Result Analysis and Discussion

1 Problem Definition

Create a NN-based model that predicts sensor values using past data as reference.

2 Contents

prediction_with_nn_for_weather_data/

```
├── report/
│   ├── images/
│   ├── report.tex
│   ├── report.pdf
│   └── foxelas_report.cls
│
├── config.ini
├── main.py
├── part1.py
├── part2.py
├── case1.txt
├── case2.txt
└── wheather_data.csv
```

Run settings can be updated in config.ini. Afterwards, run 'main.py' and see the results.
This code was tested with PyCharm using Python 3.9 and Tensorflow 2.8.3.

3 Data Preprocessing

Data type: Time series in .csv format
Variables: id, timestamp, value, identifier, value_type_id, location_id, source_id

Preprocesssing:
Scale numerical variables with MinMax Scaler.
Convert categorical variables into multiple binary variables using One Hot Encoding.
Convert string into datetime.

Datetime variables: timestamp
Numeric Variables: value

```
Reading input data...

**Statistics of numerical variables

         min     max       mean
value -0.019  1021.9  153.45302
```

Categorical Variables: identifier

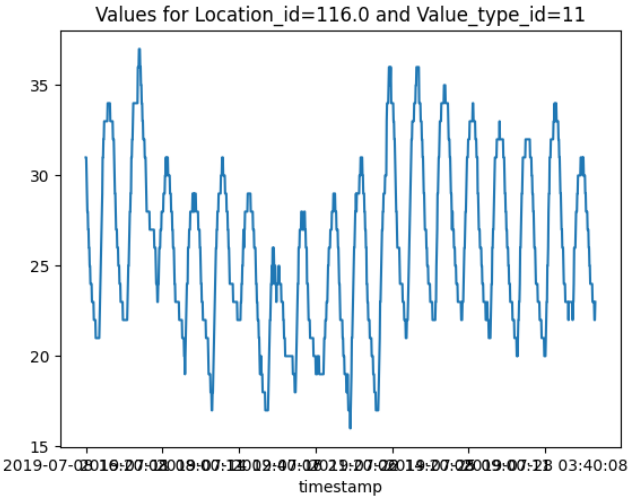
```
**Statistics of categorical variables

Variable: identifier
                Category  Frequency (%)
0                      T      11.750127
1                     RH      11.749881
2                   PMSL      11.744472
3                     WS      11.738325
4                     WD      11.632356
5                    UVI      10.075507
6                     PC       7.508870
7                      P       6.222478
8                    TDP       3.854258
9                    P1H       3.854012
10                    CC       2.947490
11                   P6H       2.567866
12                  P24H       2.567620
13                   SM0       0.252508
14                   ST0       0.251771
15                 SM100       0.200384
16                  SM30       0.200384
17                 ETref       0.199892
18                ETmodel       0.199892
19                 ST100       0.199646
20                  ST30       0.199646
21                  ST10       0.051387
22                   P3H       0.026062
23  pressureMeanSeaLevel       0.005163
```

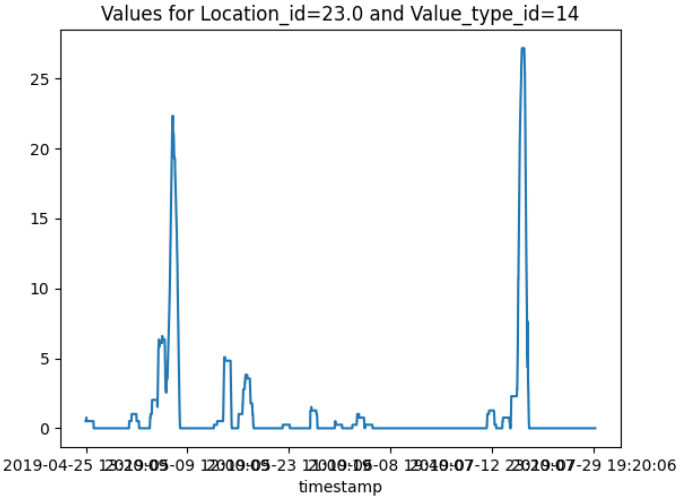
Values after preprocessing:

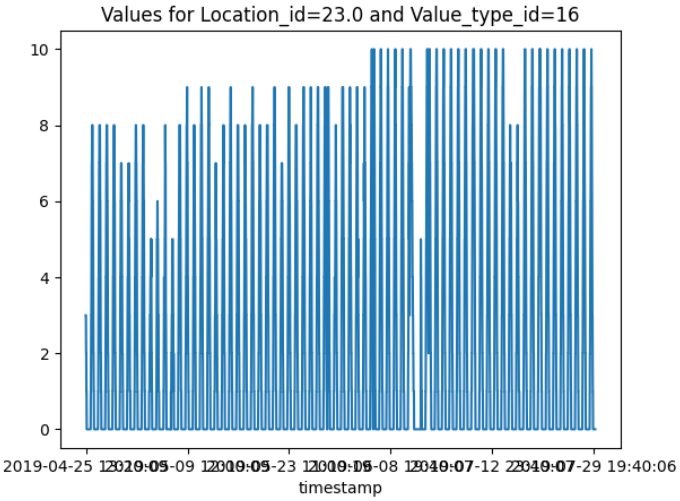
```
**Example of preprocessed data:

      id                          timestamp     value  value_type_id  location_id  source_id  identifier_CC  identifier_ETmodel  identifier_ETref  identifie
0  427436  (2019, 4, 25, 13, 20, 9, 3, 115, -1)  0.999902             20         23.0          5              0                   0                 0
1  427439  (2019, 4, 25, 13, 20, 9, 3, 115, -1)  0.011761             15         23.0          5              0                   0                 0
2  427434  (2019, 4, 25, 13, 20, 9, 3, 115, -1)  0.000019             13         23.0          5              0                   0                 0
3  427435  (2019, 4, 25, 13, 20, 9, 3, 115, -1)  0.000518             14         23.0          5              0                   0                 0
4  427440  (2019, 4, 25, 13, 20, 9, 3, 115, -1)  0.002954             16         23.0          5              0                   0                 0

Preprocessing finished.
```

Observation:
For a given location_id and value_type_id, all remaining variables are the same apart from columns 'timestamp' and 'value'. Therefore, the rest can be ignored in this case.

Visualization:







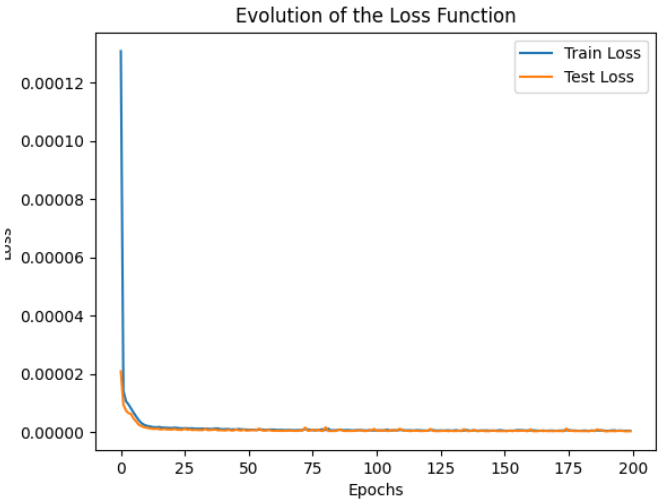
4 Prediction Model

A value should be predicted from previous values. A simple NN model is used (alternatively, LSTM and GRU networks are recommended). A look-back window is applied sequentially, in order to use the n-th previous values to predict the (n+1)-th. The dataset is split in train and test subsets, in order to evaluate performance. The model is fitted on the train data, and evaluated on both train and test sets.

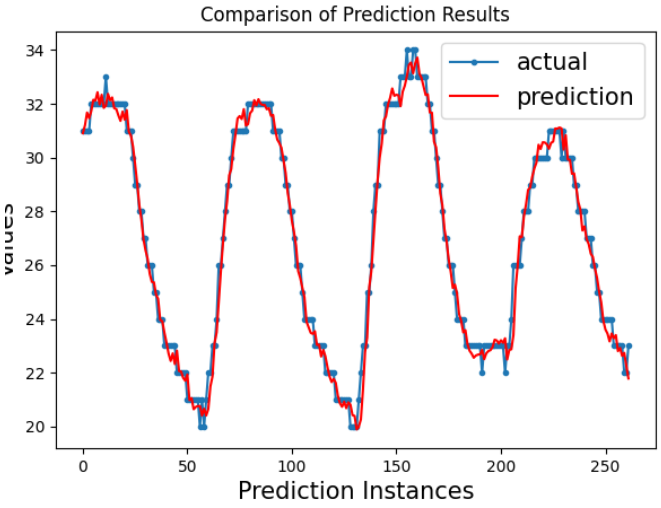
4.1 Experiment 1

Settings:
location_id = 116.0
value_type_id = 11
activation = relu
layers_number = 3
neurons_number = [32, 16, 8, 4]
loss = mean_squared_error
optimizer_name = Adam
epochs = 200
batch_size = 32
window_size = 20 split_percentage = 0.2

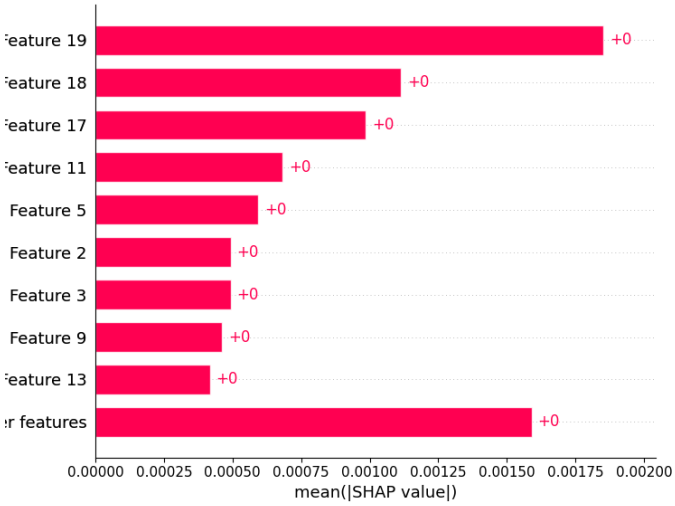
The evolution of the loss function across 200 epochs for the train and test sets. Training loss is expected lower than test loss, due to fitting of the model on the train data.



Expected and predicted values based on the past 20 values (20 point window). The model misses the steps in the data and incorrectly predicts smoother evolution of the signal.



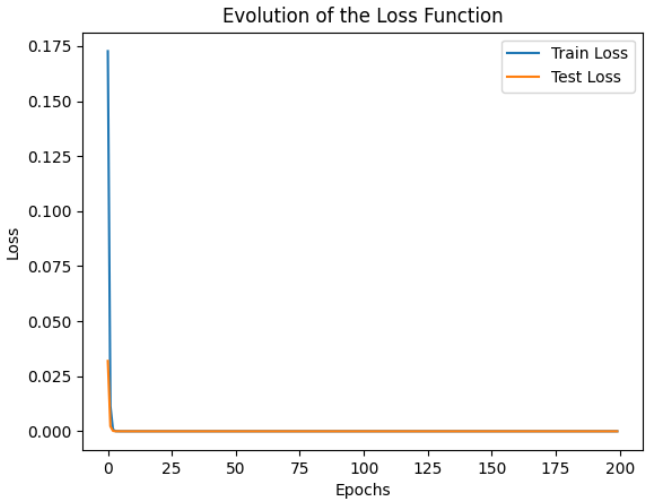
Data columns (past 20 values) are judged in terms of their contribution in the prediction using SHAP values. Values closer to the value to be predicted (last values in the window) are more influential.



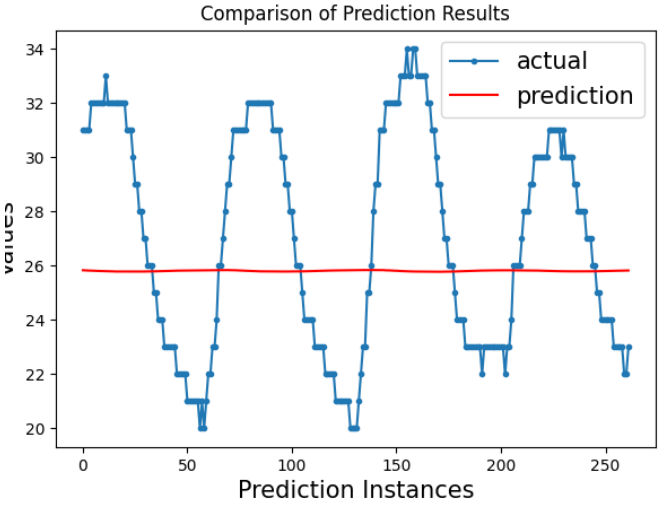
4.2 Experiment 2

Settings:
location_id = 116.0
value_type_id = 11
activation = sigmoid
layers_number = 4
neurons_number = [32, 16, 8, 4, 4]
loss = mean_squared_error
optimizer_name = SGD
epochs = 200
batch_size = 32
window_size = 20
split_percentage = 0.2

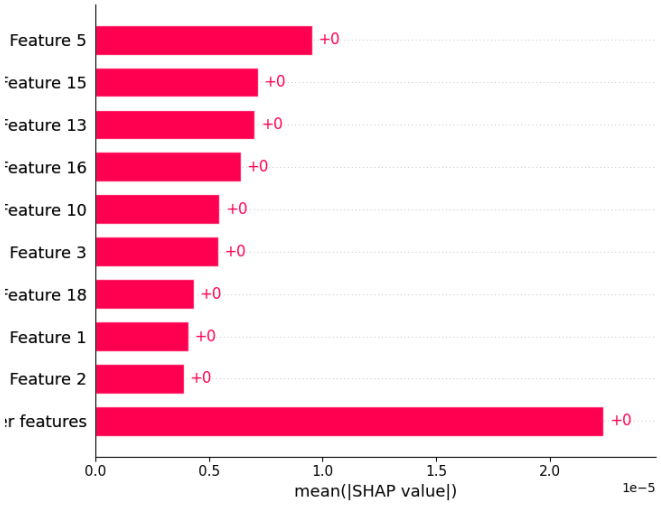
The evolution of the loss function across 200 epochs for the train and test sets. The shape of the curve at the knee shows that the model is not properly trained.



Expected and predicted values based on the past 20 values (20 point window). The model missed the prediction completely.



Data columns (past 20 values) are judged in terms of their contribution in the prediction using SHAP values. All of the values contribute equally to the (failed) prediction.



5 Modifications

The structure and the training of the model can be modified by changing the values of

---

experiment_settings = dict(location_id=116.0, value_type_id=11, activation='sigmoid', layers_number=4, neurons_number=[32, 16, 8, 4, 4], loss='mean_squared_error', optimizer_name='SGD', epochs=200, batch_size=32, window_size=pt1.window_size, split_percentage=pt1.split_percentage)

---

and then running function

---

build_and_evaluate_model_target_data()

---

in main.py.