DEEP LEARNING FOR WIND POWER FORECASTING

Tymoteusz Barcinski[1], *Jan Eberle*[1], *Sumukha Shridhar*[1]

[1]Technical University of Denmark

ABSTRACT

This paper researches in the context of wind power prediction. Wind farms are becoming crucial for the entire power grid and therefore to gain a more stable grid the accurate prediction of the future power supply of wind farms is vital. Wind power forecasting is still a very difficult non-linear, multivariate, multi-step time series problem with deep learning approaches entering the field. In this project, we investigate the power prediction of a wind farm for the next 36 hours. In this context, we make use of an Encoder-Decoder architecture as well as a transformer architecture. With data preprocessing including Variational Model Decomposition as well as hyperparameter training, we were able to predict the wind power of a wind farm for the next 36 hours using a transformer and reaching an accuracy of 0.1958 RMSE and by using an Encoder-Decoder LSTM reaching an accurancy of 0.2198 RMSE. In this context the transformer has turned out to be slightly better on the test set than the Encoder-Decoder LSTM.

Index Terms— Deep Learning, Windpower, Forecasting, Transformers, Encoder-Decoder, Long short-term memory

1. INTRODUCTION

Wind power has been a promising source of renewable energy and in recent decades, it has been incorporated into the power grid. However, it heavily depends on local weather conditions, and because of that it is known for its volatility and instability. Therefore, reliable and accurate wind power forecasting is of vital importance for power grid management. In recent times, deep learning has been promising in forecasting wind power prediction. With this paper we are researching in this field and investigating two different architectures for time-series predictions. The repository of this project can be accessed via this Github-Link [1].

1.1. Problem formulation

In this project, we used publicly available data from the Global Energy Forecasting Competition 2012 [2]. We slightly modified the problem proposed in the wind power forecasting track. This project aims to train and evaluate deep learning models which can forecast the hourly power production of the wind farm for the 36-hour horizon, given the past data and the weather forecast for the next 48 hours. The predictions are to be issued two times per day: at noon and at midnight. We decided to approach the problem as a multi-step, multivariate time series problem, to incorporate the time dependency into our models.

1.2. Related work

In the past, there were already several papers which dealt with short-term wind power prediction. Zhou Wu et. al. [3] reviews in this context different approaches including time-series-based recurrent neural networks, convolutional neural networks and auto-encoder-based approaches, focusing on real-world applications. Another work from Huijuan Wu et. al. [4] proposes an Encoder-Decoder Transformer architecture in order create a short-term prediction for wind power. And Fu et. al. [5] integrate spatiotemporal attention with a transformer architecture in order to capture the spatial dependencies among wind farms also in a short-term manner.

2. DATA

In this section, we want to take a closer look at the data we used as well as the techniques and approaches of the data preprocessing we implemented.

2.1. Data description

The data is stored in two datasets. The first one contains observed wind power (*wp*) every hour, originally normalized to be in the interval $[0, 1]$. The second one contains the weather forecast issued at noon at midnight - every 12 hours - for the next 48 hours every hour. Hence, at every timestamp, there are 4 overlapping 48 hour periods. Furthermore, there are 4 features available: wind speed (*ws*), wind direction (*wd*) measured in degrees, zonal wind component (*u*), meridional wind component (*v*). It is important to note that wind speed and wind direction convey the same information as zonal wind component and meridional wind component. The former pair describes it in the polar coordinate system and the latter pair does so in the Cartesian coordinate system. Originally, the data is available from the 1st of July 2009 at 00:00 to the

26th of June 2012 at 12:00 which constitutes 3 years of observations. However, in the competition from the 1st of January 2011 until the end there are 48 hour periods with missing wind power observations, where the forecast was supposed to be issued. Since we couldn't access these test data from the host of the competition, we decided to work only with the first 1,5 years of data where there are no missing observations.

2.2. Exploratory data analysis

To get a better understanding of the data we did an exploratory data analysis. Figure 1 therefore presents the power curve. We can notice that wind speed and wind power correlate. Moreover, the distribution of wind power is highly skewed, with many 0 values.

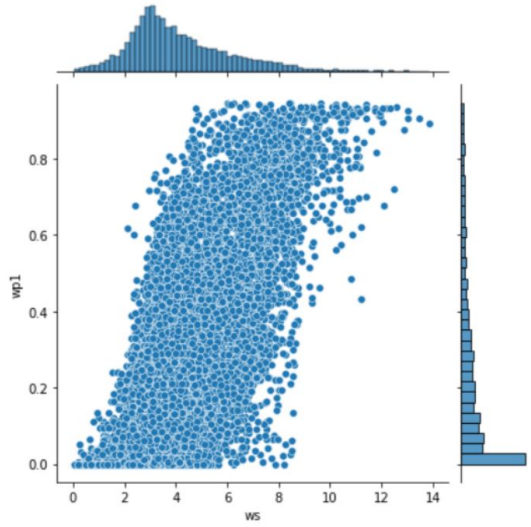
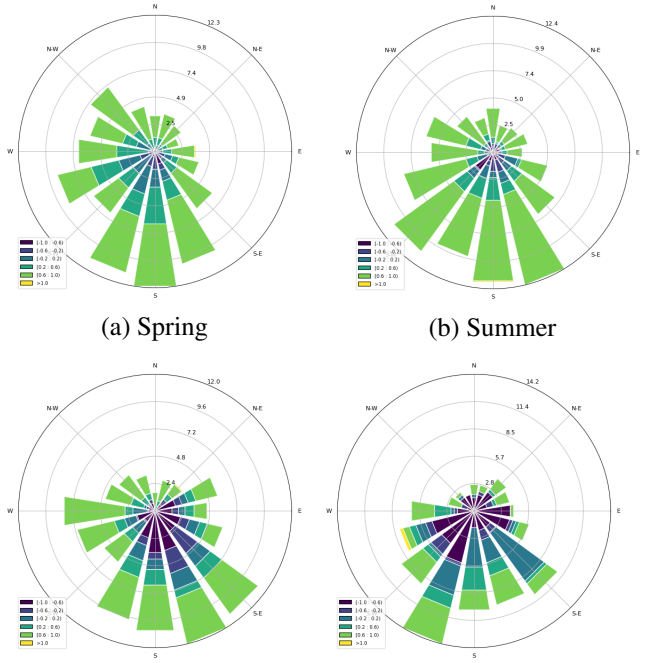


Fig. 1. Wind farm 1: wind power and wind speed plotted against each other

If we plot the data in seasonal wind roses, which can be seen in Figure 2 then the graphs look similar in terms of wind direction, but it is also noticeable that the wind speed differs depending on the season.

2.3. Preprocessing

As a first step several preprocessing steps were taken to make the data suitable for modelling. Based on the exploratory data analysis, we concluded that including the seasonality would improve the model. We introduced the feature $month$ which takes the values from 1 to 12 and the feature $hour$ which takes the values from 0 to 23. Those features, along with wind direction, exhibit cyclic nature which is not captured by their current representation. To account for that we applied sinus and cosinus transformations with the proper phase. Namely,



(a) Spring (b) Summer

(c) Autumn (c) Winter

Fig. 2. Seasonal windroses displaying the wind direction and windspeed in each season of 2010

we created the following new features:

$$wd_{sin} = \sin(wd \cdot \frac{2\pi}{360}),$$

$$wd_{cos} = \cos(wd \cdot \frac{2\pi}{360}),$$

$$month_{sin} = \sin((month - 1) \cdot \frac{2\pi}{12}),$$

$$month_{cos} = \cos((month - 1) \cdot \frac{2\pi}{12}),$$

$$hour_{sin} = \sin(hour \cdot \frac{2\pi}{24}),$$

$$hour_{cos} = \cos(hour \cdot \frac{2\pi}{24}).$$

We applied the square root transformation to the wind speed variable to deal with its skewed distribution. It was checked with the Q-Q plot that after the transformation the variable was normally distributed. Even though wind speed and wind direction convey the same information as the zonal wind component and meridional wind component we decided to keep all of them. Further, as it is commonly done with preprocessing for deep learning models, we standardized all of the explanatory variables, to improve the convergence of the optimization algorithm.

Let $\boldsymbol{x}_t$ be the vector of explanatory variables. Further, let y_t be the wind power at time t. To make the data suitable

for the models, we used the sliding window approach, where the time lag is denoted by l. For the past observations the following matrix can be constructed:

$$SL_{t,l} = \begin{bmatrix} y_{t-l} & y_{t-l+1} & \cdots & y_{t-2} & y_{t-1} \\ \boldsymbol{x}_{t-l} & \boldsymbol{x}_{t-l+1} & \cdots & \boldsymbol{x}_{t-2} & \boldsymbol{x}_{t-1} \end{bmatrix} \quad (1)$$

Moreover, the weather forecast for the next 48 hours is available, so we have a matrix:

$$X_{t,48} = \begin{bmatrix} \boldsymbol{x}_t & \boldsymbol{x}_{t+1} & \ldots & \boldsymbol{x}_{t+46} & \boldsymbol{x}_{t+47} \end{bmatrix} \quad (2)$$

The goal is to learn a function f which takes as an argument the matrices above and predicts following vector.

$$\boldsymbol{y_t} = [\hat{y}_t, \hat{y}_{t+1}, \hat{\ldots}, \hat{y}_{t+34}, \hat{y}_{t+35}]$$

The forecast needs to be issued every 12 hours, but for the training phase, it means that the model would not use the sliding windows between hours $[1, 11]$ and the amount of data available for training would be decreased by 12 times. Hence, we decided to forecast for 36 hours ahead instead of 48, as proposed in the competition, to always have weather forecast for $t \in [1, 12]$. After the matrices SL and X were constructed for every plausible t, they were split into the train, validation and test sets. For the train set as mentioned above, the stride was 1, but for validation and test sets, the stride was 12 - forecast at noon and midnight.

2.4. Variational Model Decomposition (VMD)

Deep learning models generally need huge amounts of data in-order to avoid over-fit. Data decomposition techniques like wavelet mode decomposition, empirical mode decomposition or Variational Model Decomposition are commonly used data decomposition techniques in wind power forecasting applications [6]. In our modelling approach we use Variational Model Decomposition, since it gives precise component separation of the signal [7]. For this, a python package *vmdpy* [8] is used. We decompose the wind speed into seven components with the VMD approach, since wind speed is the crucial aspect affecting the wind power generation.

3. ARCHITECTURES

Given that our problem requires multi-step prediction, we looked into sequence-to-sequence models. In this section we want to present the architectures Encoder-Decoder LSTM and Transformers, which we both investigated in this context.

3.1. Encoder-Decoder LSTM

Encoder-Decoder LSTM models have been widely used for time series forecasting tasks [9, 10]. Our model was mostly inspired by the description provided by Chenyou [11]. Figure 3 shows the general architecture, to which several modifications were made. The Encoder sequentially processes the sliding window matrix SL with past explanatory variables and target variables. At each time step, the input is processed by the Long Short-Term Memory (LSTM) unit. The final hidden state - the latent representation of the past - is given to the decoder. Since at the time of the forecast, the matrix X is known the decoder is the bi-directional LSTM. It allows the model to take into consideration both the information from the past and from the forecasted future. The decoder runs in an auto-regressive fashion: it takes as input the explanatory variables and the wind power predicted at the previous iteration. The following vector are concatenated and given to the fully connected feed-forward neural network with two linear layers and ReLU activation: The forward and backwards hidden states of the last layer of the bi-directional LSTM, explanatory variables at a given time stamp, past n predicted wind power values, where n is the hyperparameter. Following the convention of recurrent neural networks, the parameters of FFNN are shared across time. Since wind power takes values between $[0, 1]$ after the second layer the sigmoid function maps the output to the specific range. The FFNN predicts one value - the wind power prediction at a given time.

To help the model learn the structure of the data during the training phase, two alternative approaches to recursively feeding back the prediction were introduced. First was the teacher-forcing method, in which the true target value is fed into the decoder. It prevents the model from accumulating the error if bad predictions were made at the beginning of the forecasting horizon. The second one was mixed teacher forcing where at each time stamp there is a certain probability that the model would go with the teacher forcing approach. The probability decreases over time. During validation and testing, only the recursive approach is used.

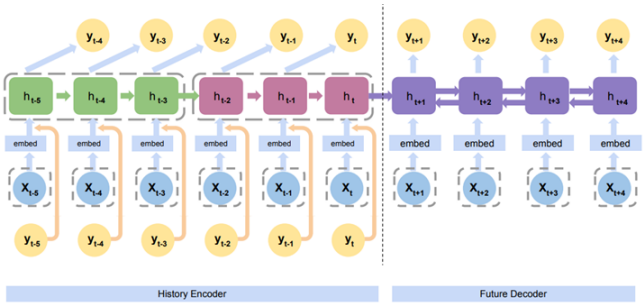


Fig. 3. Encoder-Decooder LSTM Architecture according to Chenyou Fan et. al.[11]

3.2. Transformer

In the scope of this project, we have investigated the Transformer architecture. For this purpose, we consulted different papers and the Pytorch library[12]. Especially the description of Ludvigsen[13] was used for the implementation.

The Transformer we used consists of an Encoder and Decoder Layer. The Encoder consists of several layers:

- **Positional-Encoding-Layer**, which is important to give the model relative or absolute information about the position of the tokens in the sequence.
- **Encoder-Layer**, which contains self-attention and a forward call.

The Decoder also consists of several layers:

- **Decoder-Layer**, which uses self-attention, a forward call, but also encoder-decoder-attention in between.
- **Linear-Layer**, at the very end.

In the Encoder and Decoder, several layers can be stacked on top of each other.

It is important to mention that the transformer does not process the data in an ordered sequence, but processes the complete sequence at once and applies self-attention mechanisms to learn dependencies in the sequence. In that context, the Positional Encoding Layer finds its role as well. This can also be applied to multivariate time series problems. We implemented the transformer in a manner that predicts the next 36 hours. We, therefore, used the Transformer-Encoder-Layer and Transformer-Decoder-Layer of the Pytorch library. For a generalization dropout was used in the Encoder and Decoder layers. As input, we feed the 48-hour weather forecast to the model, where 12 hours are fed to the encoder and 36 hours to the decoder in order to get a 36-hour sequence as the prediction. In Figure 4 we can see the structure of our transformer. Note that the layer count is just representative and more layers can be stacked over each other as mentioned before.

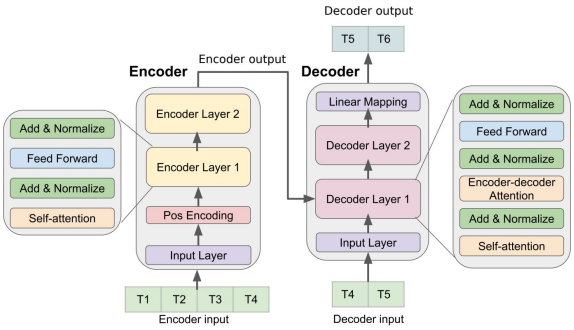


Fig. 4. Transformer Architecture according to Wu et. al.[14]

4. EXPERIMENTS

In this section, we present the results as well as the changes we have introduced to improve the respective models. We use the MSE as a loss function to train the models and RMSE to calculate accuracy.

4.1. Encoder-Decoder LSTM

For the Encoder-Decoder LSTM, we used an Adam optimizer with a weight decay of 0.033, a sliding window size of 60, and a mixed teacher forcing throughout all approaches. The train/validation/test split was 80%/10%/10%.

4.1.1. First Approach

As a first approach, we trained the model with different hyperparameters for 20 epochs each to test for the most promising. The following hyperparameters were in this approach tested.

- learning_rate = [1e-05]
- hidden_size = [300, 320, 340]
- num_layers = [3, 5]
- hidden_size_backward = [64, 128, 258]
- num_layers_backward = [1, 2, 3]

As a result the most promising parameters were: hidden_size = 300, num_layers = 5, hidden_size_backward = 128, num_layers _backward = 2. We continued to train with these parameters and experimented with the learning rate of 1e-05 and 1e-06 for 100 epochs each. In Figure 5 a) and b) we see the training and validation loss of the training with both learning rates. learning rate with 1e-05:

- training loss: 0.199379
- validation loss: 0.255689
- test loss: 0.224782

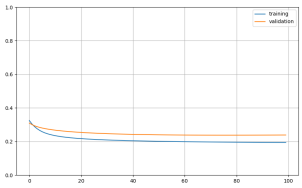
learning rate with 1e-06:

- training loss: 0.255003
- validation loss: 0.324698
- test loss: 0.299603

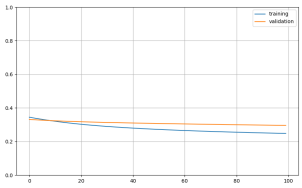
4.1.2. Second approach

In the second approach of the LSTM training, we added VMD to our preprocessing. We added the sinus and cosinus of hour of the day as well as applied a square root transformation to the wind speed. In the context of hyperparameters, we kept them the same as in the first approach. In Figure 5 c) is the training and validation loss of this approach plotted. We trained our model for 50 epochs.

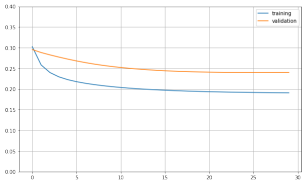
- training loss: 0.189832
- validation loss: 0.240100
- test loss: 0.219838



a) First Approach



(b) First Approach



(c) Second Approach

Fig. 5. a) First approach of the LSTM training and validation loss for 100 epochs with learning rate of 1e-05, b) First approach of the LSTM training and validation loss for 100 epochs with learning rate of 1e-06, c) Second approach of the LSTM training and validation loss for 50 epochs with changed preprocessing

4.2. Transformer

4.2.1. First Approach

As a first step we tried different parameters with a hyperparameter training. The following parameters were tested for 20 epochs:

- learning_rate = [1e-04, 1e-05, 1e-06]
- dimension_val = [520, 560, 640]
- n_heads = [5, 7, 8]
- n_decoder_layers = [4, 5, 6]
- n_encoder_layers = [4, 5, 6]

From this hyperparameter training the following settings turned out to be most promising: learning_rate=1e-05, dimension_val=640, heads=8, decoder_layers=6, encoder_layers=6. For this setting, we trained the Transformer for 400 epochs. The result: After a few epochs, the model overfits strongly with constant validation loss. In Figure 6 a) we plotted the first 50 epochs of the training loss and validation losses. Here we can see that after about 9 epochs the transformer model is starting to overfit. Reason for this could be for example too less data or a too complex model.

- training loss: 0.153044
- validation loss: 0.255201
- test loss: 0.195875

4.2.2. Second Approach

In the second attempt, we implemented VMD to prevent overfitting. We have kept the original transformer architecture but a weight decay parameter was introduced to penalize complex functions in the optimizer. We also switched the optimizer from Adam to RAdam, since no prewarming of the model might be needed with that optimizer and it stabilizes training, accelerate convergence and improves generalization[15] In Figure 6 b) we can see the training and validation loss of this approach, which are both much closer together than in the first approach. We also see that the training loss falls very rapidly in the beginning which might result from making use of RAdam.

- training loss: 0.173209
- validation loss: 0.230499
- test loss: 0.204823

4.2.3. Third Approach

In the third attempt, we changed several things to achieve a better generalization. As a reference, we used the article by Onnen [16], which presents an improvement approach of multivariant transformers in the context of electricity price predictions. The transformer architecture was implemented with fewer layers. The following parameters were used.

- learning_rate = [1e-05]
- dimension_val = [96]
- n_heads = [5]
- n_decoder_layers = [5]
- n_encoder_layers = [5]
- weight_decay = [0.011, 0.022]

The results of this training can be seen in Figure 6 c). This approach didn't show much of a difference compared to the one before and in fact, reached worst scores.

- training loss: 0.183604
- validation loss: 0.269883
- test loss: 0.225951

4.2.4. Fourth Approach

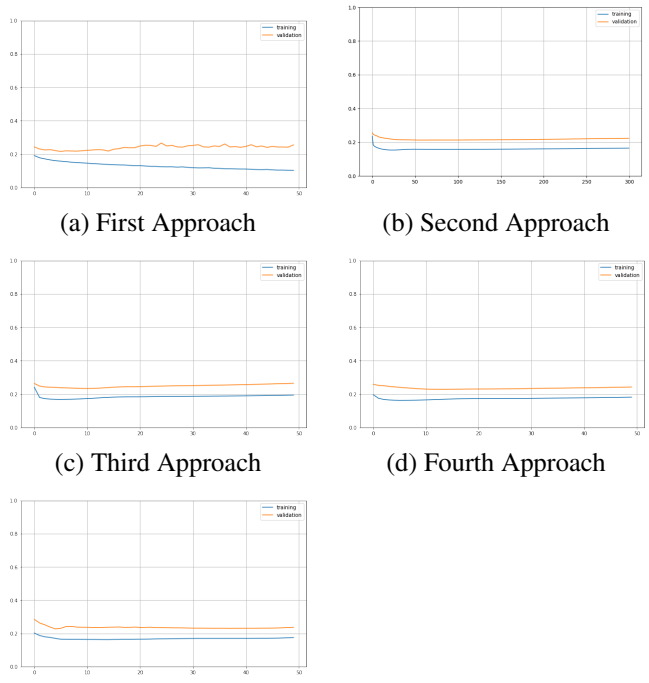
In the fourth attempt we changed the preprocessing of our data. We added the sinus and cosinus of hour of the day as well as applied a square root transformation to the wind speed. With the new changes we trained a simple model with following parameters: learning_rate = 1e-05, dimension_val = 48, n_heads = 5, n_decoder_layers = 5, n_encoder_layers = 5, weight_decay = 0.011, as well as a complex model with the same hyerparameters as the model in the first approach. The result of the training can be seen in Figure 6 d) and e). The

complex model scores shows similar results as the first approach but without overfitting.
simple model:

- training loss: 0.195004
- validation loss: 0.235483
- test loss: 0.217625

complex model:

- training loss: 0.175589
- validation loss: 0.237403
- test loss: 0.199403



(a) First Approach (b) Second Approach

(c) Third Approach (d) Fourth Approach

(e) Fourth Approach

Fig. 6. a) First approach of the transformer training and validation loss for 50 epochs, b) Second approach of the transformer training and validation loss for 300 epochs with VMD, c) Third approach of the transformer training and validation loss for 50 epochs with a simpler architecture and weight decay, d) Fourth approach of the transformer training and validation loss for 50 epochs with preprocessing changes and simple model, e) Fourth approach of the transformer training and validation loss for 50 epochs with preprocessing changes and complex model

5. CONCLUSIONS

After training and experimenting with both models, we got a good overview of the different approaches and techniques. If we go back to the experiments section, we can see from the training, validation, and test loss shown that both architectures achieve similar results. If we look at the first approach of the transformer and compare it with the following approaches, we see that we have successfully prevented the transformer from overfitting. However, no significant improvement or difference in results is visible between approaches two and four and in fact, the scores of the complex model in the last approach and the scores from the first approach are similar. In Figure 6 b) - e) we can observe that, the loss curve flattens steadily. But we do not see any noticeable improvements in the results with the VMD applied to wind speed and by using the weight decay and RAdam. In this case, we were able to prevent overfitting, but we were not able to significantly improve the transformer and push it to the limits. The best loss score on the test set with the transformer model was with the overfitted model and achieved 0.195875. In terms of the Encoder-Decoder LSTM, we can say that we successfully applied hyperparameter training and found promising parameters for training. After training with two different learning rates the higher one gave us better results after 100 epochs. By applying VMD and changing the preprocessing for the second approach the model didn't improve significantly but we got a better loss score on the test set than in the first approach. At the end we got a loss score of 0.219838 with the Encoder-Decoder LSTM. In this context, both architectures are very promising and it seems reasonable to apply them to long-term wind power prediction in the future. As for now, more work is needed to get more precise results as well as more stable ones to apply them to real world applications.

6. FUTURE WORKS

In view of future work and coming from our experiments on the DTU cluster, we observe that the Encoder-Decoder model takes a very long time to run. Thus, one computational challenge is to accelerate the Encoder-Decoder model to adapt to the same run-time as the transformers. Another modification could be to implement a Bayesian hyperparameter approach. This will again accelerate the process of hyperparameter search. Most importantly, we suggest creating an ensemble model for all 7 wind farms available in the dataset in order to predict the power production of all seven plants simultaneously. Moreover, one could compare the ensemble model with a simple model trained for each wind farm separately. One could also incorporate the sliding window approach into the transformer model to take full advantage of the past data.

7. REFERENCES

[1] Project-GitHub-Repository, "https://github.com/jerrite/deeplearning-windenergy-project2022," .

[2] Shu Fanc Tao Hong, Pierre Pinson, "Global energy forecasting competition 2012," 2014.

[3] Zhile Yang Zhou Wu, Gan Luo et al., "A comprehensive review on deep learning approaches in wind forecasting applications," 2021.

[4] Huijuan Wu, Keqilao Meng, Daoerji Fan, Zhanqiang Zhang, and Qing Liu, "Multistep short-term wind speed forecasting using transformer," *Energy*, vol. 261, pp. 125231, 2022.

[5] Jiang Wu Xinyu Wei Fangwei Duan Xingbo Fu, Feng Gao, "Spatiotemporal attention networks for wind power forecasting," .

[6] Hui Liu and Chao Chen, "Data processing strategies in wind energy forecasting models and applications: A comprehensive review," *Applied Energy*, vol. 249, pp. 392–408, 2019.

[7] Xiaodan Wang, Qibing Yu, and Yi Yang, "Short-term wind speed forecasting using variational mode decomposition and support vector regression," *Journal of Intelligent Fuzzy Systems*, vol. 34, pp. 1–10, 06 2018.

[8] Vinícius R. Carvalho, Márcio F.D. Moraes, Antônio P. Braga, and Eduardo M.A.M. Mendes, "Evaluating five different adaptive decomposition methods for eeg signal seizure detection and classification," *Biomedical Signal Processing and Control*, vol. 62, pp. 102073, 2020.

[9] Xin Wang Xiang Rong MENG Yong Zhai Hong Hai Li Rong Gui ZHANG Kuan LU, Wen Xue SUN, "Short-term wind power prediction model based on encoder-decoder lstm," 2018.

[10] Rishabh Gupta Rohitash Chandra, Shaurya Goyal, "Evaluation of deep learning models for multi-step ahead time series prediction," 2021.

[11] Yi Pan Chenyou Fan, Yuze Zhang et al., "Multi-horizon time series forecasting with temporal attention learning," 2019.

[12] Pytorch Library, "Language modeling with nn.transformer and torchtext," .

[13] Kasper Groes Albin Ludvigsen, "How to make a transformer for time series forecasting with pytorch," .

[14] Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion, "Deep transformer models for time series forecasting: The influenza prevalence case," 2020.

[15] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han, "On the variance of the adaptive learning rate and beyond," 2019.

[16] Heiko Onnen, "Transformer unleashed: Deep forecasting of multivariate time series in python," .