NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

VentusNet: Deep Learning for Wind Speed prediction

Charalambos E.Tzamos

Supervisor: **Ioannis Emiris,** Professor

ATHENS

MAY 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

VentusNet: Deep Learning for Wind Speed prediction

Χαράλαμπος Ε.Τζάμος

Επιβλέπων: **Ιωάννης Εμίρης,** Καθηγητής

ΑΘΗΝΑ

ΜΑΪΟΣ 2019

BSc THESIS

VentusNet: Deep Learning for Wind Speed prediction

Charalambos E.Tzamos

S.N.: 1115201400314

SUPERVISOR: **Ioannis Emiris,** Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

VentusNet: Deep Learning for Wind Speed prediction

Χαράλαμπος Ε.Τζάμος
Α.Μ.: 1115201400314

ΕΠΙΒΛΕΠΩΝ: **Ιωάννης Εμίρης,** Καθηγητής

ABSTRACT

In this thesis, we develop a deep neural network architecture based on recurrent layers in order to forecast wind speed sequences. Our network's input is a conjunction of wind measurements and wind speed forecasts from another model. We analyse our data into time series so that, we capitalise on the temporal nature of our data and the recurrent layers. Mean absolute error, mean squared error and the logarithm of the hyperbolic cosine are used as the evaluation metrics of our model. Based on our experimental results, we show that our model achieves to improve that model's forecast whose forecasts are used as features on our model's input.

SUBJECT AREA: Artificial neural networks

KEYWORDS: Wind speed prediction, artificial neural networks, recurrent networks, time series, deep learning

ΠΕΡΙΛΗΨΗ

Στην παρούσα πτυχιακή εργασία, αναπτύσουμε μια βαθιά αρχιτεκτονική νευρωνικών δικτύων βασισμένη σε recurrent στρώμματα με σκοπό την πρόβλεψη ακολουθιών ταχύτητας ανέμου. Η είσοδος του δικτύου μας είναι ένας συνδυασμός μετρήσεων ανέμου και προβλέψεων ταχύτητας ανέμου απο άλλο μοντέλο. Αναλύουμε τα δεδομένα μας σε χρονολογικές σειρές έτσι ώστε να επωφεληθούμε από τα χρονική φύση των δεδομένων μας και τα recurrent στρώμματα. Το μέσο απόλυτο σφάλμα, το μέσο τετραγωνικό σφάλμα και ο λογάριθμος του υπερβολικού συνημίτονου χρησιμοποιούνται ως μετρικές αξιολόγησης του μοντέλου μας. Βάσει των αποτελεσμάτων μας, δείχνουμε ότι το μοντέλο μας επιτυγχάνει να βελτιώσει τις προβλέψεις του μοντέλου του οποίου οι προβλέψεις χρησιμοποιούνται ως χαρακτηριστικό στην είσοδο του μοντέλου μας.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητά νευρωνικά δίκτυα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Πρόβλεψη ταχύτητας ανέμου, τεχνητά νευρωνικά δίκτυα, recurrent δίκτυα, χρονολογικές σειρές, βαθιά μάθηση

ACKNOWLEDGEMENTS

I would like to thank Professor Ioannis Emiris, my thesis supervisor, for his guidance and useful critiques of this thesis. I would also like to thank Emmanouil Christoforou and Iason Theodorakopoulos, for their assistance in keeping my progress on schedule and for their valuable suggestions during the development of this thesis.

Contents

List of Figures

PREFACE

This thesis, titled "*VentusNet: Deep Learning for Wind Speed prediction*", was written as a part of undergraduate program of studies at the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens.

1. INTRODUCTION

In this thesis, we explore deep neural architectures capable of predicting time sequences, viz., wind speed sequences. We model our input as time series in order to exploit the natural dependence of the elements on previous ones. In this section, we provide some basic knowledge about our methods and related work.

1.1 Wind Speed Forecasting

Due to the necessity of using renewable energy sources, wind speed forecasting became an important task. Many approaches have been proposed for such forecasting. Methods that are based on mathematical models and equations which they use historical data of exogenous variables, such as, humidity, temperature, atmospheric pressure, etc. to forecast wind speed. These models, in general, are not very efficient in terms of computational cost. Also, there are models based on statistic approaches that use time series analysis, spatial correlation, etc. For example, persistence and ARMA models, using data from exogenous variables and wind speed data, all analysed into time series, they make good quality short-term predictions. The spatial correlation models, take advantage of the relation between neighbouring sites, by using the neighbouring data as input to their model. Finally, non-linear Neural Networks are frequently used for this task, along with time series analysis. Due to the non-linearity, neural networks are suitable for the wind speed forecasting task, which is rather challenging, owing to the wind's time series non-linear and non-stationary nature.

1.2 Artificial Neural Networks

Artificial neural networks (ANN) are computational systems that are inspired by the Biological neural networks. These systems, instead of taking instruction on how to solve a task, they "learn" how to solve it by getting examples. For example, in image recognition, such networks might learn to identify the label of an image that contains a specific item by analysing example images that are already manually labelled as 1 (contains the item) or 0. An ANN is a collection of artificial neurons or connected nodes that receive input and produce output depending on the input, the weights and the activation function. They can be represented as a directed, weighted graph (see Fig 1.2), where the neurons are the vertices and the connections between them are the edges.

ANNs use optimisation algorithms to minimise (or maximise) an Objective function. During the training process of a network, the input is propagated layer by layer through the network, until it reaches the output layer, where the output is compared with the desired output, using a loss function. Propagating the error values from the output layer back through the network, we get an associated error value for each neuron. The backpropagation algorithm uses the error values to calculate the gradient of the loss function. In order to minimise the loss function, the optimiser uses the gradient to update the weights.

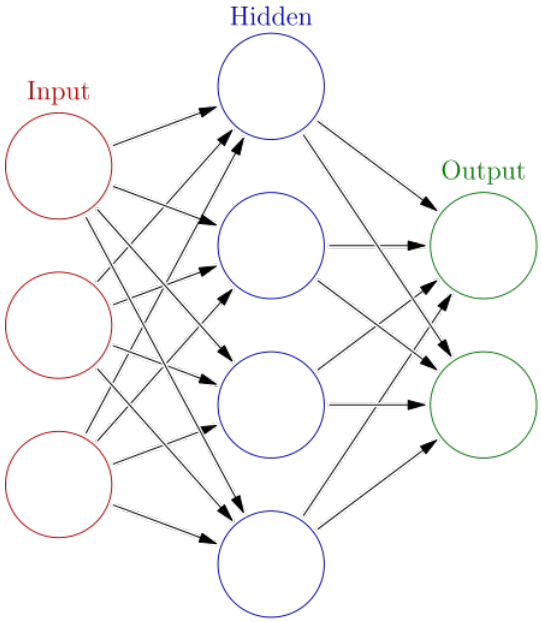

Figure 1.1: The graph representation of a simple Feed Forward neural net. The edges have certain weights. Source: Wikipedia

1.3 Time Series

In temporal data, values usually depend on previous ones. Therefore, in order to predict future values, it is desired to know a number of previously observed values. As a time series we denote a d-dimensional vector X that contains data in time order.

$$X = \{x_1, x_2, ..., x_d\}$$

Time series are used in statistics, signal processing, pattern recognition, weather forecasting, etc. and also in any domain that involves temporal measurements. The goal of time series analysis varies from forecasting and signal detection to classification and anomaly detection.

1.4 Related Work

MLPs: Tao et al. in [7] develops a DBF (deep belief) architecture with 3 layers of 100, 200 and 300 nodes. They train their model using three month data from a wind station in Mongolia, sampled every 10 minutes, to predict the wind power of the future 48 hours. Using MSE (mean squared error) and MAE (mean absolute error) to evaluate their model, they note that the model shows stability from 6 to 24h which manifests that their architecture is eligible to capture some of the hidden patterns of the wind series. In [8] they compare 3 ANN architectures (linear, backpropagation and radial basis) using data measurements from North Dakota (US). Evaluating the results using MAE, RMSE (root mean squared error) and MAPE (mean absolute percentage error), they conclude that there is not a superior architecture amongst them as the results depend on the data. Alexiadis et al. [12] instead of using the wind speed as input, they use the differences of wind speeds from the moving averages. With this technique they achieve improvements of up to 13% over persistence, while with the standard approach they achieve 9%. Sfetsos in [13, 14] compared a number of methods, namely feed forward neural networks, radial basis function

networks, traditional linear ARMA models, Elman network, Box-Jenkins model, ANFIS models (Adaptive Network based Fuzzy Inference System), a neural logic network and other linear models based on their ability to forecast hourly mean wind speed. The non-linear models manifested RMSE (root mean squared error), which was better than any of the linear methods. Regarding the one hour ahead forecast, the best model was the neural logic network that incorporated Logic Rules, which produced an RMSE 4.9% lower than the persistence approach.

CNNs: Several architectures that attempt to forecast wind speed and power using convolutional layers have been proposed, that also have promising results. [2, 3] use shallow CNN architectures to predict wind power, comparing them with other models, namely Gaussian SVR, persistence and regression. Chiou-Jye Huang et al. in [5] propose a very simple convolutional network which takes as input data from the previous 7 days (hourly) and predicts the following 3 (hourly). Using MAE and RMSE as metrics to evaluate their model, they achieve very high accuracy. They note that their proposed algorithm achieves 0.800227 and 0.999978, MAE and RMSE, respectively. The length of their dataset is one year and they note that the wind speed of their data mostly falls at a range of 1.5 m/s to 4 m/s. They also mention that there are many cases where the wind speed is higher than 5 m/s or even exceeds 10 m/s.

RNNs: [4] proposes an RNN architecture with good results for short-term forecasts, using data from 57 meteo stations. Their proposed DL-STF (Deep Learning-based Spatio-Temportal Forecasting) model achieves (in hourly forecasting for the next 6 hours) 1.18 and 1.62, MAE and RMSE, respectively. As test set, they use the time period from January 6, 2014 to February 20, 2014, which they claim to be the one with the most unsteady wind conditions. [1] using NWP data for off-shore points to make mid-term predictions, concludes that this RNN architecture has a lot of potentials with a high degree of fine-tuning. The NMAE (normalised mean absolute error) of the persistence model on their data ranges from 5.5% up to 32%, while that of their proposed model varies from 5.5% to 15.5%.

Unlike the above mentioned previous work, we build a deep neural network with multiple RNN layers. Also, our input is consisted of both measurements and predictions from another model.

2. METHODS

2.1 Multilayer Perceptron

A multilayer perceptron is a class of Feed Forward ANN. Their goal is to approximate some function f^*, where a linear method cannot. It is consisted of at least three layers of nodes: an input layer, a hidden layer and an output layer. Besides the input layer nodes, each node is a neuron that uses a nonlinear activation function. Two common nonlinear activation functions:

$$y(u_i) = tanh(u_i) = \frac{e^{2u_i} - 1}{e^{2u_i} + 1} \quad and \quad y(u_i) = u_i^+ = max(0, u_i)$$

Figure 2.1: Tanh activation function (left) and reLU activation function (right).

With nonlinear activation functions and a backpropagation algorithm that optimizes the weights on each connection, a MLP can distinguish data that is not linearly separable.

2.2 Convolutional Networks

Convolutional networks are actually regularised versions of multilayer perceptrons. Both networks are apt to overfitting data because of their fully-connectedness. In order to overcome this problem, CNNs include a kind of regularisation. They take advantage of the hierarchical pattern in data and construct more complex patterns using smaller and simpler patterns.

CNNs work in matrices to extract features relevant to the task. These networks apply the discrete convolution operation for finite matrices f, g (see equation (2.1)):

$$(f * g)(t) = \sum_m f(\alpha) g(t - \alpha) d\alpha \tag{2.1}$$

Such networks are most commonly used to process 2D or 3D image matrices (see Fig. 2.2).

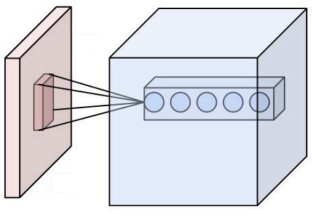

Figure 2.2: A representation that shows how a convolutional layer, working on smaller and simpler parts of a matrix, extracts features about the task. Source: Wikipedia

2.3 Recurrent Networks

A Recurrent neural network is a class of artificial neural networks where the connections between neurons build a directed graph along a temporal sequence. So, they take advantage of sequential data, in that, each output is a function of the previous elements (see Fig. 2.3). That fact makes RNNs eligible to learn from patterns in the time series, hence the prediction of the future. Unlike feed-forward neural networks, RNNs can use their internal state to process sequences of inputs.

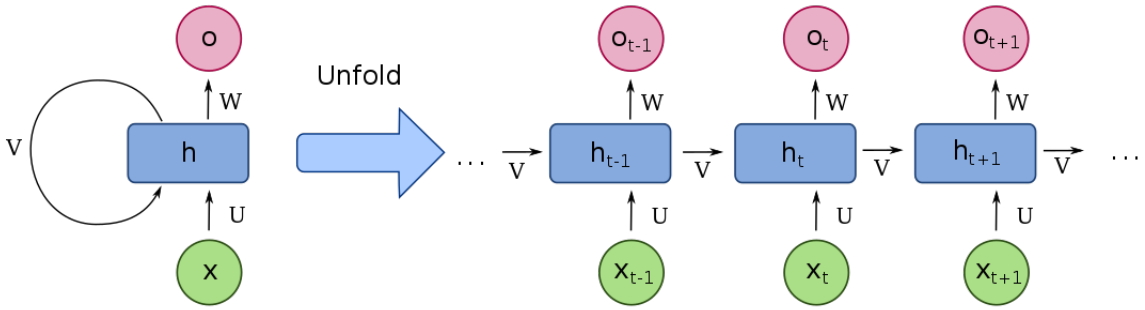

Figure 2.3: The design of a RNN layer. $x = \{x_1, ..., x_{t-1}, x_t, x_{t+1}, ..., x_d\}$ is the input vector and o is the output vector. Source: Wikipedia

There are many variations of recurrent networks. Elman and Jordan networks are two of the variations of RNNs and they are known as simple recurrent networks (see section 2.3.1). Also there are more complex RNN architectures, such as LSTM and GRU.

2.3.1 Simple Recurrent Networks

Elman and Jordan networks are very similar to each other. The only difference between these two networks is that the former uses as input for its extra set of units (context units), the output of the hidden layer while the latter uses the output of the output layer.

Elman Network

Elman Network [10] processes the input vector x_t and outputs the output vector y_t according to the following equations:

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

where h_t is the hidden layer vector, W, U and b are the parameter matrices and vector and σ_h and σ_y are the activation functions.

Jordan Network

Similarly to the Elman network, Jordan network [11] processes the input vector x_t and outputs the output vector y_t according to the following equations:

$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

where h_t is the hidden layer vector, W, U and b are the parameter matrices and vector and σ_h and σ_y are the activation functions.

2.3.2 Long Short-Term Memory

A common LSTM [15, 16] unit is composed of a cell, an input gate, an output gate and a forget gate (see Fig. 2.4). The cell remembers values over arbitrary time intervals and the three gates control the flow of information into and out of the cell. LSTMs were developed to overcome the problems of exploding and vanishing gradient that can be encountered when training traditional RNNs. The equations below, show how an LSTM operates [16].

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$
$$h_t = o_t \circ \sigma_h(c_t)$$

where $x_t \in \mathbb{R}^d$ is the input vector to the unit, $f_t \in \mathbb{R}^h$ is the forget gate's activation vector, $i_t \in \mathbb{R}^h$ is the input gate's activation vector, $o_t \in \mathbb{R}^h$ is the output gate's activation vector, $h_t \in \mathbb{R}^h$ is the output vector of the LSTM unit and $c_t \in \mathbb{R}^h$ is the cell state vector. Also $W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h}$ and $b \in \mathbb{R}^h$ are the weight matrices and bias vector parameters. Apropos the activation functions, σ_h, σ_c are hyperbolic tangent functions and σ_g is a sigmoid function.

2.4 Stochastic Optimisation

To improve the performance of a deep neural network, its important to use an optimisation algorithm. We choose to use the adaptive moment estimation (Adam) optimiser [9], which is a stochastic gradient-based optimiser. The core of Adam optimiser [9] is described by the following equations (2.2) - (2.7):

$$g_t = \nabla_\theta f_t(\theta_{t-1}) \tag{2.2}$$

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{2.3}$$

$$\upsilon_t = \beta_2 \cdot \upsilon_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{2.4}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.5}$$

$$\hat{\upsilon}_t = \frac{\upsilon_t}{1 - \beta_2^t} \tag{2.6}$$

$$\theta_t = \theta_{t-1} - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{\upsilon}_t} + \epsilon} \tag{2.7}$$

where α is the stepsize, $\beta_1, \beta_2 \in [0, 1)$ are the exponential decay rates, $f(\theta)$ is the stochastic objective function and θ_0 is the initial parameter vector. Both 1st and 2nd moment vectors m_t, υ_t are initialised to 0. Respectively, $\hat{m}_t, \hat{\upsilon}_t$ are the bias-corrected moment estimates. Also g_t^2 indicates the elementwise square $g_t \odot g_t$.

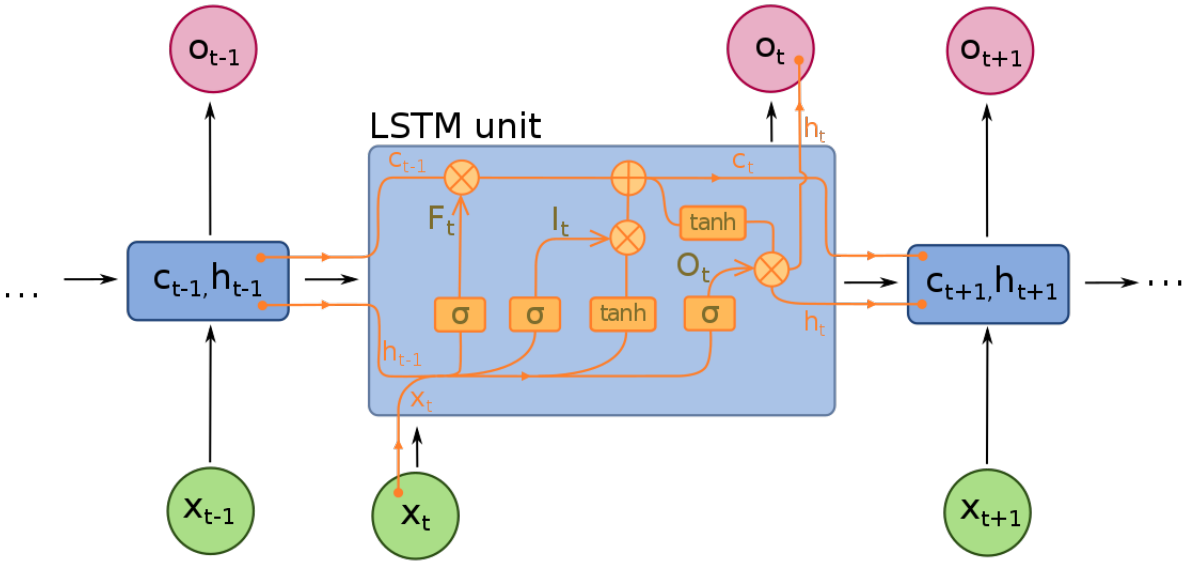

Figure 2.4: The design of an LSTM layer. In the middle unit, we can see how a LSTM unit (blue) operates. It is a visual representation of the above equations. Source: Wikipedia

2.5 Our Model

In order to predict wind speed we select to exploit both pattern recognition and simulation forecasting. WRF [6] as an NWP model, it is comprised of mathematical equations that predict the behaviour of a physical system. So, as input we choose to use both measurements and WRF predictions. The resolution of measurements and WRF predictions is 1h. Consequently, for each day there are 24 measurements and 24 predictions. We match every value in the measurement data, with the value of the next day, same hour in the prediction data. The transformed data plus extra features such as wind direction, are getting sliced into time series preserving the daily split. The final transformation plus some

extra statistic features that are processed the same way, are the network's input. Before feeding our network the input, we normalise it, by rescaling each feature individually into $[-1, 1] \subseteq \mathbb{R}$. A single input is consisted of multiple time series, one for each feature. Consuming such input, the network outputs t values - where $t \geq 24$ is the window of the time series - of which only 24 are kept, i.e. the prediction of the next 24h.

As loss function, we use the logarithm of the hyperbolic cosine (log-cosh) of the prediction error (see equation 2.8). It is approximately equal to $x^2/2$ for small x and to $|x| - \log 2$ for large x. This means that "log-cosh" works mostly like the mean squared error (MSE, see equation 2.9), but will not be so strongly affected by the occasional wildly incorrect prediction.

$$logcosh(y, \hat{y}) = \sum_{i=1}^{n} \log\left(\cosh\left(\hat{y}_i - y_i\right)\right) \tag{2.8}$$

$$mse(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.9}$$

$$mae(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{2.10}$$

We also evaluate our model using Mean Squared Error (MSE) and Mean Absolute Error (MAE) (see equations 2.9, 2.10). In the above equations, as y and $\hat{y}$ we denote the ground truth and the prediction, respectively.

3. DEEP LEARNING ON TIME SERIES

3.1 Temporal Input

Our input is a vector of elements. Each element is a point in $\mathbb{R}^d$. Each element is also a sequence of measurements in time order. So, treating our data as time series, leads us to use recurrent and time-distibuted layers into our model's network. We also generate and use features that give a different representation to our input.

$$F(X,i) = \begin{cases} 0 & \text{if } i = 0 \\ \left\lfloor \frac{X_i}{X_{i-1}} \right\rfloor - 1 & \text{if } X_i \geq X_{i-1} \\ \left\lfloor -\frac{X_{i-1}}{X_i} \right\rfloor + 1 & \text{if } X_i < X_{i-1} \end{cases} \tag{3.1}$$

The above defined function $F : \mathbb{R}^n \times \mathbb{Z} \longmapsto \mathbb{Z}$ (see equation 3.1) is the function that we use to map some of our features in order to have a different representation of our data. $X \in \mathbb{R}^n$ is the vector that contains the measurements and $0 \leq i \leq n$ is the index.

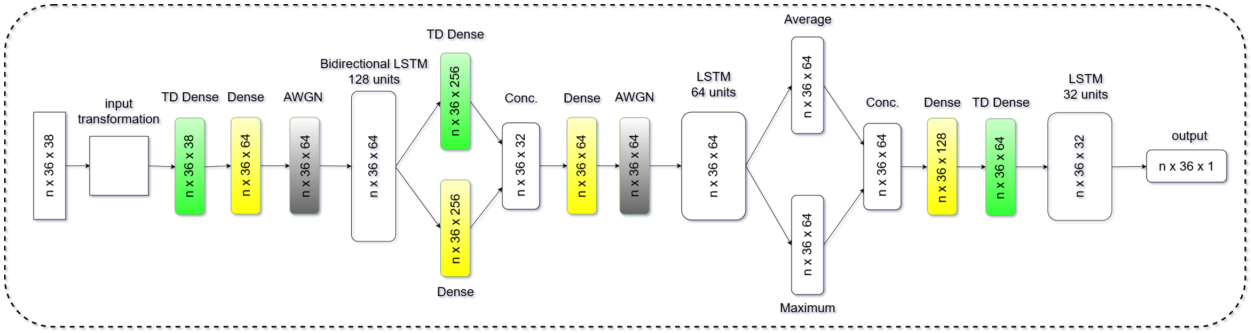

Figure 3.1: VentusNet. Time-distibuted dense (TD Dense). Additive zero-centered Gaussian noise (AWGN). Concatenation layer (Conc.)

3.2 VentusNet Architecture

Our network is a deep neural network. Its architecture is visualized in Fig 3.1. Taking advantage of the natural order of the data, we first feed them to a time-distributed dense layer and to a normal fully-connected dense layer afterwards. Time-distributed dense applies a dense layer to every temporal slice of the input. Then we add some Gaussian noise to the output in order to mitigate overfitting and give it as an input to a Bi-directional LSTM layer. Bi-directional LSTM gives as output the concatenation of the output of an LSTM and the output of an LSTM with reversed input. Using dense layers, our network aggregates the information and gives it as input to a second LSTM layer. Afterwards, using Maximum and Average layers after 2 parallel LSTM nodes, we aggregate the information further. Finally, the flow goes to a third LSTM layer and then to a dense layer with 1 node which is the output node.

4. EXPERIMENTS

This chapter is divided into two parts. First, we show detailed experiments to account for the choice of our network. Second, we analyse the space complexity of the network.

4.1 Experiments

The data that we test our network are real from wind farms. Our measurements have a lot of missing data. In wind speed measurements we handle the missing data by replacing the missing values with WRF wind speed predictions. In wind direction measurements we deal with missing data by replacing each one of the missing values with the last not missing value.

About our experiments, we trained the model on 380 samples, validated it on 43 samples and tested it on 104 samples. We also used data from neighbour wind turbines, hence the 38 features. For each neighbour turbine we have the wind speed, the wind direction and the WRF prediction. In Fig. 4.1 we can see the plot of both actual and predicted values.

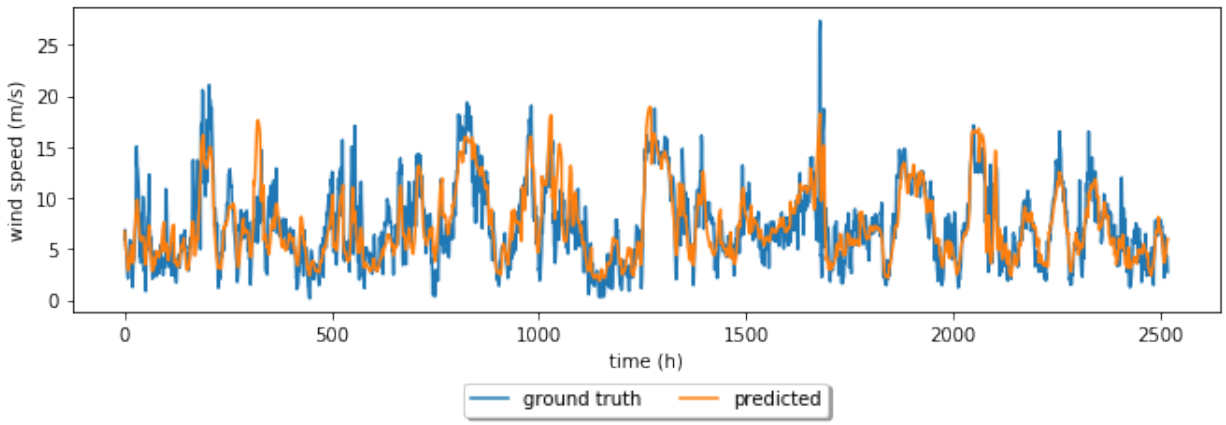

Figure 4.1: The plot of actual wind speed values (blue) and our model's prediction (orange).

With our proposed architecture and an input of 38 features, we achieve ~ 1.65 MAE and ~ 5.30 MSE. Although, we had better MAE using less features, i.e. wind speed, wind direction, WRF prediction and 2 statistical features, and also a slightly different architecture, the results were instable and overfitting was encountered in certain cases. Also, by using neighbouring data we describe the problem even better and the model might behave better with a larger dataset. Based on the results, our model beats both persistence and WRF models, where the former achieves ~ 3.56 MAE and the latter achieves ~ 1.90 MAE on the test data. The output of our persistence model for a certain time step i in a day, is the previous day's measurement for the same time step i.

The above results are based on experiments where the time series' window size is set to 36. Taking 36 time steps instead of 24, doesn't make quite a difference to the mean absolute error, but it still improves it. Additionally, with a larger dataset, the 36-timestep model might find some hidden patterns in the data that the 24-timestep model cannot.

In Fig. 4.2 we provide results of our model compared with the actual measurements with a different zoom.

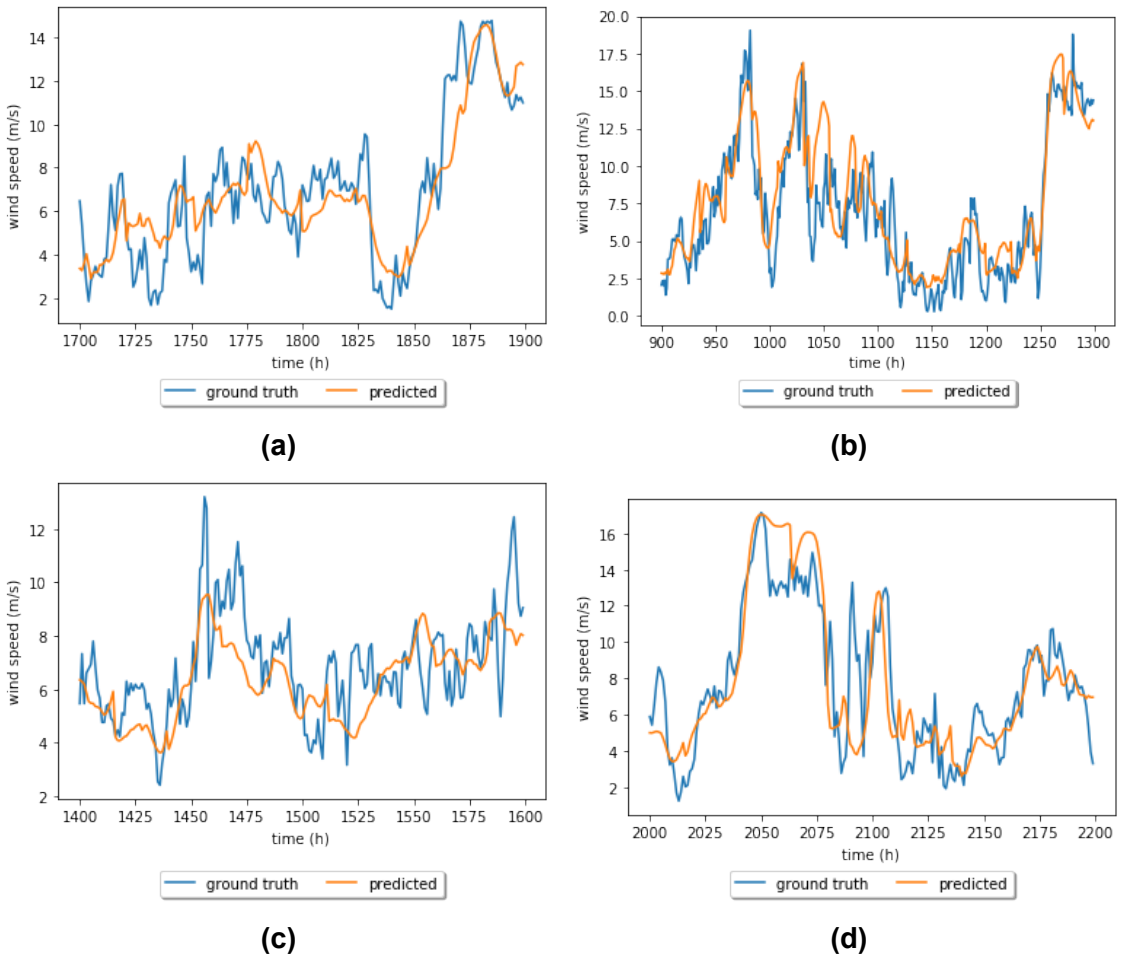

Figure 4.2: Forecasting results of VentusNet.

4.2 Space Complexity Analysis

The space complexity (number of parameters in the network) of our VentusNet is 0.3 M parameters. The Bidirectional LSTM is the major contributor on the space complexity by having approximately the $2/3$ of the total parameters of the network. Although our network is very efficient in terms of #param in the network, it outputs good quality predictions.

5. CONCLUSIONS AND FUTURE WORK

In this thesis, we propose a deep neural network VentusNet for wind speed forecasting. We analyse our data into time series and use recurrent networks to exploit the temporal neture of our data. Also we use WRF wind speed prediction as input to our model. VentusNet predicts the following 24h based on the previous 36h. As for the future work, adding convolutional layers to the network's architecture or analysing the input in a different way could improve the results. A custom loss function that is strongly affected by incorrect prediction on high values and less affected by incorrect predictions on low values, would improve the model, as the high values are more important than the low ones. Additionally, using the differences of wind speeds from the moving averages instead of the wind speed, would make the input data have a simpler representation. Also, having more data to train and test the model would specify more precisely the effectiveness of the model.

ABBREVIATIONS - ACRONYMS

ANN	Artificial Neural Network
RNN	Recurrent Neural Network
MLP	MultiLayer Perceptron
LSTM	Long Short-Term Memory
ARMA	AutoRegressive Moving Average
CNN	Convolutional Neural Network
SVR	Support Vector Regression
NWP	Numerical Weather Prediction

BIBLIOGRAPHY

[1] Balluff, S., Bendfeld, J., & Krauter, S. *Short term wind and energy prediction for offshore wind farms using neural networks*. 2015 Int. Conf. on Renewable Energy Research and Applications (ICRERA), pp. 379–382, 2015.

[2] D´ıaz, D., Torres, A., & Dorronsoro, J. R. *Deep neural networks for wind energy prediction*. Rojas, I., Joya, G., & Catal'a, A., editors, Advances in Computational Intelligence, Springer International Publishing, Cham, pp. 430–443, 2015

[3] Wang, H. Z., Li, G. G., Wang, G. B., Peng, J. C., Jiang, H., & Liu, Y. T. *Deep learning based ensemble approach for probabilistic wind power forecasting*. Applied Energy, Vol. 188, pp. 56–70, 2017

[4] Ghaderi, A., Sanandaji, B. M., & Ghaderi, F. *Deep forecast: Deep learning-based spatiotemporal forecasting*. CoRR, Vol. abs/1707.08110, 2017

[5] Chiou-Jye Huang & Ping-Huan Kuo *A Short-Term Wind Speed Forecasting Model by Using Artificial Neural Networks with Stochastic Optimization for Renewable Energy Systems*. Energies 2018, 11, 2777; doi:10.3390/en11102777, 2018

[6] Skamarock, W. C., J. B. Klemp, J. Dudhia, D. O. Gill, Z. Liu, J. Berner, W. Wang, J. G. Powers, M. G. Duda, D. M. Barker, & X.-Y. Huang *A Description of the Advanced Research WRF Version 4*. NCAR Tech. Note NCAR/TN-556+STR, 145 pp., doi:10.5065/1dfh-6p97, 2019

[7] Tao, Y., Chen, H., & Qiu, C. *Wind power prediction and pattern feature based on deep learning method*. IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC), pp. 1–4, 2014

[8] Li, G. & Shi, J. *On comparing three artificial neural networks for wind speed forecasting*. Applied Energy, Vol. 87, No. 7, pp. 2313 – 2320, 2010

[9] Kingma, D.P., & Ba, J. *Adam: A Method for Stochastic Optimization.* Comput. Sci., 1–13, 2014.

[10] Elman, & Jeffrey L. *Finding structure in time*. Cognitive Science, 14, pp. 179-211, 1990.

[11] Michael I. Jordan. *Serial order: A parallel distributed processing approach*. Technical Report 8604, Institute for Cognitive Science, University of California, San Diego, 1986.

[12] Alexiadis, M.C., P.S. Dokopoulos, H.S. Sahsamanoglou, & I.M. Manousaridis. *Short-Term Forecasting of Wind Speed and Related Electrical Power*. Solar Energy 63, pp. 61-68, 1998

[13] Sfetsos, A. *Time series forecasting of wind speed and solar radiation for renewable energy sources*. Ph.D. thesis Imperial College, UK, 1999

[14] Sfetsos, A. *A comparison of various forecasting techniques applied to mean hourly wind speed time series*. Renewable Energy 21, pp. 23-35, 2000

[15] Sepp Hochreiter, & Jürgen Schmidhuber *Long Short-term Memory*. Neural computation, 9, 1735-80. 10.1162/neco.1997.9.8.1735, 1997

[16] Felix Gers, Jürgen Schmidhuber, & Fred Cummins *Learning to forget: continual prediction with LSTM*. Proc. ICANN'99, IEE, London: 850–855, 1999