BM20A6100 ADAML Leevi Kämäräinen, Taru Haimi, Sameed Ahmed

House Energy A2, Predicting daily electricity consumption for houses.

Link to MATLAB Drive: https://drive.matlab.com/sharing/6cd31f83-5c35-4a52-80dc-494db876a681

1 Introduction

The aim of this project work is to forecast daily electric power consumption for a house using multivariate data. For estimation three different models are utilized: recurrent neural network (RNN), long short-term memory neural network (LSTM), and transformer neural network (TNN). The models' performances are compared, and additionally their forecasting power for more than one day ahead are explored.

2 Exploratory data analysis and data pretreatment

2.1 Datasets and data distributions

The primary dataset in use is hourly power usage of 2-storey house located in Houston, Texas, USA. It contains hourly power usage values, the day of the week of the time stamp and a note for whether the datapoint is weekday or weekend, and these can be seen on the table 1. The time period of data is from 01.06.2016 until to August of 2020.

Feature	Unit
Power Usage	kWh
Day of the Week	$(0-6)$
Notes	(Weekday or Weekend)

Table 1. Features and their units from power usage dataset.

The supporting data set is historical weather data also from Houston. It contains daily data for the features listed in table 2. The temperatures are originally listed in Fahrenheit, and wind speed in miles per hour, but they are chosen to be converted into metric system if there is need to visualize them, since this will make the analysis more intuitive.

In the figure 1 the distribution for the power usage is visualized. The distribution is very spiky, and there are definite parts when the values get much higher or lower which would indicate some kind of seasonalities to exist. The monthly distributions on the figure 2 indicates that during the summer months the times where the power usage is higher increases, which could for example be explained by increased cooling costs for the houses. On the weekday distributions on figure 3 however, there is no clear differences between the different weekdays. There is a slight increase in the higher power usages during the weekends, so that is one thing to consider while modelling. Overall most of the power usage values falls between 0-2 kWh, while values above that are more abnormal.

Feature	Unit
Max temperature	$^{\circ}C$
Min temperature	$^{\circ}C$
Average temperature	$^{\circ}C$
Max dew	$^{\circ}C$
Min dew	$^{\circ}C$
Average dew	$^{\circ}C$
Max humidity	$\%$
Min humidity	$\%$
Average humidity	$\%$
Max wind speed	$\frac{m}{s}$
Min wind speed	$\frac{m}{s}$
Average wind speed	$\frac{m}{s}$
Max air pressure	Hg
Min air pressure	Hg
Average air pressure	Hg
Precipitation	mm
Day of the week	(0-6)

Table 2. Features and their units from environmental dataset.

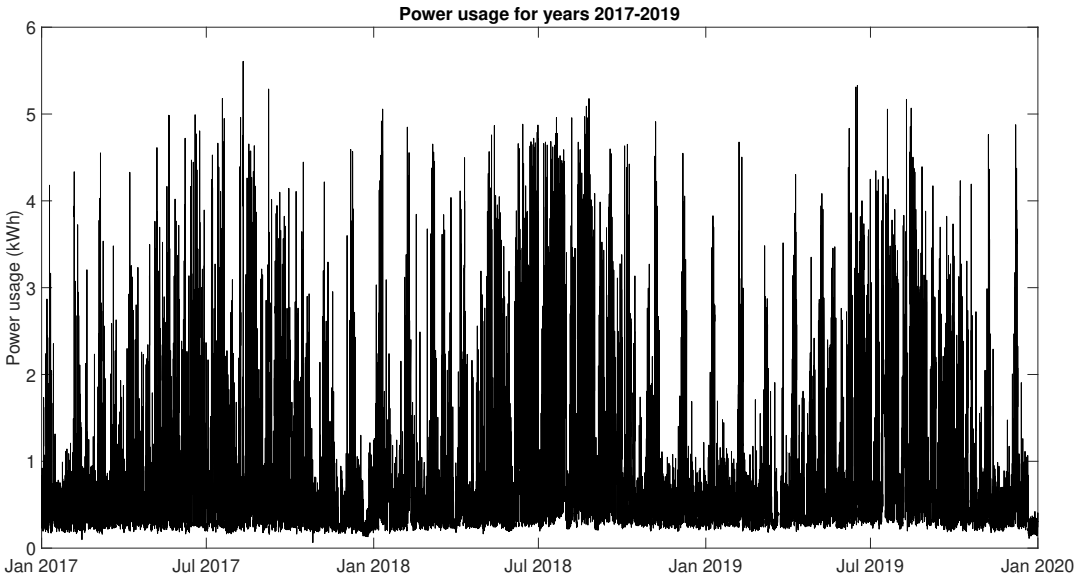

Figure 1. The distribution for the power usage of the house during the years 2017 - 2019.

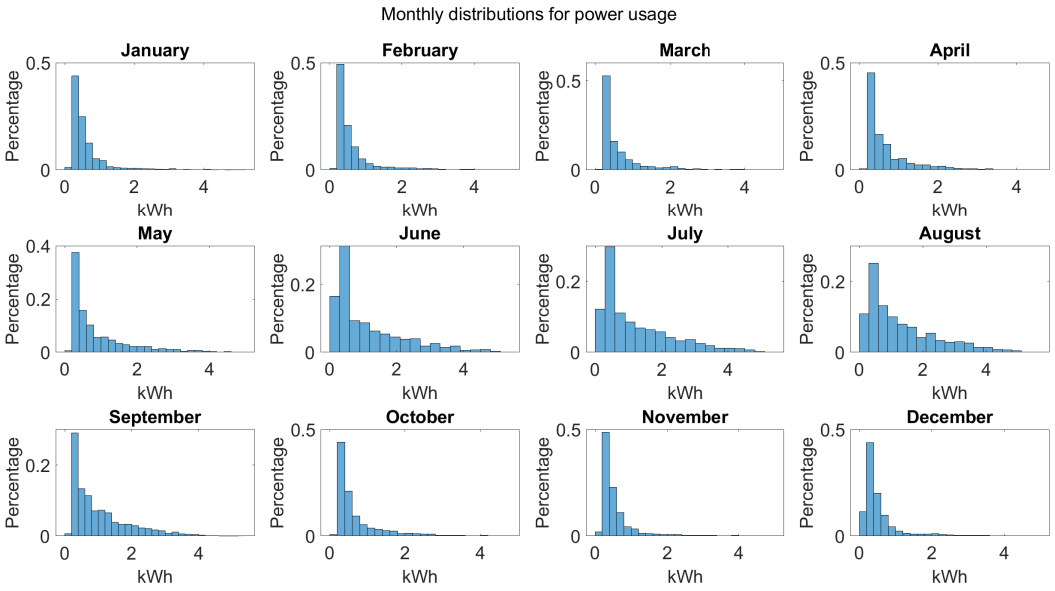

Figure 2. The distribution for the monthly power usages during the years 2017 - 2019.

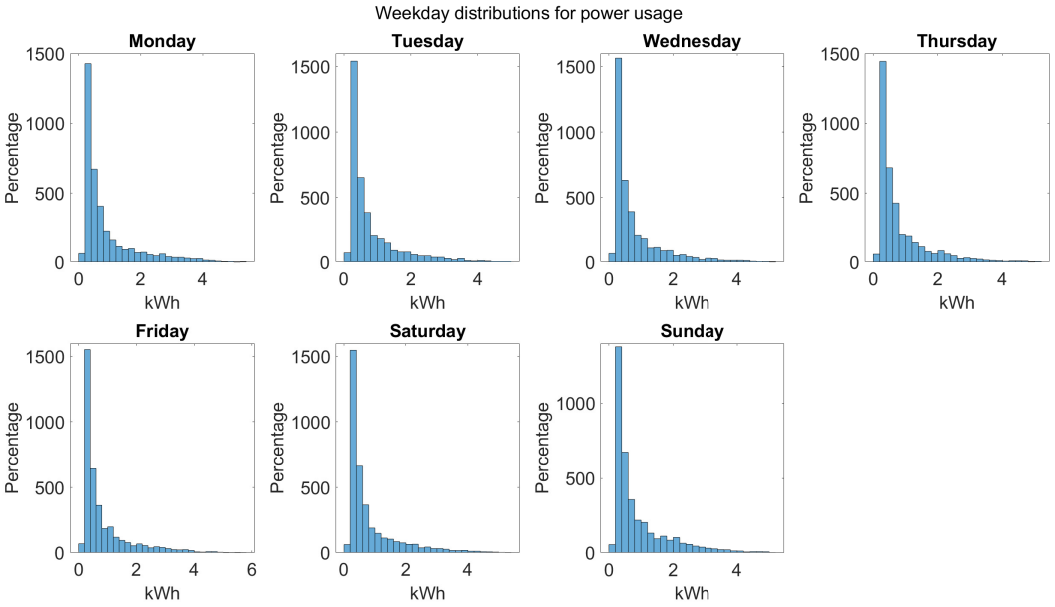

Figure 3. The distribution for the weekday dependent power usage during the years 2017 - 2019.

2.2 Data standardization

Because the datasets contain different features with different units and value ranges, they need to be normalized some way to make model building more efficient. Centering and scaling is performed with standard score (Z-score). The standardization will be done independently for each variable, so that each of the variables will have a mean of 0, and a standard deviation of 1. The equation for the standardization is

$$z = \frac{x - \mu}{\sigma},$$

where z is the standardized value, x is a variable's single datapoint, μ is the variable's mean and σ is the variable's standard deviation.

2.3 Frequency of measurements and data synchronization

All the features of first dataset from two-storey house are measured hourly, so they are continous and have same frequency within the set. The environmental features of second dataset are measured daily, so they are continous and have same frequency within the set. Therefore, both datasets are intrinsically continuously measured with same frequencies, but the sets need to be sampled into same frequency rate for further data analysis. This can be achieved by calculating the averages of each day from the hourly power usage data, because daily environmental measurements are much harder (and unreliable) to convert into hourly measurements.

Initially, neither of the datasets is in chronological order. So when exploring if the data is synchronized, both datasets are sorted so that measurements' timelines are comparable. After ordering, both datasets are intrinsically synchronous across their own features, but due to different sampling rate the sets are not initially synchronized with each other. To bring the data into the same timesteps, same procedures can be utilized as for frequency questions, so conversion of hourly measured data into daily data.

2.4 Missing values

Both datasets contain significant number of missing values for the years 2016 and 2020, meaning the beginning and end of the sets. The missing values seem to be consecutive mostly in periods of couple of weeks. To ensure data to be continous and synchronized, it's decided to exlude all the data measurements from those years for later modeling purposes.

The years 2017 - 2019 does not contain any missing values, thus the data is fully continous during those years and there is no need to fill any missing values.

2.5 Trend and seasonality analysis

The power usage data set from the years 2017-2019 is decomposed into seasonal components and a long term component. In the figure 4 there are two seasonal components visible. The seasonal component with very high frequency, is the one which represents the daily variety in the power usage. On the figure 5 it is easier to see the daily variation of the power usage. The values get lower during

the nights and higher during the day time. The other seasonal component represents the biannual variation between the colder and warmer seasons of the year. This indicates that the time of the day, and as well as the time of the year has high effect on the power usages value.

In the figure 6, it is seen that there is a slight upwards trend in the average values of the power usages. All of the seasonality was not able to be removed from the long term component, thus the long term component also takes into account biannual seasonal changes between summer and winter. In the figure 7 is the remainder after decomposing the data. There are no clear trends in the figure left, and the remainder seems to be mostly random noise that is left after decomposition.

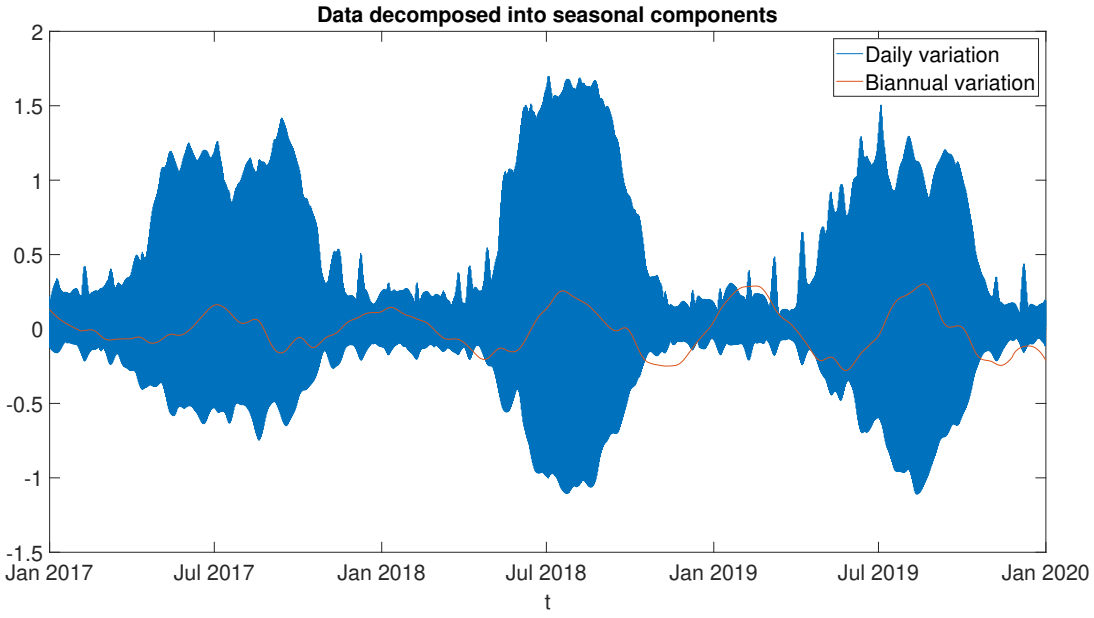

Figure 4. The seasonal components of the power usage data.

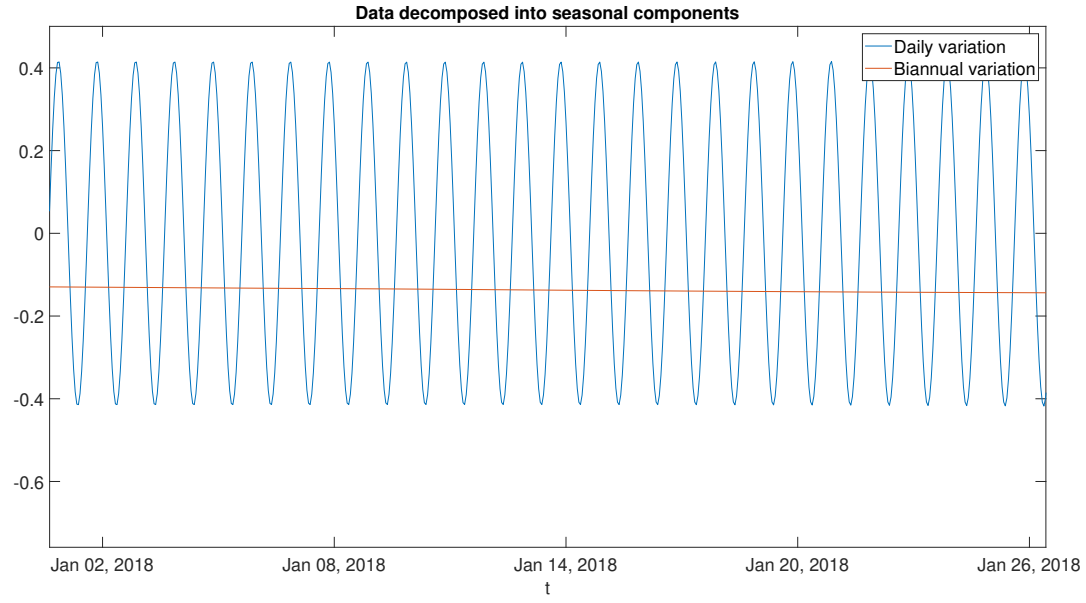

Figure 5. The seasonal components of the power usage data zoomed in to January of 2018.

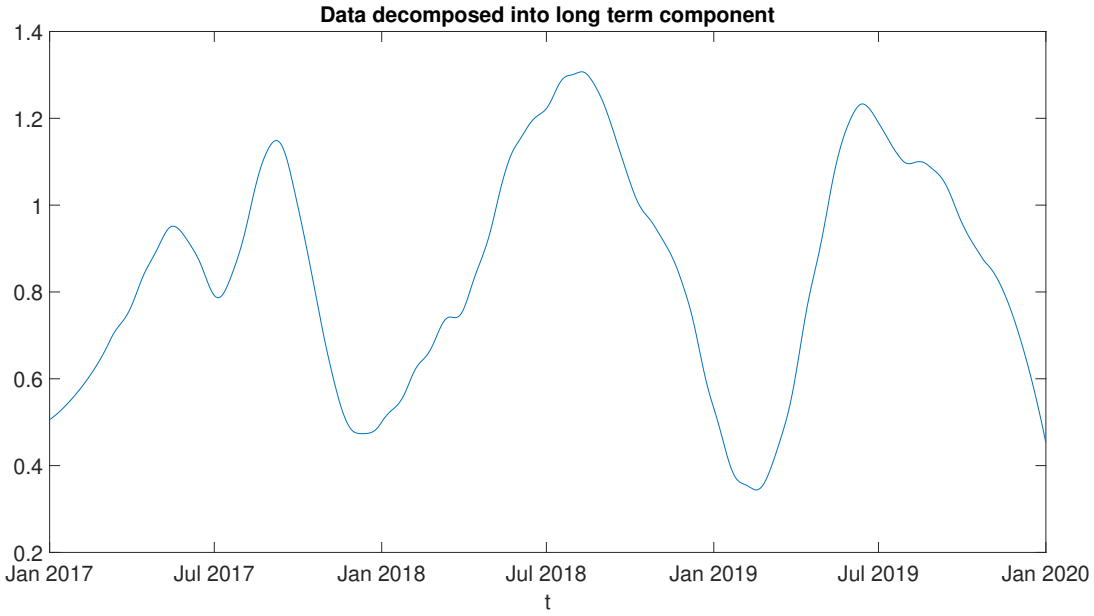

Figure 6. The long term component of the power usage data.

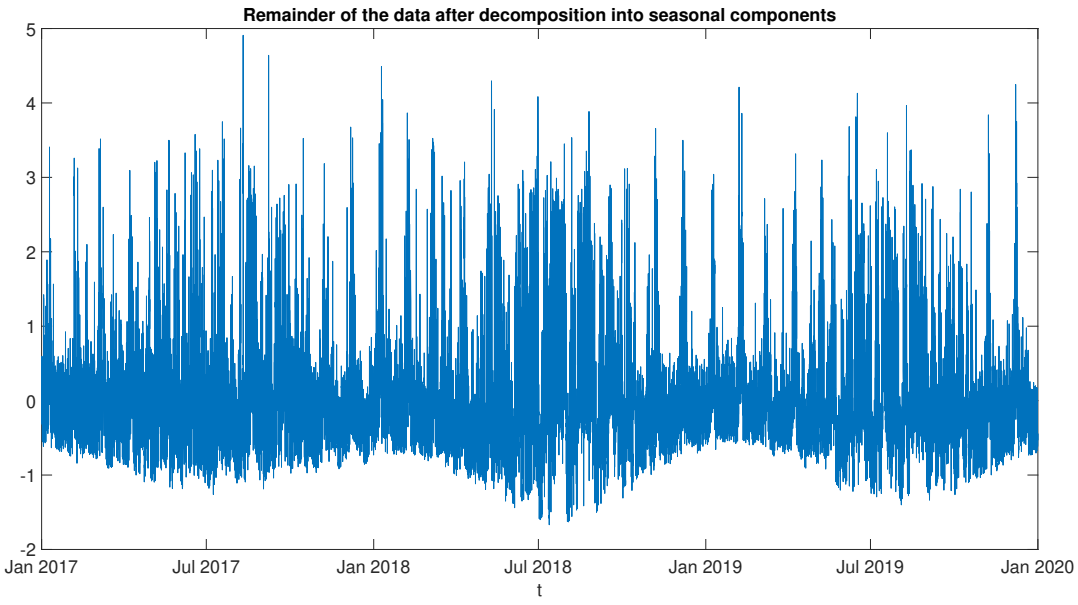

Figure 7. The remainder of the decomposition into seasonal components.

2.6 Outlier detection

STL decomposition (seasonal and trend decomposition using Loess) can be used to spot and eliminate possible outliers. From the remainder plot the possible outliers can be detected as very high or very low spikes when compared to the other data. With this method possible outliers found are presented in figure 8. The outlier detection was done on the remainder of the data after decomposition, and a sliding window method with window size of one month was used. In the sliding window if the value differed enough from the window's mean value, the value was classified to be outlier.

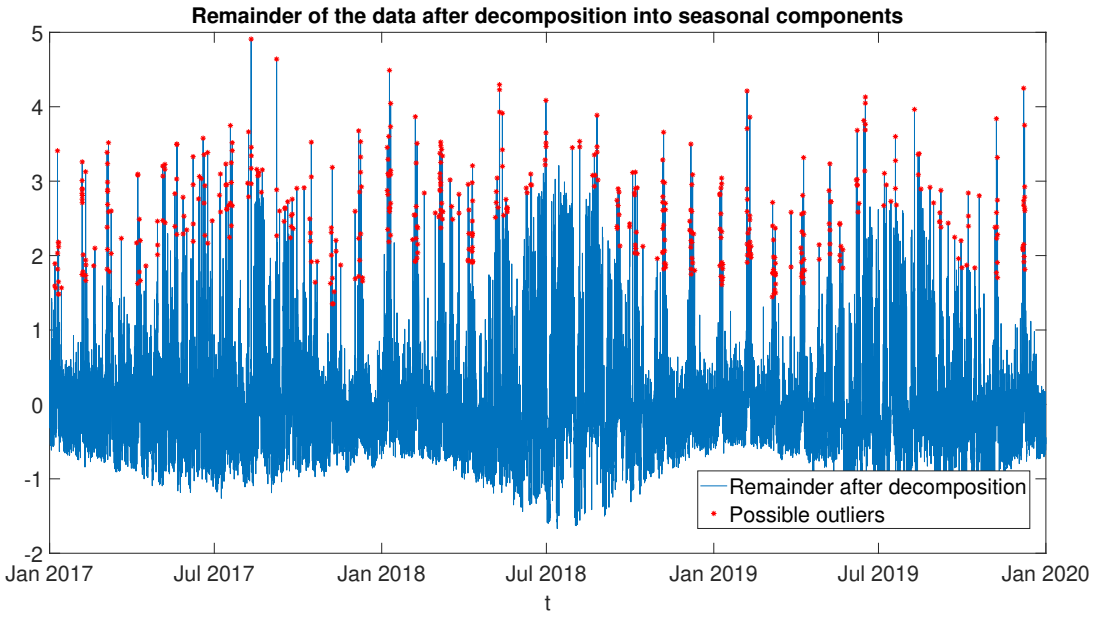

Figure 8. The remainder of the decomposition into seasonal components with possible outliers.

In the figure 9 is the results of outlier detection using standard deviations times 2 and 3 as control lines. However since the remainder of the decomposition still contains such a large variation, using for example 2 STD:s as the control line would classify large amount of data as outliers. Even using 3 standard deviations would leave a significant amount of data outside the control lines, so this method of outlier detection might not be feasible here, or it would be needed to consider using even more standard deviations.

As a result of outlier detection, the samples where the power usage is much larger than on average are classified to be outliers (using STL decomposition). Further investigation is needed however to identify which of the possible outliers are really significant, and thus should be considered while creating a predictive model. This time, it's decided not to remove any outliers, because there's not exact knowledge what kind of power consumption behavior in reality can be concerned as outlier.

2.7 Correlation analysis

The correlations between environmental averaged features and power consumption are explored and presented in the figure 10, Power being the variable to estimate and the others being variables used in estimation. It can be seen that average temperature and average dew have pretty high correlation with Power (0.72 and 0.64), so they're good alternatives to use as explanatory variables in later modeling.

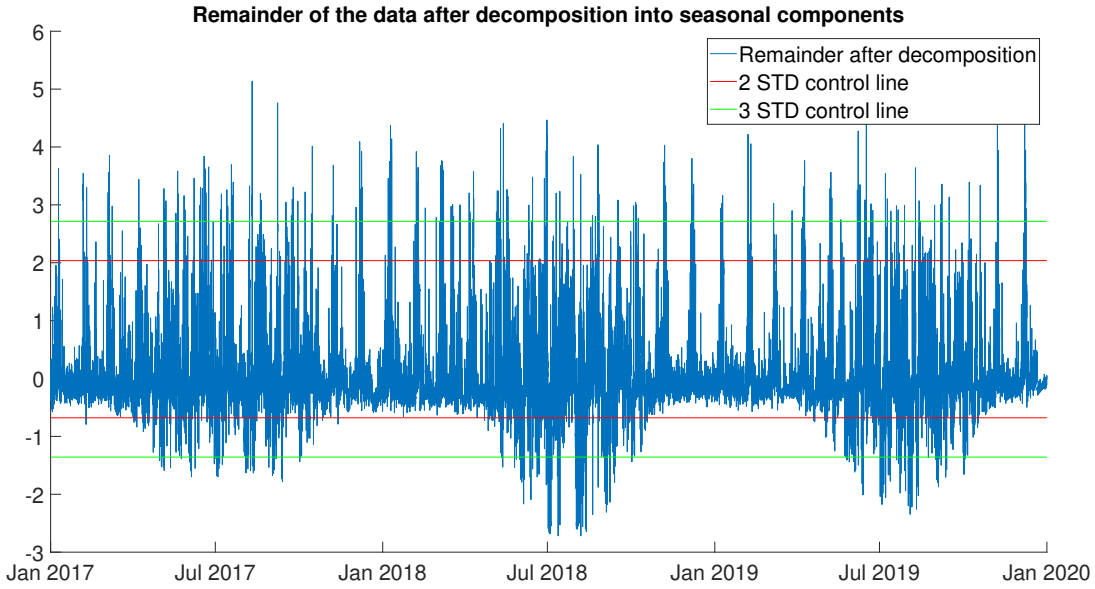

Figure 9. The remainder of the decomposition into seasonal components with control lines.

Some correlation to Power can also be seen with average wind speed and average air pressure, but they're only -0.16 and -0.30. Correlations between Power and Humidity, Precipitation and Weekday are so low that they can straightaway be dropped off from the model because they don't explain almost anything of power consumption.

When comparing temperature, dew, wind speed and air pressure between each other, it's notable that temperature and dew have very high correlation (0.93), and air pressure have also some correlation with temperature and dew (-0.62 and -0.65). These correlations between explanatory variables could to be taken in account before they're used in modeling, because using all of them might not be necessary.

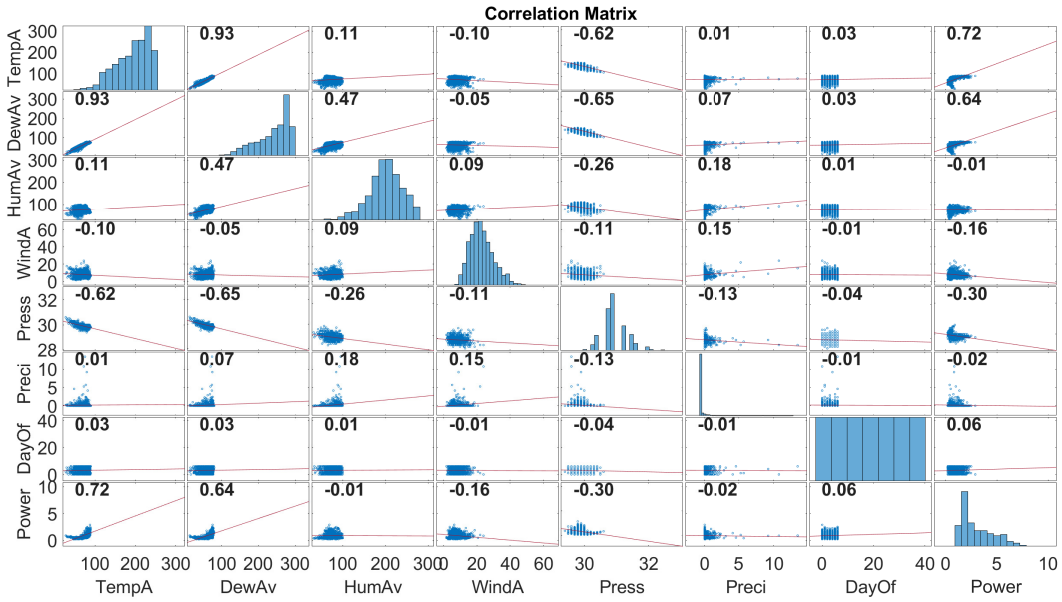

Figure 10. The correlation matrix between the explanatory variables and the calculated daily average power usage.

3 Models and model architectures

3.1 Recurrent neural network

A recurrent neural network (RNN) is a deep learning neural network, which uses past data to improve both the current and future inputs. In all its simplicity, the layer graph of the RNN can be seen in figure 11. Essentially the RNN has two weights to update, one for the inputs and one for the hidden states. The amount of hidden states will initially be one, but depending on the models performance, there is need to consider adding more hidden layers or some other layers for example dropout layers. Since the RNN architecture is also part of the LSTM architecture, it is explained in more detail in LSTM architecture part.

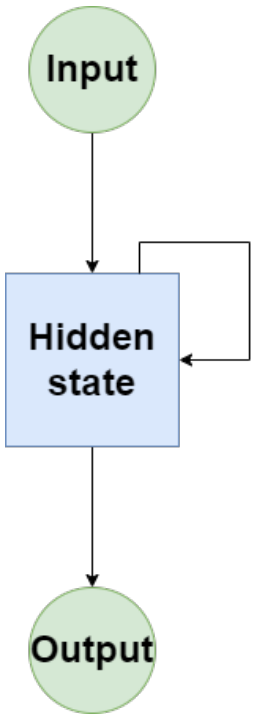

Figure 11. Recurrent neural network layer graph.

3.2 Long short-term memory neural network

Long short-term memory (LSTM) is a neural network architecture that is able to maintain memory of past information over long periods. For that, the network has special memory cells that can store, read, and erase information, and these cells are controlled by gates that regulate the information flow. The network is designed especially for sequential and timeseries data, due to cabability of remembering patterns over time. LSTM is also a certain type of RNN.

The model's architecture is described with a layer graph in figure 12. The LSTM gates receive two inputs: X_t from the current time step and hidden state H_{t-1} from the previous time step. LSTM consists three fully-connected layers that are used to implement gates: input gate I_t, forget gate F_t, and output gate O_t. The input gate determines which information from the current input should be stored to the current memory cell, the forget gate decides whether the information from the memory cell should be kept or discarded, and the output gate determines how the memory cell should have affect on the output. The gates' values are calculated as follows:

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i),$$
$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$

and

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_f),$$

where X is data samples, W weight matrices, H_{t-1} hidden state of previous time step and b the biases.

In addition to layers, there's input node $\tilde{C}_t$, which computes the input value from the current input and from the last time step's hidden state as

$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c),$$

where W_{xc} and W_{hc} are weights, and b_c is the bias. A set of multiplicative gates are used to determine the inputs to affect the memory state and when content of the memory state should influence the model output.

The internal state of the whole net takes in account both inputs (current input and hidden state) and the memory cell from the previous internal state C_{t-1}. This can be calculated by element-wise summarization and product as follows:

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t,$$

where F_t is the forget gate value, C_{t-1} the previous internal state, I_t the current input, and $\tilde{C}_t$ the input value. This internal state is then saved for the next iteration.

Finally the output from the net is calculated as element-wise product of the current internal state and the output gate value as follows:

$$H_t = O_t \odot \tanh(C_t),$$

where O_t is the value of output gate and the C_t is the current internal state. This output is saved as hidden state for the next iteration.

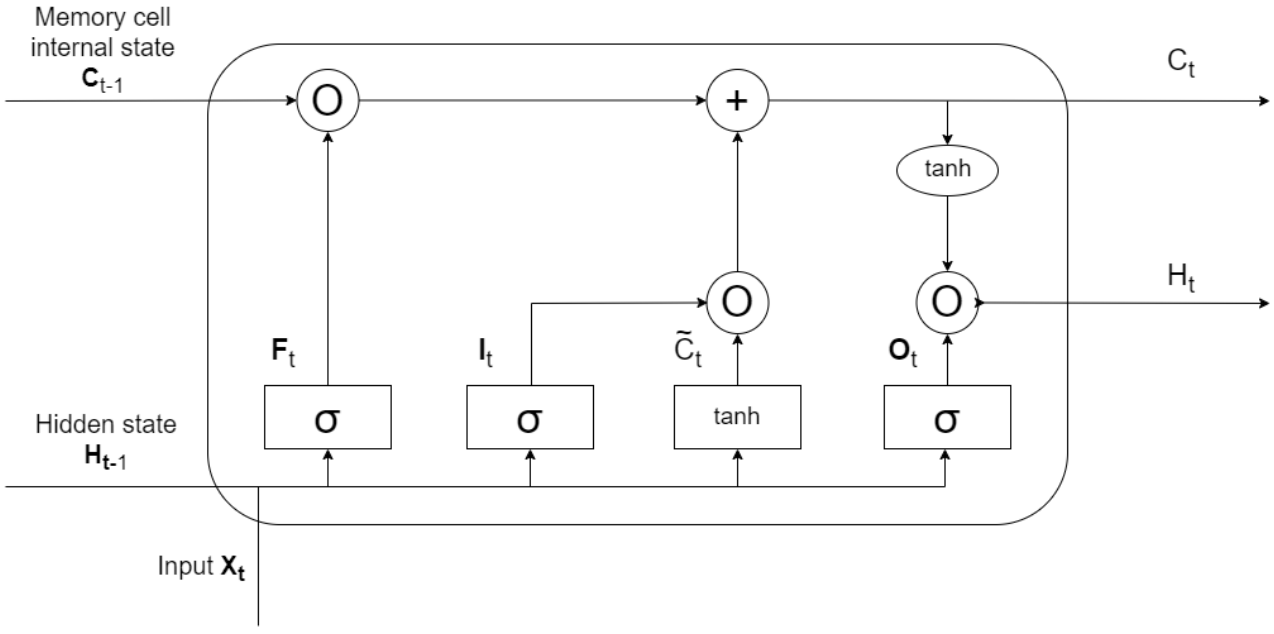

Figure 12. Long short-term memory network.

3.3 Transformer neural network

Transformer neural network (TNN) follows an Encoder-decoder architecture. A simplified figure of the architecture can be seen on the figure 13, where the encoder takes as input variable-length sequence. The decoder takes as input the encoded input and the target sequences state, and as output gives the predicted element/token which is next in the target sequence.

Figure 13. Simplified encoder-decoder architecture.

The chosen transformer architecture is similar to the one architecture that was introduced in the example codes of the course, which will result in the following layer graph in the figure 14. The hyperparameter optimization will focus on the sequence length of the sequence input layer and on the number of attention heads of the self attention layer in addition to the other hyperparameters.

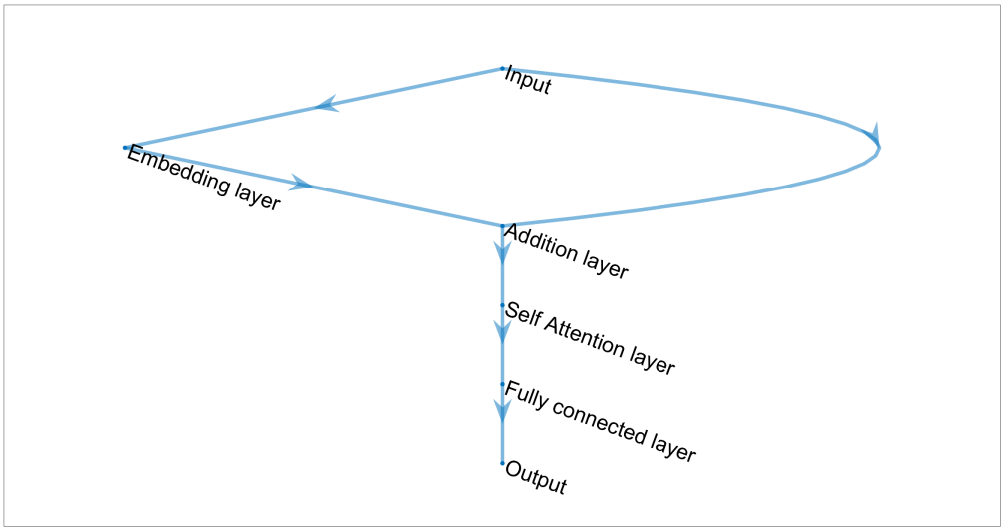

Figure 14. Initial layer graph for tranformer neural network.

The embedding layer handles the mapping of sequential indices to vectors, and encodes the information about the data positions. The addition layer combines the embedding layers and input layers element-wise. The self attention layer handles the multihead self-attentition of the inputs. And the fully connected layer combines the process by multiplying the input by the weight matrix, and adds the bias vector there.

3.4 Ablation study and model performance evaluation

While training all of the models, it's a good and usually necessary manner to try to make them perform as well as possible. This is achieved by varying the models' parameters during the training as well

as choosing suitable metrics for calculating the models' correctness both with training data and when predicting with the testing set.

The optimization based on the training process will be done by using sensitivity analysis. Depending on which model is in considered, the sensitivity analysis will focus on the following hyperparameters: learning rate, number of batches and the types of layers used in the neural network. A good learning rate is important so the training will be stable and efficient, meaning that the training will not neither miss the minima nor get stuck in any local minima and will be as efficient as it can. The number of batches refers to how many training samples are used in single iteration, and there is a balance which needs to be found. While smaller batch can be more efficient to compute a larger batch size could converge faster.

The ablation study on layer decisions will depend a lot on the network, but for example different number of hidden layers should be tried out and there is need to consider of adding different kinds of layers if needed. Few examples to reduce the risk of overfitting would be to consider adding dropout or regularization layers. For the transformer NN specifically there are attention heads and positional encodings as hyperparameters.

To evaluate how the models actually perform on their predictive task, suitable metrics to describe how good predictions are when compared to the actual values can be computed. For this task, R-squared and root mean squared error are chosen, and additionally models' accuracy and validation loss during training are visually interpreted from the graphs.

Root mean squared error (RMSE) is used to measure the quality of models predictions. It tells the square root of the average of the squares of the errors, and its equation is

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2},$$

where n is the number of datapoints predicted, Y_i is the actual power usage, and $\hat{Y}_i$ is the model's predicted values for the power usage..

The measure for goodness of fit for the model is R^2 value, and its equation is

$$R^2 = 1 - \frac{\Sigma(y_i - y_{i,est})^2}{\Sigma(y_i - \bar{y})^2},$$

where y_i is the actual power usage's value, $y_{i,est}$ is the estimated value, and $\bar{y}$ is the mean value of the power usage.

Then, it's explored how the models perform when they try to predict only one day ahead versus larger time window ahead. The models are chosen to be optimized with only one day ahead prediction, and after optimal ones are found, it's explored how those models with same parameters perform with larger time periods. This is done by retraining the model multiple times while shifting the training data to always be one day further in to the past on each iteration.

Additionally, when it's case of large dataset and neural networks, training and prediction processes can take some time and computational limitations might be reasonable to take care of. With this in mind, it's also measured how long it takes to train model. When selecting the best models and best parameters, both models' accuracy and training times need to be considered.

4 Results and discussion

All models (RNN, LSTM and TNN) are chosen to be trained with previous day's average power usage, and enviromental variables average temperature, pressure and dew. Datasets are divided into training, validation and testing sets. The data is divided into training and testing sets with relation 80:20. During training processes, the models use crossvalidation and divides the training set into training and validation sets.

4.1 RNN model

MATLAB's built-in neural network layrecnet allowed to change as hyperparameters the number of hidden layers and the number of delays for the layers, as well as the training function for the back-propagation. Additionally, the network's learning rate is varied.

4.1.1 RNN model optimization

The RNN-model is optimized by varying different hyperparameters, such as learning rate, number of hidden layers, layer delays and the backpropagation's training function. The varied parameters, and resulting models' root mean square errors (RMSEs), R^2-values and training times are presented in the table 3. As it can be seen, the variation of these parameters doesn't really have a significant effect on to the sizes of RMSEs (which is anyway pretty small) or R-squared (which is anyway pretty large). The models' training times had some variation, and some training processes were run until the epoch limit. Based on these result, the most optimal model would have the learning rate of 0.0005, 1 hidden layer, layer delays of 5-9 and training function to be gradient descent, because the RMSE is smallest, R^2 is highest, and the training time is good. However, the differences are very small in general, so it's hard to determine at this point which kind of model would really be the most optimal one. Additionally, the sizes of RMSE and R^2 depended a lot on the current run time.

LR	HL	LD	TF	**RMSE**	R^2	**Training time [s]**
0.01	1	5-9	Gradient descent	0.39	0.88	45
0.001	1	5-9	Gradient descent	0.44	0.85	104
0.0005	1	5-9	Gradient descent	0.37	0.89	20
0.0005	2	5-9	Gradient descent	0.39	0.88	51
0.0005	3	5-9	Gradient descent	0.38	0.89	768
0.0005	1	3-9	Gradient descent	0.55	0.75	1
0.0005	1	5-13	Gradient descent	0.38	0.88	7
0.0005	1	5-9	Levenberg-Marquardt	0.41	0.87	2
0.0005	1	5-9	Bayesian regularization	0.44	0.84	55

Table 3. RNN model optimization with hyperparamaters learning rate (LR), number of hidden layers (HL), layer delays (DL), and used training function (TF).

4.1.2 RNN model results

RNN model's results can be seen on the figure 15, where the final RNN model was trained using 5 to 9 input delays and one hidden layer with size of 10. The training function for backpropagation was gradient descent. The root mean square error for the predictions is 0.37 and the R^2 value is 0.89, which indicates that the model is performing promisingly.

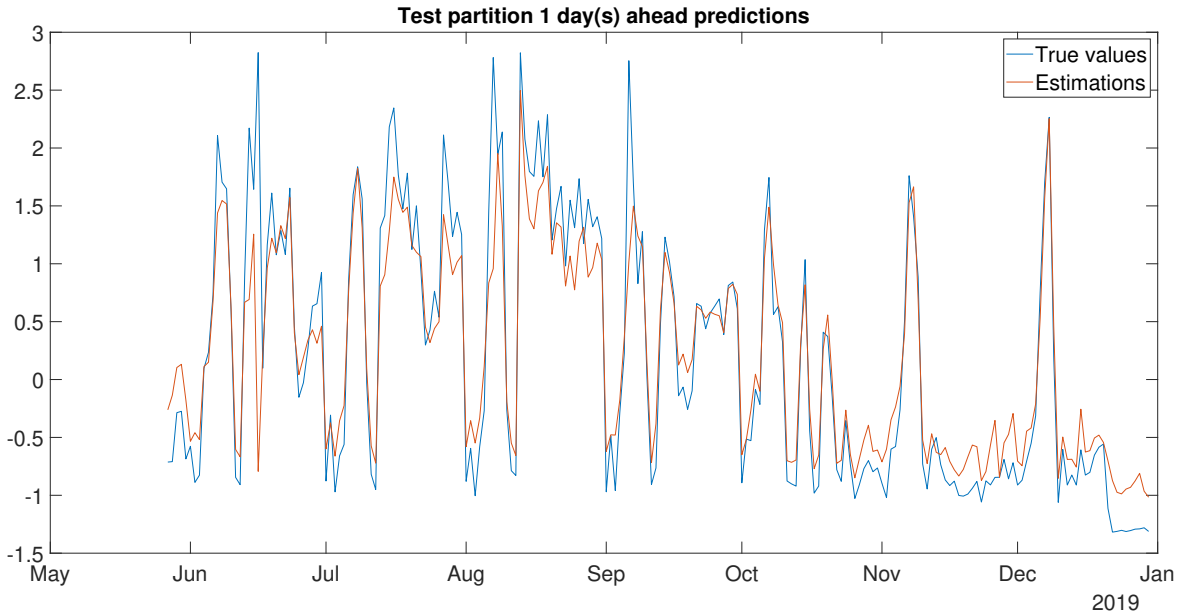

Figure 15. Predictions made with RNN model compared to the real values for the last half of the year 2019.

In the figure 15 it can be seen that the model generally performs very well when predicting the next days' power usages, with the general direction of the values being correct. The most definite errors for the models come from when the true values are unusually high or low (the largest peaks). In these cases, the model predicts the values to be a lot less than the real values. The cause for the bad performance when predicting these large peaks could be that the model is missing some significant variable which could explain the larger peaks, or simply that the peaks are outliers.

The model's forecasting power is investigated in the figures 16 and 17. In the figure 16 the RMSE values almost doubles when trying to predict two days ahead, and triples when trying to predict three days head. However the RMSE values do get more stable after 5 days ahead and further predictions, and stays around RMSE of 1.1. The reason for why the RMSE values does not increase further can be seen on the figure 176. The figures for 5 days and further ahead starts to converge more on around the middle of the data. This of course means that in general the RMSE values will also be around the same size, since the distances between predicted values and the ground truth remains roughly the same. As a result the model seems to perfom well only when predicting 1 day ahead, while on some cases it could also be used to predict two days ahead.

There is also interesting phenomena of the RMSE values slightly decreasing when predicting around 30 days ahead. When predicting 28 - 32 days ahead the model seems to perform slightly better compared to other days over 5. This could indicate that the model is able to find some monthly seasonality between two consecutive months power usage values.

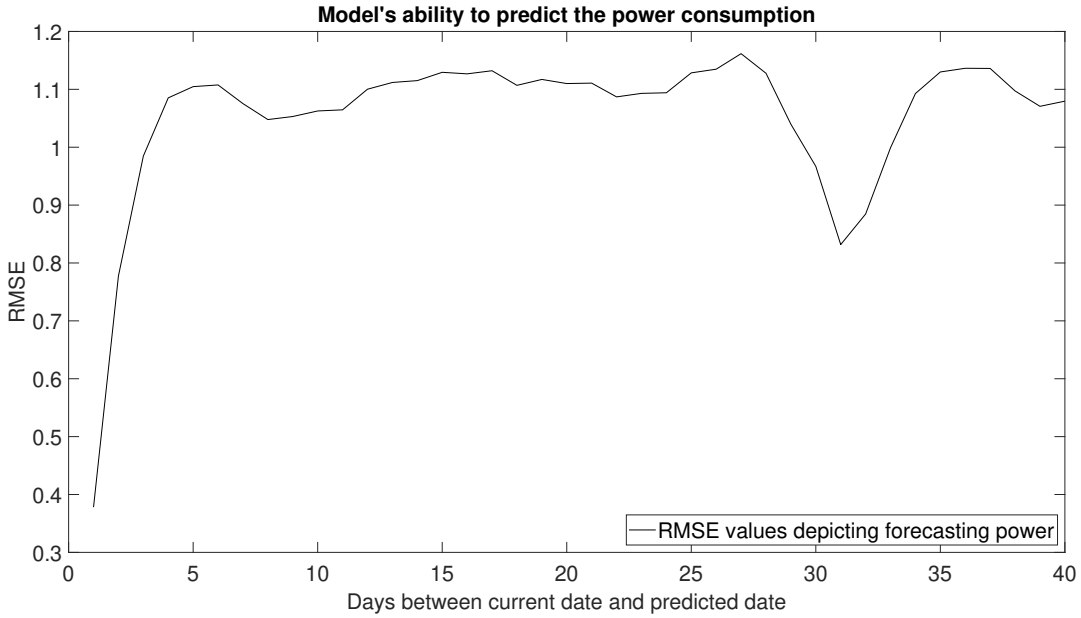

Figure 16. RNN model's forecasting power investigated with RMSE value compared to the days between current date and predicted date.

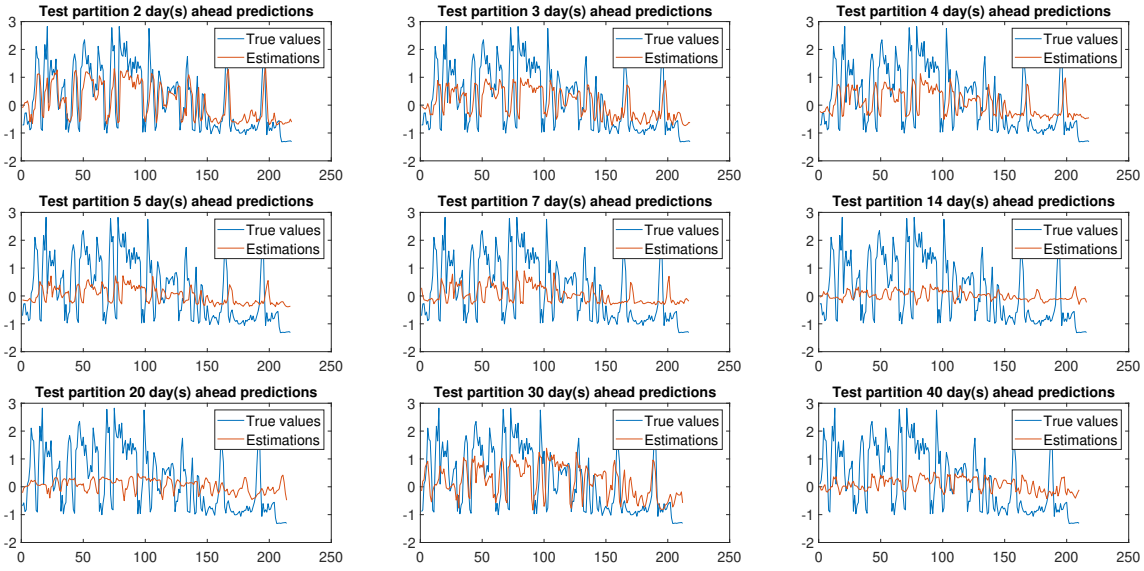

Figure 17. RNN model's forecasting power investigated with different number of days ahead predictions.

4.2 LSTM model

4.2.1 LSTM model optimization

In the table 4 are the results of the hyperparameter optimization for the LSTM model. To compare between different hyperparameters, the ones with the lowest and reasonable training time can be assumed to be the best. However since all of the training times were reasonably low, the most weight was given to the RMSE values.

It seems that increasing the amount of LSTM layers increases the error of the predictions, thus it was decided to keep to only have one LSTM layer. Lower learning rate did decrease the error and increase the R2 value, thus learning rate of 0.0005 was selected as optimal. When optimizing batch size, the batch size of 32 gave the best results. With learning rate of 0.0005, batch size of 32 and only 1 LSTM layer, the sequence length between longest and shortest did not differ much, so we decided to use the longest sequence length.

Overall it seems that the Batch Size and Sequence length did not have that much of an effect on the RMSE values, while changing the learning rate and no. of LSTM layers did have greater effect. Thus the lowest RMSE value was achieved using learning rate of 0.0005, 1 LSTM layer, 32 as batch size and sequence length as ”Longest”, and they were used as the optimal hyperparameters.

LR	LL	BS	SL	**RMSE**	R^2	**Training time [s]**
0.05	1	128	Longest	0.82	0.46	8
0.01	1	128	Longest	1.6	-1.1	5
0.001	1	128	Longest	0.71	0.59	5
0.0005	1	128	Longest	0.55	0.75	5
0.0005	2	128	Longest	0.74	0.56	8
0.0005	3	128	Longest	0.94	0.29	12
0.0005	1	64	Longest	0.54	0.76	5
0.0005	1	32	Longest	0.51	0.79	5
0.0005	1	32	Shortest	0.51	0.79	5

Table 4. LSTM model optimization with hyperparamaters initial learning rate (LR), number of LSTM layers (LL), batch size (BS) and sequence length (SL).

4.2.2 LSTM model results

LSTM model's results can be seen on the figure 18, where the final LSTM model was trained using the chosen optimal parameters. The root mean square error for the predictions is 0.51 and the R^2 value is 0.79, which indicates that the model is performing promisingly.

The actual power usage is shown in blue, and the predicted power usage is shown in orange. The predicted power usage follows the actual power usage mostly well, but is not able to fully follow the directions the ground truth goes in to. There are also discrepancies between the two especially for the larger peaks and valleys.

Overall, the LSTM model is able to predict the general trend of the power usage with reasonable accuracy. However, it tends to underestimate the magnitude of the sharp peaks in power usage.

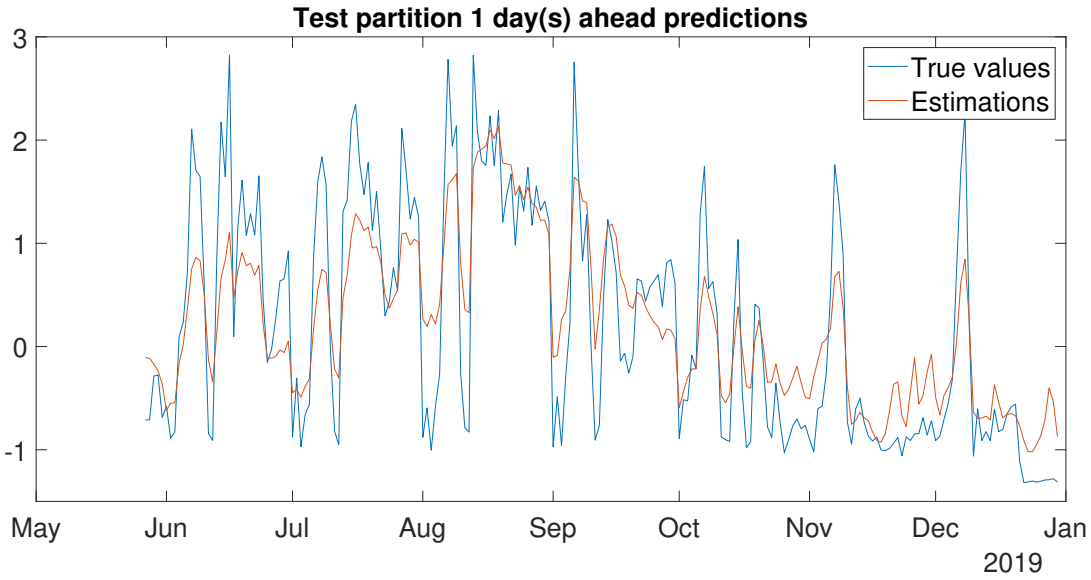

Figure 18. Predictions made with LSTM model compared to the real values for the last half of the year 2019.

This could be due to a number of factors, for example it is possible that the sharp peaks in power usage in the test partition are due to factors that were not present in the historical data. It is possible that the model is not able to learn to predict the sharp peaks in power usage because it is overfitting the training data.

To improve the accuracy of the model, it could help to train the model on more data, including data with sharp peaks in power usage. Experiment with different model architectures and hyperparameters could help to find a model that is able to predict both the general trend of the power usage and the sharp peaks accurately.

The model's forecasting power is investigated in the figures 19 and 20. In the figure 19 the RMSE values slightly increases when trying to predict two days ahead, but increases greatly when trying to predict three days head or further into future. However the RMSE values seems to be slightly increasing generally, but overall the values behave very unstabily. This could indicate that the LSTM model might not be the best option for this issue, or that the model would need further modifications in the model structure to achieve better results. Overall it seems that LSTM learns the general trends from the data, and is somewhat able to predict on which direction the data is going. But when predicting more days to the future, it seems the LSTM model is always trying to predict something since it is not converging, even if that something is completely wrong. The same phenomena of RMSE values slightly decreasing on around 30 days ahead predictions happen on the LSTM model.

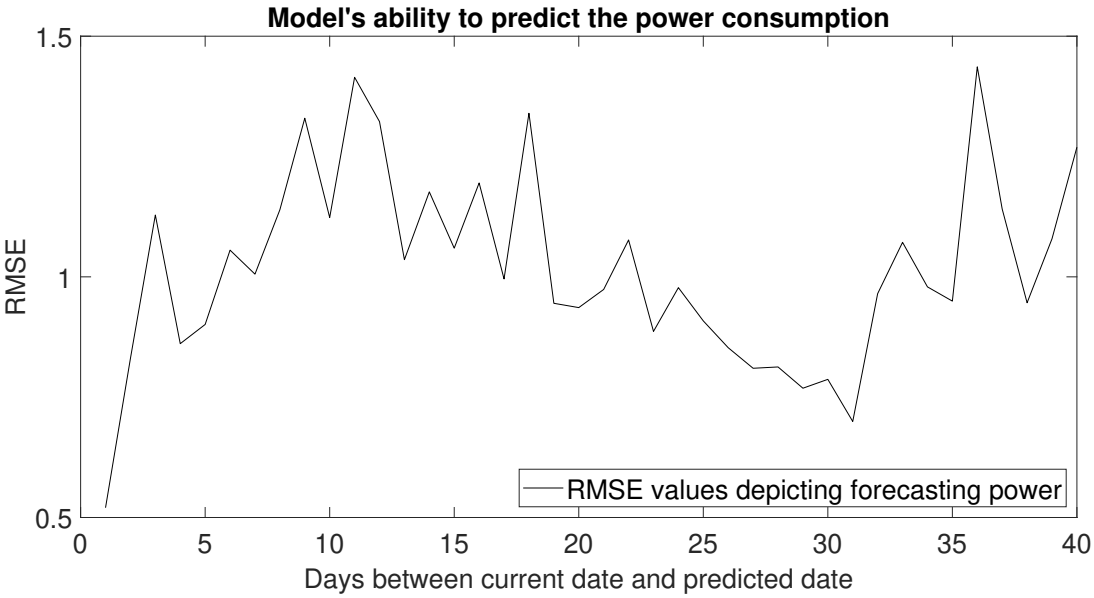

Figure 19. LSTM model's forecasting power investigated with RMSE value compared to the days between current date and predicted date.

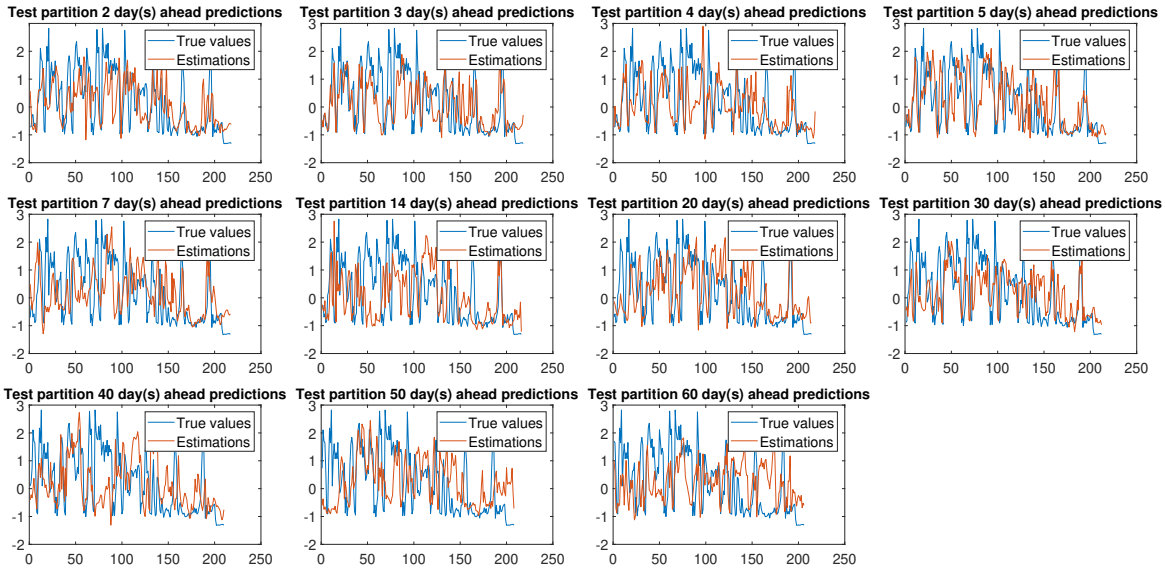

Figure 20. LSTM model's forecasting power investigated with different amount of days ahead predictions.

4.3 TNN model

4.3.1 TNN model optimization

The optimization of transformer neural network is committed by varying the model's hyperparameters initial learning rate, mini batch size, sequence length and number of heads. The time taken in trainings, the resulting root mean squared errors (RMSE) between true and predicted values and models R^2-values can be seen in table 5.

LR	BS	SL	H	**RMSE**	R^2	**Training time [s]**
0.0005	16	80	8	0.41	0.86	12
0.0001	16	80	8	0.78	0.51	12
0.001	16	80	8	0.30	0.93	11
0.005	16	80	8	0.50	0.80	10
0.05	16	80	8	0.59	0.72	10
0.05	16	80	8	0.60	0.71	12
0.001	8	80	8	0.29	0.93	10
0.001	32	80	8	0.32	0.92	10
0.001	64	80	8	0.33	0.91	10
0.001	128	80	8	0.29	0.93	10
0.001	8	40	8	0.37	0.89	18
0.001	8	20	8	0.54	0.76	28
0.001	8	100	8	0.36	0.89	28
0.001	8	80	4	0.34	0.91	10
0.001	8	80	16	0.31	0.92	12
0.001	8	80	2	0.48	0.82	9

Table 5. TNN model optimization with hyperparamaters initial learning rate (LR), batch size (BS), sequence length (SL) and number of heads (H).

Based on the results presented in the table, most of varied parameters didn't make any remarkable changes to the model performance, since most of the training times and calculated RMSEs are kind of similar with each other. Perhaps it's most notable that too small initial learning rate and sequence length make the RMSE increase a lot, and also with small sequence length the training time increases significantly.

Based on this hyperparameter optimization, the most optimal model would be with initial learning rate of 0.001, batch size of 8, sequence length of 80 and number of attetion heads being 8.

4.3.2 TNN model results

The model's predictions against true values are presented in figure 21. By visually exploring the results, it can be seen that most of the time the model detects quite well the changes and even some spikes of power consumption, even though the magnitude is not always quite correct. Overall the model's predictions follows the ground truth very closely, with the most difficulties being in the largest spikes and on the more stable areas. The resulting model gives RMSE of 0.29 and R^2 of 0.93.

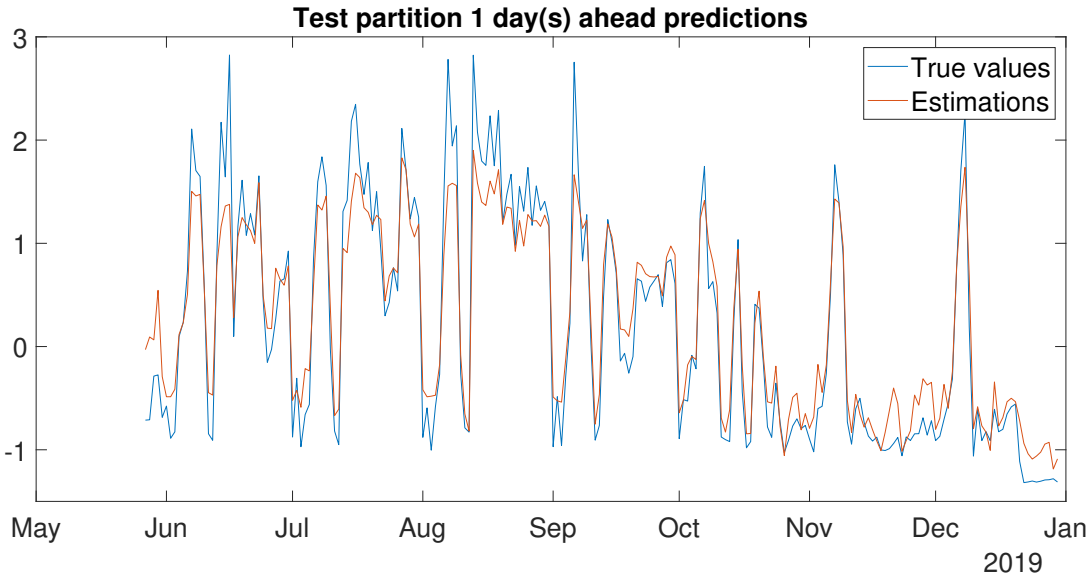

Figure 21. Predictions made with optimizecd transformer NN model compared to the real values for the last half of the year 2019

In addition to predicting power consumption only one day ahead, the model is also tested how well it could estimate the consumption with larger time windows between current date and estimated date. The power consumption is estimated 1-40 days to the future, and the development of RMSEs can be seen in the figure 22. RMSE is increasing sharply during first four days from 0.3 to 1.1, but after that it balances and stays there for the rest of the tested timeline.

The reason for this can be seen when plotting the estimated values against true values in figure 23. When predicting 2-4 days ahead and comparing them to the figure 21 with only one day ahead predictions, the prediction graphs are flattening a lot during this prediction time window. After 4-5 days between current and predicted dates, all the resulting estimation graphs look pretty similar to each other and they are balancing close to value 0.5. So to say, the prediction is somewhat succesfull still with 2 and 3 days prediction window to the future, but already 4 days (and further days after that) are clearly too hard to predict reliably, at least with this model. The only exception again is around the 30th day, where the model seems to gain the increasement of model performance.

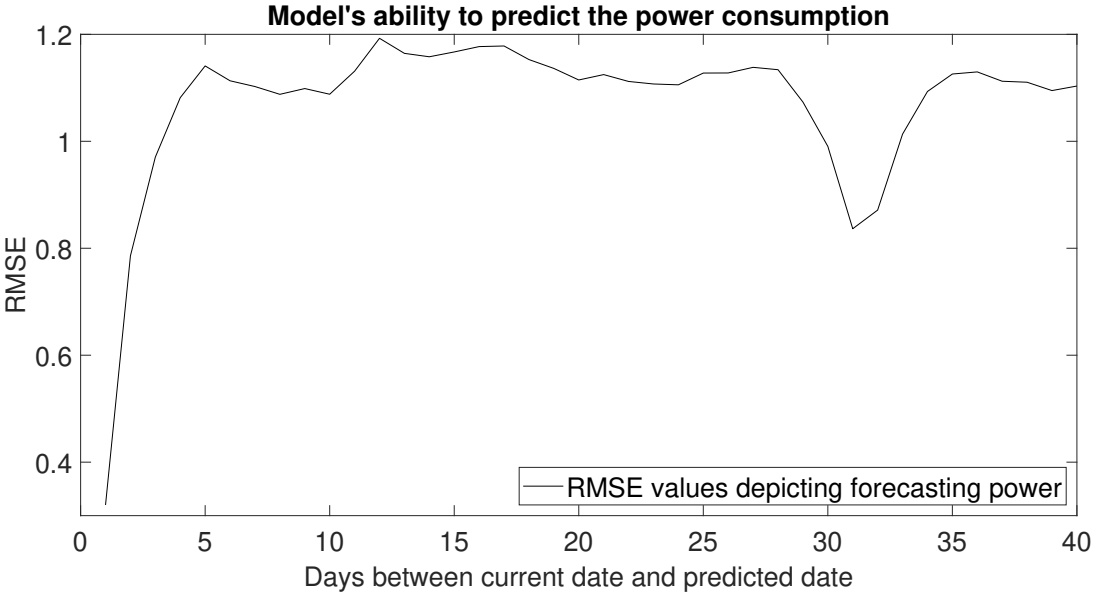

Figure 22. Investigating the forecast power of the model by comparing the RSME values with regards to the number of days predicted into the future.

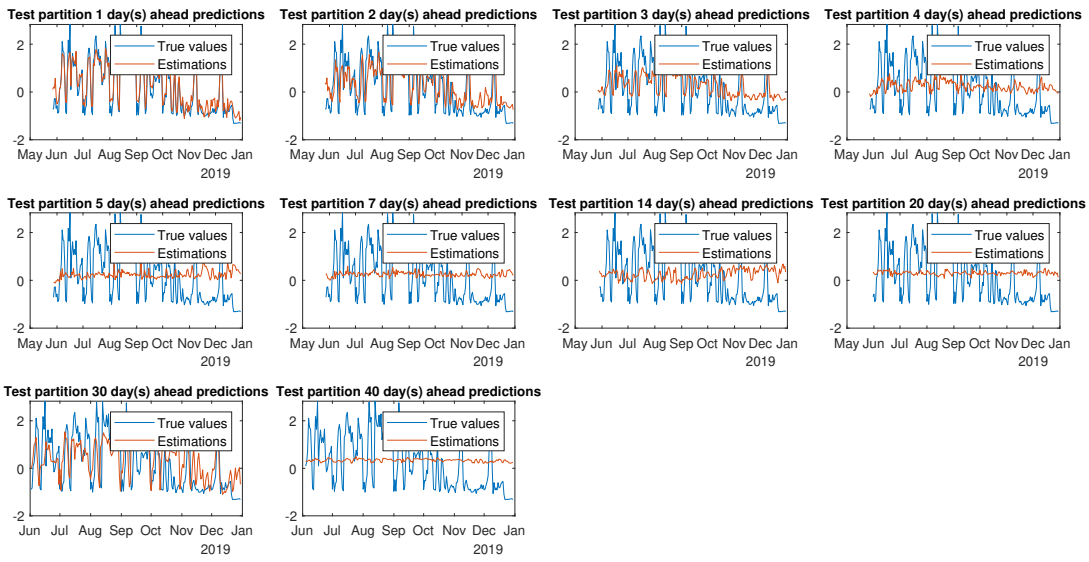

Figure 23. Comparing the power consumption estimation results.

4.4 Model comparison

Overall all of the models performed well but they all had also troubles when predicting the largest spikes of power usages. There are also many cases where the models predicts small peaks where the actual values are more stable. One thing to take note of is that the months when the model predicts the larger peaks incorrectly are summer months, while during the winter months the model predicts small peaks when the power usage values seems to be more stable.

Whether the large peaks in the data are outliers or not are up to debate, since they could as well as be just normal behaviour for the summer months. Especially during the summer months power usage might greatly increase with regards on how ofter air conditioning is used. The models also might be missing some other variable which could explain the discrepancies during the largest mispredictions. Since the power usage data seems to behave differently between the summer months and winter months, having to separate models for the seasons also could help in improving the overall performance. Then also the outliers could be investigated more and possibly removed.

Current model uses only the fully continous data between the years 2017-2019, since the years 2016 and 2020 had significant amount of missing values. However since the model needs to predict only the power usage based on the previous day's data, there might not be a need to exclude those two years. The excluded two years contains around 500 samples and including them would increase the training data from around 1000 samples to 1500 samples. Increasing the amount of data available could increase the performance of the model as well.

Something extra that was noticed was that all of the different models had slight improvement in their performance when predicting around 30 days ahead. This could indicate that the models could have some predictive power when trying to predict power usage values one month ahead, and this could be utilized in further research.

Additionally when thinking of how to improve model, the selected explanatory variables from environmental dataset could be investigated more and be decided differently due to multicollinearity. Perhaps only previous day's power consumption and temperature would have been enough to predict power consumption reliably.

5 Conclusions

The power consumption of house was estimated with previous days' power consumption and environmental data (temperature, dew and air pressure) with RNN, LSTM, and TNN models. When comparing the models performance, it is clear that overall the transformer neural networks has the best performance out of the three options with the R^2 value of 0.29 and RMSE of 0.93. However the RNN model was not far behind in the performance, with some iterations of the models being better than the worst iterations of TNN models. The worst performing model was clearly the LSTM model, even though it had decent performance as well.

Surprisingly the LSTM model was the model which took the least amount of time to train, while the RNN was the one which had the highest training times. However since the training times was reasonably low for all of the methods, the training time was not given much weight when deciding the best models.