Machine Learning Engineer Nanodegree

Wind Energy Forecast for the Sotavento's Wind Farm

Oscar Daniel Garcia Lino
June 7th, 2019

I. Definition

Project Overview

As a renewable energy source, wind power generation will play a bigger role in the future [1]. However, one of the main problems of wind energy is its dependency on environmental conditions. In the electricity grid, at any moment, balance must be maintained between electricity consumption and generation. The nature intermittency of wind adds complexity to the management of wind energy but this intermittency can be partly mitigated through forecasting techniques, which aim at reducing the uncertainty of future wind power generation of a wind farm or portfolio [2].

Wind power forecasting models can be classified into two categories depending on their methodology: Physical approach and Statistical approach. The physical approach is based on lower atmosphere or numerical weather predictions, in the other hand, the statistical approach is based on vast amount of historical data without considering meteorological conditions. There is a third approach, the hybrid approach, which combines physical methods with statistical methods, particularly uses weather predictions and time series analysis. [3]

Problem Statement

In this project, I am proposing a hybrid approach feeding a recurrent neural network (RNN) specifically a Long Short-Term Memory (LSTM) with historical time series data to forecast wind power production for a short term (8 hours ahead) for the Sotavento's wind farm in Galicia, Spain [4].

Solution Statement

There are several time-series forecasting techniques like auto regression (AR) models, moving average (MA) models, Holt-winters, ARIMA, to name a few [4]. These techniques are very popular and traditional for time series forecasting but they have a bottleneck. This is because is not easy to incorporate new signal in the forecasting model like events or weather data for example [5]. More powerful methods need to be used to be able to incorporate multiple signals in forecasting models. This is where deep learning comes into play. Long short-term memory (LSTM) recurrent neural network (RNN) is becoming popular for time series forecasting. Unlike other machine learning algorithms, LSTM are capable of automatically learning features from sequence data, it supports multi-variate data, and it can output a variable length sequence that can be used for multi-step forecasting [6].

The dataset that I am going to use for this project represents a multivariate time series of power-related variables that can be used to model and forecast power production for the Sotavento's wind farm. For purposes of this project, I am planning to develop a LSTM model to forecast hourly power production eight hours ahead. Wind farm hourly time series data from 2010 up to 2018 is going to be used for training the model; 20% of the test data is used as validation data. The test data set is going to contain data from January to April 2019.

Metrics

There are many different performance measures to choose from when it comes to time series predictions. For this project I am proposing to use scale-dependent errors as evaluation metrics. Scale-dependent errors are accuracy measures that are based only on e_t and cannot be used to make comparisons between series that involve different units [7]. The forecast that I am planning to execute in this project involves only one output variable, therefore scale-dependent errors seems suitable in this case.

The two most commonly used scale-dependent measures are based on the absolute errors or squared errors [7]:

$$Mean\ absolute\ error: MAE = mean(|e_t|)$$

$$Root\ mean\ squared\ error: RMSE = \sqrt{mean(e_t^2)}$$

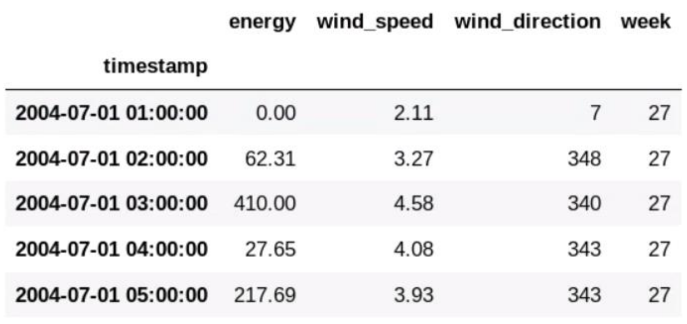
The performance metric chosen for this problem will be the RMSE for each lead time from hour 1 to hour 8. A single score summarizing the performance of a model on the test dataset will be implemented which will help in model selection. This implementation is done in the evaluate_forecasts() function. The function returns the overall performance score and an array of RMSE scores for each hour forecasted.

II. Analysis

Data Exploration

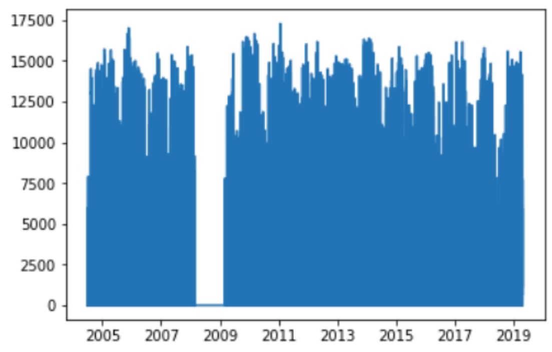
The dataset to be used was obtained from the website for the Sotavento Experimental Wind Farm project. The Sotavento project was promoted by the Government of Galicia for the promotion, training and research of renewable energy sources [4]. The website offers real time data visualization of the wind farm SCADA data. Historical time series data is also available and accessible for public domain.

The available historical data goes from July 2004 until now (keeps updating daily). The dataset was manually copied into a comma-separated values file. There are only three variables available in the historical data, these are the wind speed (m/s), wind direction (º) which are weather related features, and power energy produced (kWh) which is going to be the output feature and what I intend to forecast. The hourly wind speed and wind direction is the average value during the hour, and the hourly energy value is power produced during that hour. The week of the year is an extra feature added to catch any seasonality during the year. The table below shows a snapshot of these variables:



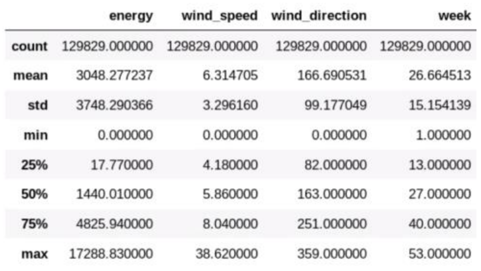
	energy	wind_speed	wind_direction	week
timestamp				
2004-07-01 01:00:00	0.00	2.11	7	27
2004-07-01 02:00:00	62.31	3.27	348	27
2004-07-01 03:00:00	410.00	4.58	340	27
2004-07-01 04:00:00	27.65	4.08	343	27
2004-07-01 05:00:00	217.69	3.93	343	27

The trend below shows an overview of the energy over the entire dataset. As observed, the dataset shows a big data gap between 2008 and 2009. For the purpose of this project, I am not considering to use any of the data before 2009, this to avoid any complication when training the model with the missing data in 2008.



There are a couple of remediation that need to be done to the data before feeding it to the neural network. For instance, decimals are separated with a coma (‘,’) and thousands with a period (‘.’) (ex 1.808,72). This was causing some issues when importing the csv file into a pandas data frame. Also, missing values were represented with a hyphen in the data source, a transformation is done to replace missing data with 0’s instead. The data transformation is executed in the *dataPrep()* function.

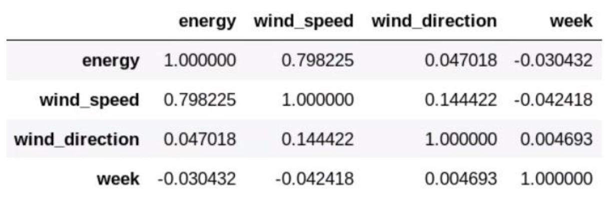
The pandas’ *describe()* function shows an statistical overview of each of the features in the dataset. The max parameter quickly highlights that there are not any outliers for any of the features. It also highlights that the maximum power generation capacity for the wind farm is over 17 MWh:



	energy	wind_speed	wind_direction	week
count	129829.000000	129829.000000	129829.000000	129829.000000
mean	3048.277237	6.314705	166.690531	26.664513
std	3748.290366	3.296160	99.177049	15.154139
min	0.000000	0.000000	0.000000	1.000000
25%	17.770000	4.180000	82.000000	13.000000
50%	1440.010000	5.860000	163.000000	27.000000
75%	4825.940000	8.040000	251.000000	40.000000
max	17288.830000	38.620000	359.000000	53.000000

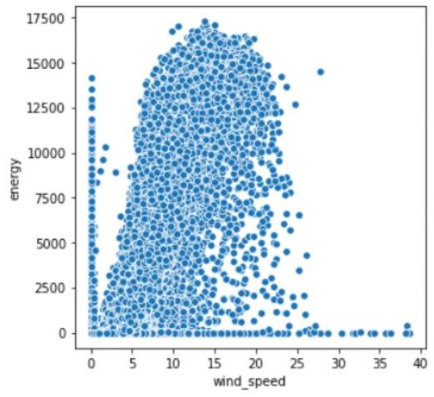
Exploratory Visualization

Calculating the correlation between the different features we can observe that there is a strong positive correlation between the wind speed and the energy produced. There is almost not existent correlation between the energy output and the wind direction or time of the year.



	energy	wind_speed	wind_direction	week
energy	1.000000	0.798225	0.047018	-0.030432
wind_speed	0.798225	1.000000	0.144422	-0.042418
wind_direction	0.047018	0.144422	1.000000	0.004693
week	-0.030432	-0.042418	0.004693	1.000000

The scatter plot below confirms the correlation between wind speed and energy. This correlation is expected as wind energy is high dependable of wind speed.



Algorithms and Techniques

The proposed model to resolve the forecasting problem is a type of Recurrent Neural Network (RNN) called Long Short-Term Memory (LSTM). Recurrent nets are a type of artificial neural network designed to recognize patterns in sequences of data, such as numerical times series data emanating from sensors. These algorithms take time and sequence into account, they have a temporal dimension [9].

RNN take as their input not just the current input example they see, but also what they have perceived previously in time, and these inputs are combined to determine how they respond to new data. It is often said that recurrent networks have memory [9]. Adding memory to the neural network helps to understand information in the sequence that can be used by the network to perform tasks. The main obstacle for RNN is the vanishing gradient problem [9]. The intention of this document is not to explain this problem as there are many resources on the web explaining it [10].

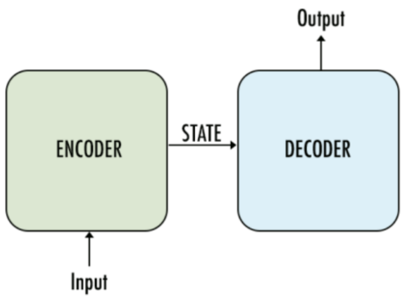
LSTM was proposed by the German researchers Sepp Hochreiter and Juergen Schmidhuber as a solution to the vanishing gradient problem [9]. LSTMs allow recurrent nets to continue to learn over many time steps. LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computer's memory. The cell makes decisions about what to store, and when to allow reads, writes and erasures, via gates that open and close [9].

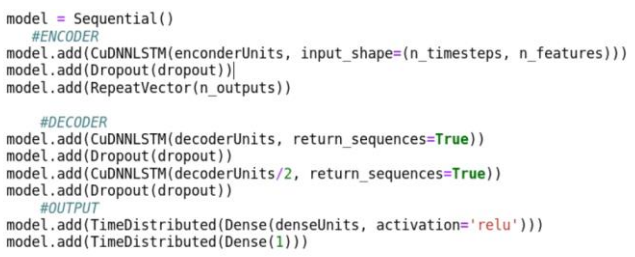
Given the gated architecture and its ability to manipulate its memory state LSTMs are very good in extracting patterns from input features, where the input data spans over long sequences such as time series data. This one of the reasons why LSTMs can be valuable for forecasting problems. Moreover, classical linear methods can be difficult to adapt to multivariate or multiple input forecasting problems, but LSTMs can almost seamlessly model this type of problems

Specialized architectures have been developed that are specifically designed to make multi-step sequence predictions, generally referred to as sequence-to-sequence or seq2seq predictions. Engineers at google were the first one to propose and coined a seq2seq model architecture to enhance the google translator. The architecture was a deep LSTM encoder-decoder network [11]. An encoder-decoder model can also be used for time series forecasting.

There are several encoder-decoder models setups for example a CNN-LSTM Encoder-Decoder model, a ConvLSTM Encoder-Decoder model or an Encoder-Decoder LSTM model. In this document I am proposing the use of an encoder-decoder LSTM model to forecast wind energy for the Sotavento's wind farm.

An encoder-decoder LSTM is a model comprised of two sub-models: one called the encoder that reads the input sequences and compresses it to a fixed-length internal representation, and an output model called the decoder that interprets the internal representation and uses it to predict the output sequence [6]. A high-level view of the architecture is shown below:





```
model = Sequential()
   #ENCODER
model.add(CuDNNLSTM(enconderUnits, input_shape=(n_timesteps, n_features)))
model.add(Dropout(dropout))
model.add(RepeatVector(n_outputs))

    #DECODER
model.add(CuDNNLSTM(decoderUnits, return_sequences=True))
model.add(Dropout(dropout))
model.add(CuDNNLSTM(decoderUnits/2, return_sequences=True))
model.add(Dropout(dropout))
    #OUTPUT
model.add(TimeDistributed(Dense(denseUnits, activation='relu')))
model.add(TimeDistributed(Dense(1)))
```

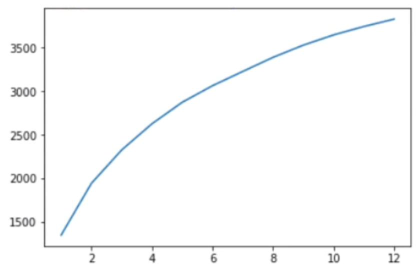
Benchmark

I am going to use a persistence model as benchmark model to stablish a baseline to compare my forecasting model. A persistence model is the simplest way of producing a forecast [12]. The persistence model uses the value at the previous time step (t-1) to predict the expected outcome at the next time step (t+1). In other words, it assumes that for example the energy produced by the wind farm for the next hour is going to be same as the energy produced during the past hour. The persistence model is going to be treated as univariate time series forecasting problem and only the wind farm energy generated is going to be considered for the forecast.

The overall RMSE score value gotten with the persistence model is 3826.681.

The RMSE score over each individual forecasted hour is:

```
p=1 RMSE:1350.691
p=2 RMSE:1943.927
p=3 RMSE:2326.600
p=4 RMSE:2626.504
p=5 RMSE:2873.293
p=6 RMSE:3063.012
p=7 RMSE:3226.669
p=8 RMSE:3388.404
p=9 RMSE:3528.054
p=10 RMSE:3645.406
p=11 RMSE:3741.995
p=12 RMSE:3826.681
```



III. Methodology

Data Preprocessing

As mentioned in the data exploration section, there are few processing steps that must be done before feeding the data to the model.

The pre-processing is done in three separate functions.

DataPrep() function:

• The wind_speed feature ranges in value from 0 m/s to 36.62 m/s, the decimal values are separated by a coma ','. This causes some anomalies when converting the csv data into a pandas dataframe. There is a step to change the column format from ‘xx,xx’ to ‘xx.xx’.

• Similar to the wind_speed feature, the Energy column format must be transformed from ‘xx.xxx,xx’ to ‘xxxxx.xx’.

• Missing values are represented with a ‘-’. There is a step to replace all missing values with cero.

• The timestamp column is a string in the format D/M/YYYY HH:MM. The column gets transformed to a pandas date time format, when doing this the dayFirst parameters must be indicated as True. This column is not used as feature for the LSTM model but it is helpful when doing any data visualization. The timestamp column is also set as the index for the dataframe.

• The wind_speed, wind_direction and energy columns are converted from string type to numeric.

• A new column is added to the dataframe with the week number in the year.

• There is a step to indicate if the LSTM model wants to be considered as uni-variate or multi-variate type of forecast. For the uni-variate the output variable “energy” is the only variable considered as input to the model. In a multi-variate forecast, the wind_speed, wind_direction and week are passed as inputs to the model and the energy as the target output variable.

split_dataset() function:

•The data gets split into train and test datasets. Train contains two datasets, train_x containing the input features, and train_y with the output feature. Similar to the train dataset, the test dataset is split in two datasets for input and output features.

• The features and target variable have different ranges in values for this reason the test and train datasets are scaled to (-1,1).

• The train and test datasets are restructured into windows of 8 hours. The number of hours in the window must reflect the number of hours that want to be forecasted in one step.

to_supervised() function:

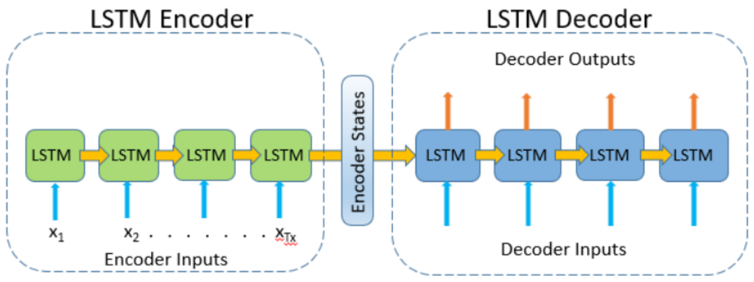
• Before training the model, the training data must be transformed to supervised. Specifically, the organization of data must be transformed into input and output patterns where the observation at the previous time step is used as an input to forecast the observation at the current time step.

Implementation

The implementation of this project was highly inspired by Dr. Jason Brownlee in his tutorial 'How To Develop LSTM Models for Multi-Step Time Series Forecasting of Households Power Consumption' [6] and parts of the ideas and code used in this projects were taken from his tutorial.

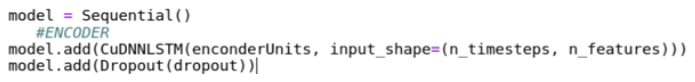
As it is well known, training deep learning models can take from few minutes to days. The hardware used for the training and development of a deep learning model can be a factor for the success of getting to acceptable results. This project was implemented using a GeForce RTX 2070 graphics card, and Keras with Tensorflow 2.0 Beta as backend. I would like to mention that setting this up from scratch and have it working in a conda's virtual environment was quite a challenge!

In this project, I followed a sequential sequence to sequence (seq2seq) model approach, that is the sequential model hast two parts -an encoder and a decoder. Each part is a different neural network and they get combined into one bigger network. In big terms, the task of the encoder network is to understand the input sequence, and create a smaller dimensional representation of it. This representation is then forwarded to a decoder network which generates a sequence of its own that represents the output [13].



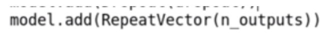
Manohar B 2019, Time Series Forecasting with deep stacked unidirectional model, Toward Data science, Accessed 25 May 2019 < https://towardsdatascience.com/time-series-forecasting-with-deep-stacked-unidirectional-and-bidirectional-lstms-de7c099bd918 >

In a closer look to the final model used in this project, the encoder is defined as a single LSTM hidden layer with 1500 units. The encoder reads the input sequence which contains the last 24 hours for the three input features (wind speed, wind direction and week)



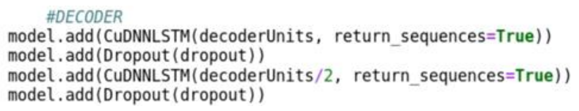
```
model = Sequential()
    #ENCODER
model.add(CuDNNLSTM(enconderUnits, input_shape=(n_timesteps, n_features)))
model.add(Dropout(dropout))
```

The encoder squashes the input data into a single feature vector. This internal representation of the input sequence is repeated multiple times, once for each time step in the output sequence. This is accomplished by adding a RepeatVector layer to the model and passing the number of outputs, remember that in this case I am training the model to forecast 8 hours.



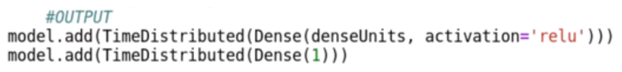
```
model.add(RepeatVector(n_outputs))
```

The encoder's representation of the input sequence is then passed to the decoder. The decoder consists of two LSTM hidden layers, the first layer with the same number of units in the encoder's hidden layer. The second layer with half of the units in the first decoder layer. I found this configuration to return a better performance. The decoder outputs a value for each hour to be forecasted, this represents the basis for what to predict for each hour in the output sequence.



```
    #DECODER
model.add(CuDNNLSTM(decoderUnits, return_sequences=True))
model.add(Dropout(dropout))
model.add(CuDNNLSTM(decoderUnits/2, return_sequences=True))
model.add(Dropout(dropout))
```

Lastly, a fully connected layer is used to interpret each time step in the output sequence before the final output layer. It is important to mention that the final output layer predicts a single hour in the output sequence, and not the eight hours in the sequence. To be able to return the eight sequential forecasted hours, we need to add a TimeDistributed layer. The TimeDistributed layer applies a specific layer, the Dense layer in this case, to every sample it receives as input from the decoder. Therefore, wrapping up the fully connected layer and the final output layer in a TimeDistributed layer give us as a result the output sequence with all the steps wanted to be forecasted.



```
    #OUTPUT
model.add(TimeDistributed(Dense(denseUnits, activation='relu')))
model.add(TimeDistributed(Dense(1)))
```

This configuration allows the LSTM decoder to figure out the context required for each step in the output sequence and the wrapped dense layers to interpret each time step separately, yet reusing the same weights to perform the interpretation [6].

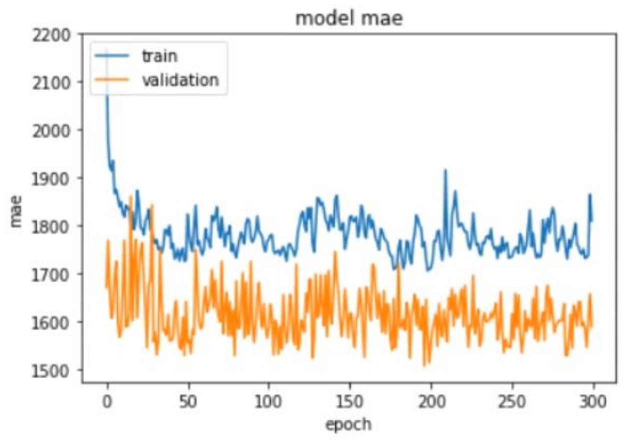
Refinement

I found the process of finding the best model architecture to be an art. There are many options to choose from, and many things to consider. For example, the deepness of the encoder and decoder network, one hidden layer, two hidden layers, perhaps more hidden layers? All this process is very time consuming. Depending the number of hidden layers, and units for each layer the training time can be few minutes or many hours.

There are many other features to tune or decide other than the number of hidden layers. These features are discussed below.

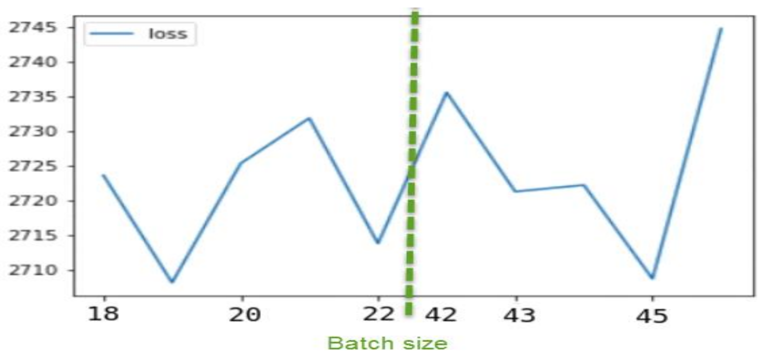
Number of epochs

One of first things I had to decide is the number of epochs to consider for tuning the model. After trying different models, I found the loss function to converge at around 30 epochs, therefore I decide to use 30 epochs for tuning the other parameters of my model.



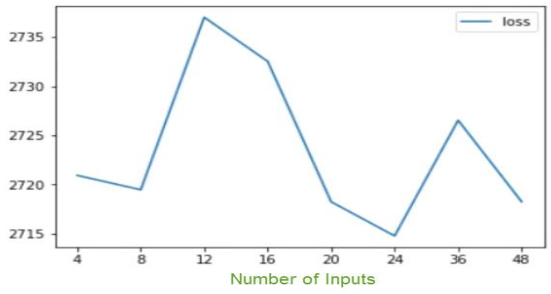
Batch size

The second parameters I considered to tune was the batch size. I found the batch size of 19 and 45 to return the best loss function performance. The model's training time gets reduced with bigger batch sizes, therefore I decided to use a size of 45.



Number of Inputs

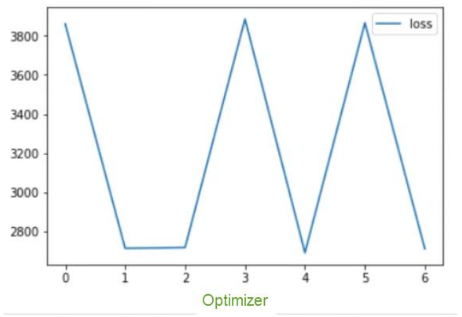
The number of inputs corresponds to the number of input hours passed to the model to do the forecast. From the trend below, it can be observed that passing the previous 24 hours to the model returns the better loss performance.



Optimizer

I tried several different optimizers. From the results, it can be observed that Adam returns the best performance, therefore Adam is the optimizer used for my final model.

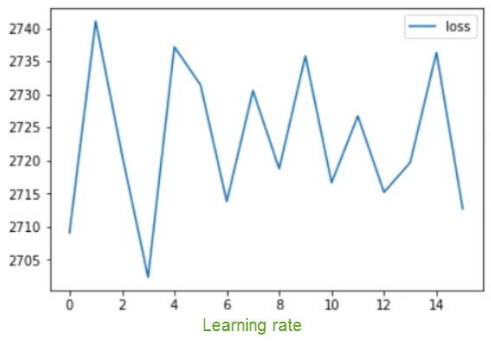
Optimizer	Loss
SGD	3860.06106
RMSprop	2715.255275
Adagrad	2719.595222
Adadelta	3884.866782
Adam	2692.765952
Adamax	3865.369282
Nadam	2713.099164



Adam’s learning rate

By default, Adam’s learning rate is .001. I decided to find out if that was the optimal learning rate. My findings were that a learning rate of .00085 returns a better performance.

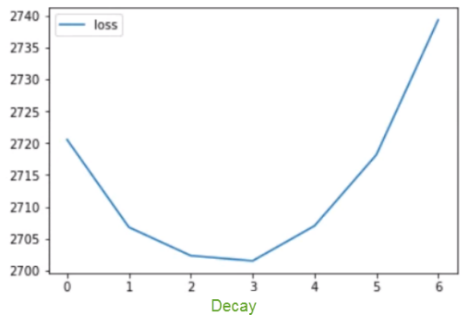
	Learning rate	Loss
0	0.0007	2709.052
1	0.00075	2741.059
2	0.0008	2720.958
3	0.00085	2702.306
4	0.0009	2737.186
5	0.00095	2731.41
6	0.00098	2713.759
7	0.001	2730.551
8	0.0012	2718.732
9	0.0015	2735.821
10	0.0017	2716.628
11	0.002	2726.713
12	0.0023	2715.169
13	0.0025	2719.693
14	0.003	2736.33
15	0.0035	2712.682
16	0.004	3887.212



Optimizer’s decay

The model was tested with different decay values and a decay of .007 turned out to be the best choice.

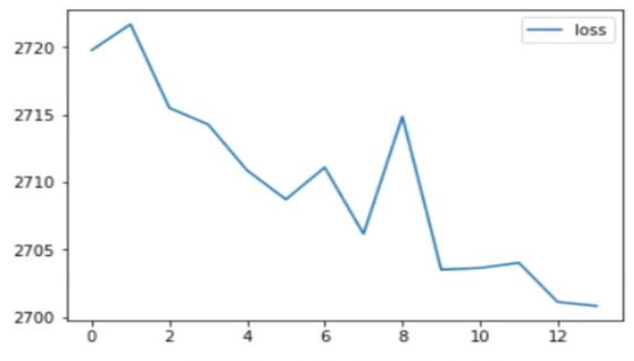
	Decay	Loss
0	0	2720.56
1	0.002	2706.789
2	0.005	2702.333
3	0.007	2701.514
4	0.01	2707.007
5	0.03	2718.161
6	0.05	2739.354



Encoder tuning – Total units for hidden layer

As previously mentioned, the encoder consists of a unique hidden LSTM layer. The model was trained with different number of units in the encoder's hidden layer. From the results it can be observed that the model's performance can be improved by increasing the number of units in the encoder. Increasing the number of units also increases the training time.

The model was trained with this number of units: [12, 24, 48, 64, 80, 120, 150, 200, 240, 280, 300, 320, 400, 500]. The results can be observed in the chart below. Noted that the x-axis legend corresponds to the positional index of the unit's array above.

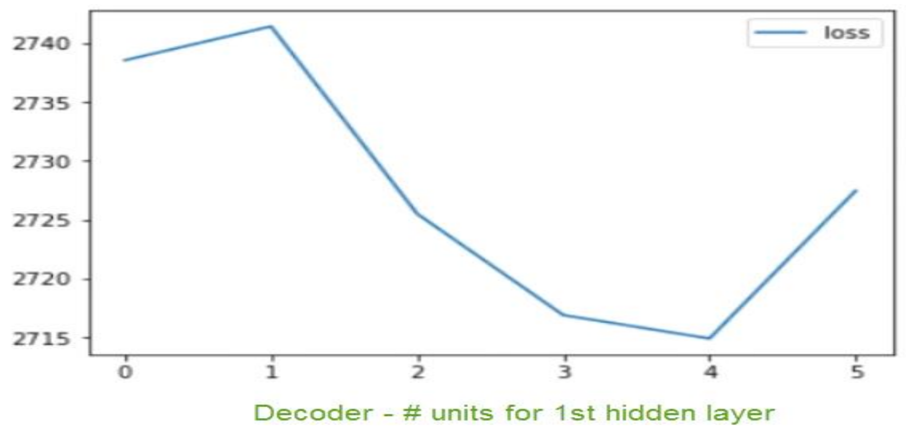


Encoder - # units for hidden layer

Decoder tuning – one LSTM hidden layer

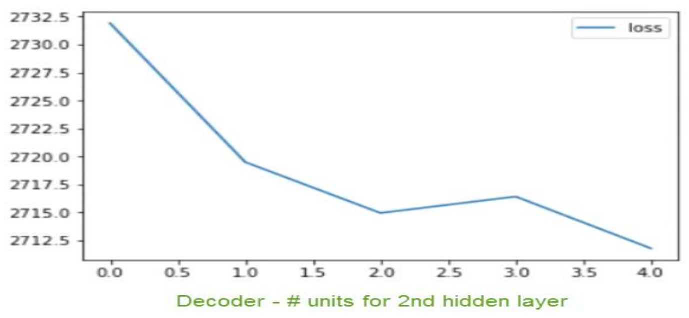
The model was trained with one hidden layer in the encoder and one hidden layer in the decoder. As mentioned in the encoder tuning section, increasing the number of units in the encoder increases the training time. I decided to use 300 units in the encoder for tuning the decoder.

The model was trained with these number of units in the decoder's LSTM hidden layer: [12, 24, 80, 150, 300, 500]. I found that the model performance is best when the decoder's hidden layer has the same number of units as the encoder's hidden layer. For this specific case, the model performance was best with 300 units in the decoder's hidden layers.



Decoder tuning – second LSTM hidden layer

The model was trained with two hidden layers in the decoder. The encoders hidden layer was kept with 300 units, also the decoders first hidden layer was kept with 300 units. The decoder’s second hidden layer was tuned with these number of units: [24, 80, 150, 300, 500]. The models performance looks to get better when increasing the number of units in the decoder’s second hidden layer.

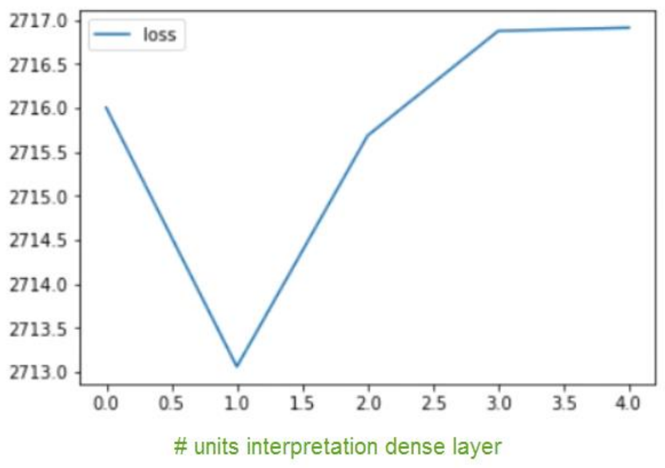


Tuning interpretation Dense layer

The model was trained with these number of units in the interpretation dense layer: [500, 300, 150, 24, 12]. The units for the encoder-decoder hidden layers was kept as follows:

- Encoder hidden LSTM layer: 300 units
- Decoder 1st hidden LSTM layer: 300 units
- Decoder 2nd hidden LSTM layer: 150 units

As shown in the trend below, the model performance was better with 300 units in the interpretation dense layer. My conclusion from this is that the model performance gets better when the number of units is the same across the different hidden layers.



IV. Results

Model Evaluation and Validation

During the development of this project a validation set corresponding to 20% of the training dataset was used.

Based on the results described in the refinement section, the parameters used for training the my final LSTM-LSTM encoder-decoder model is as follows:

Number of inputs = 24

Encoder hidden LSTM units = 1500

Decoder 1st hidden layer units = 1500

Decoder 2nd hidden layer units = 1500

Interpretation dense layer units = 24

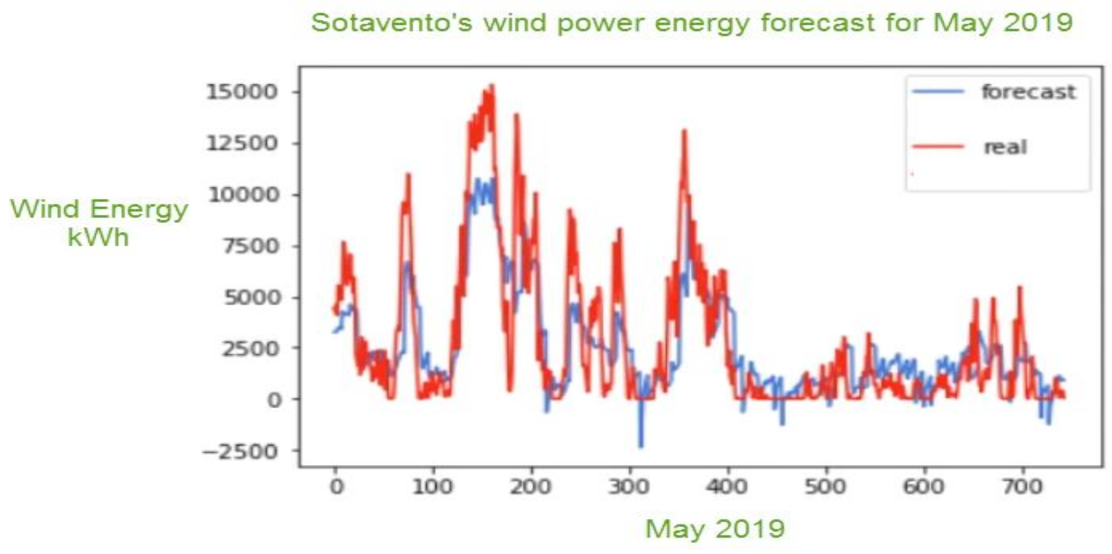
Adam optimizer learning rate=0.00089

Adam optimizer decay = 0.007

Number of epochs = 30

Batch size = 45

The model was used with data never seen by the model. This data corresponds to the data published by the Sotavento's project for the month of May 2019. Remember that the training data contained data for the period 2010 to 2018. The test data contained the period January-April 2019. The forecast results for the month of May can be observed below:



Justification

Overall the LSTM-LSTM encoder-decoder model proposed in this project returns better results than the benchmark model. The RMSE performance obtained with the benchmark model was 3826.681 while the performance obtained with the seq2seq model is 2804.309. Although the performance in the seq2seq model is better, I believe there is still possibilities to improve it more.

V. Conclusion

Reflection

Getting to the final results was challenging, and enjoyable at the same time. The part of this project that took me more time was deciding the final model architecture. I tried several different architectures. For instance, I tried a CNN-LSTM encoder-decoder architecture, when training this model the loss and validation function were decreasing but at the end when evaluating the model on the test dataset I was able to get only a flat line. My conclusion was that the model was only learning some sort of a mean value. I tried adding more units and layer to the CNN encoder but I was not able to get any good results. Perhaps more data is needed to be able to use a CNN model as the encoder.

After a couple of days trying to find a good CNN encoder model without any clear success, I finally decided to move on and use an LSTM encoder model. With the LSTM encoder I was able to get some promising results quickly.

Tuning the LSTM-LSTM encoder-decoder model took me several days. I was able to tune many different parameters as discussed in the refinement section. With the results obtained, the model forecast results follow the trend of the real observation, but it is not able to forecast quick changes or picks. I believe the final model and solution fits my expectations, but I think there is still some room for improvements before using it to take decisions.

I was able to try different number of inputs to the model. One of the models I tried was a multivariate input model, using only the wind power energy feature as the input to the model. I also tried two different multi-variate model, one where the input features were the wind speed, wind direction, week and wind energy, and a second one where the wind energy was not considered as input feature. Among the different number of input features, the univariate model was the one with the better performance, followed by the multivariate features model that includes the wind energy. The multivariate model that did not included the wind energy was the one with the worst performance of the three.

Although the third model type was the worst, I ended up deciding to use it as my final model. One reason for this decision was that the performance was the worst but it was not really too bad, it was just slightly worst. The second reason was that removing the wind energy as input feature increases the potential of the model to be reusable to forecast long-term periods, not just a short eight hours' period. This is because the model depends only on the wind speed and wind direction features, the week can be easily calculated. There are several sources from where to take wind speed and wind direction forecasts for long time periods (days and even weeks). These other sources with weather forecasts could be used as input to my model to forecast wind energy for longer time periods.

With the results obtained I can confirm seq2seq LSTM models can be suitable to forecast time series data, and I look forward to try it in some other different scenarios like heat power forecast demand for district energy facilities. District Energy Systems are networks of hot and cold water pipes, typically buried underground, that are used to efficiently heat and cool buildings using less energy than if the individual buildings were to each have their own boilers and chillers [14].

Improvement

According to the literature, there are other seq2seq models that could return better results that an LSTM-LSTM model. A different encoder-decoder architecture is the CNN-LSTM. I mentioned that I tried to implement the network for this project but I did not have good success, but I still would like to give this a try in the future to be able to compare the results. Another architecture is a ConvLSTM network, which also could return some promising results.

Using a deeper LSTM-LSTM network could potentially improve the performance of the model. I did try a deeper network but soon I realized that it will take days to finish training it, so I decided not to go on that route as the deadline for the project was getting closer.

References

[1] AESO, "aeso," [Online].

[2] A. C.-T. O. L.-G. Cristobal Gallego-Castillo, "CORE - A review on the recent history of wind power ramp forecasting," [Online]. Available: https://core.ac.uk/download/pdf/78495744.pdf. [Accessed 23 April 2019].

[3] P. G. X. H. Xiaochen Wang, "ScienceDirect - A Review of Wind Power Forecasting Models," [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1876610211019291. [Accessed 22 April 2019].

[4] "Towards Data Science," Towards Data Science Inc, 17 January 2018. [Online]. Available: https://towardsdatascience.com/using-lstms-to-forecast-time-series-4ab688386b1f. [Accessed 27 April 2019].

[5] D. Yuan, Director, *Two Effective Algorithms for Time Series Forecasting.* [Film]. InfoQ, 2018.

[6] P. Jason Brownlee, "How to Develop LSTM Models for Multi-Step Time Series Forecasting of Household Power Consumption," Machine Learning Mastery, 10 October 2018. [Online]. Available: https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/. [Accessed 27 April 2019].

[7] R. J. H. a. G. Athanasopoulos, "Forecasting: Principles and Practice," OTEXTS, [Online]. Available: https://otexts.com/fpp2/accuracy.html. [Accessed 27 April 2019].

[8] S. G. Foundation, "sotaventogalicia," Sotavento Galicia, S.A., 2005. [Online]. Available: http://www.sotaventogalicia.com/en. [Accessed 19 April 2019].

[9] "skymind," skymind, [Online]. Available: https://skymind.ai/wiki/lstm. [Accessed 2 June 2019].

[10] "superdatascience.com," superdatascience, 22 Aug 2018. [Online]. Available: https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem. [Accessed 2 June 2019].

[11] O. V. Q. V. L. Ilya Sutskever, "Sequence to Sequence Learning with Neural Networks," Google, 14 Dec 2014. [Online]. Available: https://arxiv.org/pdf/1409.3215.pdf. [Accessed 10 Jun 2019].

[12] "Persistence Method," University of Illinois, [Online]. Available: http://ww2010.atmos.uiuc.edu/(Gh)/guides/mtr/fcst/mth/prst.rxml. [Accessed 1 May 2019].

[13] F. SHAIKH, "analyticsvidhya," 15 March 2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/03/essentials-of-deep-learning-sequence-to-sequence-modelling-with-attention-part-i/. [Accessed 12 June 2019].

[14] "what-is-district-energy," FVB Engineering, [Online]. Available: http://www.fvbenergy.com/what-is-district-energy/. [Accessed 10 June 2019].