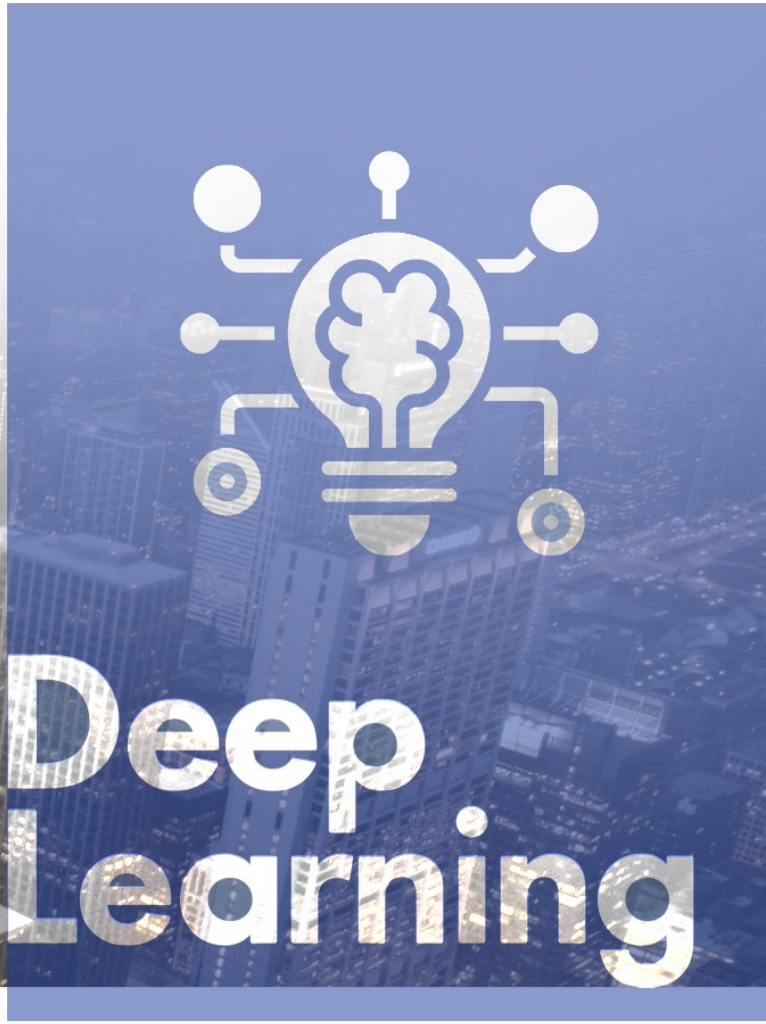
Frankfurt School
of Finance & Management



Prediction of Electricity Contract Price

With Deep Learning and Machine Learning Models

Qianjing Liang (8434350), Daniel Barbie (4939697), Jacob Umland (8436406), Fan Jia (8436217), Jan Faulstich (8439328)

1 Motivation

Challenge

- Multivariate-windowed time-series regression with single-step forecasting
- Multiple parallel & varying-length underlying time series from different data-generating entities (i.e. contracts)
- Windowed time series in tabular structure with other features attached

Approach

- Start simple – dummy & linear models as baselines
- Utilize classic machine learning methods
- NNs approaches and advanced concepts
- On a per-model basis, experimentation with hyperparameters
- Draw conclusions & identify explanations

Findings

- Surprisingly, the dummy regressor was hard to beat
- Simple & complex models seemed to be stuck at the same MAE plateau
- Boosted trees performed the best disregarding “time-seriesness” of the problem
- Poor generalization across models (due to dataset shifts?)

2 EDA

<1> Understand the Data:

The data is about electricity contract trades. There are 2 sets of data with window size 5 and 15 for the first difference of the trading information. Both datasets include contract-wise features as shown below on the left and trading-related features on the right.

- **total_hours**: hours from the first trade we know for a given contract until delivery start of the same contract
- **dlvry_weekend**: whether the electricity is to be delivered on a weekend (binary)
- **dlvry_bank_holiday**: whether the electricity is to be delivered on a bank holiday (binary)
- **dlvry_day_sin** & **dlvry_day_cos**: sine/cosine-transformed day of delivery
- **dlvry_weekday_sin** & **dlvry_weekday_cos**: sine/cosine-transformed weekday of delivery
- **dlvry_hour_sin** & **dlvry_hour_cos**: sine/cosine-transformed hour of delivery
- **lasttrade_weekend**: whether most recent trading hour was on a weekend (binary)
- **lasttrade_bank_holiday**: whether most recent trading hour was on a holiday (binary)
- **lasttrade_day_sin** & **lasttrade_day_cos**: sine/cosine-transformed day of most recent trading hour
- **lasttrade_weekday_sin** & **lasttrade_weekday_cos**: sine/cosine-transformed weekday of most recent trading hour
- **lasttrade_hour_sin** & **lasttrade_hour_cos**: sine/cosine-transformed hour of most recent trading hour

The integer-named columns are a series of values obtained at successive time steps, which can be seen as time series. The data set with window size 5 has 24 time series columns (6 features * 4 time steps) and the window size 15 data set has 84 (6 features * 14 time steps). Using the overview below, we made the mapping of the integer-named columns more intuitive.

Open (1st diff)		Col - 0		Col - 6		Col - 12		Col - 18	
High (1st diff)		Col - 1		Col - 7		Col - 13		Col - 19	
Low (1st diff)		Col - 2		Col - 8		Col - 14		Col - 20	
Close (1st diff)		Col - 3		Col - 9		Col - 15		Col - 21	
Volume (1st diff)		Col - 4		Col - 10		Col - 16		Col - 22	
Min left until dlvry starts		Col - 5		Col - 11		Col - 17		Col - 23	
Time Line	T-4		T-3		T-2		T-1		T

for window_size = 5

Open (1st diff)		Col - 0		Col - 6		Col - 12		Col - 18		Col - 24		Col - 30		Col - 36		Col - 42		Col - 48		Col - 54		Col - 60		Col - 66		Col - 72		Col - 78
High (1st diff)		Col - 1		Col - 7		Col - 13		Col - 19		Col - 25		Col - 31		Col - 37		Col - 43		Col - 49		Col - 55		Col - 61		Col - 67		Col - 73		Col - 79
Low (1st diff)		Col - 2		Col - 8		Col - 14		Col - 20		Col - 26		Col - 32		Col - 38		Col - 44		Col - 50		Col - 56		Col - 62		Col - 68		Col - 74		Col - 80
Close (1st diff)		Col - 3		Col - 9		Col - 15		Col - 21		Col - 27		Col - 33		Col - 39		Col - 45		Col - 51		Col - 57		Col - 63		Col - 69		Col - 75		Col - 81
Volume (1st diff)		Col - 4		Col - 10		Col - 16		Col - 22		Col - 28		Col - 34		Col - 40		Col - 46		Col - 52		Col - 58		Col - 64		Col - 70		Col - 76		Col - 82
Min left until dlvry starts		Col - 5		Col - 11		Col - 17		Col - 23		Col - 29		Col - 35		Col - 41		Col - 47		Col - 53		Col - 59		Col - 65		Col - 71		Col - 77		Col - 83
Time Line	T-14		T-13		T-12		T-11		T-10		T-9		T-8		T-7		T-6		T-5		T-4		T-3		T-2		T-1		T

for window_size = 15

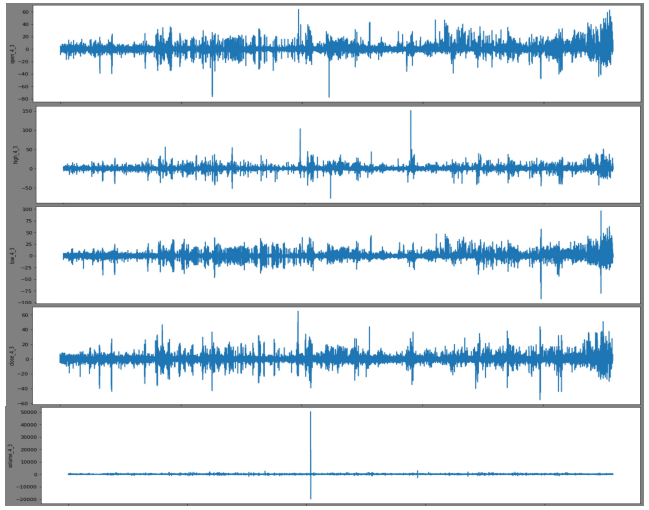
<2> Rename:

Give meaningful names to the columns with integer names, in line with the structure shown in <1>.

```
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   total_hours             91512 non-null  float64
 1   dlvry_weekend           91512 non-null  float64
 2   dlvry_bank_holiday      91512 non-null  float64
 3   dlvry_day_sin           91512 non-null  float64
 4   dlvry_day_cos           91512 non-null  float64
 5   dlvry_weekday_sin       91512 non-null  float64
 6   dlvry_weekday_cos       91512 non-null  float64
 7   dlvry_hour_sin          91512 non-null  float64
 8   dlvry_hour_cos          91512 non-null  float64
 9   lasttrade_weekend       91512 non-null  float64
 10  lasttrade_bank_holiday  91512 non-null  float64
 11  lasttrade_day_sin       91512 non-null  float64
 12  lasttrade_day_cos       91512 non-null  float64
 13  lasttrade_weekday_sin   91512 non-null  float64
 14  lasttrade_weekday_cos   91512 non-null  float64
 15  lasttrade_hour_sin      91512 non-null  float64
 16  lasttrade_hour_cos      91512 non-null  float64
 17  open_4_3                91512 non-null  float64
 18  high_4_3                91512 non-null  float64
 19  low_4_3                 91512 non-null  float64
 20  close_4_3               91512 non-null  float64
 21  volume_4_3              91512 non-null  float64
 22  minutes_4_3             91512 non-null  float64
 23  open_3_2                91512 non-null  float64
 24  high_3_2                91512 non-null  float64
 25  low_3_2                 91512 non-null  float64
 26  close_3_2               91512 non-null  float64
 27  volume_3_2              91512 non-null  float64
 28  minutes_3_2             91512 non-null  float64
 29  open_2_1                91512 non-null  float64
 30  high_2_1                91512 non-null  float64
 31  low_2_1                 91512 non-null  float64
 32  close_2_1               91512 non-null  float64
 33  volume_2_1              91512 non-null  float64
 34  minutes_2_1             91512 non-null  float64
 35  open_1_0                91512 non-null  float64
 36  high_1_0                91512 non-null  float64
 37  low_1_0                 91512 non-null  float64
 38  close_1_0               91512 non-null  float64
 39  volume_1_0              91512 non-null  float64
 40  minutes_1_0             91512 non-null  float64
```

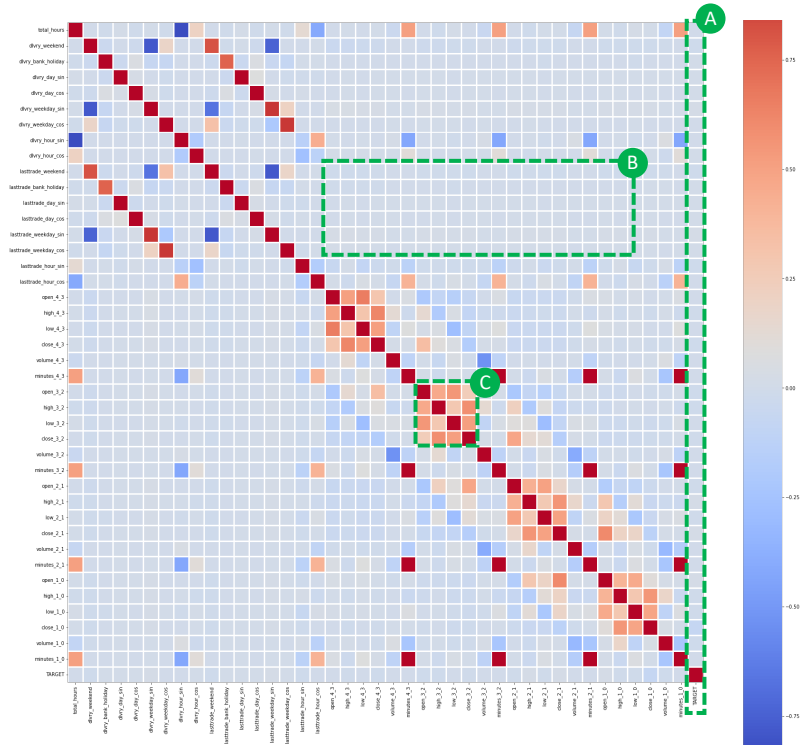
<3> Visualization Example:

Multiple rows could, in theory, stem from the same contract, as its trading history can be split into multiple sliding windows. We can use the "total_hours" as a proxy primary key for identifying the contract for each row of the data. The MIN and MAX amounts of rows attributed to a specific contract are 1 and 44 (window size 5), and 1 and 24 (window size 15). This also indicates the difficulty that the data is not a continuous time series, but rather is composed of many time series that might occur in parallel.



<4> Correlation matrix:

- A Target variable is not significantly correlated with any predictor
- B Most features are uncorrelated
- C Strong correlations between the same features at different time steps and between the open, high, low and close prices differences at a specific time step, as marked in green boxes



3 Preprocessing

Data

Sine/Cosine Re-transformation

Re-transforming sine/cosine-transformed cyclical features in order to provide tree-based models (splitting on one column per split) with suitable data. Since the sine/cosine transformation creates separate sine and cosine columns per cyclical feature, tree-based approaches may split on the cosine column only, which disregards the corresponding sine column in representing the actual value.

Tree-based Models

File Examples:
X_train_window_size_5_tree
X_train_window_size_15_tree

Reshaping data: 2D → 3D

Reshaping the data from 2D to 3D in order to make it compatible with RNNs (creating dimension along time steps) while keeping the time series structure intact.

RNNs

File Examples:
X_train_unflatten_5_ohlcv
X_train_unflatten_15_ohlcv
X_train_unflatten_all_5
X_train_unflatten_all_15

Normalization

Applying normalization in order to align feature scales and prevent scale-sensitive models (e.g., SVM) from considering different feature scales with corresponding unwanted distortions of weightage.

SVM

File Examples:
X_train_window_size_5
X_train_window_size_15

4 Baseline ML Models

Conventional ML models serve as baselines to be compared with DL approaches. The following results are based on data with window size **5**

Dummy Regressor

The Dummy Regressor served as a starting point of our modeling process and one of the baselines for result comparison. The Dummy Regressor is just predicting one value for all rows by using simple rules.

Results:
Test MAE = 2.2588
Test MSE = 21.1589

Linear Regression

As expected, the linear models performed poorly for our data, as indicated by the R2 scores for both degree 1 and degree 2 models. The near 0.0 R2 scores for both validation and testing sets suggested that dependent variable could not be explained by our data.

Results:
Test MAE = 2.2783
Test R2 = -0.0047

SVM

The SVM model utilizes the kernel trick to project the original dataset to a higher dimension to tolerate errors within a certain range, making the model more robust. The input data of the model has been normalized. Using GridSearchCV, the best combination of parameters for SVM model is:
{ Kernel: 'poly', C = 20 }.

Results:
Test MAE = 2.2552
Test MSE = 21.1625

KNN

The optimal K was found through GridSearch in the range(3, 25). KNN is a simple model that is prone to underfitting as it only takes into account K other observations for a prediction. In our case it was the worst of the baseline models.

Results:
Test MAE = 2.7276
Test MSE = 23.3259

5 Tree-based Models

The following results are based on data with window size **5**

Random Forest

```
rfmodel = RandomForestRegressor(random_state=3315,
                                max_depth=6,
                                n_estimators=500,
                                max_features='sqrt',
                                min_samples_split=10,
                                criterion='mse',
                                n_jobs=-1)
rfmodel.fit(X_train_window_size_5_tree, y_train_window_size_5)

result = evaluate_model(rfmodel, X_test_window_size_5_tree, y_test_window_size_5)
performance_df = performance_df.append({'Model': 'Random Forest',
                                        'MAE score (on test set)': round(result, 4)},
                                        ignore_index=True)
```

Parameters

Training the model

Evaluation

Results:
MAE = 2.2755
MSE = 21.8467

LightGBM

```
train_data = lgb.Dataset(X_train_window_size_5_tree, label=y_train_window_size_5)
valid_data = lgb.Dataset(X_valid_window_size_5_tree, label=y_valid_window_size_5)
test_data = lgb.Dataset(X_test_window_size_5_tree, label=y_test_window_size_5)

param = {'num_leaves': 60, 'objective': 'regression_l1', 'metric': 'mae'}
num_round = 20
bst = lgb.train(param, train_data, num_round, valid_sets=[valid_data])

result = evaluate_model(bst, X_test_window_size_5_tree, y_test_window_size_5)
performance_df = performance_df.append({'Model': 'LightGBM',
                                        'MAE score (on test set)': round(result, 4)},
                                       ignore_index=True)
```

Preparation

Parameters & Training

Evaluation

Results:
MAE = 2.2495
MSE = 21.1023

Key Takeaway: Compared to Random Forest, which trains all decision trees independently, the estimators within LightGBM can learn from each other, increasing its ability to learn from the data and therefore producing better performance results.

6 MLP & CNN

<1> Model architecture

For both MLP and CNN, we found that a simple network structure (2 dense layers for MLP, 2 convolutional and Max Pooling layers with 2 dense layers for CNN) with dropout and Adam optimizer yielded the best results. For the CNN, we needed to roll the data into 3D.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 300)	12900
dropout (Dropout)	(None, 300)	0
dense_1 (Dense)	(None, 100)	30100
dropout_1 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 1)	101

Total params: 43,101
Trainable params: 43,101
Non-trainable params: 0

MLP

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 4, 22)]	0
conv1d (Conv1D)	(None, 4, 5)	225
max_pooling1d (MaxPooling1D)	(None, 3, 5)	0
conv1d_1 (Conv1D)	(None, 3, 5)	55
max_pooling1d_1 (MaxPooling1	(None, 2, 5)	0
dense (Dense)	(None, 2, 100)	600
dropout (Dropout)	(None, 2, 100)	0
dense_1 (Dense)	(None, 2, 100)	10100
dropout_1 (Dropout)	(None, 2, 100)	0
dense_2 (Dense)	(None, 2, 1)	101

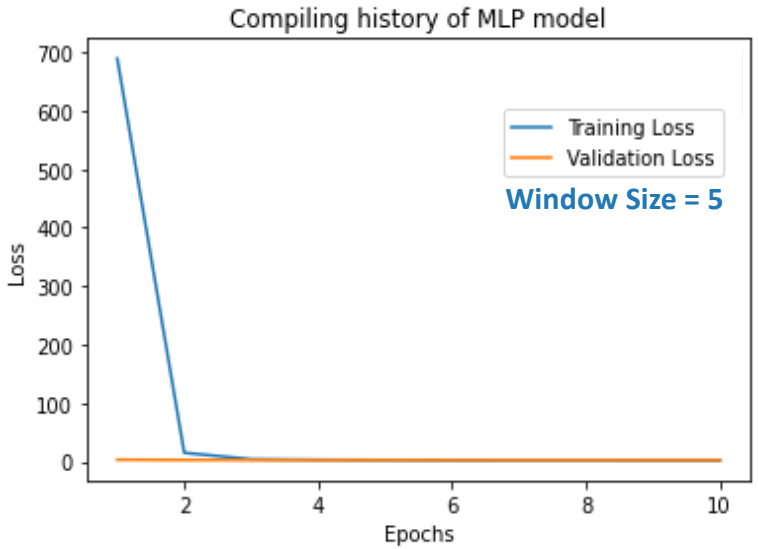
Total params: 11,081
Trainable params: 11,081
Non-trainable params: 0

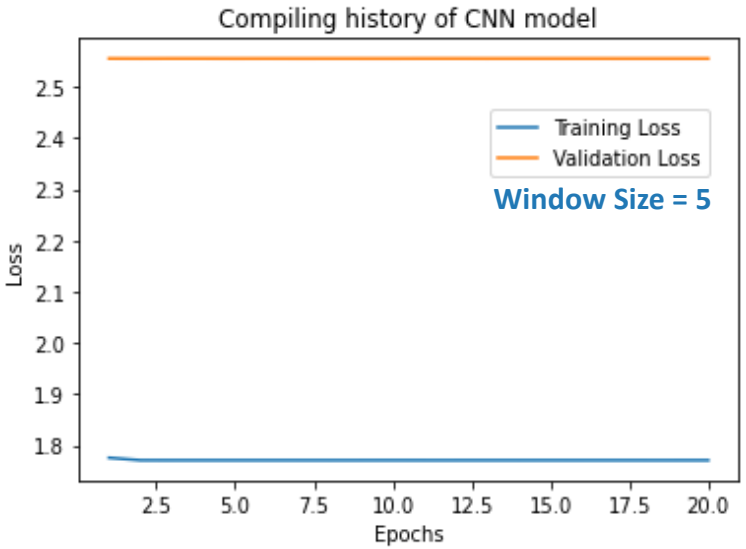
CNN

<2> Convergence

After the first two iterations, it seemed that neither MLP nor CNN decreased their losses. Also, the validation loss remained larger than the test set loss.

It should be noted that CNN's power has not been fully unleashed in our case, as the kernel size was bounded by the dimension of the timesteps (4 / 14), whereas normally a CNN would be applied to images data, for example, with at least 32x32 pixels.





7 RNNs – LSTM

<1> Model architecture

We used LSTM and GRU RNNs. The findings for the architectures are comparable to MLP and CNN models: Higher complexity did not increase model performance. Our LSTM model used 3 "LSTM and Dropout" layer pairs followed by 3 "Dense and Dropout" layer pairs. The GRU model used 2 GRU layers followed by 2 "Dense and Dropout" layer pairs. For the RNNs to work, we needed to roll the data into 3D, using the timesteps as the 3rd dimension.

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 4, 22)]           0
_________________________________________________________________
lstm (LSTM)                  (None, 4, 300)            387600
_________________________________________________________________
dropout (Dropout)            (None, 4, 300)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 4, 300)            721200
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 300)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 4, 300)            721200
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 300)            0
_________________________________________________________________
dense (Dense)                (None, 4, 100)            30100
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 100)            0
_________________________________________________________________
dense_1 (Dense)              (None, 4, 100)            10100
_________________________________________________________________
dropout_4 (Dropout)          (None, 4, 100)            0
_________________________________________________________________
dense_2 (Dense)              (None, 4, 100)            10100
_________________________________________________________________
dropout_5 (Dropout)          (None, 4, 100)            0
_________________________________________________________________
dense_3 (Dense)              (None, 4, 1)              101
=================================================================
Total params: 1,880,401
Trainable params: 1,880,401
Non-trainable params: 0
```

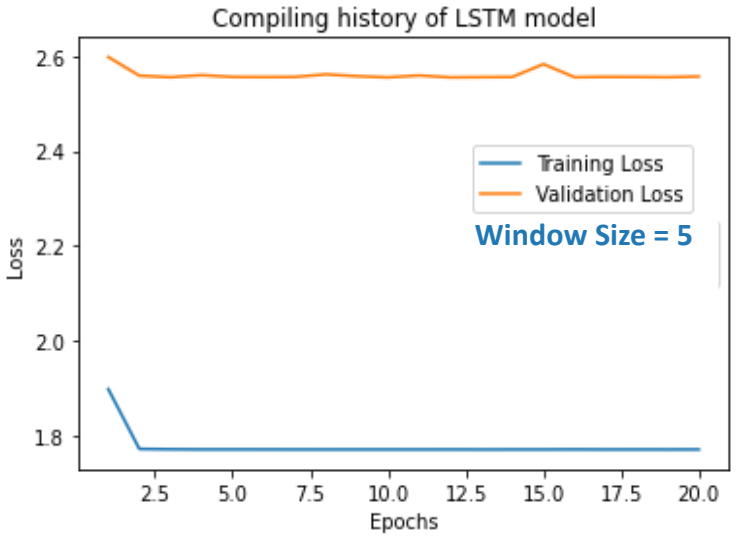
LSTM

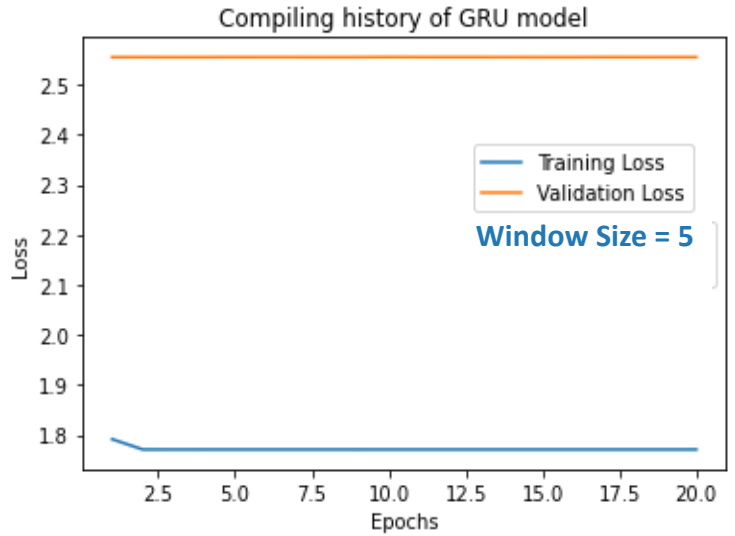
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 4, 22)]           0
_________________________________________________________________
gru (GRU)                    (None, 4, 200)            134400
_________________________________________________________________
gru_1 (GRU)                  (None, 4, 200)            241200
_________________________________________________________________
dense (Dense)                (None, 4, 100)            20100
_________________________________________________________________
dropout (Dropout)            (None, 4, 100)            0
_________________________________________________________________
dense_1 (Dense)              (None, 4, 100)            10100
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 100)            0
_________________________________________________________________
dense_2 (Dense)              (None, 4, 1)              101
=================================================================
Total params: 405,901
Trainable params: 405,901
Non-trainable params: 0
```

GRU

<2> Convergence

As with the other NN-based models, the RNNs did not continuously decrease the loss after the initial convergence which happened in the first two epochs. An explanation for this behavior could be that the models were too complex for the task at hand.





8 Transformer

<1> Model architecture

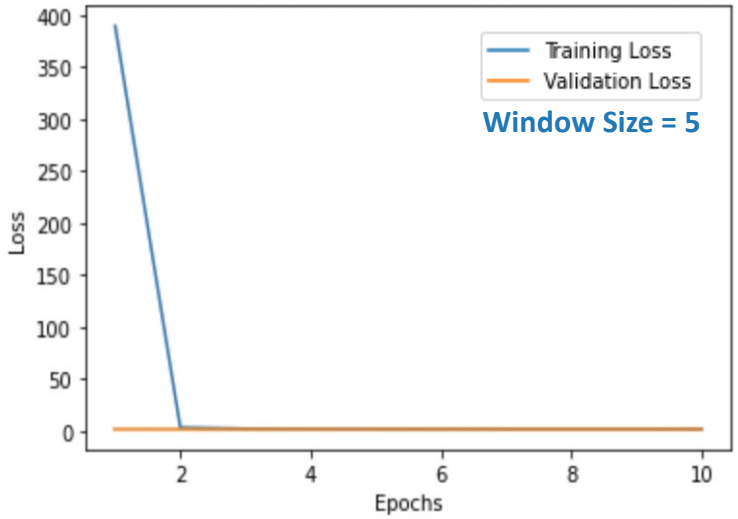
Since our task is to predict, we can only apply a self-attention algorithm which we also further simplified and used only a query and skipped the “key + value” store. Additionally, we used a Dense Layer for the Embedding of our features, followed by an attention layer. Afterwards, we concatenated the attention and the input layer with 2 “LSTM + Dropout” layer pairs. The output is a Dense layer with a linear activation function.

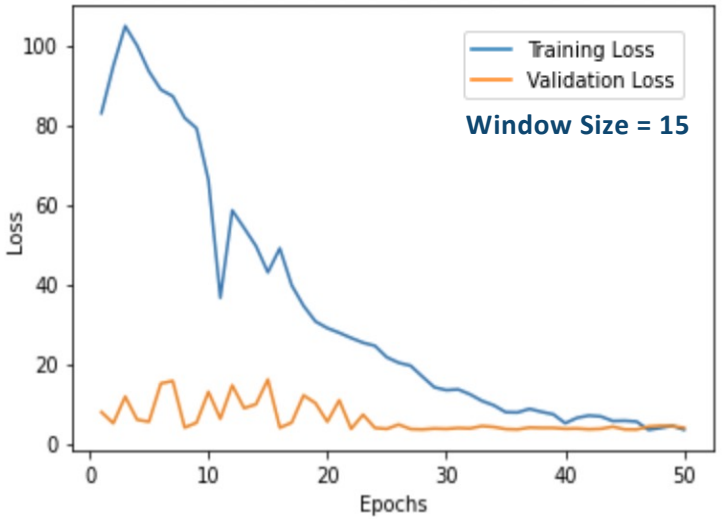
```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 34)]         0
__________________________________________________________________________________________________
dense (Dense)                   (None, 34)           1190        input_1[0][0]
__________________________________________________________________________________________________
attention (Attention)           (None, 34)           0           input_1[0][0]
                                                                 dense[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 68)           0           input_1[0][0]
                                                                 attention[0][0]
__________________________________________________________________________________________________
Dense_1 (Dense)                 (None, 256)          17664       concatenate[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 256)          0           Dense_1[0][0]
__________________________________________________________________________________________________
Dense_2 (Dense)                 (None, 256)          65792       dropout[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 256)          0           Dense_2[0][0]
__________________________________________________________________________________________________
output_layer (Dense)            (None, 1)            257         dropout_1[0][0]
==================================================================================================
Total params: 84,903
Trainable params: 84,903
Non-trainable params: 0
```

Transformer

<2> Convergence

In the graphs, one can see the convergence of the model for the data with window size 5 on the left and the data with window size 15 on the right. After two epochs, the self-attention algorithm learned the training data already while the model converged very well on the window size 15 after 50 epochs. However, the MAE on the validation set did not converge in both models very well even with many dropouts.





9 Summary

Results Window Size = 5

	Model	MAE score (on test set)	MSE score (on test set)
4	LightGBM	2.2495	21.1023
7	Mulit-Layer Perceptron	2.2550	NaN
5	Support Vector Regressor	2.2551	21.1624
11	Self-Attention	2.2551	21.1618
8	CNN	2.2552	21.1624
10	GRU	2.2553	21.1627
9	LSTM	2.2558	21.1615
0	Dummy Regressor	2.2588	21.1589
3	Random Forest	2.2755	21.8467
1	Linear Regression	2.2783	21.2577
2	Linear Regression (poly transformed)	2.4848	22.6935
6	K-Nearest Neighbor	2.6874	23.3259

<1> Boosted decision trees have the best performance

<2> Performances of all NNs are very close to each other

- RNN (more suitable for sequence data) and CNN (more suitable for image data) showed very similar performance results.
- During hyperparameter tuning, we discovered that the complexity of the models also did not make much difference.
- When training the NNs, the validation & training losses fluctuated around certain values, regardless of the types of optimizers/ learning rates (scheduler)/ activation functions/ initializers we used, the models could not reach lower values.
- Therefore, it is very likely that NNs is not suitable for solving our problem. Generally, NNs are helpful in identifying underlying patterns in data. However, our datasets involved not only one time series but multiple sequences of different contracts, which might add a lot of noises. And without a large amount of data to make up for this drawback, it could be one of the reasons why NNs were unable to perform well.

<3> Comparison between NNs and Decision Trees

- Even though NNs are often compared to Decision Trees because both approaches can handle nonlinear relationships within the data, NNs have some disadvantages in solving multi-classification problems.
- Additionally, in our data, the trading info of electricity contracts has very small time steps with varying lengths, which make it also hard for the LSTM model to remember important features and information of the time series. In contrast, gradient boosted decision trees don't even consider the time series or the underlying autocorrelation. However, XGBoost-like models tend to overfit significantly, but we were able to mitigate this problem by dropping features with high correlations and by increasing the independence between the variables.

10 Summary

‹4› Error Analysis

The most important finding of our modelling was that most of our models have a hard time beating the dummy regressor

- By looking into the two distributions down below, one can see on the right the very dense distribution around 0 of the training set target variable. The small variance also explains why the Dummy Regressor did so well by just predicting one value and using the MAE as the loss function.
- Additionally, we suspect the same behavior of our best model, the LightGBM, which predicts only values in a dense space between −0.3 ~ 0.2 as well.
- In summary, we must say that our modeling was not good enough to capture the target distribution very well to significantly beat the Dummy Regressor.

‹5› Shift in Data Distribution

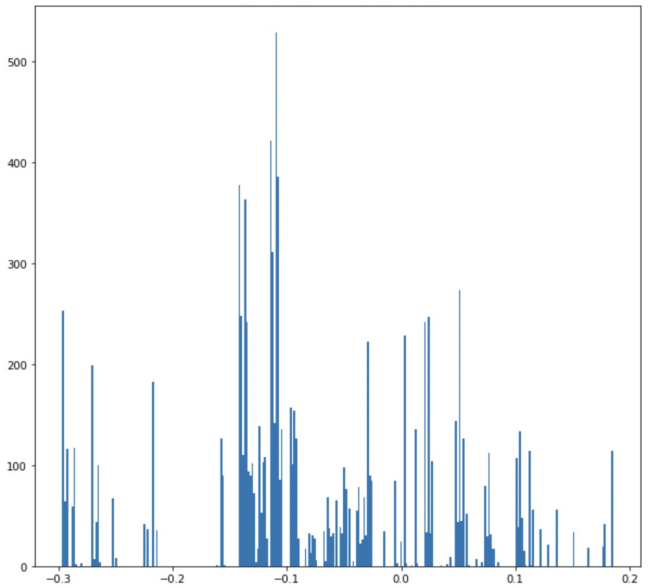
‹1› Covariate shift (Input)

- Significant shift in feature "total_hours" from train set over validation and test set
- "Dlvry_bank_holiday" only represented in the training set
- Some candle features of t-4 and t-1 are systematically different in validation and test set in relation to the train set

‹2› Prior prob. Shift (Target)

- Target variable of the training set is dragged closer to zero
- 2x increase in mean from test to valid and another 2x increase from valid to test set
- Generalization is more difficult if the distribution means change that much between the data sets

LightGBM prediction distribution



Train target distribution