Basis network consist of input layer, hidden layer and output layer. Hidden layer is consisted of neurons with Gaussian activation function. Input neurons and hidden neurons are fully connected. In between hidden layer and output layer all the neurons are also fully connected.

As the output function of the hidden neurons, Gaussian function, known widely from statistics, is mostly used. Gaussian function is suitable for this network because it expresses the similarity of the input to its prototype. The similarity is thought expressed in the terms of Euclidean metrics not in the terms of similarity for a human eye, even though in many cases both tend to be the same.

Neurons in the input and the output layer are perceptrons. The main task of the perceptrons in the output layer is to add the inputs of the hidden neurons in such a manner that the output function would be approximated as closely as possible.

Radial basis network as approximator

Radial basis networks work well as approximators as well as classifiers. For approximations, Radial basis networks with continuous output functions of hidden neurons are used. The approximation is based on the fact that two arguments which lie in the output space close to each other would have also similar functional value. Radial basis neurons thus work as representatives of clusters of mutually similar patterns. Their activity is proportionately equal to the distance from the prototype. A correct setting of centers of the Radial Basis neurons is the first and very important phase of learning. Output neuron connects the outputs of the RBF neurons and assign them a corresponding functional value. Setting and optimization of the output weights is done in the second learning phase. (Guillén, 2007)

Radial basis networks as classifier

Radial basis network used as classifier can use continuous as well as discrete output functions of the RBF neurons. Continuous output functions carry the information about a rate with which is the given pattern classified. On the other side, complementary rules have to set other for evaluating the output. The simplest is a threshold value.

Classification is based on the distance of patterns. Similar patterns have shorter distance and thus fall in the same category. Centers of the RBF neurons are set in such a way so that RBF neurons are capable to represent clusters of patterns without having the knowledge about classification of the patterns into classes. Outputs of the network represent the classes into which the patterns are classified. For networks with continuous output functions classifications is just a special case of approximation, where required output values are either 0 or 1 and not in the interval (0; 1). (McCormick, 2013)

5.4 Recurrent neural network

In contrary to Feed Forward neural networks, where connections in between neurons take only one direction, the Recurrent neural networks contain at least one cyclic connection.

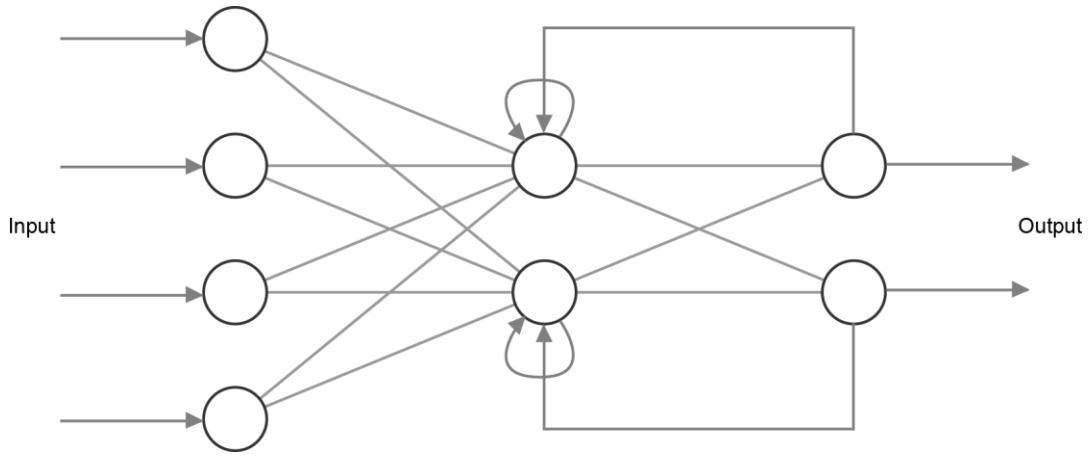


Figure 1.9 Example of Recurrent neural network

Recurrent neural networks are similar to Feedforward networks consisting of neurons divided to input, hidden and output layers, but they contain connections running backwards through the network. An example of a recurrent network is shown fig. 1.9.

Learning of recurrent neural networks is mostly a supervised learning process. The network is learned on a data, in such a way, to be capable to generalize inputs based on similarities or common characteristics of data used for learning. Currently, more learning algorithms are used for learning recurrent networks, for example, recurrent learning in real time, backpropagation through time or extended Kalman filtering. (Brownlee, A Tour of Recurrent Neural Network Algorithms for Deep Learning, 2015)

5.5 Long / Short Term Memory

Long Short Term Memory (LSTM) unit was designed in 1997 by researches S. Hochreiter a J. Schmidhuber as a response to a problem of vanishing gradient. Recently LSMT networks are one of the most frequently used networks. LSTM network is a recurrent network equipped with a special gate mechanism, which controls the access to inner state of the cell. Gates can prevent rest of the network to alter the inner state of the cell when propagating the error back and the gradient doesn't diminish so fast compared to conventional recurrent networks. (Weber, 2017)

Inner state in LSTM cell in every step is determined by a number of factors, which either write or read information of the current state using the closures. Gates are implemented using a sigmoid activation function in the interval of 0 to 1 which determines if the information is let through or not. 0 means no signal passes through and 1 means that everything passes through.

The first forget gate f_i^t in time t and cell i determines which information will get lost from the inner state of the cell and is calculated using the following formula:

$$f_i^t = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^t + \sum_j W_{i,j}^f h_j^{t-1})$$

Where σ is the sigmoid, x^t is current input vector and h^t is an actual vector which contains the output of all LSTM cells. The threshold value is b^f, U^f is input weight and W^f are weights in between the recurrent layers.

Second gate g^t_i or the input gate is calculated similarly like the first gate but contains its own set of parameters:

$$g_i^t = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^t + \sum_j W_{i,j}^g h_j^{t-1})$$

This gate participates on decision which information is passed through to save to the inner state. The inner state s^t_i is then updated with the following formula into which aside from gates f^t_i and g^t_i enters also the previous state s_i^{t-1}

$$s_i^t = f_i^t * s_i^{t-1} + g_i^t \sigma(b_i + \sum_j U_{i,j} x_j^t + \sum_j W_{i,j} h_j^{t-1})$$

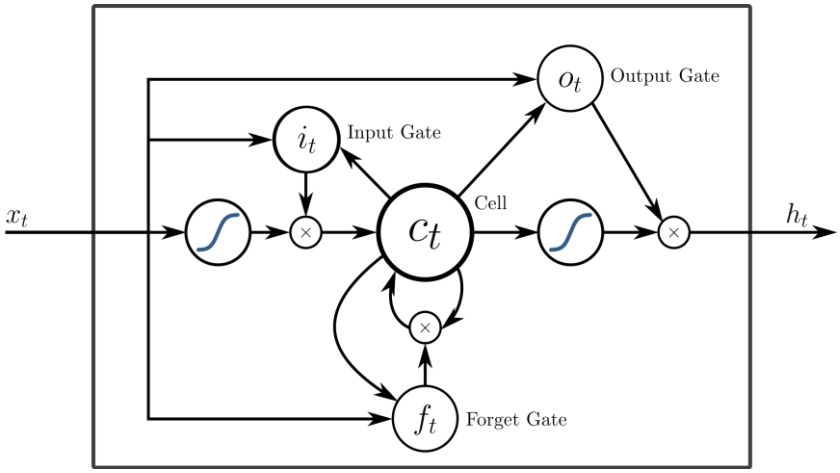


Figure 1.10 LSTM cell (Wikipedia, Long short-term memory, 2018)

The output of LSTM cell is influenced by the last output gate, which determines which parts of inputs are passed through. The formula of an output gate is similar to the previous two:

$$q_i^t = \sigma(b_i^q + \sum_j U_{i,j}^q x_j^t + \sum_j W_{i,j}^q h_j^{t-1})$$

The output is calculated with the following formula:

$$h_i^t = tanh(s_i^t) q_i^t$$

We could notice interesting insight into what is exactly happening in LSTM cells in an experiment published by Graves and Schmithuber where LSMT network was used to recognize written characters based on inputs of continuous stream of the position of a pen on a paper. So each time step the LSMT in the experiment is fed with a simple 1D vector of two input variables corresponding to the pen

position on the paper, which is shown on fig. 1.11. In the same picture, we can see the inner states of the LSTM cells and their values represented in shades of grey corresponding to the normalized value converted to RGB with all channels of the same normalized value in the second row. The last row is a representation of what the network looks at in the moment the decoder is outputting the character recognized by the network. (Hiton, 2016)

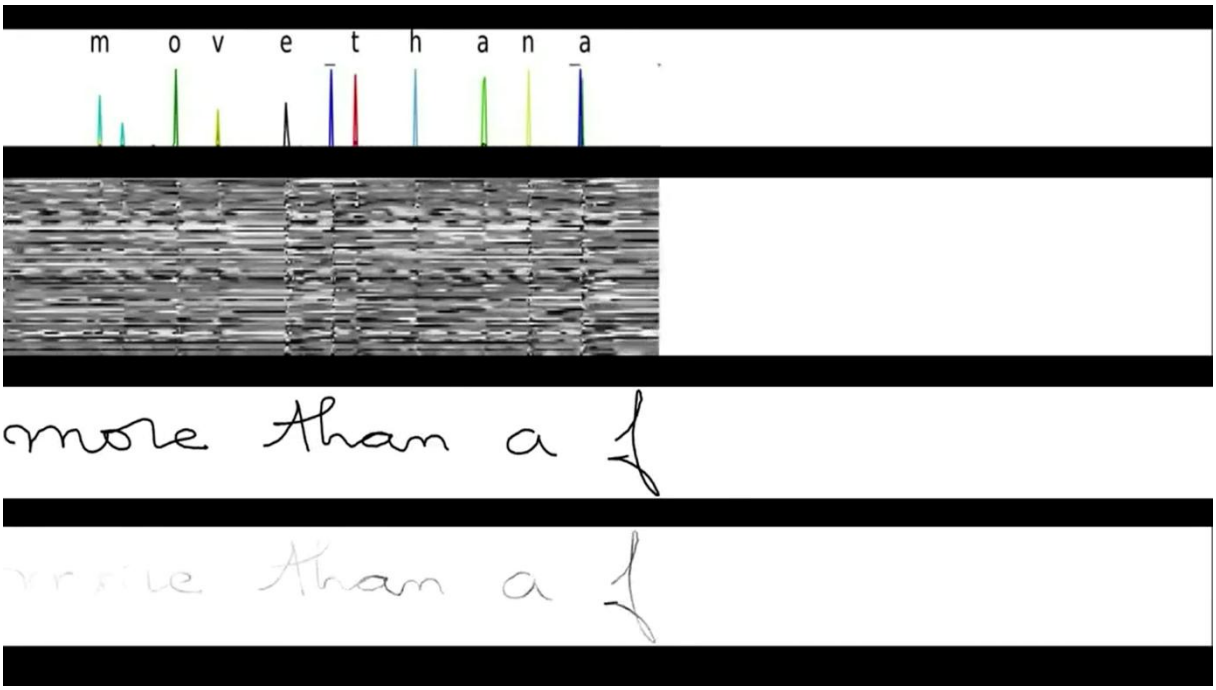


Figure 1. 11 Visualization of inputs and outputs from LSTM learned to recognize characters from written text (Hiton, 2016)

In 1997 a paper was published that introduced a recurrent neural network that solved the problem of remembering information. LSTM network is constructed from special modules, which are designed to allow information to be stored through a principle called gating. Information can be gated in and can be used by gating it out at the moment when it's needed. In between the two different states, which consist of two steps, first remembering the information through a gate in and releasing the information without forgetting it through gate out when needed by the rest of the network, where a decision is made about releasing the information. The basic difference from a normal neuron is that any information that arrives in between the two mentioned states, gate in and gate out, is not allowed to affect the information stored in the neuron remembering the information. (Hochreiter, 1997)

In figure 1.10 the schematic of a LSTM cell is sketched. Firstly information is enabled to enter the cell using an input gate, we call this process a gate in and it precedes the input gate to be set. The mechanism that decides if the information is stored or not is contained in the rest of the recurrent network not inside the LSTM cell. A network learned through backpropagation in time decides if the info is to be stored by setting the right write gate on thus enabling the information to enter the cell from rest of the network. The information previously stored in a cell using the write gate is stored as long as the forget gate is not set. The forget gate is as well controlled by the rest of the network and as long as the forget gate is not set the information previously written in the LSMT unit is kept unchanged in the cell. Analogically a read gate is used to retrieve the information from a memory

cell either directly written or being kept through time. Information on the output is affected by the state of the read gate and it enters rest of the network and influences the states of the rest of the LSTM cells. And the read gate is analogically controlled by the rest of the network. (Kang, 2017)

C_t is the neuron keeping the information inside the LSTM cell, as long as the forget gate is not on the neuron containing the remembered value is fed with its output value to the input with a weight of one, so it can keep its analog value. In order to store a value from rest of the system taken on input, the input gate has to be set and the value on input is able to enter into the neuron remembering the information allowing it to hold the information as long as the forget gate is not set. And analogically to read information from the neuron inside the LSTM cell, which is remembering the information, read gate has to be set in order for the value to pass to the output of the LSTM cell and to the rest of the network. (Srivastava P. , 2017)

6 LEARNING A NEURAL NETWORK

An important feature of neurons and neural networks is their capability to learn. Learning is the algorithm of setting the weights and thresholds of neurons so that the system is capable to approximate the learned output function based on the input function.

After the discovery of perceptron and publication of the study in 1969 about its limitations it has been later found out that networks with hidden units can learn difficult tasks. Without any hidden units neural networks are very limited in mapping inputs to outputs, they can learn. Adding additional layers between input and output layer makes neural networks way more powefull, but with the number of hidden layers and hidden units another problem emerges, which lies in the alghoritm setting the weights of activities in the connections of the hidden units.

The core of alghoritms used for learning neural networks is to repeatedly by using a trial and error approach design the features out of the inputs without knowing anything about the task neural network has to perform except of the pairs of expected inputs and expected ouputs. Features are actually being guessed by the learning aghoritms in an iterative process that improves its the guess each learning step until a certain limit reachable by the alghoritm. The learning process is a perfect task which can be computerized using learning alghoritms.

According to (Hiton, 2016) on the topic of perturbing weights, one way to achieve learning is to use a principle called perturbing weights, which randomly perturbs one weight to see if it improves performance and if so the change is kept in the network and saved. The problem with this algorithm

is that it requires making multiple forward passed on the training set just to set one weight. In this sense, the backpropagation algorithm performs much better and is way more efficient. Another problem with perturbing one weight is that by the end of learning nearly every large change in any weight would lead the model to worse performance because the weights need to have the right relative values. So the weights would need to change very slightly and that makes this learning method very inefficient. (Hiton, 2016)

6.1 Backpropagation

The basic idea is that the gradient can be effectively calculated for every layer by backpropagating from output to input. The goal of learning is to empirically minimize the error, respectively global error function. Overall error expresses the probability of wrong classification by the model and therefore there is an effort to minimize it in relation to all the parameters of a model. Every node of feedforward signal spread (in black) is also a node of backpropagation (in red) shown on fig. 1.12.

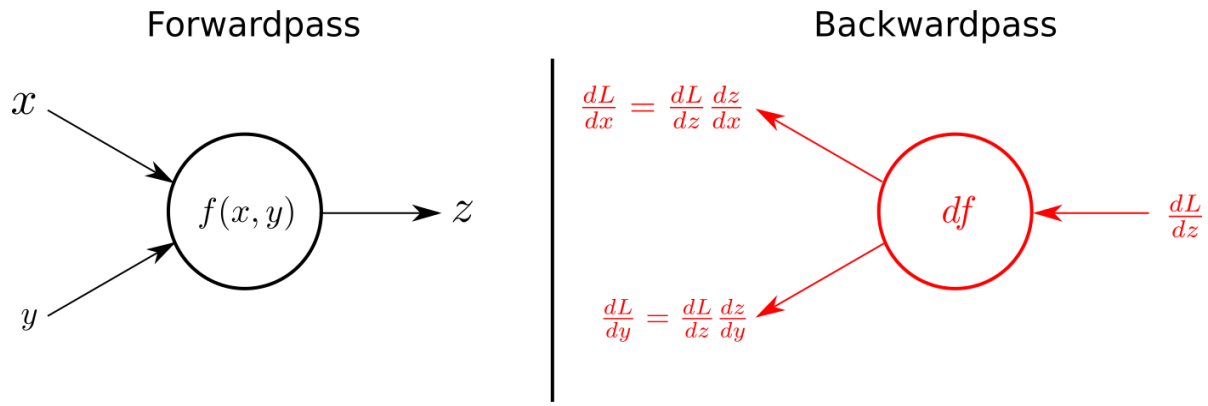


Figure 1.12 Representation schematic of feedforward (green) pass and backpropagation of error derivatives(red) (Kratzert, 2016)

The algorithm of learning process in backpropagation consists of four steps:

- Feed forward calculation
- Backpropagation of input layer
- Backpropagation of hidden layers
- Adding gradients to the weights

The first step is running the input vector through the model. This step is identical to the use of the model, where the input vector is brought in to the input and the result is a prediction. The second step is a calculation of the gradient for the output layer using a partial derivation of the error function on network parameters. The third step is calculating the gradient for the remaining hidden layers. In

the fourth step, the thresholds and weights are changed. The last step doesn't have to be run after every training sample, depending on the batching approach mentioned in the next paragraph.

Backpropagation is an alghoritm that enables to learn neural networks with layers of hidden units fast. It enables to calculate the speed of change of the error on the output of a network depending on the change of weight of an activity between the hidden units. The whole method stands on using error derivatives of activities. Each time activity in the networks is changed it either makes the error larger or smaller. The problem with hidden layers and hidden units comes in the point where each activity has effect on many other units. And all those effects have to be combined in order to learn the network. When the error derivatives are computed for all the hidden neurons in the same time it can be actually figured out how fast the error changes with the change of the activity. So the alghorithm actually starts backwards, because it calculates the effect of changing activities first and based on the correct direction of the change it calculates the correct change in the input weights of the activities of neurons in the previous layers, therefore we call the principle Backpropagation. It stands on the fact that once error derivatives of activities are calculated error derivatives of input weights can be more easily calculated for the neuron that is being observed.

The error derivatives are calculated for the layers closer to the output layer first and are combined using the same weights which are used for a forward pass to calculate the error derivatives in the layer further from the output units in the directions of the inputs and in the opposite way a forward pass through the network happens. (Mazur, 2015)

6.2 Training RNN with backpropagation

Backpropagation through time is used as a standard algorithm to train a recurrent neural network. The algorithm is not much more complicated than normal backpropagation if the recurrent neural network is unfolded into a feedforward network that contains one layer for each time step in the recurrent network as a forward pass. Untangling the recurrent connections in to a layer in a forward pass enables us learning the recurrent neural network the same way we learn a feedforward network. (Guo, 2013)

The algorithm starts from the network in its initial state to get to another state in time that is represented as another layer in the feedforward network to which it unfolds its recurrent connections. The difference to a normal multilayer feedforward network is that weights are kept the same between the layers while they represent the weights in the recurrent network. Weights are then kept constrained during different time steps which are unfolded into the layers in the feedforward network.

So the same weights are used again to get to another state. Basically, the recurrent neural network is represented by a feedforward network where the weights are constrained to have the same value at every layer and backpropagation is fast enough at learning when the weight constraints are present. (Hiton, 2016)

Actually, backpropagation through time is rather a name for the feed forward network with shared weights. Forward pass in recurrent neural network builds a series of activities of all the units at each time step. Backward pass, on the contrary, passes the activities backward off the stack of layers and computes error derivatives each time step backward from layer to layer with the weight constraints. After the backward pass derivatives are added for each weight at all time steps. All the copies of the

same weights are changed by the same amount proportional to the average of all the error derivatives. (Wikipedia, Backpropagation through time, 2018)

The only difference compared to standard backpropagation is the initialization of the activity of hidden or output units. So initial states have to be learned as well and have to be treated rather like additional parameters than activities which are learned the same way weights are learned. As well as weights are being learned in backpropagation the initial states can be learned the same way by following the negative gradient. The same way weights are initialized randomly, the initial states are also randomly initialized and at the end of each training sequence, they are backpropagated through time thus through all the layers of feedforward network in order to get the gradient of the error function all the way to the initial states. (Hiton, 2016)

6.3 Batching

Weight adjustments in the process of learning a neural network don't necessarily have to be performed immediately after every feed-forward calculation depending on the selected approach. There are three possible options: weight adjustments after iteration of one case, iteration of set of cases or after the whole training set. Each of those has its advantages and disadvantages.

On-line learning, or so-called "stochastic learning", is when the learning algorithm is done for every case separately. Value of the gradient is calculated only for that particular sample and parameters of the learned network are changed before running next iteration.

While in this case values of the gradients are dependent only on one training sample, they show a significant noise, where the error oscillates around a descending trend. Stochastic learning is used because: (Brownlee, A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size, 2017)

- Online learning is frequently faster than batch learning
- Results of online learning are frequently better than in case of batch learning
- Changes are observable in the process of learning

The difference in speed is observable when learning on big redundant data samples. For example, if all the data would contain every sample ten times, batch learning would calculate the same values ten times but weights would be changed only once. In case of stochastic learning, nothing is changed, after running a complete epoch on the data, the result is the same like for running 10 epochs on the redundant data. In practice data are rarely redundant, nevertheless, the singular cases are frequently so similar, for example in classification tasks, that they could almost be considered identical.

Another advantage stems from the negative characteristics of this type of learning. Non-linear networks tend to have several local minima with different depths. Strong noise in stochastic learning sometimes causes the algorithms to get out of the local minima trap and find other better solution. On the contrary batch learning can find local minima closer to the initial initialization.

Batch learning is a type of learning where the change of parameters is done after the run of epoch on the whole training set. That means that from the learning algorithm first three steps are run for every sample from the training set. Gradients are gradually added or averaged and the final step of changing weights is done on the very end.

Despite the above-mentioned disadvantages compared to on-line learning there are reasons to choose batch learning: (Brownlee, A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size, 2017)

- Convergence conditions are obvious.
- There are many speedup techniques practiced only for batch training
- Theoretical analysis of dynamic weights and convergence speed is simpler.

The main reason is again the noise from which the stochastic learning suffers. In case of stochastic learning the noise enables finding of better local minima, but the consequence is the inability to converge to absolute minima. Instead of finding a precise local optimum, the convergence stops because of the oscillation of weight values. This fluctuation is possible to minimize by lowering the learning rate or by adaptive size of the batch. In practice the first is used either by planned indiscrete lowering, lowering dependent on the number of epochs or dynamically changed based on the error.

Mini-batch learning is the third option. The main effort of this approach is to use the advantages of both presented methods. In this case, for training we have to set one more extra parameter, which is the batch size. Batch size in proportion to the number of samples in the training set determines if the learning would be more stochastic or batch like. The method also enables to change the size of mini-batch based on the learning process.

6.4 Overfitting

Overfitting is the biggest problem neural networks are being challenged with. Overfitting is illustrated on fig. 1.13. From the graph it's obvious that while the model is adapting for the given training dataset, from a certain point the test error starts to increase. It's caused by the model being adapted to specific cases and it's not capable of predicting on new data different from the training data.

A well-learned model can generalize characteristics found in training on all the data generated by the domain. This capability is dependent on network complexity, which could be affected by the number of hidden units, model type or regularization. And the capability of a model to generalize prediction is measured by test error.

Figure 1.13 Overfitting – The training error in blue and test error in red. Typically training error is constantly decreasing, while test error stops to decrease from certain point. (Scheau, 2016)

If the parameter setting in previous epochs provided a better generalization, then the network is being overfitted. If the situation is exactly opposite the model is underfitted. If an underfitted model is iterated through more loops of the training process to better cover the training data, the forecast on the test data will get more precise. Underfitting can also be caused from insufficient network capacity, which doesn't have enough capacity to approximate the output function. Under fitting can be recognized when the error on training data and on test data has similar value.

6.5 Dropout

One way to reduce overfitting is the dropout method. The idea is to disrupt the structure of neural network by stochastically switching off some of the neurons while learning, thus assigning a zero activation value to random neurons in a layer to which dropout is applied. A verified approach to lower the test error is to average the results of many different models. A standard method to achieve that is to learn many networks separately and average the result. A random dropout enables to create large volumes of slightly different networks in a reasonable time. It's almost certain that during learning for every case a different structure of the model is used. Every hidden neuron is assigned a probability with which during feed-forward calculation it's assigned a zero value. The consequence of this method is that nodes cannot cooperate when searching the characteristics, while any of their neighboring neurons in the layer to which dropout is applied can be switched off.

In the learning process, a dropout mask on the hidden layer is applied to determine which nodes are going to be deactivated. The dropout mask changes for every training sample to reach good enough regularization.

In order to combine a number of models without training large number of models in the same time dropout can be used. During each time step in training a number of hidden units are randomly omitted, but only during training. That allows a slightly different architecture for each training sample to be reached. This approach actually corresponds to having a different model for each training sample.

Dropout could be thought of as model averaging. It's analogous to model averaging while only few of the models get trained and they get one training example at a time and most of the models will actually never be sampled. Dropout can actually be thought of as extreme case of bagging. (Hiton, 2016)

It is an efficient way to average a large number of neural networks. Let's consider a neural network with one hidden layer. Each time a training sample is presented to the network only a subsample of hidden units are used for a forward pass and learning. While each time a training sample is used each hidden unit is assigned a probability with which it will be omitted and only after the omitting a training sample is run through the network. In that case, each training case is presented to a random sample from a large number of network architectures. The whole process happens only during training time, the dropout mask is not applied during the normal model run. The difference though is that the models share weight in between each other and when a hidden unit is used it has the same weights as in other architectures, which presents one advantage while the models get regularized. (Baldi, 2014)

Different options are available during training time. All the different architectures can be sampled and their output distributions can be used to form another distribution which is better in prediction using a geometric mean. The way which is actually used in applied neural networks is to use all the

hidden units but lower their outgoing weights respective to the probability with which a dropout during training was applied. For a network with one hidden layer this way actually exactly corresponds to calculating a geometric mean of the prediction of all the subsampled models. (Yadav, 2016)

For a topology that contains more than one hidden layer the same principle can be applied for every hidden layer. Thus making the division of weights proportional to the dropout probability for every hidden layer, even though, this algorithm applied to multiple hidden layers does not exactly correspond to calculating a geometrical mean of all the output distributions of all the subsampled models but the approximation works good enough. (Baldi, 2014)

Dropout is actually a way of learning the hidden units to work with many random sets of other hidden units. By using dropout it is more likely that the units will learn something individually useful rather than being useful because a particular subset of hidden units learned to collaborate together during the learning. And therefore, the subsystem is going to do something that is individually useful and different from what other subsystems do. (Srivastava N. H., 2014)

6.6 Initialization

The first step before training a neural network is the initialization of parameters (setting all the weights and thresholds on initial values). There is no precise algorithm that would guarantee an optimal starting point, but there are some rules that should be followed. If the weights are set on zero values the gradient value for backpropagation would be also zero and no learning can be achieved. When the weights are all set to the same value all the hidden nodes would behave the same and the network would be symmetric. For correct weight initialization, random values are used.

Values of the random initialization should in the case of logistic activation functions lie in their linear part. If the values would be chosen too big neurons would saturate easily and the result would be slow learning. (Dolezel, 2016)

6.7 Model complexity

In the process of creating an architecture of neural network it's necessary to decide about the number and size of hidden layers. Input and output layer is always a fixed part of a model, but the number of hidden layers is possible to change. The sizes of mandatory layers are set by the structure of input and output, respectively desired output. It's obvious that if input information consists of five values, the input layer will contain five neurons.

The number of hidden layers determines the computational capacity of a neural network. As a simple illustration on fig. 1.14 shows how increasing number of hidden neurons and thus capacity of network improves generalization capability of the neural network until it starts overfitting and the error for test set starts growing because the network learns the sampling error in the training set. (Hiton, 2016)

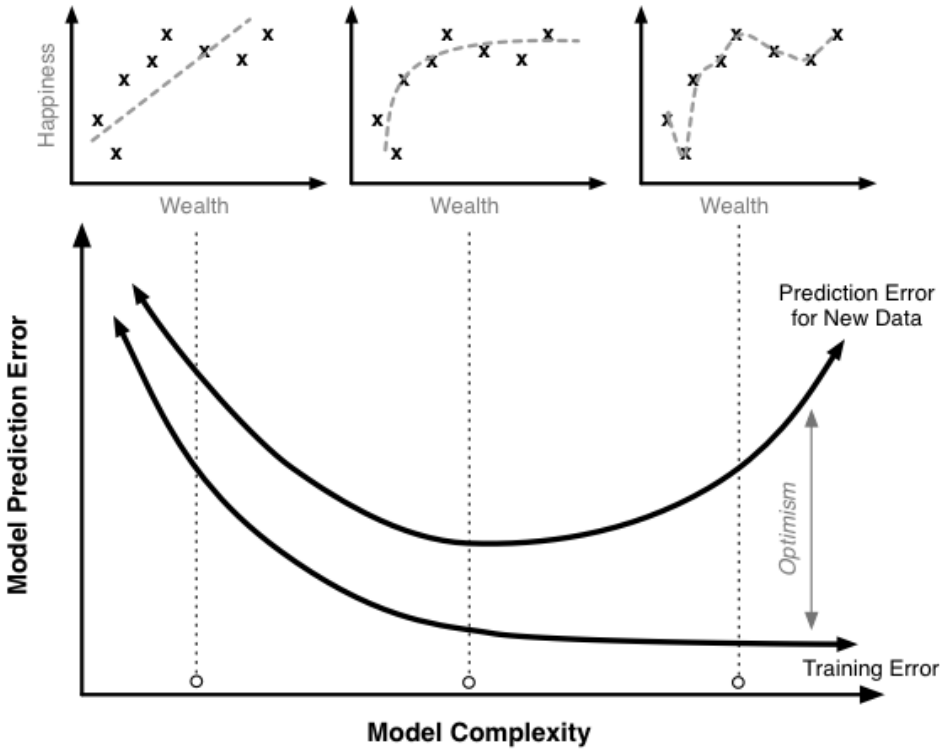


Figure 1.14 Quality of generalization being determined by model complexity. (Fortmann, 2012)

It's nearly impossible to get a precise output function for complex models. In most of the implementations capacity of the network is used big enough that it excludes underfitting.

6.8 Combination of different models

Combination of different models in neural networks contributes to reducing variance while keeping low bias. Models averaged together can gain better performance than single models respectively any single model.

The more the models predict differently from each other the larger the effect of combining different models can get. On a limited training set size, overfitting tends to occur. On the other hand, averaging prediction of different models reduces overfitting especially when the models make different predictions. As an example, we can take simple regression, where the squared error consists of two different parts - bias and variance. Generally, variance term is large in case the model is overfitting while it has too much capacity and it learns to predict the sampling error in the training set. Hinton mentions that overfitting occurs because of a different sampling error in the test set and training set, he explicitly states that if another training set of the same size from the same distribution is obtained the model would fit it differently because of different sampling error. And the bias term is large when the model has too small capacity to fit the data. (Hiton, 2016)

Regularities in the data are not able to be fitted by a model if the capacity of a neural network is chosen too little. And in the opposite case when the capacity of the model is too big it enables the network to fit a sampling error in a set used to train the network. But as long as the sampling error is different for the training set and for the test set overfitting occurs. Using many different models a

better equilibrium can be reached between overfitting sampling error and fitting regularities. (Didcock, 2015)

Variance reduction occurs from averaging models because it is possible to average out the variance. That enables to have high variance and low bias on high capacity models. The point in combining different predictions is to have predictions that guess pretty well but have very different errors. In general, if individual predictions from various different models disagree a lot, the combined predictor tends to be better when averaged over test cases.

It is important to notice though that the math behind averaging more different models is dependent on the presence of Gaussian noise in the sampling errors. If the present noise is not Gaussian averaging does not perform significantly better.

In order to make the predictions differ different attitudes towards models can be chosen. It's possible to rely on the fact that training algorithms can get stuck in different local minima, but according to (Hiton, 2016) this is not a very successful approach. However, there is another way how to achieve substantial differences in predictions made by the models. It's possible to combine different machine learning algorithms that don't rely only on neural network. Algorithms like decision trees or support vector machines. A different number of hidden neurons or a different number of hidden layers can be included in models which can be afterward combined together to make improvements into prediction capabilities. Different activation functions, as well as different types of weight penalties or different learning algorithms, can be applied to achieve the same result in generating models with different predictions. (Hiton, 2016)

Interesting inspiration into generating different models comes from a technique called bagging, where the model is trained on different subsamples of the training data by sampling the training set with replacement. Sampling a set with replacement means, that some of the data points will be missing and some of the data points will be duplicated. Bagging is successfully used on a method called random forest where it works really well on averaging models by using a lot of trees trained using bagging. In neural nets the problem that comes with bagging is that it is very computationally demanding. While on random forests the computational complexity does not play such an important role because random forests are relatively easy to compute, on neural networks it is an important factor. (Hiton, 2016)

7 OPERATION OF ELECTRONICS IN ANEMOMETERS

Digital anemometers are relatively simple devices with complex electronic systems that could introduce sampling errors to measurements of wind speed. The following chapter focuses on different parts of electronics present in anemometers in order to explain how the errors in measurements introduced by cup anemometers respectively anemometers with rotary parts can occur.

7.1 Voltage divider

A voltage divider by itself is probably not a large source of error in digital cup anemometers, even though the tolerance value of resistors used to build a voltage divider can cause that the voltage divider does not split the voltage into equal parts thus resulting in a malformed distribution of wind speed values on a histogram.

A voltage divider is an electrical circuit constructed from resistors. An example of two resistor voltage divider can be seen on figure 1.30. The main purpose of the voltage divider is to divide reference voltage into two or more voltages that add up to the reference voltage. Voltage divider works based on Ohm's law. The most common use of voltage divider is to create reference voltages in use for comparation. The most interesting use for the purpose of this work is the use in combination with operational amplifiers to convert the observed voltage to an n-bit number representing the voltage.

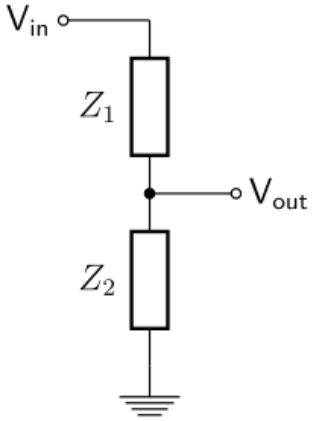


Fig 1.30: Simple voltage divider consisting of two resistors. (Wikipedia, Voltage divider, 2018)

The relationship between output voltage V_{out} input voltage V_{in} and the value of the two resistors Z_1 and Z_2 is a simple mathematical formula which underlies Ohm's law. The equation to calculate the output voltage from the resistance and input voltage is:

$$V_{\text{out}} = \frac{Z_2}{Z_1 + Z_2} \cdot V_{\text{in}}$$

A simple two resistor voltage divider can then divide the source voltage into two different voltages which correspond the ratio between the resistances of the resistors in the divider. A voltage divider containing more resistors can segment the source voltage into number of segments that corresponds to the number of resistors. If the values of the resistors in the voltage divider are the same, the voltage divider can segment the source voltage into equal parts each having 1/n voltage drop where n corresponds to the number of resistors. Exactly this principle is used in A/D converters to split the reference voltage.

7.2 Operational amplifier as Comparator

Operational amplifiers in A/D converters by themselves could also bring a certain level of error into sampling the voltage from a generator. Because operational amplifiers are constructed from semiconductors different kind of noise can enter and influence their operation. One of the noises influencing the input and output of operational amplifier that could be the most interesting one in the application of digital anemometers is temperature noise. (Texas Instruments, 2007)

An operational amplifier is another electronic circuit present in A/D converters. The detailed principle of operational amplifiers will not be explained for the purpose of this work. Only the circuit, where operational amplifier works as a voltage comparator will be briefly explained. When an operational amplifier is used as comparator it enables to digitalize the value of measured voltage. Operation principle can be understood from the schematics on fig. 1.31, where one input to the comparator is connected to a reference voltage created by a voltage divider from the source voltage. On the second input a triangle shaped function is connected to the input to illustrate the principle. As soon as the input voltage V_{IN} crosses the threshold of V_{REF} or the half of V_{CC} (both are equal when the resistance of the resistors R is the same) the comparator get saturated and it will output the voltage V_{OUT} on to the output thus enabling to read a digital value if a certain voltage threshold was crossed. As soon as the V_{IN} drops again below the threshold of the reference voltage V_{REF} it is possible to read a digital zero on the output of the comparator.

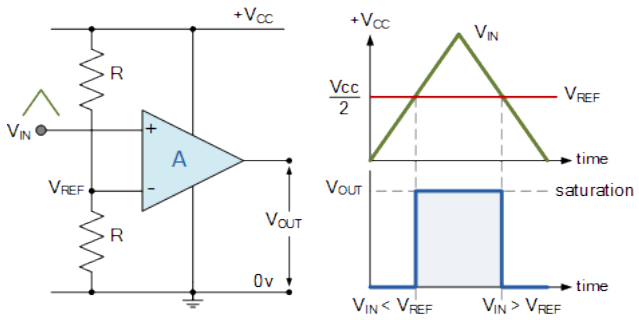


Fig. 1.31 Operational amplifier operating as voltage comparator. (Electronic tutorial, 2016)

7.3 A/D converter

Analog to digital converter is an electronic circuit that enables to convert analog voltage signal to digital bit represented series of numbers. In every application that processes continuous inputs from electrical sensors A/D converters are used to read the voltage and convert it through a relatively simple mechanism to an n-bit number. Most of the cheap market available A/D converters bundled into processors and microcontrollers are 8bit or 16bit A/D converters. The principle of A/D converter will be explained in this chapter.

An A/D converter works based on comparators which compare a reference voltage to a measured voltage and output a binary value when the threshold is crossed or when it's not crossed. A/D converter constructed from the voltage divider and comparators connected to each segment of the reference voltage which then encodes its outputs through a digital encoder in order to read a digital value of the measured voltage V_{IN} can be seen on fig. 1.32. The principle of operation is surprisingly simple. The reference resistor ladder, which is a voltage divider segmenting the voltage to n different voltage segments is connected to a series of n comparators which have one common input V_{IN} which is the voltage measured by the converter and each has a different voltage input which corresponds to a step in the source voltage of (1/n) * V_{REF}.

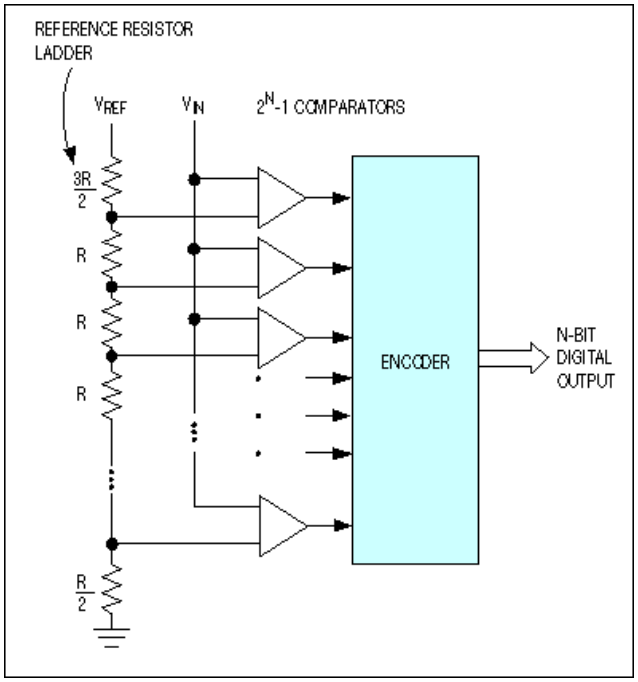


Fig. 1.32 A/D converter constructed from operational amplifiers and resistor voltage divider. (Errington, 2007)

So if an A/D converter has for example 8 comparators it would contain a voltage divider dividing the voltage to 8 different segments and each of the comparators would have the input of the measured voltage and the reference voltage was for example 8 volts the first comparator would take reference voltage of 8 volts on the input, second comparator 7 volts, third comparator 6 volts all the way to one volt. This enables to encode the voltage into a digital information, because if the measured voltage V_{IN} would, for example, be 6.2 volts the first comparator would not get saturated, the

second neither but the third which gets saturated after crossing the threshold of 6V would already be saturated and the encoder is then able to read the digital one form the output of the third comparator thus reading that the measured voltage crossed 6V because the third to eight comparators are saturated.

This technique of analog to digital value conversion is quite clever but has few major drawbacks, which is mainly the resolution of the sampling. No matter what if analog voltage needs to be converted to a digital number and stored the voltage input function needs to be sampled. For illustration, the upper part of fig. 1.33 shows how a simple sinusoid input function gets sampled by an AD converter. As seen on the picture output of the AD converter does not have a smooth character but a rather stepwise character of the output function strongly depending on the resolution and sampling rate of the AD converter. It is necessary to note that the mentioned architecture of A/D converter is a flash converter one of the more precise architectures, other cheaper variants exist that vary in precision and principles of operation.

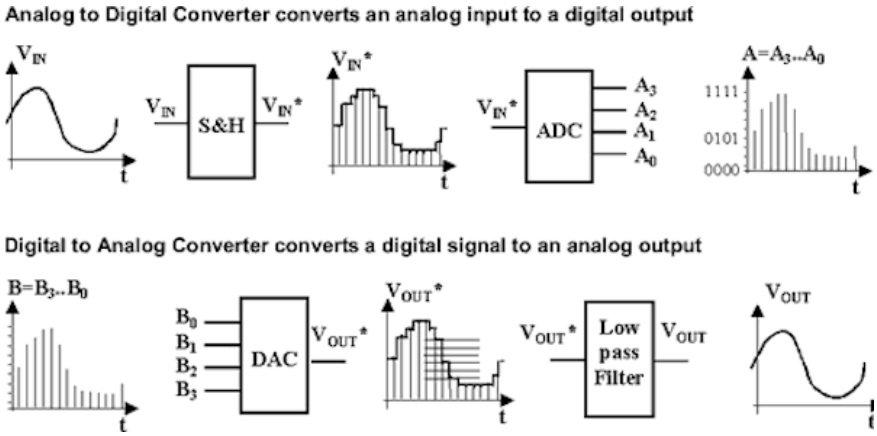


Figure 1.33 Sampling of sinusoid voltage function into its digital representation. (Fremer, 2015)

It is necessary to realize that operational amplifier used as comparator very well resembles a neuron with binary activation function with one input and one output, but without a learning function. Thus an 8 bit A/D converter with 256 operational amplifiers behaves more or less as a neural network with set weights, one logistic input neuron one output neuron and 256 hidden units with binary activation function.

7.4 Optical rotation sensor

Optical rotation sensor is constructed from three different parts, a source of light mostly a LED diode and a photodiode that enables to sample the information if stream of light is broken or free and a cog wheel that enables to sample the speed in a periodical manner when the axis of the anemometer is spinning.

On fig 1.34 a simple rotation sensor is shown with the two parts, an optical sensor and a simple cogwheel. Optical rotation sensors are relatively precise and relatively immune to any kind of noising and should theoretically be better for application in anemometers. But nevertheless, they carry a resolution which is dependent on the number of holes in the cog wheel. And the number of the holes

respectively the output resolution should be observed while it can introduce a particular error in the data, which might together with other sampling methods used in post-processing the sampled number of rotation to convert to a wind speed unit create other errors when the processing is done by 8bit controllers where the code for processing and sampling the rotation uses a data type that causes the data to be again resampled to lower resolution.

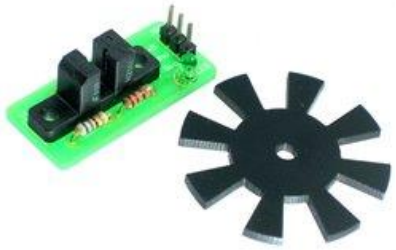


Figure 1. 34 optical sensor soldered on a PCB with 8 cog cogwheel. (Nasrulhaq, 2012)

The exact course of the output signal on the output of the photodiode is illustrated on fig. 1. 35, to better illustrate how the wind is exactly sampled. Each revolution of the cup anemometer consists of multiple impulses per revolution depending on the number of nipples or teeth in cogwheel blocking the light in between the photodiode and the LED diode in case only one value from the cogwheel encoder is read. A simplistic construction was explained, more sophisticated cogwheels with encoder and decoder mechanism exist that enable to retrieve more precise position of the cogwheel.

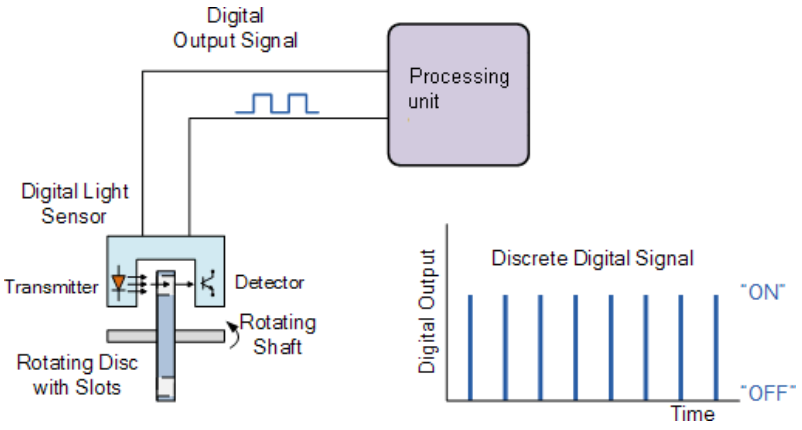


Figure 1. 35: Principle of rotation sensor explained and illustrated with the output signal. (Electronics Tutorial, 2015)

7.5 Magnetic rotary sensor

Magnetic rotary sensor works similar to an optical rotation sensor, but it works based on magnetic field of a static magnet revolving around a relay switch that connects two contacts in the proximity of the magnet which rotates together with the cups of anemometer thus enabling to count the number of revolutions per unit of time and therefore convert it to units of wind speed. Magnetic rotation sensors is reliable but come in a variant that has the lowest resolution from all the above-mentioned ways how

digital anemometers can be constructed, because it can count only by whole revolutions. So when the electronics samples the revolutions it happens relatively often that the time sampling window does not fit exactly into whole revolutions thus creating sampling errors larger in lower wind speeds and smaller sampling errors in faster wind speeds when the sampling time frame stays of fixed length. Figure 5. 6 shows a rotation cup anemometer with a magnetic relay switch enabling to sample the number of revolutions. It is necessary that there are commercially available constructions of magnetic rotary sensors and incremental magnetic rotary encoders that enable to read parts of revolutions quite precisely.

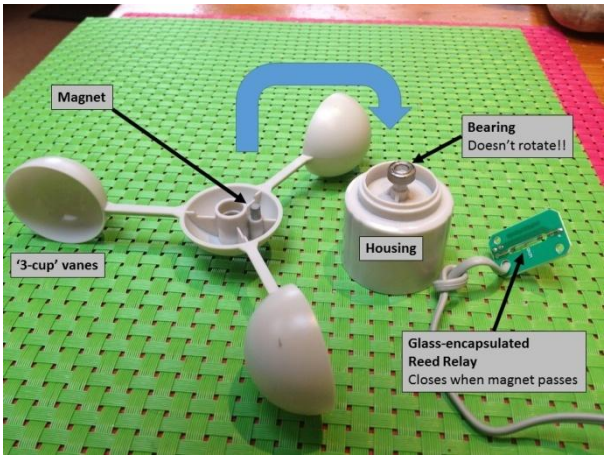


Figure 1. 36. Low quality rotary anemometer with magnetic relay switch for reading only whole revolutions.

8 NUMERICAL WEATHER PREDICTION

Numerical Weather Prediction model is actually a complicated program. It uses modified equations to calculate how energy and gas masses in the atmosphere are going to be transferred. For any calculation of future state, it is necessary to know the current state. The current state is observed by measuring atmosphere's physical and chemical properties. It is actually mainly measurements of temperature and air pressure. All this information is collected from meteorological stations, balloon probes, radio locators or satellite measurements.

8.1 Global models

All Numerical Weather Prediction models are based on mathematical equations, but the process of calculations differs from model to model. Some models include in the calculations shape of the surface of earth, radiation, turbulence or warm streams, some include only subset of the features to make the computation simpler and faster.

One type of models are global forecast models, those simulate the behavior of atmosphere around the whole earth. The programs have to calculate the forecast for the whole planet in units of hours, so that the results are available in sufficient time advance so that they can be used, which is extremely demanding on computational resources. For calculation of these models, the most powerful computers are used, but even that is not sufficient to generate the outputs in precise resolution. Outputs of the models are simplified and resulting states of the atmosphere are converted to a network where for example one output pixel represents area of 50 x 50 km. The equations used for calculations are being simplified as well. Global models slice the atmosphere in more levels by height to represent what is happening in different height levels above the ground.

Global models usually provide a forecast for 10 to 20 days ahead and they calculate tens of different variants of a possible evolution in the future. Large meteorological services work with global models, from which one of the most known is American model GFS or European model ECMWF. The simplification that leads to computational efficiency respectively that leads to a calculation of the model in short enough time is also a particular disadvantage of the approach. Global models based on the simplification cannot, for example, see mountains which actually affect the flow of air, temperature or creation of precipitation.

8.2 Local models

Therefore local models exist that don't calculate the forecast for the whole earth but only for a certain limited area. Local models are calculated based on the outputs of global models and they make the results more precise for a given area.

While local models work only with a smaller area they are capable to process results in more fine resolution, possibly even less than 2km, and thus they are able to better fit the local topology and thus the state and development of weather.

Another advantage of local models is the possibility of their more frequent run, thus providing more up to date forecast. Everything depends again on the resolution, more fine resolution leads to more precise result but also to more computationally demanding and thus longer processing.

8.3 Thermal Convection

Thermal wind is relatively stable wind caused by theradiation of sun. Thermal winds work on the principle of thermal convection. When for example a large enough mountain slope gets heated by the sun a large amount of air gets heated up from the mountain surface which rises along the slope of the mountain and the air that starts to escape the system needs to suck air from surrounding area which creates a flow of air and generates surface wind speed. Thanks to this effect a relatively stable wind conditions occur. A windsurfing spot that works based on Convection is for example Lake Garda in the northern part around Riva del Garda and Torbole.

Thermal convection in meteorology is mainly a vertical movement of air, caused by temperature differences between air particles and the surrounding atmosphere. It is in its core caused by Archimedes lift principle on air particles, which on increase of the temperature above the temperature of the surrounding atmosphere gain lower density and therefore lower mass and thus starts to spontaneously rise up in height.

Thermal convection represents mainly vertically oriented movements of particles in the area of the surrounding atmosphere. An elevated flow of air rises up to such a height, until their kinetic energy is completely used into energy transfer to friction, turbulent change with surrounding atmosphere or temperature transfer.

8.4 Inaccuracy in models

There is a whole set of reasons, why the forecast of a meteorological model is not precise. Firstly it is improper knowledge about the current state of the atmosphere. Pressure and temperature on world's surface is measured using meteorological stations, but those are tens of kilometers apart in Europe, hundreds of kilometers apart in Africa and even more far apart in polar areas of the earth. Problem is that with the missing measurement in between inputs to the meteorological models have to interpolated thus bringing errors in the inputs.

Even less is known about the state of the atmosphere in heights, which is necessary for gaining an overview and the calculation of current situation which is rather fundamental. Due to a substantial dynamics of atmosphere even small differences in equations used for the estimation of initial state lead after a couple of days of forecast lead to absolutely different results. Therefore different ensembles of forecasts are created which are calculated by global and some of the local models.

Other cause of inaccuracy origins from simplifications of equations used to calculate the interpolations or the future states.

9 WIND DATA SPECIFICS

Wind forecasting has its specific from different perspectives, the data sets with wind speed data have differences from each other depending on the location where the wind data was sampled. The measuring devices have their specific functionality that introduces errors into the measurements and the electronics post processing data from the sensors create a combination of systems that introduce a sampling error and other systematic errors into the measurements.

It's important to describe the datasets on which the experiments were conducted to illustrate what observations were conducted before doing the experiments and what limitation the experiments

suffered because of steps chosen in the data preparation. In this chapter, two spots are chosen to describe in detail how the learning validation and test sets for the experiments looked like.

Two different data sources were used in order to create discrete functions of observed variables which served as learning inputs to neural networks architectures proposed in chapters 10, 11 and 12.

9.1 Measurements

One part of the available data is wind measurement collection from stations placed in Europe. The data were collected with a physical device able to measure the wind speed, wind direction and temperature. The measuring device consists of three sensors, outdoor transmitter, and a receiver. Components of the measuring device used for collecting the data displayed on server windguru.com are pictured in fig 1. 37.

Figure 1.37 measuring device components: Anemometer outdoor transmitter and indoor receiver. (Windguru, 2017)

Data from stations are archived on a server and values for average wind speed in ten minute intervals are stored and were used for data analysis and learning the neural network.

9.2 WRF 9km forecast

A second important data source that was used for the conducted experiments is predictive output of Numerical Weather Prediction model WRF with a geospatial resolution of 9km. The WRF forecast model output contains many different variables but only a few were chosen to be used in the experiments, because of better data accessibility than other specific reasons. On fig 1.38 an output from WRF 9km of wind speed variable is shown, red color indicates strong wind, white no wind.

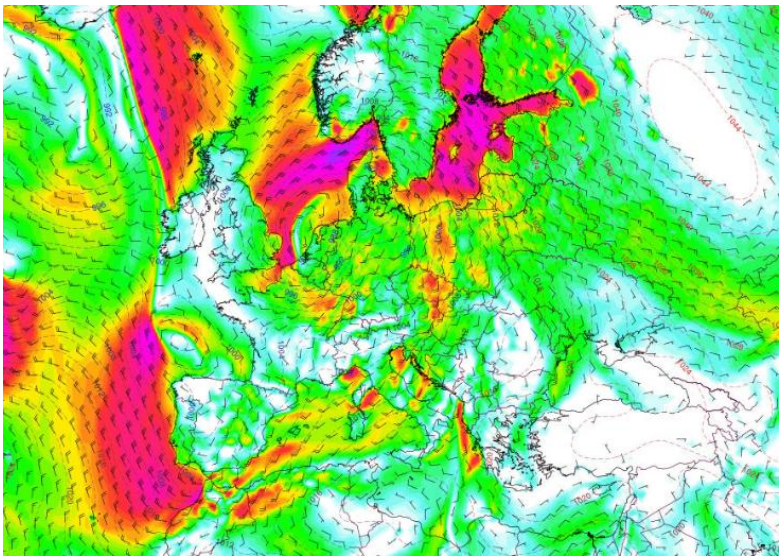


Figure 1.38 Output from WRF9km converted to an image. Red indicates strong wind, green medium wind, white no wind.

For the specific purposes of the conducted experiments, only certain points were drawn out of the WRF 9km model outputs. Those points correspond to the placing of the weather station from which the measurements were collected. The predictors from the numerical weather prediction model can be then connected as discrete functions on the inputs of proposed neural networks models. Only very simplified and information reducing outputs were drawn out of the model calculations, the observed variables contained a forecast for 24^{th} hours ahead predictions of wind speed, wind direction, and temperature. The WRF 9km output contains a forecast for each hour up to 72 hours in the future. The model is calculated four times a day at 00:00, 6:00, 12:00 and 18:00. Only forecasts for 24 hours ahead and the four daily initializations were taken and the missing values were interpolated using simple linear interpolation.

9.3 Complex terrain example – Lake Garda

Some of the data from different measuring spots were drawn out of the dataset to observe the characteristics of variables affecting the functionality of the neural networks as well as to observe different distributions of variables in the dataset.

One reason Lake Garda was chosen for the experiments is that it has rather special characteristics described in the literature and is very well known to have special local thermal winds. It is a common knowledge that on Lake Garda two types of thermal winds occur throughout the summer. The first on the early morning where the wind blows in the direction from the water towards villages placed on the north of the lake and the second thermal wind starting from the afternoon blowing the exact opposite direction.

While the measuring stations can be purchased and operated freely by everybody, the correct installation is not always guaranteed. Therefore a knowledge that the station was correctly installed on an open space, not for example in turbulent conditions behind an obstacle, was one of the reasons this station was chosen for the preliminary data analysis.

From a three months period through the summer an average for every hour of wind direction and wind speed was calculated and plotted into a graph shown on fig. 1.39. The two graphs seem to confirm the presence of the specific two-phase thermal effect. First half of the day the wind blows in direction around 40 degrees declined from north and in the second half of the day the wind blows in average in 220 degrees declined from north. When we observe the angles we can notice a 180 degrees declination difference in between.

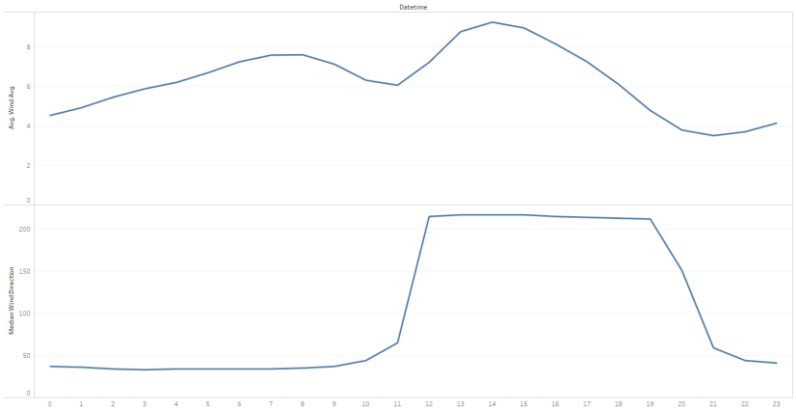


Figure 1.39: Average values of wind speed and wind direction during the day on Lake Garda taken from the measurement dataset

From the same period a scatter plot of wind direction and wind speed was made that clearly shows the two clusters of wind thermics present on Lake Garda, the plot is shown in fig 1.40.

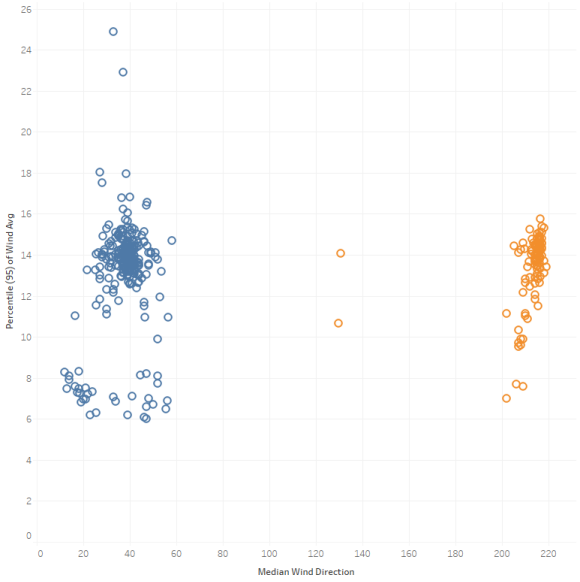


Figure 1.40 Scatter plot showing measurements of wind speed and wind direction

From the above data analysis, we can be rather sure that measurements are relatively correct and the fact that the data corresponds to the thermal effect know to be present on Lake Garda is ensuring us that the spot has certain periodic characteristics.

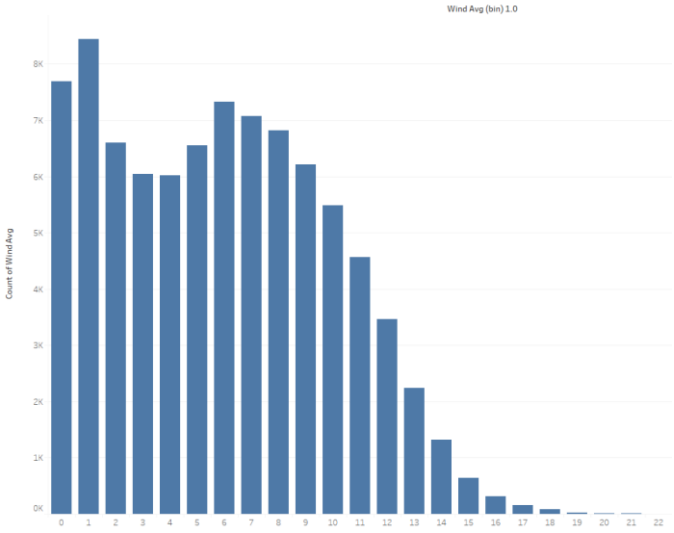


Figure 1.41: Wind speed distribution over on the whole two-year dataset on Lake Garda

A frequency plot of wind speed is plotted on fig. 1.41. The values for the wind speed frequency occurrence plot are taken from two years of measurement from 2016 to 2018 on Lake Garda. On the plot an obvious multimodality of the wind speed distribution can be seen making the dataset more interesting than dataset from measurement station placed on flat land. It is necessary to note that the left part of the distribution might not be exactly correct because the measurement device with which this dataset was collected with states a minimal measurable wind speed which can malform the distribution over the first three bins of the plot. (Davis Instruments Corp., 2018)

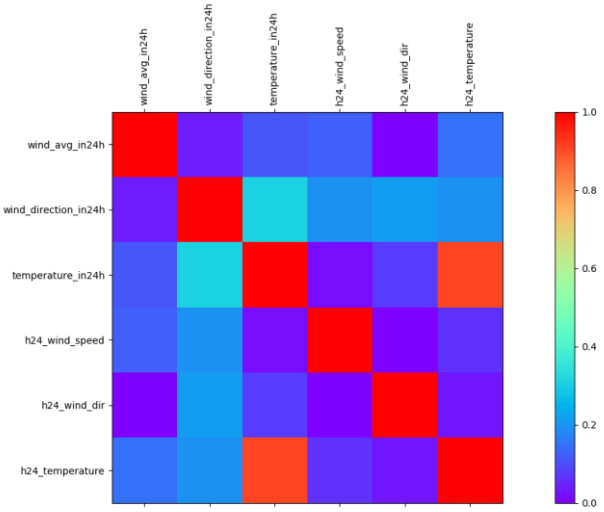


Figure 1.42: Correlation between predictors from WRF9km forecast and measured values.

Correlation plot on fig. 1.42 with predictors from WRF9km forecast and measurements of wind speed, wind direction and temperature obtained by sampling the values with a weather station show correlations between 24 hours ahead forecast and measurements only for predictor of temperature.

Data from the WRF model outputs were also examined to gain a better insight into the second available dataset. A model output of wind speed u-component variable is shown next to a satellite picture from the same region on fig. 1.43 which helps to understand the conclusions of (Larraondo P. I., 2017) that single point approach, as well as a square space average over an area, produced similar results.

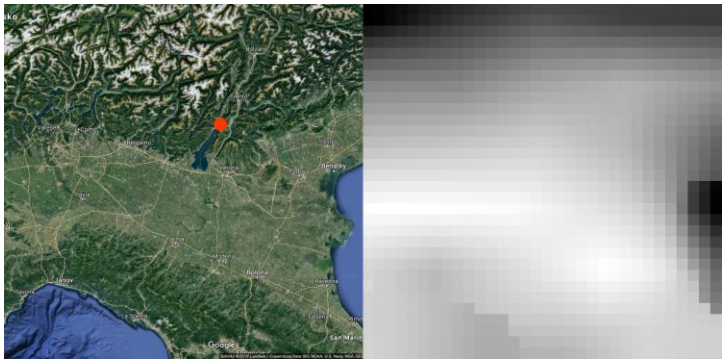


Figure 1.43: Satellite image on the left with a marked point of the measuring station and output of WRF 24 hours ahead forecast of wind speed for the same region.

As it's possible to observe the predictions are of relatively low resolution considering the fact that on the satellite picture we can observe the terrain of the Alps northern from Lake Garda which forms different channels and obstacles for a wind to flow in. And considering the fact that the effect we described based on the analysis of the measurements is probably very dependent on the position of the measuring station in the terrain, the output of the WRF model doesn't seem to be recalculated on the terrain or is so much averaged that viable information gets lost, which is of course understandable considering the purpose of the use. Therefore a mechanism based on neural networks is suggested to recalculate the outputs of the WRF 9km model output and is further described in chapter 11.

9.4 Flat landscape example - Zloncice

Another spot chosen for the experiment is from the opposite spectrum. An example from a city Zloncice in the Czech Republic was drawn out of the dataset to demonstrate how the wind speed prediction differs in a terrain where no local thermal effect affects the wind speed or wind direction.

This spot was chosen because it contains very little repeating effects in the data. On fig 1.44 we can see the scatter plot of wind direction and wind speed. Contrary to the example of Lake Garda where two clusters of wind direction could be clearly seen on fig. 1.40 in the case of data from a measurement station placed on flat landscape the scatter plot of wind direction and wind speed shows no signs of regular thermal effect with different directions on the contrary wind direction seems to be rather evenly distributed along wind speed.

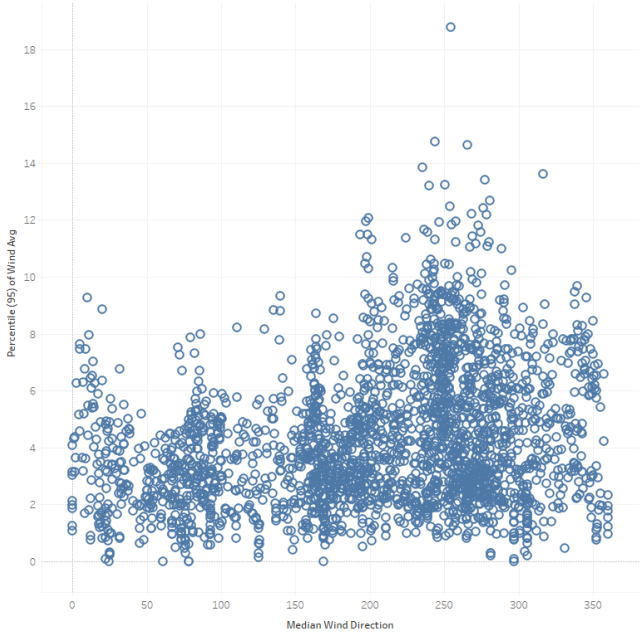


Figure 1.44: Scatter plot of wind direction and wind speed of dataset in Zloncice in Czech Republic

If we look at the scatter plot there seem to be no well distinguishable clusters in the data telling us that the dataset has no dominant wind direction or prevailing effect where wind would blow only from a certain direction even though we can see that there are areas in the scatter plot where the plotted discrete values of measured wind direction and wind speed occur more frequently. As we can see later in the experiments conducted and described in following chapters of this work the characteristics of the dataset and its distribution influences the prediction capabilities of different neural network architectures. For the completion of the comparison with Lake Garda, a plot with average daily wind speed is included in fig. 1.45

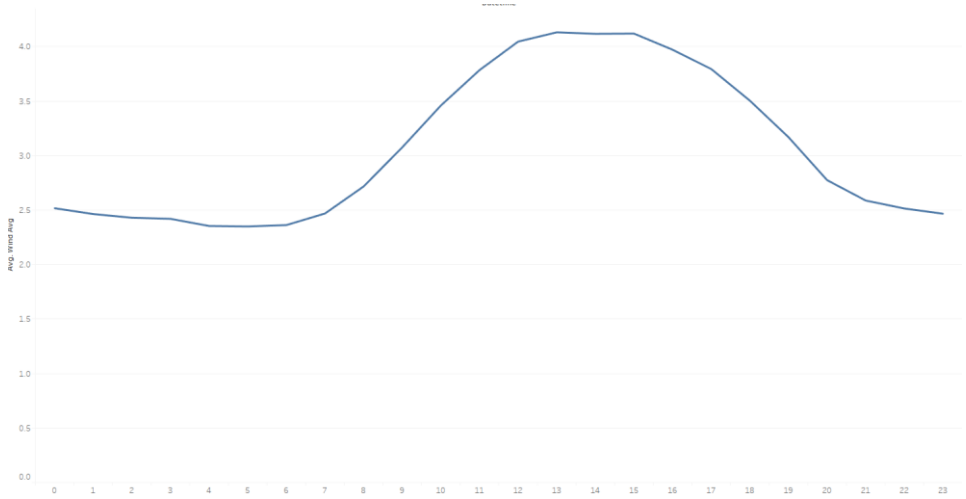


Figure 1.45: Average daily wind speed on the dataset in Zloncice in Czech Republic

On figure 1.46 a wind speed frequency plot can be seen for the dataset in Zloncice. Compared to the same type of plot for measurements obtained from Lake Garda this particular dataset from Zloncice

seems to exhibits unimodality which should make theoretically the data more resistant towards error occurrence caused by using wrong down sampling mathematics described in chapter 9. 12.

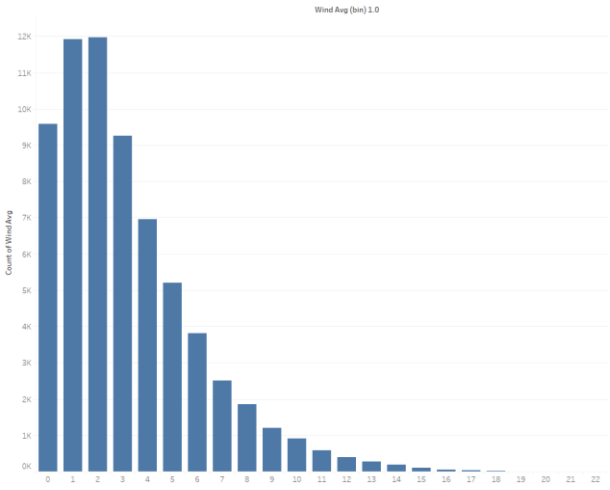


Figure 1.46: Frequency occurrence plot for wind speed dataset Zloncice

Correlation between variables available from measurements and WRF 9km forecast were plotted in order to observe how well the predictors and other available variables correlate in between each other. Temperature forecast from WRF 9km model is correlating with the values sampled by the measurement station. Relatively high correlation can be observed between the WRF 9km predictors of wind speed and measurements obtained by a measurement station. Sadly correlations between the measurements and predictors of wind direction are extremely low as can be seen in figure 1.47. Variables from WRF 9km forecast for 24 hours ahead start with prefix h24 measurements sampled from the station end with suffix in24h.

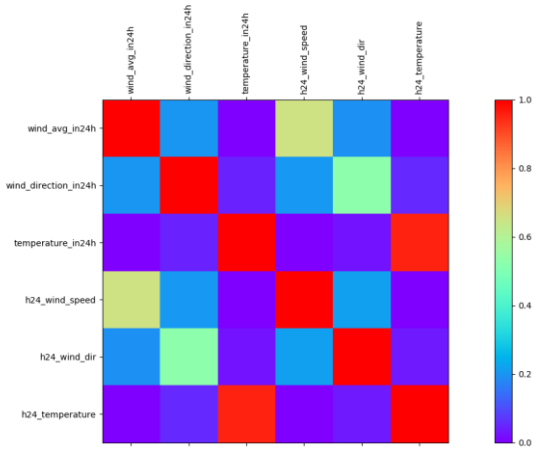


Figure 1.47 Correlation between predictors from WRF9km forecast and measured values

9.5 Error caused by anemometers with generators

Anemometers are devices consisting of a rotational part and static part depending on the construction voltage gets induces in the winding of the stator or rotor of the generator in the anemometer. Anemometer construction with a generator is pictured in figure 1.70. While anemometers contain rotational parts that have certain mass and thus while moving accumulate and release kinetic energy they might introduce a degree of elasticity in reaction to a current wind speed while it might have stopped blowing the rotational part keeps spinning from its kinetic energy still producing electricity in the windings of the generator which are later read by an electronic circuit to obtain readings about wind speed.

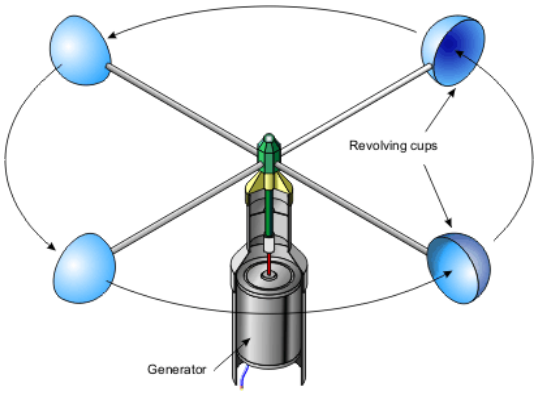


Figure 1. 70 Decomposition of four cup rotational anemometer showing cups and generator. (Compact Analysis, 2016)

The main question concerning accuracy is if the mass and the kinetic energy which is transferred from the wind to accelerate the anemometer has the same character in acceleration, as when the rotational part of the anemometer is breaking by resistance in the bearings between stator and rotor thus creating some asymmetry in measurements. Another problem of this construction is minimum current necessary to pass through the windings in the generator necessary to measure the voltage and speed at which the rotor of the anemometer is revolving.

9.6 Error caused by A/D converter in the measurement hardware

One interesting source of error in the measurements of wind speed comes from the principle of conversion of analog to digital signal. Depending on the construction of the anemometer and it's processing logic data has to be processed and converted from analog signal to digital. Most common operation principle to convert measured analog values to digitally sampled bits is measuring a voltage by series of comparators connected to a voltage divider in a circuit called A/D converter further explained in chapter 7.3.

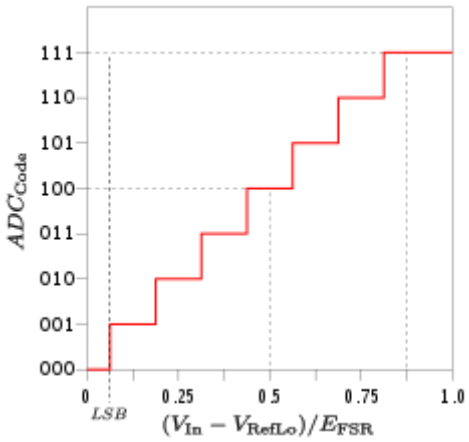


Figure 1.71: Three bit A/D converter with 8 voltage steps

As seen on fig.1.72 we can see a histogram of wind speed from a measurement station. The histogram with plotted values occurrence of 0. 1 knots step shows an interesting pattern. The places where the graph has holes in between the bars are actually missing values. So for example, 1.4 knots 1.9 knots and 2.4 knots are values that never occur in the measurement dataset. As well some of the values have significantly more frequent occurrence than other values. If we take into consideration the way A/D converters operate to convert the voltage to digital number it might lead to a conclusion that the values that are more frequent in the dataset were the values of wind speed converted with the generator to a voltage which fell close to the reference voltage of the comparators on the A/D converter contained in the measuring device. And the missing values in the dataset are caused by upsampling the resolution of the A/D converter to a bigger resolution.

In the moment when an 8 bit A/D converter is used we practicaly reduce the resolution to only 256 values but it can be thought of it as if we prepend this hypothetical network with set weights before the processing network.

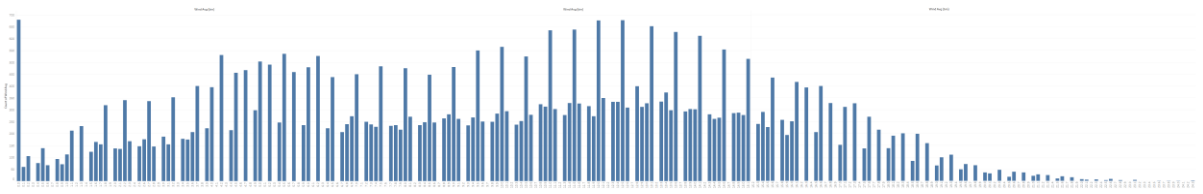


Fig 1.72: Histogram of measured wind speed sequenced in bins of 0.1 knots

9.7 Errors caused by low sampling rate in microcontrollers

Another source of error which might be relevant especially to anemometers using optical rotation sensors might be a problem of too fast sampling causing an error to enter wind speed measurements while the sampled number of rotations might then lead to inaccurate reading when the number of rotations reach low values, thus creating a data which are resampled in a resolution that corresponds to the ratio between cog wheel resolution and the sampling time. So in order to increase the

resolution either a longer sampling period is necessary or an rotary sensor with better cogwheel resolution.

9.8 Errors caused by downsampling in optical and magnetic sensors

There might be a significant error source in the data produced by rotational anemometers based on low-resolution cog wheel in optical sensor or by anemometers with a magnetic sensor that can record only full revolutions of the anemometer. The resolution itself would not be a problem if the sampling window is large enough to even out the sampling error caused by low-resolution output from the sensor.

9.9 Other sources of error

When the same histogram as in figure 1.72 is plotted with a step of 1 knot we can better notice how the distribution of the wind speed looks like when it comes from the measurement device. The histogram can be seen on fig. 1.73.

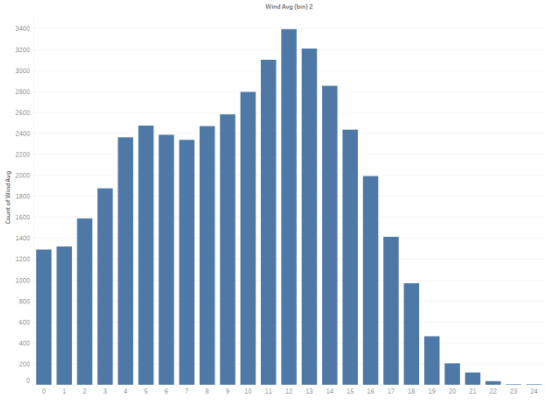


Fig 1.73: Histogram of measured wind speed sequenced by 1 knot

It is possible to notice that the distribution of wind speed is obviously read wrong by the measurement device at least on one end of the distribution, on the left part. Most of the commercially available rotational anemometers have a minimum wind speed which they can read. So a lot of wind speed values between zero and the minimum measurable speed fall to zero, while the anemometer is not able to spin because of the resistance of the rotational parts or delay in transfer of potential to kinetic energy.

9.10 Interference of resolutions

Another possible error source might after a quick look at figure 1.72 with the histogram of wind speed be interference of two different sampling resolutions, while some of the electronics parts need to resample and rescale the input resolution from electrical sensors or A/D converters into a n-bit

number depending on the architecture of the microchip present for processing the signal from the electronics or depending on the architecture of the software processing the signal.

Digital interference can be a cause of the way figure 1.72 looks like but it has few drawbacks of the explanation. One of the drawbacks is that obviously, the interference is not of the same frequency along the whole spectrum, if we closely examine the spectrum we can see that some of the values are correctly sampled each 5th decimal value and some are sampled correctly each 2nd decimal value. And the overall question is if the means are then sampled correctly across the spectrum of wind speed in the input datasets or if due to the possible interference of digital resolutions and some other frequencies (like increasing sampling precision due to increasing resolution of wind speed from slower winds to faster winds) and if thus if the means are not calculated wrong for the histogram with different bin size in figure 1.73.

9.11 Weibull distribution

Weibull distribution is a continuous probability distribution named after Swedish mathematician. The weibull distribution function is used in many different applications like for example reliability engineering or analyzing error rate in the components of electrical systems. (Novotny, 2002)

Weibull distribution is also used in prediction of wind speed. The distribution enables different parameterization and looks very different for different values of its parameters. The one we could observe in the experiments conducted in this work and in the work of other researches aimed at wind prediction has a rather specific look while the distribution of lower wind speed values seem to be more frequent in the measurements than higher values thus making the distribution skewed to the left especially in flat land areas. On fig 1.74 Weibull distribution with different parameterization is shown.

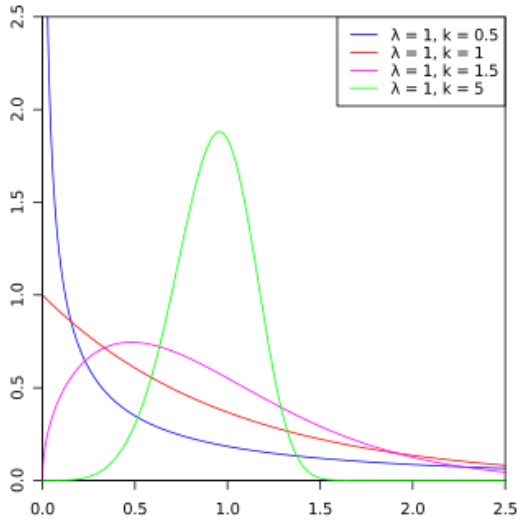


Figure 1.74: Weibull distribution sketched with different values of parameter k.

The parameter around which the different wind speed frequency distribution corresponds most precisely to measurements is sketched on figure 1.75 with parameter k=2.

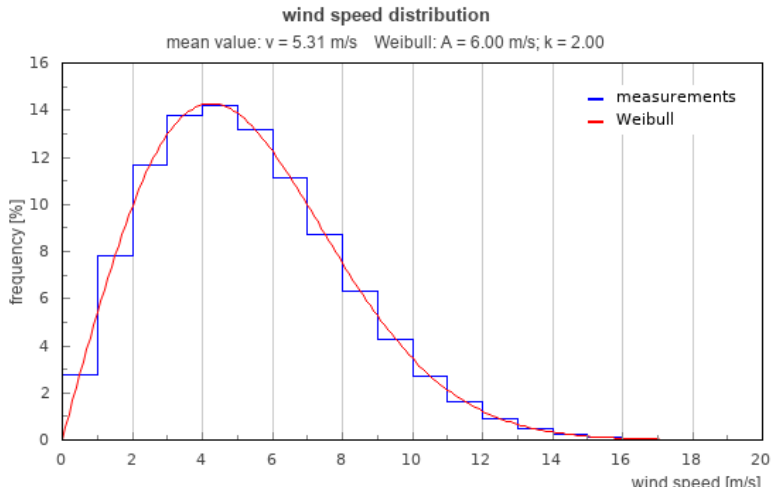


Figure 1.75: Weibull distribution fitted to wind speed measurements with k=2

9.12 Error in downsampling mathematics

Another possible source of error respectively a point where the real wind distribution gets malformed might be the mathematics of downsampling the wind speed data onto a larger time frame than the sampling time frame.

Especially in the datasets used for conducting the experiments averaging is used to calculate wind speed for 10-minute intervals which are archived in the dataset used to conduct the experiments in this work. While the wind speed distribution might not correspond exactly to the results of the downsampling mathematics the values could be shifted.

One reason to think that is that there might be a substantial reason to think that wind speed distribution is not unimodal but bimodal or multimodal. The main problem of averaging a multimodal distribution is the malformed probabilistic view on multimodal data based on one single average. While a multimodal distribution has two different probability peaks each time a dataset with multimodal distribution is downsampled using average important information about the distribution might get lost. It is also possible that the effect of averaging multimodal wind speed data has no significant effect on the quality of predictions when the malformed distribution is both on the inputs as well as on the output of a machine learning model.

In theory, it could introduce an error that is corresponding to a proportion of the distance of the two probabilistic moduses of the real distribution which explains better the conclusion of (Zhao, 2017) why relationship between power output and wind speed is probabilistic and not deterministic.

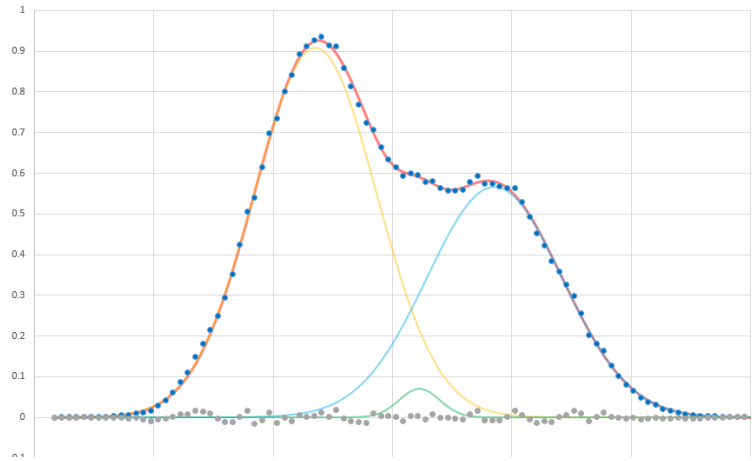


Figure 1.76: Multimodal probabilistic distribution.

On fig. 1.76 a multimodal probabilistic distribution is plotted. As it can be observed an average made from a multimodal distribution is not corresponding well to the probabilistic division of the values. While wind speed tends to have a distribution similar to Weibull distribution (Kadhem, 2017) or multimodal distribution (Zhang, 2013), averaging measurement in order to downsample dataset might not always make sense. In the case of Weibull distribution the situation might not be so tragic as in case of multimodal distribution, while averaging a multimodal distribution introduce a systematic error in the downsampling process and the question is if the same error does not appear multiple times on the way where information sampled by a measurement device travel from the physical device all the way to the machine learning model and if the accumulation of the error is only additive, multiplicative or in the worst case exponential.

10 FORECAST WITH FULLY CONNECTED LSTM

In this chapter a fully connected LSTM neural network is applied to predict wind speed measurement from one single station 24 hours ahead. The presented approach and results are relatively simplistic in its nature, because the network is capable to only repeat remembered values based on previous input. A better design would be a model that would accept measured inputs from multiple stations,

but to keep the simplicity of the experiments a simple fully connected LSTM network was used to conduct the experiments. LSTM networks can keep previous input information in special memory cells (LSTM cells) which contain gates enabling the cells to remember and forget information. Using this mechanism, a fully connected LSTM network accepting as input a 1D vector each time step can predict the times series.

10.1 Network topology

The network consists of two layers of LSTM cells. Each layer contains 50 stacked cells and the layers are mutually interconnected. On the input linear neurons fully connected with the 50 LSMT cells are accepting the input information. The second LSTM layer contains 50 cells which accept a sequence from the previous layer and those are fully connected to one output neuron.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 1, 50)             10800
_________________________________________________________________
lstm_2 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 31,051
Trainable params: 31,051
Non-trainable params: 0
_________________________________________________________________
```

Figure 2.0: Comparison of 24 ahead forecast made by LSTM network (green) and measured wind speed (blue)

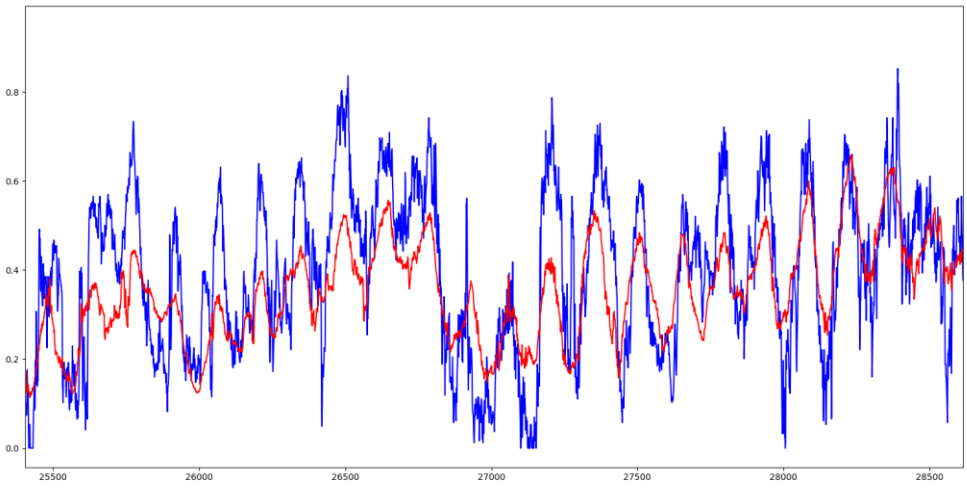


Figure 2.1: Comparison of 24 ahead forecast made by LSTM network (green) and measured wind speed (blue)

On fig. 2.1 we can see the output from the LSTM network learned on inputs from measurements and the expected output of measurements shifted to 24 hours ahead. When zooming the graph, it can be noticed that the LSTM network rather well resembles the daily period of the learned outputs. The LSTM network is very good copying the distribution of wind speed course of previous inputs to the day it is predicting adjusting the previous inputs it then repeats on the output based on decisions from inputs and inner states of LSTM cells. On fig. 2.2 graph of predictions from the same model is plotted. We can see the above-mentioned phenomenon happening in more detail. The network predicts well the period and is synchronized with the expected output but it exhibits a certain error function.

Each peak on fig. 2.2 corresponds more or less to course of wind speed during one day. When the graph is analyzed day by day it's observable that position of minima and maxima of daily wind speed are approximated by the network quite correctly. The question is if the precision achieved is dependent on the number of hours ahead for which the forecast is done. While predicting a relatively periodic data (in this case the period of one day is relatively well visible) the task is much simpler because value from previous day relatively well correlates with the value of the upcoming day.

Directions of the approximated functional derivations of course of the wind speed are also rather quite correct and they reach quite optimal steepness.

If a straight line would be plotted through the rather noised rising edge of the forecast on the beginning of the day and analogously a similar line would be plotted thought the rising edge on the expected wind speed, the steepness of these straight lines would be quite correctly assumed by the LSTM network. The research question at this point is what data features can be used in the model that could help lower the error that occurs in the approximation.

It's also interesting to notice on fig. 2.2 how the network quite well estimates the position of daily maxima and minima, but for the three days where the error is obviously the biggest, it fails to predict the temporary trend function.

According to a rather different experiment mentioned in chapter 5.5 conducted by Graves and Schmithuber where LSTM network was used to recognize written characters the network was able to reliably recognize a handwritten character only on the beginning of a sequence of a new character.

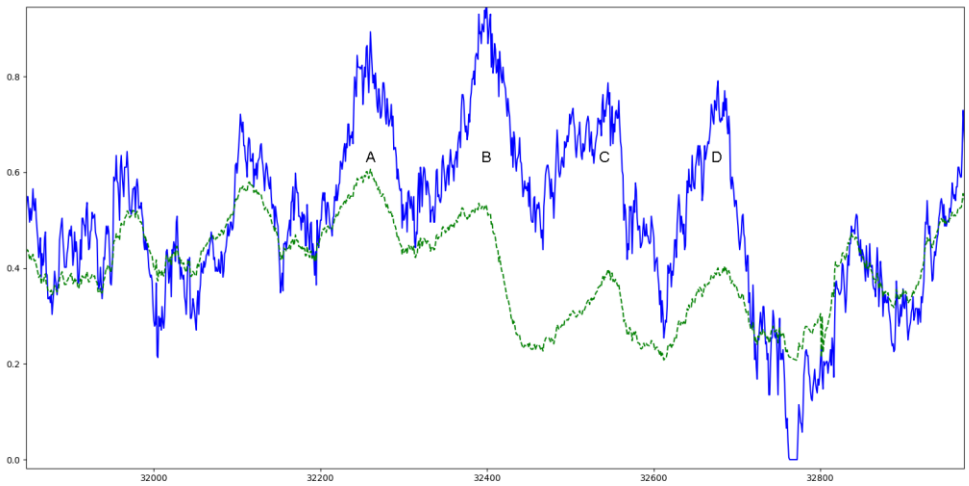


Figure 2. 2 forecasts done by LSTM network (green) an expected wind speed values in (blue) [include marks and letters for days]

If we look at the four peaks where error for the prediction is obviously the largest it's important to realize that output – the green dashed line is calculated inside the LSTM network based on the blue line – real measurement input, but from one peak before. Thus it might be possible that the first peak A was outputted based on a knowledge of the previous peak and its ancestors, that is where the error function occurs. At the peak B where local maxima of the scope of the graph is reached the knowledge about peak A has just been presented to the network. And knowledge about where the local maxima of the wind speed occur is known when the network gets the information about the peak C and at that moment it is predicting the peak D. If the graph is observed more closely it's possible to observe that prediction of peak D might include the knowledge about the maxima in peak B, but of course delayed, because the features could not have been recognized by the network before. So while a half wave of spanning across 4 days (peak A,B,C,D) occurred in the data it is possible that the network recognized the presence of this half wave only after it has recorded it.

Different performance was observed for different terrains and quality of data for which the LSTM network predicted the wind speed.

10.2 Training

Input data to train the network consisted of measurements samples for every 10 minutes with average wind speed, wind direction and temperature for outputs plotted in fig 2. 3 and 2. 4. Expected output data, in this case, is a series of wind average wind speed shifted by 24 hours as opposed to the input.

For the experiment, data was split into a training set containing 60% of the data and test set containing the remaining 40% of the whole dataset. On fig. 2.3 the predictions on the training set are plotted in blue and predictions for the test set, which was not used for learning only for predictions, are plotted in green. On fig 2.4 the results from experiments conducted on a dataset from Lake Garda are shown.

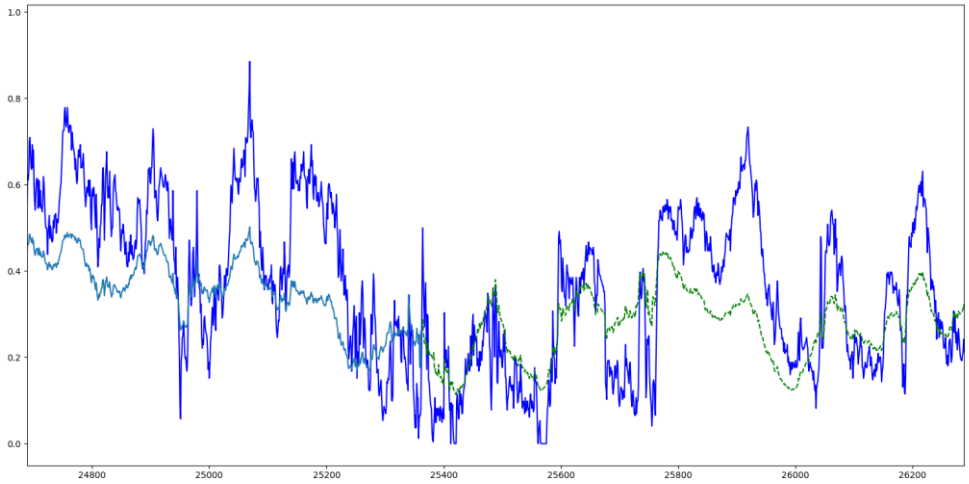


Fig. 2. 3 Predictions done on the training set and predictions done on the test set with comparison of expected real measurements for Lake Garda.

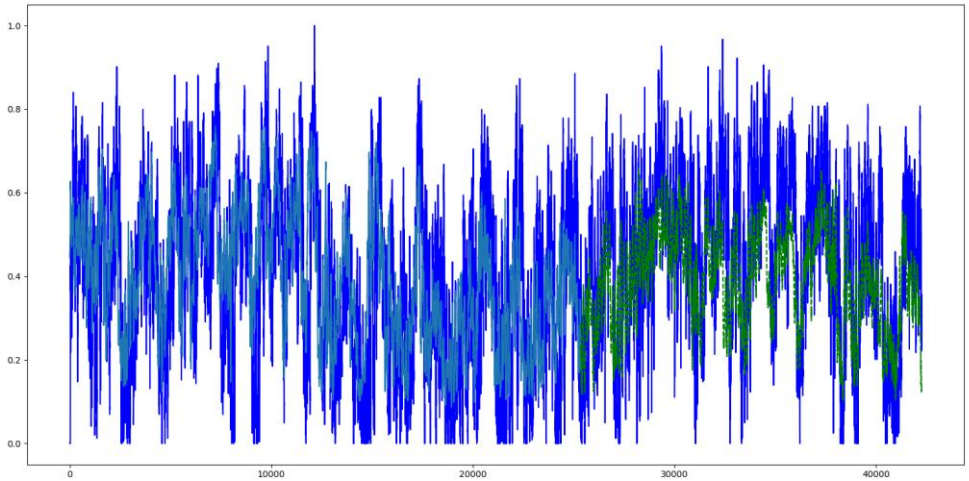


Fig 2.4 The whole dataset for Lake Garda – dark blue, predictions for training set – light blue and predictions for test set – green.

10.3 Results

Four different datasets were used to learn and test the network performance in prediction of wind speed for 24 hours ahead. Four different locations and datasets were chosen with relatively different wind characteristics. Performance of prediction capability was measured using normalized root means squared error and normalized mean absolute error.

Table 1: Results of experiments with LSTM network making prediction from historical measurements

Station location	Wind speed distribution	Terrain	NRMSE	NMAE
Rhodos, GR	Multimodal	Complex	0.16	0.13
Lake Garda, IT	Multimodal	Complex	0.12	0.10
Zloncice, CZ	Unimodal	Flat	0.11	0.10
Prague, CZ	Unimodal	Flat	0.13	0.10

11 MODIFICATION OF NWP WITH MLP

In the following chapter an artificial neural network topology that transforms WRF 9km forecast into a modified forecast that better fits local measurement is introduced and described. Following paragraphs describe how two similar neural networks perform and how the output looks based on the distribution of the input data.

11.1 Network topology

For a neural network fitting the outputs of WRF 9km model to local measurements, a simple architecture at each time step processes two different predictors from the WRF 9km forecast, was trained to recalculate them into wind speed predictions for 24 hours ahead to datapoints from the measurements.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1, 100)            600
_________________________________________________________________
gaussian_noise_1 (GaussianNo (None, 1, 100)            0
_________________________________________________________________
dense_2 (Dense)              (None, 1, 500)            50500
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 500)            0
_________________________________________________________________
```

```
dense_3 (Dense)              (None, 1, 500)            250500
_________________________________________________________________
dense_4 (Dense)              (None, 1, 100)            50100
_________________________________________________________________
flatten_1 (Flatten)          (None, 100)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 101
=================================================================
Total params: 351,801
Trainable params: 351,801
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.0: Topology of the model using summary() function in Keras.

The network consists of four hidden layers two input neurons and one output neuron. First, hidden layer is consisted of 100 neurons with sigmoid activation function fully interconnected with the input layer and the second hidden layer of 500 neurons fully connected to the third hidden layer of the same size. The last hidden layer consists of 100 neurons fully connected to one output neuron, which based on activations from previous layers transforms the WRF9km predictors to better fit the purpose of local forecast. The output of summary() function in Keras shows us insight into the topology on figure 3. 0.

Fig 3.1 shows the architecture of this model with two input variables and one forecasted variable.

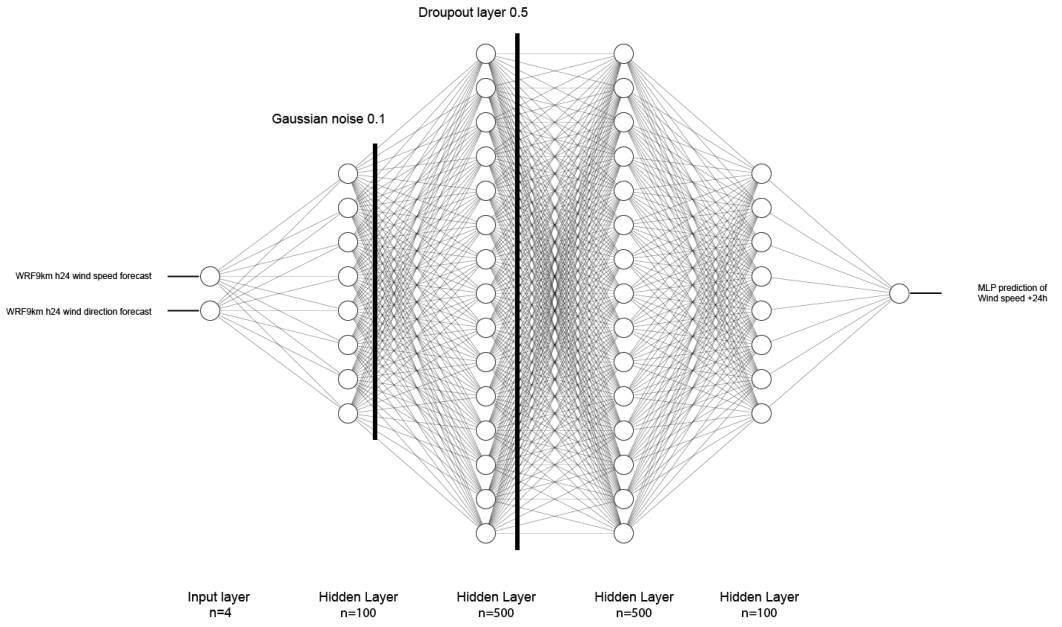


Fig 3. 1: Neural network topology transforming WRF 9km to better fit wind speed measurements

WRF 9km predictions dataset and measurement dataset contain variables in different time granularity – once every six hours for WRF forecast and every 10 minutes for the measurements – the character

of the output function from a neural network learned on linearly interpolated input data for WRF forecast model outputs a discrete function with connects data points which each fit the 6 hour initialization period of the WRF model.

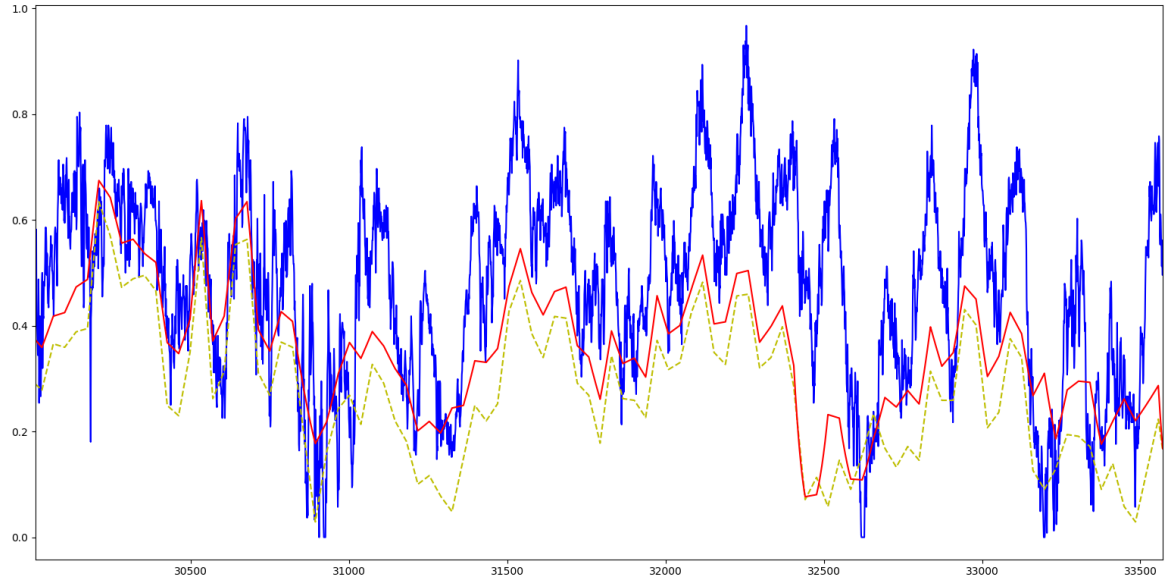


Fig 3.2: Real wind speed measurements (blue), recalculated WRF9km +24h wind speed (red) and original WRF9km forecast (yellow)

It's rather interesting to notice how the neural networks which lags a source of oscillation on the input lags the oscillation on the output as well. For the purpose of experimentation, a similar network was constructed that accepted another two inputs – wind direction and temperature.

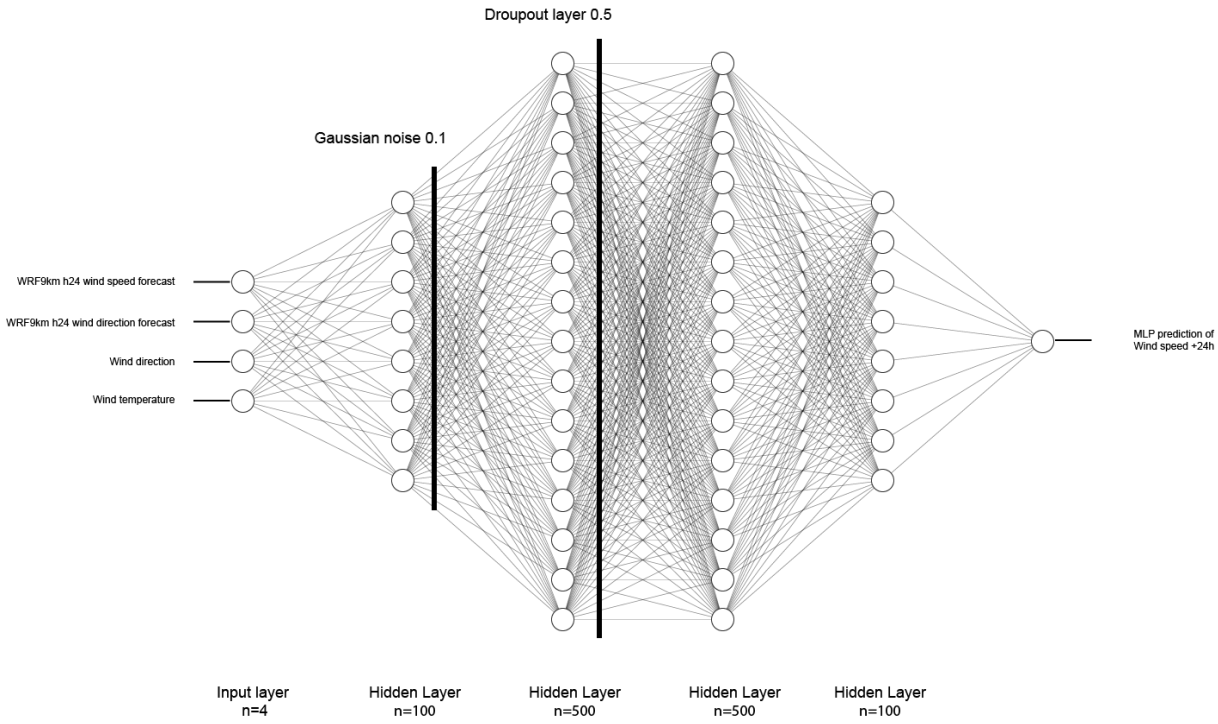


Fig 3.3: MLP network with four input variables and predicted wind speed in 24 hours as output

Performance of the second network was not considerably higher, but in most of the experiments was higher compared to the network with only two inputs. The second network displays considerably higher oscillation on the input as well as on the input which seems to lower the output error. On fig 3.4 it's possible to see the oscillation of the output curve. When the two output functions are compared we find the second one has a bit better performance measured in NRMSE.

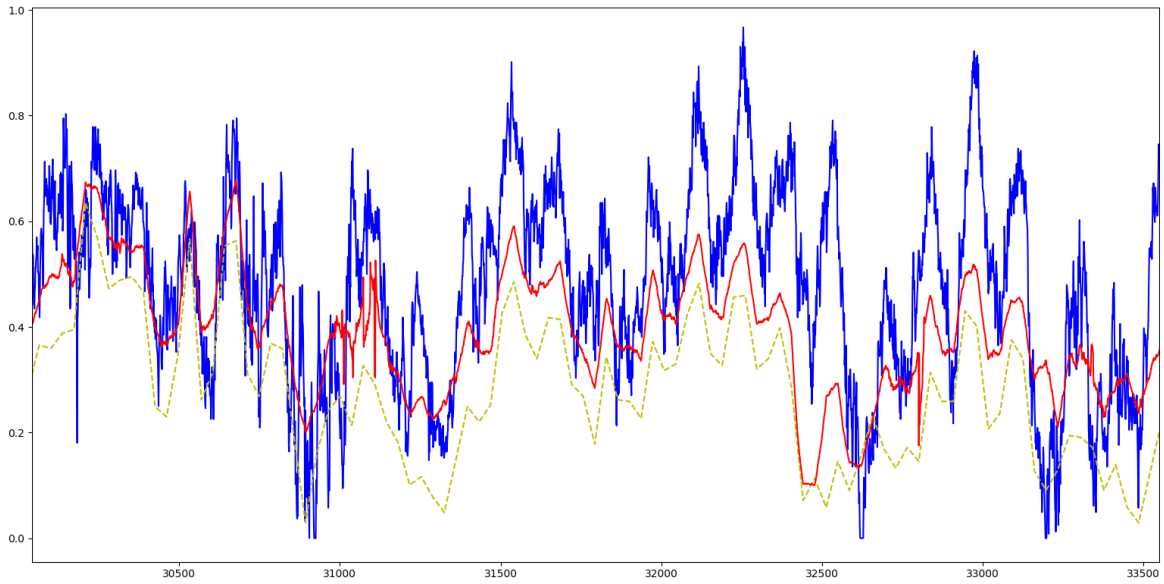


Fig 3.4: More oscillating outputs of second neural network with more inputs

The second neural network with the additional inputs exhibits better performance on the NRMSE metrics but it obviously carries some crucial prediction mistakes and errors. Even though the model performs better by approximately 0.01 RMSE in comparison with the model containing only WRF forecast inputs, connecting current wind direction to the input obviously causes also an interesting short oscillation on the output with relatively high amplitude. An example of the short period high amplitude occasional oscillation is illustrated on fig. 3.5.

Fig 3.5: Oscillation of the output forecast for ANN using current wind direction and temperature as input

The basic question that arises from these observations is, why current wind direction fed to a simple MLP network is a factor that helps predict wind speed 24 hours ahead. Further experiments would be needed to find out why the wind direction plays an important role and if it plays an important role only in 24 hours ahead forecasts. Possible theoretical explanations could be that the progress of the wind direction function from the previous day is closely related to the progress of the wind direction of the upcoming day.

On the other hand including wind direction to the neural network creates unfavorable oscillations of the output. As observed from the graph on fig. 3.5. Interesting is also to interpret what happens afterward on the output. As we can observe the oscillation are not present in the output of network reading only the WRF forecast, on the contrary, it is present only in the topology accepting wind direction as input. A visualization of the inputs can help us to further examine this phenomenon.

If we look at the network as an oscillator that based on the input recounts the continuous input function into a continuous output function it could be presumed that a certain change of the inputs created the short oscillation on the output. As we have carried out the two experiments in separated conditions and we learned the network on different inputs and the same output we cannot compare these two because both networks reached different values of weights in the network. Thus the models are rather different.

But we could for the purpose of theorization put this factor aside and assume that if the learned networks learned using backpropagation of the same expected output even thought additional inputs were used and if both networks converge to having the same character of output while having other parameters fixed we could then assume that oscillations of the outputs are not caused only by including these inputs in the network but they are caused by backpropagating the expected output to the inputs. So the network has to be seen in reverse in the same time we are making the presumptions of its character. To further explain this, the output function of the same network but from a different time frame is plotted on fig. 3.6. A specific part of the output function from fig. 3.4

is used for the illustration. It is possible to observe a drop in the output function followed either by a continuous output or a quick rise to the previous level.

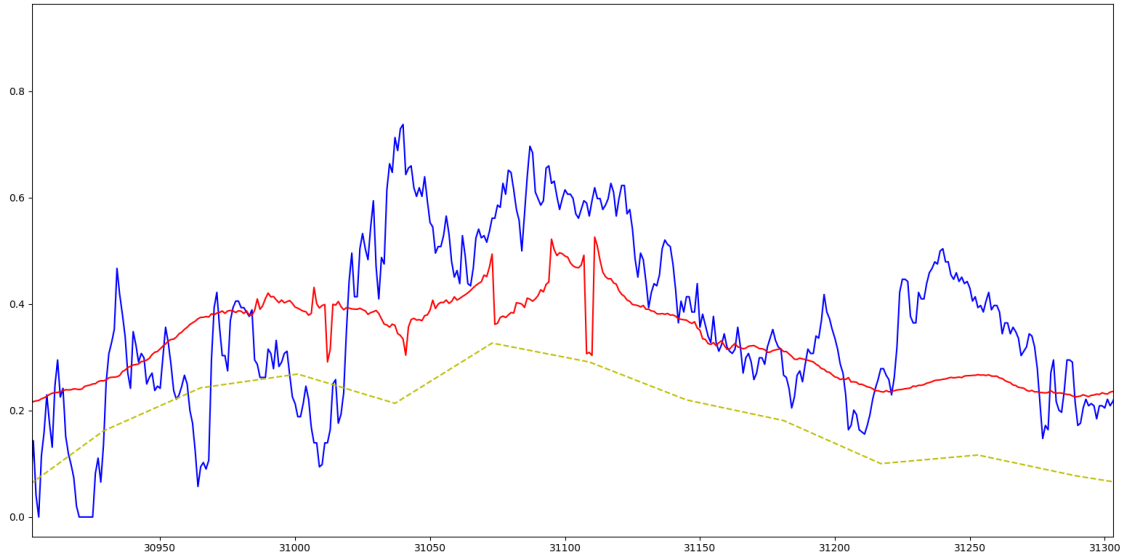


Figure 3.6: Example of sudden drops and oscillations of the output from the same network output

Obviously backpropagating the expected output, which is in this case real wind speed to a combination of inputs from WRF forecast and current wind direction and temperature introduces positive and negative effects that influence the model performance.

Nevertheless, it's very interesting to notice that the phenomenon of sudden drops or quick oscillations occurs frequently around the points in the WRF forecast which are initial points of the linear interpolation function used to interpolate the low-resolution data to match the resolution of the of other inputs and output. The question is how is it possible that the drops are timed in a manner that is that they occur around the point where the trend of the interpolated WRF forecast changes.

It's arguable that the point where the derivation changes is a point where the network receives new information about the WRF forecast, because without any recurrent connections it MLP cannot keep the information. The points of the continuous input function were linearly interpolated but the information the discrete function carries stayed the same. The right question is if we could prevent the drops and oscillations by introducing a different interpolation method. But anyways at this point, we could notice obvious problems with interpolation of the WRF forecast data. By linearly interpolating the values in the forecast we give recurrent network the information about a forecast of the WRF model that will happen in the future by providing the network with a change of the speed with which the interpolated function increases or decreases. Thus it's disputable if the predictive performance of this network is affected by the fact that the inputs might be leaking the value of predictions which would normally not be known in the case we would run the model to predict in real time. Probably not, because the network does not contain any recurrent connections.

To further explain this it's interesting to explain the how the leak might happen in the linear interpolation and what causes only sudden drops and what causes the drops and rises and why we don't see the phenomenon in all the points where the derivation of the linear input function changes.

It is probably relatively hard to explain and we would need to conduct a lot of experiments to either confirm or disapprove the hypothesis.

It's also disputable if the phenomenon occurs in case of local minima or local maxima in the input function. In fig. 3.5 we can see the effect happening in local minima, in fig. 3.6 we can see it happening both in local maxima and out of local minima or maxima.

On fig 3.7 it's possible to observe the graph of modified WRF forecast using the second MLP network on the flat land terrain in Zloncice. As opposed to the graph in fig. 2.5 in the previous chapter where the LSTM network qualitatively underperformed in the prediction of wind speed the MLP network performs very well in remapping the WRF 9km forecast to better match the values measured from the measurement station. One observable fact which is possible to notice is that the MLP architecture with WRF 9km forecast inputs performs well in predicting peaks which are way above the average wind speed. This is probably due to a fact that a global effect in the atmosphere is affecting the wind speed and information about this effect is contained in the WRF forecast and enables the neural network to recalculate the forecast to better match the reality and produce a better forecast.

Even though as well as in the example with an LSTM network applied to the dataset in Zloncice the NRMSE values of this model performance are not so different but the quality for human interpretation of the forecast is way better.

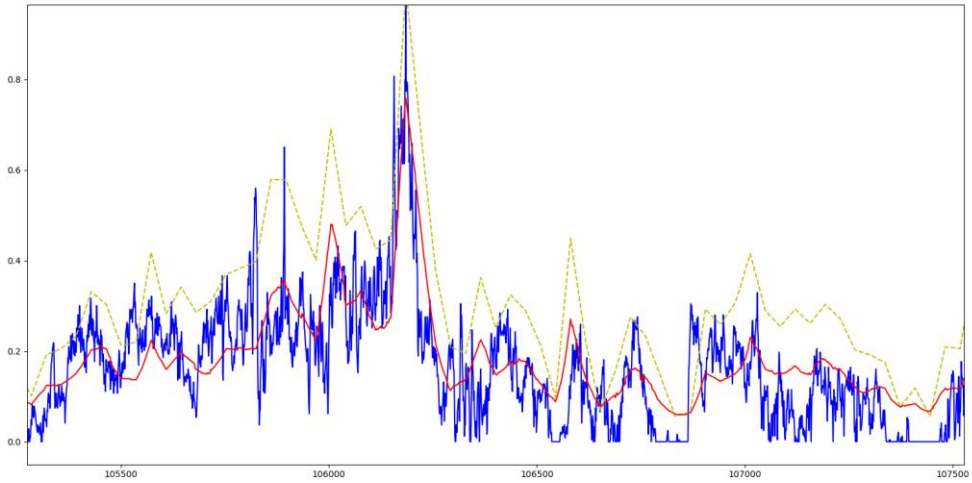


Figure 3. 7: Modified WRF forecast using MLP network on dataset from Zloncice

Other more interesting example drawn out of experiments conducted with the second MLP network on the dataset from Lake Garda shows how including wind direction from previous day affects the improvement of forecasting performance of the MLP network. The graph is depicted on figure 3. 8 As it was previously observed on the dataset from Lake Garda in chapter 9.3 the knowledge of wind direction can help to predict the wind speed. While the thermal winds on Lake Garda are dependent on the sun warming a surrounding terrain as mentioned in chapter 8.3 about thermal Convection and

surrounding topology that contributes to creation of flow of wind in a mountain channel, they occur on regular basis and the timing of this effect is bound to wind direction change. For each out of the two observed most frequent wind directions a different wind speed mean is observed as it is possible to see on figure 1. 39 and 1.40.

It is possible to observe that the second MPL network learned on dataset from lake Garda learned to include the knowledge about wind direction in the forecast which manifests itself by relatively steep jumps up and down on the red curve. The network in order to minimize the error has learned that a change of wind direction from previous day has an effect on the forecasted wind speed mean. And because the whole effect observed on Lake Garda has a rather precise timing while it occurs because of other periodic effect which is sun radiation heating up a surrounding slope it has relatively similar course two consecutive days.

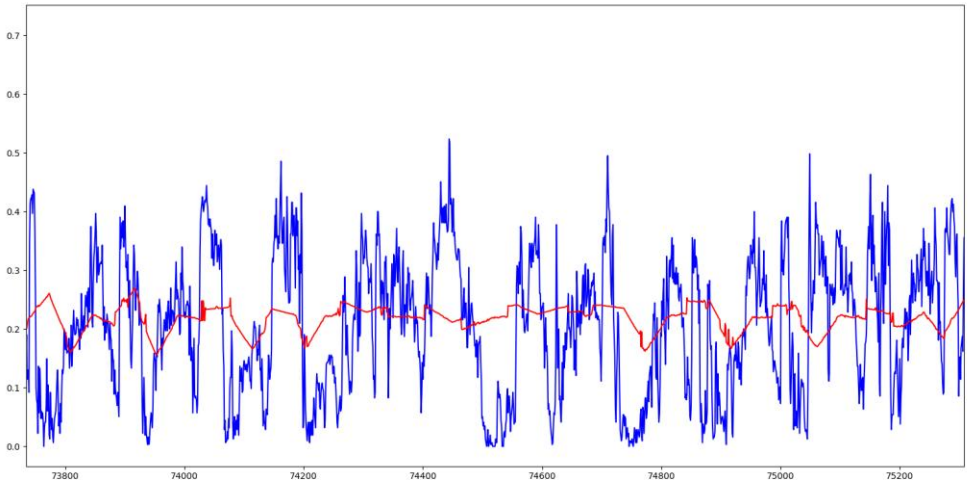


Figure 8. 3: Modified WRF 9km forecast using wind direction from previous day (red) and measured wind speed (blue)

11.2 Training

Training was conducted on a dataset with values of two forecast variables of WRF model and real wind at the time of the 24h ahead forecast in the case of the first neural network. A different dataset with two other input variables – current wind direction and the temperature was used to train the second network that performed a little better and showed better oscillation of the output function.

Both networks were trained using backpropagation and learned on 20 epochs. Outputs of the first hidden layer were randomly noised by adding each forward pass a noise using Gaussian distribution, mean 0 and dispersion 0. 1. On the second layer a dropout of 0. 5 was applied, meaning each learning iteration half of the neurons were randomly set to 0.

11.3 Results

Experimented network was tested on four different datasets from four different locations because of different sampling error dispersion and different wind characteristics to prove the performance enhancement on very different locations. Performance of prediction capability was measured using normalized root means squared error and normalized mean absolute error. Results of measurements are mentioned in table no. 2.

Table 2: Results of error measurement for MLP network using WRF 9k outputs

Station location	Wind speed distribution	Terrain	NRMSE	NMAE
Rhodos, GR	Multimodal	Complex	0.15	0.12
Lake Garda, IT	Multimodal	Complex	0.13	0.11
Zloncice, CZ	Unimodal	Flat	0.11	0.10
Prague, CZ	Unimodal	Flat	0.10	0.08

12 ENSEMBLE MLP NETWORK

Proposed ensemble neural network based on multilayer perceptron, takes two different predictors that perform similar, but make different errors in predictions. Because the two ensembles make different errors it is possible to use them collectively for prediction with obtaining a better prediction result.

Two previously examined models are combined and their predictors are combined in order to produce a better prediction. This principle works only when the input ensembles aim well at the expected value but their predictions disagree a lot, therefore the design was used for models that predict based on two different data sources. The first model makes predictions based on historical wind speed values, the second ensemble fits WRF 9km outputs to local measurements.

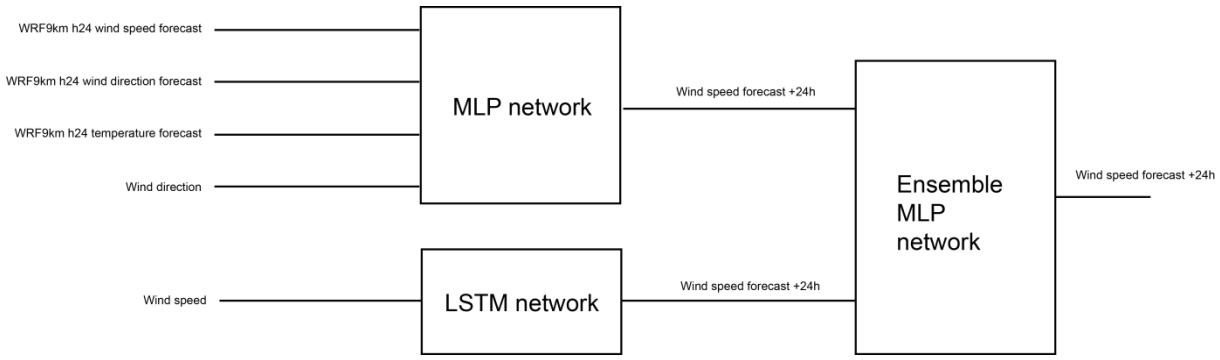


Figure 8.4: Two trained models are used as two ensembles that serve as inputs for a third model.

12.1 Network topology

The ensemble network takes only two inputs, one is the predictor from LSTM network and the second one is a predictor form MLP network transforming the WRF forecast. Summary of the model implemented in Keras is on figure 8.4.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 1, 100)            400
_________________________________________________________________
dropout_2 (Dropout)          (None, 1, 100)            0
_________________________________________________________________
flatten_2 (Flatten)          (None, 100)               0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 101
=================================================================
Total params: 501
Trainable params: 501
Non-trainable params: 0
_________________________________________________________________
```

Figure 8.5: Summary of the model implemented in Keras

While both models predict with a different error it was presumed that the combination of both can leverage a better precision. Making arithmetic mean didn't improve the predictions, suggesting that the error noise does not have a Gaussian distribution. The topology of the network is illustrated on fig. 8.6.

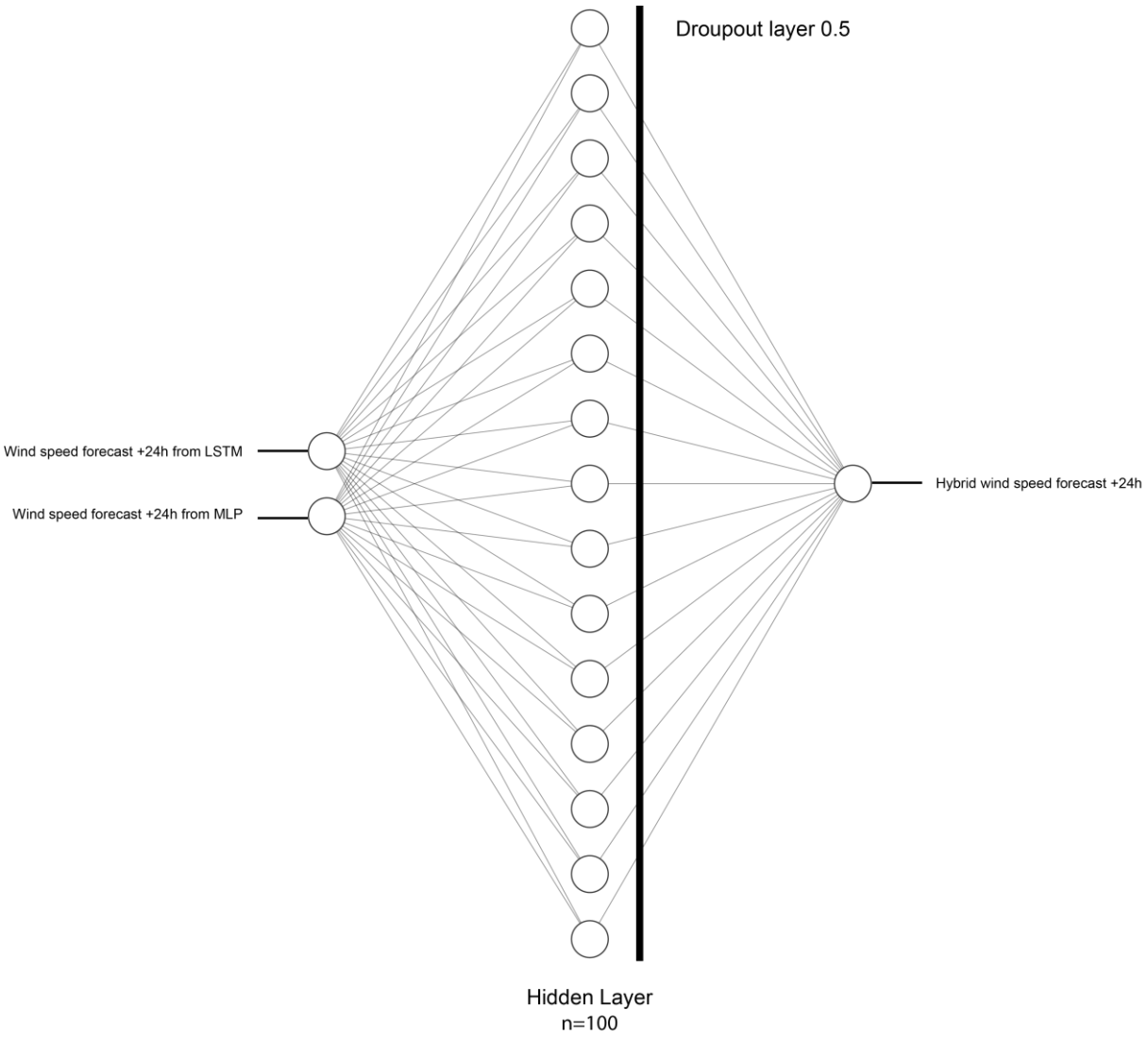


Figure 8.6: Ensemble network with one hidden layer

The network contains dropout layer to increase regularization of the model which is in detail described in chapter 6.5. The dropout mask is used only for learning, for the normal model run the dropout mask is removed.

12.2 Training

Training of the network was done on the outputs of the two previous models and as expected output shifted measurement data was used. For a training run a dropout layer on the hidden layer with 100 neurons was applied with a Gaussian distribution of probability with mean 0.5. In average on each

training sample only half of the hidden units were used in order to increase regularization of the model.

12.3 Results

The ensemble network seems to perform better than any single model on three out of four different test sets. Results can be seen in table no. 3. Performance of predictive capability was measured using normalized root means squared error and normalized mean absolute error.

Table 3: Normalized error of ensemble network predictions

Station location	Wind speed distribution	Terrain	NRMSE	NMAE
Rhodos, GR	Multimodal	Complex	0.13	0.11
Lake Garda, IT	Multimodal	Complex	0.12	0.10
Zloncice, CZ	Unimodal	Flat	0.11	0.09
Prague, CZ	Unimodal	Flat	0.10	0.08

12.4 Reliability of forecast for different time horizons

All the above published results were measured on datasets shifted by 24 hours, for a practical application in wind speed forecasting for windsurfing sailing or kiting a forecast for couple of days is necessary and more valuable. A table with NRMSE for a 1 day ahead forecast, two days and three days ahead forecast is displayed in table no. 3

Table 4: Normalized error produced by ensemble network for different time horizons

Station location	NRMSE +24h	NRMSE +48h	NRMSE +72h
Rhodos, GR	0.13	0.16	0.17
Lake Garda, IT	0.12	0.12	0.13
Zloncice, CZ	0.11	0.11	0.11
Prague, CZ	0.10	0.11	0.11

13 IMPLEMENTATION INTO SERVER WINDGURU.COM

13.1 Windsurfing forecast windguru.com

WindGURU is a serviceproviding weather forecast mainly for the fans of windsurfing, kitesurfing and yachting. The forecasts of WindGURU adopt outputs of numerical weather prediction models. The sense of the website is to provide forecasts in a simply organized manner. Predictions are displayed using tables representing the course of forecast on a given windsurfing spot. WindGURU with the use of numerical prediction models can display forecast of any place on the world. Forecasts are displayed in form of tables which show how weather evolves in certain location in upcomming days. Windguru also offers a service called Windguru Station which comes with a hardware to measure wind speed and wind direction. The data from the station are streamed on the site and displayed next to the forecast. Using better prediction from neural networks models, the service can display even more precise forecast.

13.2 Reliability of forecasts

Reliability of few days ahead forecast is a big issue, but reliability of numerical weather prediction is not any better therefore it would make sense to include the forecasts into the service windguru.com. Nevertheless with further research an implementation of different architecture mentioned in chapter 15 like convLSTM precisions of the models could be improved. In case precision of the models would not significantly improve a mechanism only for one day to three days ahead forecast could be implemented into the interface of windguru.com which can still provide an interesting insight into weather forecasting complementing the NWP displayed on the interface of windguru.com

13.3 User interface

Visualization of the forecast cannot sadly be very well visualized on a map while the experimented and proposed mechanisms using neural networks work only for a single spot wind speed forecast because of the nescessity to train the models on expected outputs available from measurement station which can therefore represent only one point on a map.

In order to visualize the forecast tables with hourly means during the day would be preferable to use for the representation of the forecast. Like the outputs of NWP forecasts on fig. 13.1.

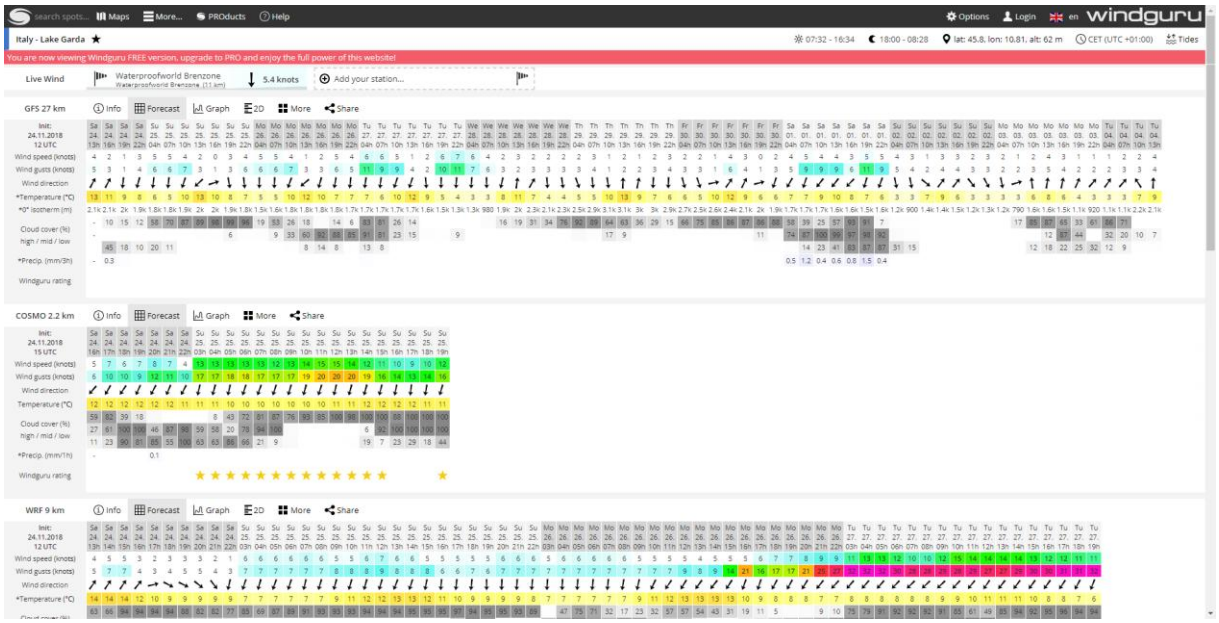


Figure 13.1: Windguru interface with windspeed forecast displayed in tables.

13.4 Proposed REST architecture

Models for the most visited windsurfing spots based on artificial neural networks could be created in a way that they could accept updated inputs from numerical weather prediction models and measurements using API endpoints from the archive of forecasts of server windguru.com and archive of measurements. The interface is sadly confidential, but a proposed REST API endpoints could be used that fit the architecture of Windguru services.

The frameworks used for the experiments conducted in this thesis rely heavily on python tensorflow and keras framework. A virtual machine running a linux distribution with configured Python, Tensorflow and Keras environments can be created for this purpose.

API endpoint to retrieve list of forecasted stations

In order to retrieve a list of forecasted stations a following endpoint could be set up.

```
Request: GET /stations/
Request body: (empty)
Response body:
[{
   "id": 23,
   "name": "Lake Garda",
   "location": [40. 7, -74]
},
{
   "id": 66,
   "name": "Rhods",
   "location": [40. 7, -73]
}]
```

API endpoint to retrieve forecast for three days ahead

The API endpoint providing forecast could be made to output a forecast three days ahead based on the archived data provided from existing Windguru API. Downsampling the forecast outputs to hourly means would be necessary to provide the forecast in the same interface as the interface used to display WRF forecast.

```
Request: GET /forecast/station/23
Request body: (empty)
Response body:
[{
   "id": 23,
   "name": "Lake Garda",
   "location": [40. 7, -74]
   "forecast": [5. 2, 5. 4, 5. 5, 6. 2, 6. 3, 6. 4, 7. 2, 5. 8, 5. 2, 5. 4,
                5. 5, 6. 2, 6. 3, 6. 4, 7. 2, 5. 8, 5. 2, 5. 4, 5. 5, 6. 2,
                6. 3, 6. 4, 7. 2, 5. 8, 5. 2, 5. 4, 5. 5, 6. 2, 6. 3, 6. 4,
                7. 2, 5. 8, 5. 2, 5. 4, 5. 5, 6. 2, 6. 3, 6. 4, 7. 2, 5. 8,
                5. 2, 5. 4, 5. 5, 6. 2, 6. 3, 6. 4, 7. 2, 5. 8, 5. 2, 5. 4,
                5. 5, 6. 2, 6. 3, 6. 4, 7. 2, 5. 8, 5. 2, 5. 4, 5. 5, 6. 2,
                6. 3, 6. 4, 7. 2, 5. 8]
}]
```

14 POSSIBLE IMPROVEMENTS

There are more possible caused of forecast imprecisions and more different approaches to make the precision better. Sampling error in measurements enters the MPL neural network from output making it possible to reach only a certain precision while the information encoded in the sampling procedure in the data is obfuscated by application of operations which are irreversible and contribute to a loss of information. Revision of hardware in the electronics would be necessary to perform

The learning algorithm has it's limitation in order what it can learn or it gets trapped in a local minimum instead of global minima in the backpropagation of gradient into the weights in the network. Therefore experimentation with different learning algorithms could be performed.

One point inputs from measurements and WRF forecast don't contain enough substantial data to enable to calculate future values of wind speed. Therefore multiple pixel time series inputs could be used to improve the precision.

The ANN making correction to WRF forecast output operates each time only with the distributions of two datasets the input dataset and output dataset and is very limited in what it can do while it is not able to evaluate the information on a historical timeframe but only on a single value vector on the input, thus having to decide which value out of the output distribution to draw based only on the 1D input vector containing information about wind in one specific time point. For example on the dataset of Lake Garda current wind direction input which has a rather step drop and rise characteristics also makes the same rises and drops on the output a network that could perform the wind speed adjustment with a delay based on wind direction should perform better.

The MLP network transforming the forecast can analyze only one point at a time a parallel input with multiple values would better help the network learn how change in the distribution of wind speed frequency occurrence could possibly improve the forecast.

The ensemble network has only two ensembles, more models could be trained with different functionality to improve the ensemble forecast. More different ensembles like simple average distribution could be fed into the inputs of the MLP performing transformation of WRF9km model to make the network understand the average course of wind respectively use this model as ensemble.

A more precise NWP model could be as basis for the transformations, like COSMO 2.9 km to improve the precision of NWP prediction inputs.

Only forecasts for 24 hours ahead for the initializations of WRF9km model were taken and interpolated into dataset with 10 minute intervals measurements. Instead of only one prediction point each 6 hours a forecasts for 25^{th} to 29^{th} hour can be taken to improve the precision.

15 FURTHER RESEARCH

It has been presented that a certain sampling error enters measurements. More research has to be done in order to figure out how much the sampling error affects the forecasting abilities of numerical weather prediction models as well as predictions made by neural networks.

It has been showed that wind speed distribution is not always unimodal and further research on the multimodality of wind speed distribution has to be done in order to better understand predictions of wind speed. Perhaps wind speed prediction viewed from the perspective of neural networks has to be

changed in order to improve learning algorithms. Learning algorithms in wind prediction need probably different error calculations mechanism.

More has to be done in the area of wind speed measurements archivation. The contemporary approach toward wind speed data archivation comes with mathematical operations that might contribute to loss of information in wind speed measurement. When only single value for each time interval is archived it contributes to substantial loss or precision in the information stored thus reducing the data resolution. The question is how much we actually reduce the resolution of wind speed data when we reduce an information about a whole spectrum of speeds that were present in a time interval and average them to get a single value.

We can probably almost never evade this approach towards measurement, because no matter how precise we want to go with wind speed measurement we will always get to a point where we hit a limit by measuring accumulated energy or potential in a time interval and deriving a single value from that cumulative potential or cumulative energy. Nevertheless different experiments could beconducted to figure out how the measurement technique affects resolution of the data and its predictability presuming that the speed with which error accumulation occurs is linear or near linear with increasing size of sliding window. Theoretically we down sampled the resolution of data when we archived a single value for a given time frame, lets say 10 minutes. So each ten minutes we record a number which corresponds to a mean of values measured during the whole interval. Lets presume we can record 300 measurements for a 10 minute interval and then we take all the values count a mean out of these values and record it in our database. If we have a database of wind speed mean each ten minutes we get to 144 values a day. So considering the speed with which the information vanishes is constant, which might not hold true, it is analogous to down sampling measurements from one value each ten minutes to approximately one value every two days. And we could then try what predictive capability will we reach adjusting the forecast ahead window proportionally thus instead of 24 hours ahead in the previous resolution of 144 values per day adjust the prediction window proportionally so under 300 days ahead and try the same forecasting horizon on a dataset which was not down sampled.

But from practical point of view the question is if there are even available datasets that long, that retained the same calibration of the measurement device with the same error dispersion on such a long time frame. Therefore it is possible that we would need to go the exact opposite way and try to sample in lower resolution by a more precise method (like ultrasonic or electrical resistance measurement) enabling us to sample on such short time frames with enough precision to conduct such experiments. Because the fundamental question is what predictive capability has the change in wind speed distribution for prediction of wind speed course and at what resolution is this distribution change recognizable if this change contains any information valuable for prediction.

Another possible drawback of wind speed measurement might be the concept of measurement itself. Some researches already suggested that from the point of machine learning, it seems, for some reason that dependency of generated output power on wind speed is not exactly deterministic. There might be a couple of reasons for that. First reason might be the imprecision of measurement of one or both variables. The imprecision that can theoretically occur on wind speed measurement which were mentioned and discussed in this thesis already. Nevertheless the same principles can apply to output power measurement and errors and imprecision can occur in the devices used for measurement the same way it can occur in anemometers. A second explanation, assuming that the devices used for measurement and the methods used to measure contribute only partially to the error, can be, that

some other variable that we did not include in the calculations or the models enters the system which in other cases has a negligible effect. A third reason can be, that what we measure as wind speed is rather a potential that enters only partially into the equation with released energy and some other variable also affects the amount of released energy. And the fourth reason could be that the efficiency of the generator changes based on external variables not entering the system. Anyways the question is what is more precise as prediction input into machine learning models, whether it is wind speed the way how we can measure it or whether it is the amount of transferred energy which we can measure, supposing the interdependence between the two is not determinist from any reason, being it the measurement method or another explaining variable which is missing in out models.

From the perspective of wind speed prediction and from the perspective of neural networks, we might be hitting the limit of what is predictable from a single spot dataset while the sampling errors enter the inputs outputs and learning algorithms we use in neural networks or any other machine learning algorithms. The research question is how much the measurement precision and downsampling corresponds to error occurrence. Simple experiments could be done using different downsampling algorithms and simulations of error occurrence.

The predictive approaches presented in this work are extremely simplistic in their use of memory blocks, more has to be done in research of different LSMT and GRU architectures like convLSTM for weather prediction. Alternatively, different architectures of neural networks with memory units have to be developed in order to process geospatial information that is not placed in a grid with equal space between the different sampling points. This specifics of weather data sampling makes the task much more difficult because for the interpolation of missing points we need to use operations that are either precise and computationally resourceful or we need to feed the input information into neural networks, not as data in a grid form with equal distance in between data pixels. This factor makes it impossible to use convolutions that proved to be very powerful in other disciplines like image recognition.

Alternatively approaches used in numerical weather prediction can be combined with neural networks to simplify certain tasks in weather prediction.

16 CONCLUSION

Several ANN topologies for wind speed prediction were introduced as well as ensemble ANN of two ensembles a MLP and LSTM network, which performed better in the tests than a single model,

LSTM or MLP network alone. Further research on the topic of wind speed forecasting is needed to answer what causes the effects observed on the experimental networks and if the error of predictions could be minimized by connecting forecasts from separated models into another ANN using different topology of connections in between LSTM cells and perceptron cells.

As observed on four different data sets from different locations in Europe with different features and rather different distributions of measurement the ensemble neural network seems to perform better than particular single models and averaging the models did not increase precision, suggesting distribution of error in the two predictions is not Gaussian. As well it turns out that it is possible using a simple ANN consisting of multilayer perceptron accepting one value of input variables each time to transform a forecast output from numerical weather prediction model namely WRF with resolution 9 x 9 km into a forecast that better fits local measurements. It has been also showed that Long Short-Term Memory network can predict wind speed 24 hours ahead using only historical wind speed as input with precision similar to ANN transforming WRF forecast into a better performing forecast.

A problem which is necessary to mention in interpretation of the results of the experiments conducted is the way the metrics behave. Experiments were conducted on datasets with different wind conditions different daily distributions and placing in the landscape. Even thought experiments were conducted on normalized datasets and thus values of the error metrics measured in the experiments correspond to their normalized values, there is a substantial problem in interpretation of the results. On the first sight it can seem the models proposed reach better predictive capabilities on datasets with unimodal distributions and flat land, but that interpretation can be misleading. Because when we closely observe the graphs we can notice that normalization has very substantial effect on NRMSE or NMAE, while the dataset from Rhodos which produced the largest normalized error contains values that are scattered rather differently throughout the dataset in concrete more equally dispersed around the median and mean. On the other hand dataset from Zloncice that produces noticeably lower values of NRMSE and NMAE has most of its values scattered in relatively small interval in the bottom of the graph with a couple of outliers that caused that the normalization procedure could have squished the normalized error. And absolute error from the values of wind speed in knots is also incomparable because the characteristics of the two places are totally different.

Different theoretical approaches toward wind speed data processing were presented as well as different approaches toward post-processing the data in order to create a wind speed forecast. As a deeper insight into the wind measurement data was done it turns out that there are different reasons why wind prediction using neural networks works and fails in the same time.

There are a couple of reasons why the specific use of neural networks increase the precision of wind speed forecast of numerical weather prediction models using different transformations learned using learning algorithms. Different effects of including different variables have been observed and through iterative process included and omitted as inputs and expected outputs of machine learning models presented in this work in particular neural networks.

One observation conducted from examining the results and signal functions produced in the experiments is concerning an adequate architecture for single spot wind speed forecasting. The three presented experimental architectures are lacking a few capabilities that could improve the signal processing of wind speed and wind speed numerical forecast predictors in order to create a better performing forecasting mechanism respectively forecasting model based on artificial neural networks.

Even thought the idea of third ensemble network connecting the two different models together should have leveraged the benefits of both approaches thus memory cells in neural networks as well as signal processing using a multilayer percepton the approach is really weak in its performance. What would be really needed for this purpose would be an architecture, that would enable to store information for longer periods of time which would be later available for post processing the data thus leveraging both approaches better. Some solution could be hopefully found in modification of convLSTM architectures or similar.

17 REFERENCES

Baldi, P. S. (2014). The Dropout Learning Algorithm. *Artif Intell. 2014* , pp. 78-122.

Brownlee, J. (2017). *A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size.* Retrieved 11 7, 2018, from Machine Learning Mastery: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

Brownlee, J. (2015). *A Tour of Recurrent Neural Network Algorithms for Deep Learning.* Retrieved 11 22, 2018, from Machine Learning Mastery: https://machinelearningmastery.com/recurrent-neural-network-algorithms-for-deep-learning/

Castellania, F. B. (2014). Wind energy forecast in complex sites with a hybrid neural network and CFD based method. *Energy Procedia, Volume 45* , pp. 188-197.

Compact Analysis. (2016). *BEST TYPES OF ANEMOMETER TO BUY TODAY.* Retrieved 10 12, 2018, from Compact Analysis: https://compactanalysis.com/wp-content/uploads/2016/09/Best-Types-of-Anemometer-to-Buy-Today.png

Davis Instruments Corp. (2018). *Standard Anemometer - Installation manual.* Retrieved 11 18, 2018, from Davis Instruments.

Devaraja, D. Y. (2002). Radial basis function networks for fast contingency ranking. *International Journal of Electrical Power & Energy Systems, Volume 24, Issue 5* , pp. 7-11.

Didcock, N. J.-J. (2015). Regularisation methods for neural network model averaging. *Engineering Applications of Artificial Intelligence, Volume 41* , pp. 128-138.

Dolezel, P. S. (2016). Weight Initialization Possibilities for Feedforward Neural Network with Linear Saturated Activation Functions. *IFAC-PapersOnLine* , pp. 49-54.

Electronic tutorial. (2016). *Op-amp comparator.* Retrieved 10 22, 2018, from Electronic tutorial: https://www.electronics-tutorials.ws/opamp/opamp105.gif

Electronics Tutorial. (2015). *Sensors and Transducers.* Retrieved 10 21, 2018, from Electronics Tutorial: https://www.electronics-tutorials.ws/io/io50.gif

Errington, J. (2007). *Analog to Digital converters (ADC's).* Retrieved 10 25, 2018, from John Errington's Data Conversion Website: http://www.skillbank.co.uk/SignalConversion/images/E33Fig01.gif

Eseye, A. Z. (2017). *A Double-Stage Hierarchical Hybrid PSO-ANN Model for Short-Term Wind Power Prediction.* Retrieved 10 30, 2018, from https://www.researchgate.net/profile/Abinet_Eseye/publication/317693740_A_double-stage_hierarchical_hybrid_PSO-ANN_model_for_short-term_wind_power_prediction/links/59f6b3b6a6fdcc075ec60579/A-double-stage-hierarchical-hybrid-PSO-ANN-model-for-short-term-wi

Fetzner, M. (2015, 4 16). *Load and Wind Power In-Feed Forecast.* Retrieved 11 1, 2018, from https://www.ethz.ch/content/dam/ethz/special-interest/itet/institute-eeh/power-systems-dam/documents/SAMA/2015/Fetzner-MA-2015.pdf

Filik, B. F. (2017). Wind Speed Prediction Using Artificial Neural Networks Based on Multiple Local Measurements in Eskisehir. *Energy Procedia, Volume 107* , pp. 264-269.

Fortmann, S. (2012). *Accurately Measuring Model Prediciton Error.* Retrieved 10 29, 2018, from Scott Fortmann: http://scott.fortmann-roe.com/docs/MeasuringError.html

Fremer, M. (2015). *Analog to digital converter Shootout.* Retrieved 11 22, 2018, from Analog Planet: https://www.analogplanet.com/images/styles/600_wide/public/0315AD.gif

Ghaderi, A. S. (2017). *Deep Forecast: Deep Learning-based Spatio-Temporal Forecasting.* Retrieved 11 6, 2018, from https://arxiv.org/pdf/1707.08110.pdf

Guillén, A. R. (2007). Output value-based initialization for radial basis function neural networks. *Neural Processing Letters, Volume 25, Issue 3* .

Guo, J. (2013). *BackPropagation Through Time.* Retrieved 11 17, 2018, from http://ir.hit.edu.cn/~jguo/docs/notes/bptt.pdf

Hiton, G. (2016). *Neural Networks for Machine Learning.* Retrieved 10 1, 2018, from Coursera: https://www.coursera.org/learn/neural-networks

Hochreiter, S. S. (1997). Long Short-Term Memory. *Neural Computation 9* , pp. 1735-1780.

Jiang, J. (2016). *TensorFlow - A system for large-scale machine learning.* Retrieved 22 11, 2018, from Github - Jiang Jaho: https://jahojiang.github.io/

Kadhem, A. W. (2017). Advanced Wind Speed Prediction Model Based on a Combination of Weibull Distribution and an Artificial Neural Network. *Energies* .

Kang, E. (2017). *Long Short-Term Memory (LSTM): Concept.* Retrieved 10 25, 2018, from Medium: https://medium.com/@kangeugine/long-short-term-memory-lstm-concept-cb3283934359

Kaufman, J. (2015, 12 24). *Detecting Tanks.* Retrieved 10 2, 2018, from Jeff Kaufman: https://www.jefftk.com/p/detecting-tanks

Kosinov, S. A. (2017, 3 28). *An overview of the main types of neural network architecture.* Retrieved 10 13, 2018, from Neural network architecture overview: http://0--key.github.io/machine-learning/neural-network/architecture/overview.html

Kratzert, F. (2016). *Understanding the backward pass through Batch Normalization Layer.* Retrieved 11 23, 2018, from Flair of machine learning: https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html

Lagandula, A. (2018). *Perceptron Learning Algorithm: A Graphical Explanation Of Why It Works.* Retrieved 10 15, 2018, from Towards Data Science: https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975

Larraondo, P. I. (2017). *Automating weather forecasts based on convolutional networks.* Retrieved from https://deepstruct.github.io/ICML17/1stDeepStructWS_paper_2.pdf

Larraondo, P. I. (2017). *Automating weather forecasts based on convolutional networks.* Retrieved 11 22, 2018, from https://deepstruct.github.io/ICML17/1stDeepStructWS_paper_2.pdf

Lauret, P. D. (2014). A Neural Network Post-processing Approach to Improving NWP Solar Radiation Forecasts. *Energy Procedia* , pp. 1044-1052.

Maarten H.P.Kole, G. J. (2012). Signal Processing in the Axon Initial Segment. *Neuron, Volume 73, Issue 2* , pp. 235-247.

Mazur, M. (2015). *A Step by Step Backpropagation Example.* Retrieved 11 2, 2018, from Matt Mazur: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

McCormick, C. (2013, 8 15). *Radial Basis Function Network (RBFN) Tutorial.* Retrieved 10 22, 2018, from Chris McCormick: http://mccormickml.com/2013/08/15/radial-basis-function-network-rbfn-tutorial/

Nasrulhaq, O. (2012). *Rotary encoder sensor.* Retrieved 10 21, 2018, from Early Warning Fault Detection System: http://2.bp.blogspot.com/-Q8Hj8jQlMaw/UKnnj4PHBHI/AAAAAAAAALo/Rf1BlTthCk8/s1600/optical+sensor.jpg

Novotny, R. (2002). *Weibullovo rozdělení při analýzách bezporuchovosti.* Retrieved 10 12, 2018, from Elektrorevue: http://www.elektrorevue.cz/clanky/02017/index.html

Ranganayaki, V. D. (2016). An Intelligent Ensemble Neural Network Model for Wind Speed Prediction in Renewable Energy Systems. *ScientificWorldJournal* .

Rasp, S. L. (2018). *Neural networks for post-processing ensemble weather forecasts.* Retrieved 2018, from Meteorological Institute, Ludwig-Maximilians-Universitat: https://arxiv.org/pdf/1805.09091.pdf

Sahoo, D. P. (2014). *Online Deep Learning: Learning Deep Neural Networks on the Fly.* Retrieved 11 13, 2018, from https://arxiv.org/pdf/1711.03705.pdf

Scheau, C. (2016, 11 16). *Regularization in deep learning.* Retrieved 11 20, 2018, from https://chatbotslife.com/regularization-in-deep-learning-f649a45d6e0

Shi, X. C.-Y. (2015). Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *Advances in Neural Information Processing Systems 28* .

Srivastava, N. H. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research 15* , pp. 1929-1958.

Srivastava, P. (2017). *Essentials of Deep Learning : Introduction to Long Short Term Memory.* Retrieved 10 29, 2018, from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/

Texas Instruments. (2007). *Noise Analysis in Operational Amplifier Circuits.* Retrieved 10 14, 2018, from http://www.ti.com/lit/an/slva043b/slva043b.pdf

Voyant, C. M.-L. (2012). Numerical weather prediction (NWP) and hybrid ARMA/ANN model to predict global radiation. *Energy* .

Weber, N. (2017). *Why LSTMs Stop Your Gradients From Vanishing: A View from the Backwards Pass.* Retrieved 11 18, 2018, from weberna's blog: https://weberna.github.io/blog/2017/11/15/LSTM-Vanishing-Gradients.html

Wikipedia. (2018). *Axon hillock.* Retrieved 11 1, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Axon_hillock

Wikipedia. (2018). *Backpropagation through time.* Retrieved 10 28, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Backpropagation_through_time

Wikipedia. (2018). *Long short-term memory.* Retrieved 11 4, 2018, from Wikipedia: https://de.wikipedia.org/wiki/Long_short-term_memory#/media/File:Long_Short_Term_Memory.png

Wikipedia. (2018). *Perceptrons (book).* Retrieved 10 4, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Perceptrons_(book)

Wikipedia. (2018). *Voltage divider*. Retrieved 11 23, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Voltage_divider#/media/File:Impedance_voltage_divider.svg

Windguru. (2017). *Quick install guide (Windguru Station 2).* Retrieved 10 22, 2018, from Windguru: https://stations.windguru.cz/img/station2-quick.jpg

Yadav, V. (2016, 11 6). *Why dropouts prevent overfitting in Deep Neural Networks.* Retrieved 10 22, 2018, from Medium: https://medium.com/@vivek.yadav/why-dropouts-prevent-overfitting-in-deep-neural-networks-937e2543a701

Zhang, J. C. (2013). A Multivariate and Multimodal Wind Distribution model. *Renewable Energy, Volume 51* , pp. 436-447.

Zhao, J. G. (2017). Multi-step wind speed and power forecasts based on a WRF simulation and an optimized association method. *Applied Energy, Volume 197* , pp. 183-202.

Zhu, H. L. (2015). *A Power Prediction Method for Photovoltaic Power Plant Based on Wavelet Decomposition and Artificial Neural Networks.* Retrieved 11 1, 2018, from https://www.mdpi.com/1996-1073/9/1/11/htm

APENDIX A

A. SOURCE CODE

Complete source code with experimental datasets is available on github:

https://github.com/grosaktom/ann-windspeed-forecast

A.1 Neural network

```
import numpy
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import GaussianNoise
import math
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import functions_single

#-----------------------------------------------------------------------------------------
#           Create dataset
#-----------------------------------------------------------------------------------------



#wrf_file = 'data/station_data_1033_2016010100-20180823-rhodos-wrf9.csv'
#measurement_file = 'data/station_data_1033_2016010100-20181018-rhodos-mereni.csv'

#wrf_file = 'data/station_data_632_2016010100-20181124-garda-wrf9.csv'
#measurement_file = 'data/station_data_632_2016010100-20181124-garda-mereni.csv'

#wrf_file = 'data/station-data-55-zloncice-wrf.csv'
#measurement_file = 'data/station-data-55-zloncice-measurement.csv'

wrf_file = 'data/station_data_525_2016010100-20181121-hajovka-wrf9.csv'
measurement_file = 'data/station_data_525_2016010100-20181121-hajovka-measurements.csv'

epochs = 10

merged_df = functions_single.merge_datasets(wrf_file, measurement_file)

#-----------------------------------------------------------------------------------------
#           Pick variables and normalize dataset
#-----------------------------------------------------------------------------------------


input_variables = ['wind_direction']
input_variables_mlp = ['h24_wind_speed','h24_wind_dir','h24_temperature',
'wind_direction','temperature']
input_variables_lstm = ['wind_avg', 'temperature']
output_variables = ['wind_avg_in24h']
```

```
merged_df_scaled = functions_single.normalize_dataset(merged_df,
input_variables=input_variables, output_variables=output_variables)
merged_df_scaled_lstm = functions_single.normalize_dataset(merged_df,
input_variables=input_variables_lstm, output_variables=output_variables)
merged_df_scaled_mlp = functions_single.normalize_dataset(merged_df,
input_variables=input_variables_mlp, output_variables=output_variables)
ratio = 0.6
training_x, training_y, test_x, test_y = functions_single.split_dataset(merged_df_scaled,
input_variables=input_variables, output_variables=output_variables, ratio=ratio)
training_x_mlp, training_y_mlp, test_x_mlp, test_y_mlp =
functions_single.split_dataset(merged_df_scaled_mlp, input_variables=input_variables_mlp,
output_variables=output_variables, ratio=ratio)
training_x_lstm, training_y_lstm, test_x_lstm, test_y_lstm =
functions_single.split_dataset(merged_df_scaled_lstm, input_variables=input_variables_lstm,
output_variables=output_variables, ratio=ratio)


merged_df_length = len(merged_df)
training_length = int(ratio * merged_df_length)


model_MLP = Sequential()
model_MLP.add(Dense(100, input_shape=(1, len(input_variables_mlp) )))
model_MLP.add(GaussianNoise(0.1))
model_MLP.add(Dense(500 ))
model_MLP.add(Dropout(0.5))
model_MLP.add(Dense(500 ))
model_MLP.add(Dense(100 ))
model_MLP.add(Flatten())
model_MLP.add(Dense(len(output_variables), activation='sigmoid'))
model_MLP.compile(loss='mean_squared_error', optimizer='adam')
model_MLP.fit(training_x_mlp, training_y_mlp, epochs=epochs, batch_size=20, verbose=2)



train_predict_MLP = model_MLP.predict(training_x_mlp)
test_predict_MLP = model_MLP.predict(test_x_mlp, batch_size=20)

train_predict_plot_MLP = train_predict_MLP[:,0]
train_predict_plot_MLP = numpy.reshape(train_predict_plot_MLP, ( training_length, 1))

test_predict_plot_MLP = numpy.empty(merged_df_length)
test_predict_plot_MLP[:] = numpy.nan
test_predict_plot_MLP[len(train_predict_plot_MLP)+1:merged_df_length] = test_predict_MLP[:,0]


model_LSTM = Sequential()
model_LSTM.add(LSTM(50, return_sequences=True, input_shape=(1, len(input_variables_lstm) )))
model_LSTM.add(LSTM(50 ))
model_LSTM.add(Dense(len(output_variables), activation='sigmoid'))
model_LSTM.compile(loss='mean_squared_error', optimizer='adam')
model_LSTM.fit(training_x_lstm, training_y_lstm, epochs=epochs, batch_size=20, verbose=2)

train_predict_LSTM = model_LSTM.predict(training_x_lstm)
test_predict_LSTM = model_LSTM.predict(test_x_lstm, batch_size=20)


train_predict_plot_LSTM = train_predict_LSTM[:,0]
train_predict_plot_LSTM = numpy.reshape(train_predict_plot_LSTM, ( training_length, 1))

test_predict_plot_LSTM = numpy.empty(merged_df_length)
test_predict_plot_LSTM[:] = numpy.nan
test_predict_plot_LSTM[len(train_predict_plot_LSTM)+1:merged_df_length] =
test_predict_LSTM[:,0]

train_predict_LSTM_reshaped = train_predict_LSTM.reshape(train_predict_LSTM.shape[0], 1)
train_predict_MLP_reshaped = train_predict_MLP.reshape(train_predict_MLP.shape[0], 1)
train_hybrid = numpy.dstack([train_predict_LSTM_reshaped, train_predict_MLP_reshaped,
training_x])
# print(train_hybrid.shape)

test_predict_LSTM_reshaped = test_predict_LSTM.reshape(test_predict_LSTM.shape[0], 1)
test_predict_MLP_reshaped = test_predict_MLP.reshape(test_predict_MLP.shape[0], 1)
test_hybrid = numpy.dstack([test_predict_LSTM_reshaped, test_predict_MLP_reshaped, test_x])
```

```
model_HYBRID = Sequential()
model_HYBRID.add(Dense(100, input_shape=( 1, train_hybrid.shape[2] )))
model_HYBRID.add(Dropout(0.5))
model_HYBRID.add(Flatten())
model_HYBRID.add(Dense(1, activation='sigmoid'))
model_HYBRID.summary()
model_HYBRID.compile(loss='mean_squared_error', optimizer='adam')
model_HYBRID.fit(train_hybrid, training_y, epochs=epochs, batch_size=20, verbose=2)


train_predict_HYBRID = model_HYBRID.predict(train_hybrid)
test_predict_HYBRID = model_HYBRID.predict(test_hybrid)

train_predict_plot_HYBRID= train_predict_HYBRID[:,0]
train_predict_plot_HYBRID = numpy.reshape(train_predict_plot_HYBRID, ( training_length, 1))

test_predict_plot_HYBRID = numpy.empty(merged_df_length)
test_predict_plot_HYBRID[:] = numpy.nan
test_predict_plot_HYBRID[len(train_predict_plot_HYBRID)+1:merged_df_length] =
test_predict_HYBRID[:,0]

scaled_wind_avg = numpy.array(merged_df_scaled['wind_avg_in24h'])

# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(training_y[:,0], train_predict_MLP[:,0]))
print('MLP train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(test_y[:,0], test_predict_MLP[:,0]))
print('MLP test Score: %.2f RMSE' % (testScore))

# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(training_y[:,0], train_predict_LSTM[:,0]))
print('LSTM train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(test_y[:,0], test_predict_LSTM[:,0]))
print('LSTM test Score: %.2f RMSE' % (testScore))

# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(training_y[:,0], train_predict_HYBRID[:,0]))
print('Hybrid train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(test_y[:,0], test_predict_HYBRID[:,0]))
print('Hybrid test Score: %.2f RMSE' % (testScore))

# calculate root mean absolute error
trainScore = mean_absolute_error(training_y[:,0], train_predict_HYBRID[:,0])
print('Hybrid train Score: %.2f MAE' % (trainScore))
testScore = mean_absolute_error(test_y[:,0], test_predict_HYBRID[:,0])
print('Hybrid test Score: %.2f MAE' % (testScore))

plt.plot(scaled_wind_avg, "k")
plt.plot(train_predict_plot_LSTM, "c")
plt.plot(train_predict_plot_MLP, "y")
plt.plot(test_predict_plot_LSTM, "b")
plt.plot(test_predict_plot_MLP, "r--")
plt.plot(test_predict_plot_HYBRID, "g--")

plt.show()
```

A.2 Data pre-processing

```
import numpy
import matplotlib
import pandas
from sklearn.preprocessing import MinMaxScaler
from pandas import datetime
from pathlib import Path
numpy.random.seed(7)


def date_parser(x):
        return datetime.strptime(x, '%Y-%m-%d %H:%M:%S')

def date_parser_wrf(x):
        return datetime.strptime(x, '%Y-%m-%d %H:%M:%S')
```

```
def transpose_wrf_variables(df, variables):
    df_transposed = None
    for num in variables:
        df_filtered = df.loc[df['id_var'] == num]
        df_filtered.columns = [str(col) + '_' + variables[num] for col in df_filtered.columns]
        if (df_transposed is None):
            df_transposed = df_filtered.copy()
            df_transposed = df_transposed.rename(columns = {'datum_'+variables[num]:'datum'})
        else:
            df_transposed    =    pandas.merge(df_transposed.set_index("datum",    drop=False),
df_filtered, how='inner', left_on='datum', right_on='datum_'+variables[num])

    df_transposed.reset_index()
    df_transposed.set_index('datum', inplace=True)
    df_transposed = df_transposed.drop('id_var_wind_speed', 1)
    return df_transposed


def read_measurements(measurement_file = ''):
    if ( measurement_file == '' ):
        raise Exception('You need to supply a forecast file path!')
    file = Path(measurement_file)
    if ( file.is_file() == False ):
        raise Exception('File does not exist!')

    df_measurements = pandas.read_csv( measurement_file,
                          delimiter=',' ,
                          parse_dates=['datetime'],
                          index_col='datetime',
                          squeeze=True,
                          date_parser=date_parser ,

usecols=['datetime','wind_avg','wind_min','wind_max','wind_direction','temperature' ],
                          engine='python' )

    return df_measurements

def read_wrf( forecast_file = '' ):
    if ( forecast_file == '' ):
        raise Exception('You need to supply a forecast file path!')
    file = Path(forecast_file)
    if ( file.is_file() == False ):
        raise Exception('File does not exist!')

    df_wrf = pandas.read_csv(   forecast_file,
                            delimiter=',',
                            parse_dates=['datum'],
                            index_col=False,
                            squeeze=True,
                            date_parser=date_parser_wrf,
                            usecols=['datum', 'id_var', 'h0', 'h24', 'h48', 'h78'],
                            engine='python',
                            skipfooter=3)

    return df_wrf

def normalize_dataset(df, input_variables, output_variables):
    scaler = MinMaxScaler(feature_range=(0, 1))
    df_scaled = df.copy()

    df_scaled[input_variables] = scaler.fit_transform(df[input_variables])
    df_scaled[output_variables] = scaler.fit_transform(df_scaled[output_variables])

    return df_scaled

def split_dataset(merged_df_scaled, input_variables, output_variables, ratio):
    merged_df_length = len(merged_df_scaled)
    training_length = int(ratio * merged_df_length)

    training_df = merged_df_scaled[0:training_length]
    test_df = merged_df_scaled[training_length + 1:merged_df_length]

    training_x = numpy.array(training_df[input_variables])
    training_y = numpy.array(training_df[output_variables])

    test_x = numpy.array(test_df[input_variables])
```

```
    test_y = numpy.array(test_df[output_variables])

    training_x = numpy.reshape(training_x, (training_x.shape[0], 1, training_x.shape[1]))
    test_x = numpy.reshape(test_x, (test_x.shape[0], 1, test_x.shape[1]))

    return training_x,training_y,test_x,test_y


def shift_measurements(df_measurement_real, days):

    df_measurement_shifted = df_measurement_real.copy()
    df_measurement_shifted.index = pandas.to_datetime(df_measurement_shifted.index)
    df_measurement_shifted = df_measurement_shifted.shift(days, freq='D')
    df_measurement_shifted = df_measurement_shifted.rename(
        columns={'wind_avg':  'wind_avg_in24h',  'wind_max':  'wind_max_in24h',  'wind_min':
'wind_min_in24h', 'wind_direction': 'wind_direction_in24h',
                 'temperature': 'temperature_in24h'})
    return df_measurement_shifted



def merge_datasets(wrf_file, measurement_file):

    df_measurement_real = read_measurements(measurement_file)
    df_measurement_shifted = shift_measurements(df_measurement_real, -1)
    df_measurement  =  pandas.merge(df_measurement_real,  df_measurement_shifted,  how='inner',
left_index=True, right_index=True)


    df_wrf = read_wrf( wrf_file )

    variables_to_transpond = {
                    1: 'wind_speed',
                    2: 'wind_dir',
                    3: 'temperature',
                }

    df_transposed = transpose_wrf_variables(df_wrf, variables_to_transpond)
    df_transposed.index = pandas.to_datetime(df_transposed.index, unit='s')
    df_upsampled = df_transposed.resample('10min')
    df_wrf_interpolated = df_upsampled.interpolate(method='linear')

    merged_df     =     pandas.merge(df_wrf_interpolated,     df_measurement,     how='inner',
left_index=True, right_index=True)
    merged_df.dropna(     inplace=True,     subset=['wind_avg',     'wind_min',     'wind_max',
'wind_direction','temperature','wind_avg_in24h','wind_direction_in24h','temperature_in24h','h2
4_wind_speed','h24_wind_dir','h24_temperature','h0_wind_speed','h0_wind_dir','h0_temperature'
])

    return merged_df
```