October 2, 2022

NHITS model , ECOD

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pytorch_lightning as pl
from pytorch_lightning.callbacks import EarlyStopping
import torch

from pytorch_forecasting import Baseline, NHiTS, TimeSeriesDataSet
from pytorch_forecasting.data import NaNLabelEncoder
from pytorch_forecasting.data.examples import generate_ar_data
from pytorch_forecasting.metrics import SMAPE, MQF2DistributionLoss,␣
 ↪QuantileLoss, RMSE, TweedieLoss
```

[2]:
```
from pyod.models.ecod import ECOD
from collections import defaultdict
df_dict = defaultdict(object)
clf_dict = defaultdict(object)
clf_col_dict = defaultdict(list)
for i in range(37):
    df_dict[i] = pd.read_csv('./data/train/train_'+str(i)+'.csv')
    df_dict[i]['series'] = i

    ##t_diff feature
    df_dict[i]['t_diff'] = df_dict[i]['  _    ()'] - df_dict[i]['  _    ()'].
 ↪shift(1)
    df_dict[i]['date']=df_dict[i]['datadate'].apply(lambda x: pd.
 ↪to_datetime(str(x), format='%Y%m%d'))
    df_dict[i].drop(columns='datadate',inplace=True)

    # np.nan
     _cols=list(df_dict[i].filter(regex=' ').columns)
    df_dict[i][ _cols]=df_dict[i][ _cols].replace(' ',np.nan)
    df_dict[i][ _cols]=df_dict[i][ _cols].astype(float)
```

```
    #    column drop
    df_dict[i].drop(columns=df_dict[i].filter(regex='  ').columns,inplace=True)
    df_dict[i].drop(columns=df_dict[i].filter(regex=' ').columns,inplace=True)
    df_dict[i].drop(columns=df_dict[i].filter(regex='  ').columns,inplace=True)

    original_cols=list(df_dict[i].drop(columns='t_diff').columns)

    ## null value    column
    for col in original_cols:
        if sum(df_dict[i][col].isna())>550:
            df_dict[i].drop(columns=col,inplace=True)

    ## weekday,month, nan  0
    df_dict[i].fillna(0,inplace=True)
    df_dict[i]['month'] = df_dict[i]['date'].dt.month
    df_dict[i]['weekday'] = df_dict[i]['date'].dt.weekday

    ## ECOD   features
    a=[' ( )', '  ', '  ( )', '  ', '   _   (kg)']
    b=list((df_dict[i].filter(regex=' ')).columns)
    c=list((df_dict[i].filter(regex=' ')).columns)
    d=['month','weekday']
    e=list(df_dict[i].filter(regex=' ').columns)
    f=list(df_dict[i].filter(regex=' ').columns)
    g=list(df_dict[i].filter(regex=' ').columns)
    h=list(df_dict[i].filter(regex=' ').columns)


    clf_col_dict[i] = a+b+c+d+e+f+g+h

    ##ECOD col
    clf_dict[i] = ECOD()
    clf_dict[i].fit(df_dict[i][clf_col_dict[i]])
    df_dict[i]['ECOD']=clf_dict[i].decision_scores_
```

[3]:
```
#DF
df = pd.DataFrame()
for i in df_dict.keys():
    df = pd.concat((df,df_dict[i]),axis=0)
```

[4]:
```
#time idx for pytorch forecast
df['time_idx'] = df.index
```

[5]:
```
#  columns
temp=list(df.filter(regex='  ').columns)
```

[6]:
```
#DF       nan value  0
df.fillna(0,inplace=True)
```

[7]:
```
df.reset_index(drop=True,inplace=True)
```

[8]:
```
from pytorch_forecasting.data.encoders import GroupNormalizer,EncoderNormalizer
# create dataset and dataloaders
max_encoder_length = 14
max_prediction_length = 28

training_cutoff = df["time_idx"].max() - max_prediction_length

context_length = max_encoder_length
prediction_length = max_prediction_length

training = TimeSeriesDataSet(
    df[lambda x: x.time_idx <= training_cutoff],
    time_idx="time_idx",
    target="  _    ( )",
    group_ids=["series"],
    # only unknown variable is "value" - and N-HiTS can also not take any␣
 ↪additional variables
    time_varying_unknown_reals=["  _    ( )",'t_diff','ECOD']+temp,
    max_encoder_length=context_length,
    max_prediction_length=prediction_length,
    allow_missing_timesteps=True,
    target_normalizer=EncoderNormalizer(transformation=dict(forward=torch.
 ↪log1p))

)

validation = TimeSeriesDataSet.from_dataset(training, df,␣
 ↪min_prediction_idx=training_cutoff + 1)
batch_size = 128
train_dataloader = training.to_dataloader(train=True, batch_size=batch_size,␣
 ↪num_workers=0)
val_dataloader = validation.to_dataloader(train=False, batch_size=batch_size,␣
 ↪num_workers=0)
```

[9]:
```
##Tweedie loss
from pytorch_forecasting.metrics.point import TweedieLoss


early_stop_callback = EarlyStopping(monitor="val_loss", min_delta=1e-4,␣
 ↪patience=10, verbose=False, mode="min")
trainer = pl.Trainer(
    max_epochs=100,
```

```
    gpus=0,
    enable_model_summary=True,
    gradient_clip_val=1.0,
    callbacks=[early_stop_callback],
    limit_train_batches=30,
    enable_checkpointing=True,
)


net = NHiTS.from_dataset(
    training,
    learning_rate=0.09,
    activation='ReLU',
    log_interval=10,
    log_val_interval=1,
    weight_decay=1e-2,
    backcast_loss_ratio=0.0,
    hidden_size=64,
    loss=TweedieLoss()
)
trainer.fit(
    net,
    train_dataloaders=train_dataloader,
    val_dataloaders=val_dataloader,
)
```

```
C:\Users\USER\anaconda3\envs\pytorch_timeforecast\lib\site-
packages\pytorch_lightning\trainer\connectors\accelerator_connector.py:447:
LightningDeprecationWarning: Setting `Trainer(gpus=0)` is deprecated in v1.7 and
will be removed in v2.0. Please use `Trainer(accelerator='gpu', devices=0)`
instead.
  rank_zero_deprecation(
GPU available: False, used: False
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
C:\Users\USER\anaconda3\envs\pytorch_timeforecast\lib\site-
packages\pytorch_lightning\utilities\parsing.py:268: UserWarning: Attribute
'loss' is an instance of `nn.Module` and is already saved during checkpointing.
It is recommended to ignore them using
`self.save_hyperparameters(ignore=['loss'])`.
  rank_zero_warn(
C:\Users\USER\anaconda3\envs\pytorch_timeforecast\lib\site-
packages\pytorch_lightning\utilities\parsing.py:268: UserWarning: Attribute
'logging_metrics' is an instance of `nn.Module` and is already saved during
checkpointing. It is recommended to ignore them using
`self.save_hyperparameters(ignore=['logging_metrics'])`.
```

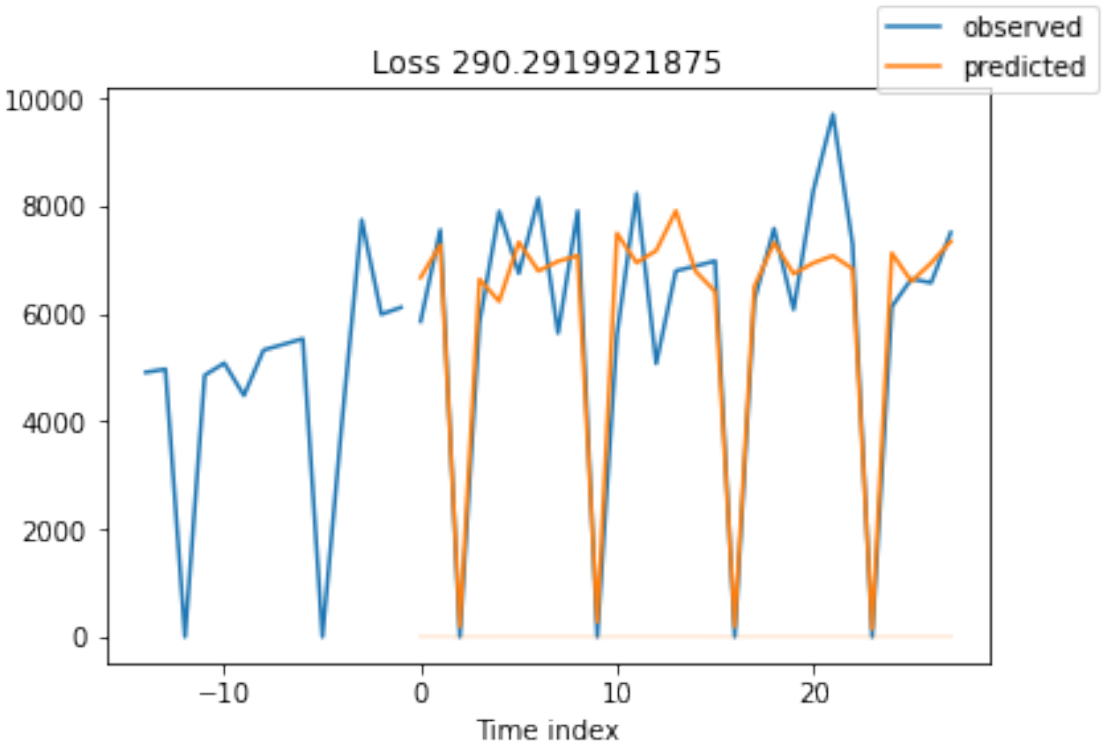
```
  rank_zero_warn(

  | Name            | Type           | Params
-------------------------------------------------
0 | loss            | TweedieLoss    | 0
1 | logging_metrics | ModuleList     | 0
2 | embeddings      | MultiEmbedding | 0
3 | model           | NHiTS          | 18.7 K
-------------------------------------------------
18.7 K    Trainable params
0         Non-trainable params
18.7 K    Total params
0.075     Total estimated model params size (MB)

Sanity Checking: 0it [00:00, ?it/s]

C:\Users\USER\anaconda3\envs\pytorch_timeforecast\lib\site-
packages\pytorch_lightning\trainer\connectors\data_connector.py:236:
PossibleUserWarning: The dataloader, val_dataloader 0, does not have many
workers which may be a bottleneck. Consider increasing the value of the
`num_workers` argument` (try 8 which is the number of cpus on this machine) in
the `DataLoader` init to improve performance.
  rank_zero_warn(
C:\Users\USER\anaconda3\envs\pytorch_timeforecast\lib\site-
packages\pytorch_lightning\trainer\connectors\data_connector.py:236:
PossibleUserWarning: The dataloader, train_dataloader, does not have many
workers which may be a bottleneck. Consider increasing the value of the
`num_workers` argument` (try 8 which is the number of cpus on this machine) in
the `DataLoader` init to improve performance.
  rank_zero_warn(
C:\Users\USER\anaconda3\envs\pytorch_timeforecast\lib\site-
packages\pytorch_lightning\trainer\trainer.py:1892: PossibleUserWarning: The
number of training batches (30) is smaller than the logging interval
Trainer(log_every_n_steps=50). Set a lower value for log_every_n_steps if you
want to see logs for the training epoch.
  rank_zero_warn(

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

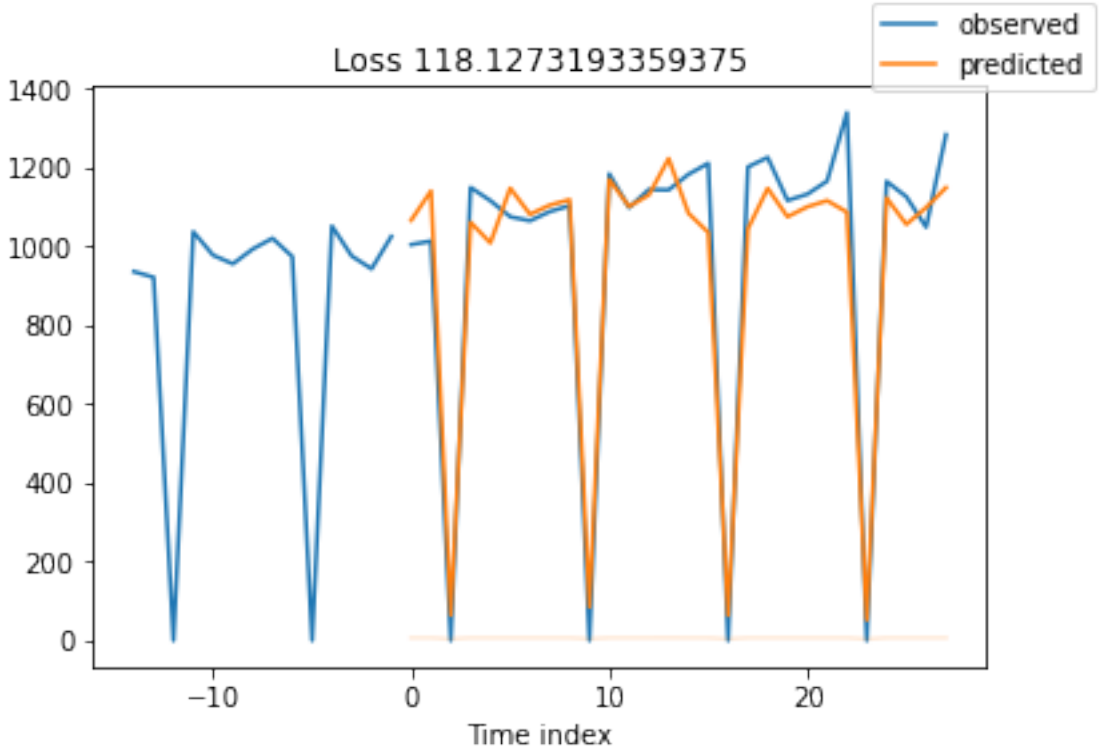
```
Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]
```

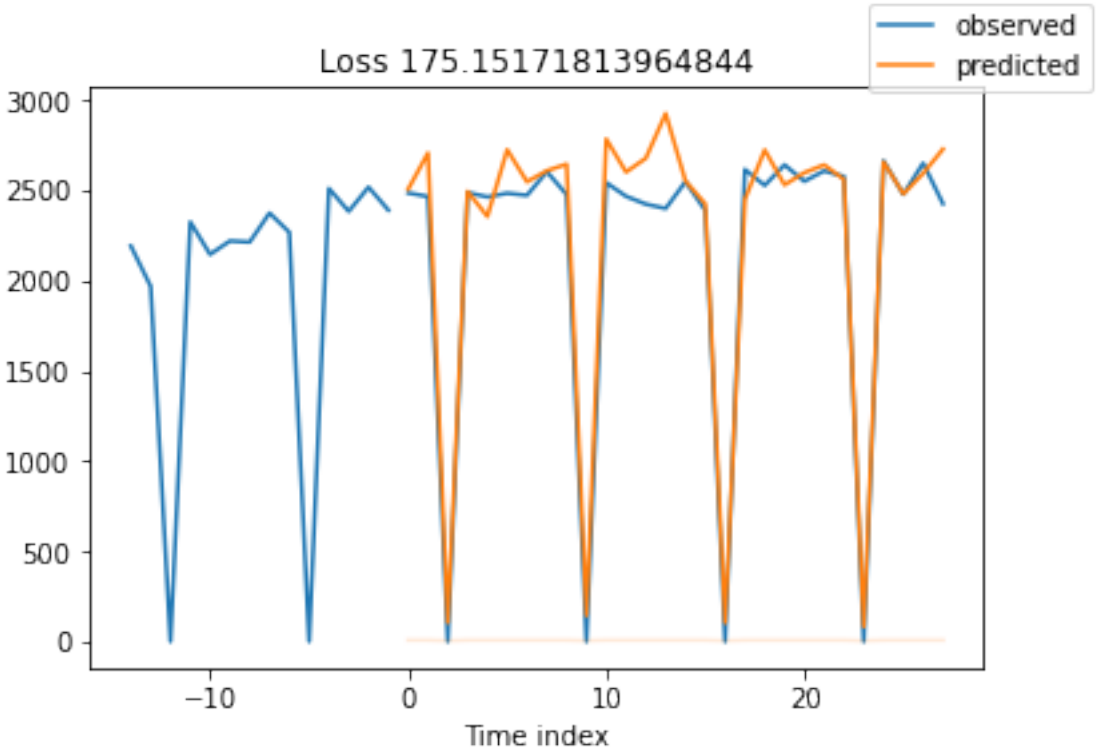
[10]:
```
best_model_path = trainer.checkpoint_callback.best_model_path
best_model = NHiTS.load_from_checkpoint(best_model_path)
```

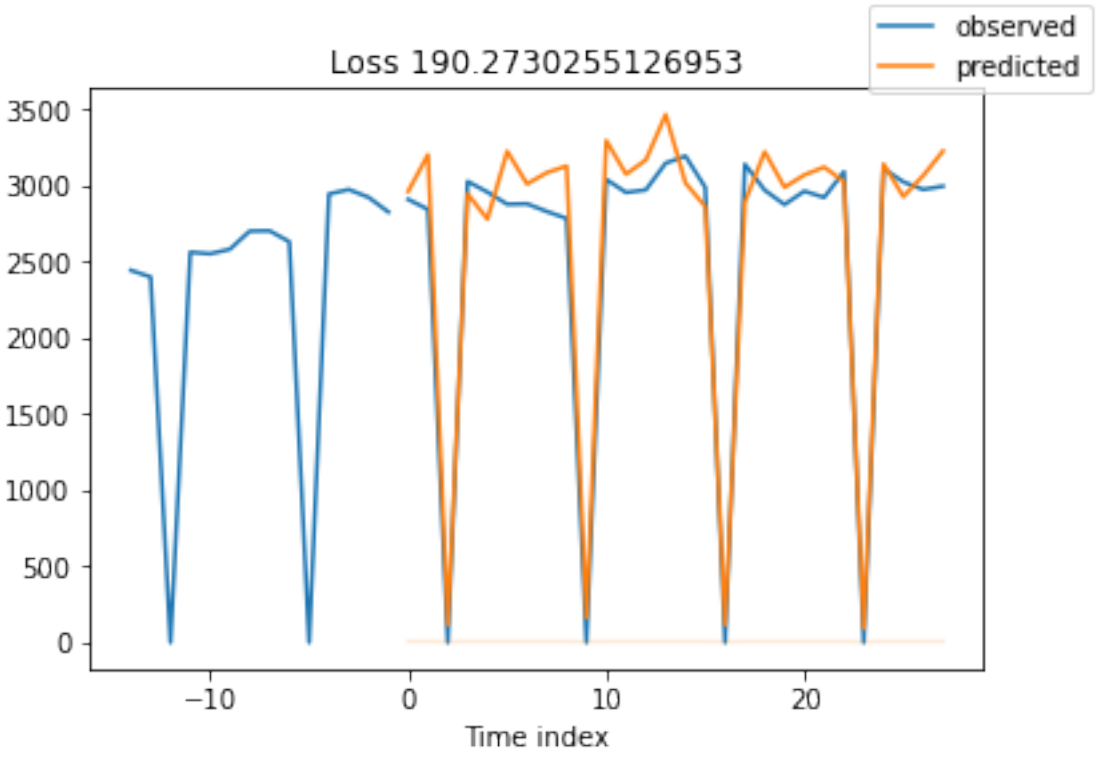
[11]:
```
raw_predictions, x = best_model.predict(val_dataloader, mode="raw",␣
 ↪return_x=True)
for idx in range(37):  # plot 10 examples
    best_model.plot_prediction(x, raw_predictions, idx=idx,␣
 ↪add_loss_to_title=True);
```

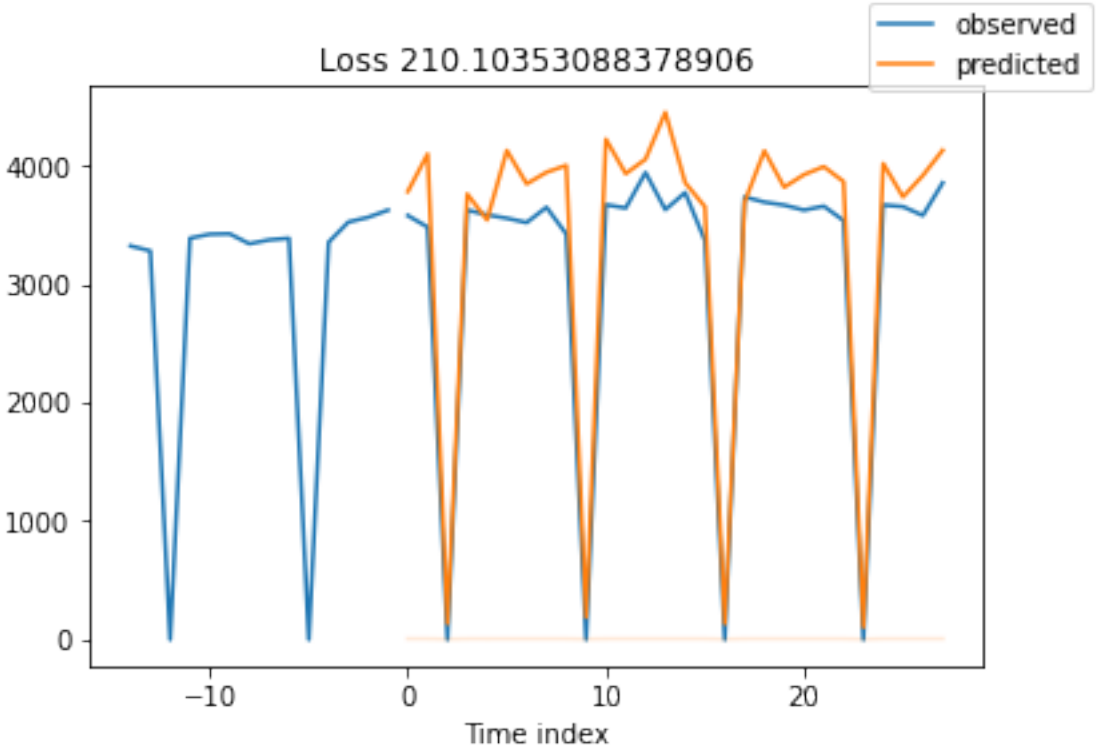
```
C:\Users\USER\anaconda3\envs\pytorch_timeforecast\lib\site-
packages\pytorch_forecasting\models\base_model.py:797: RuntimeWarning: More than
20 figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`). Consider using `matplotlib.pyplot.close()`.
  fig, ax = plt.subplots()
```

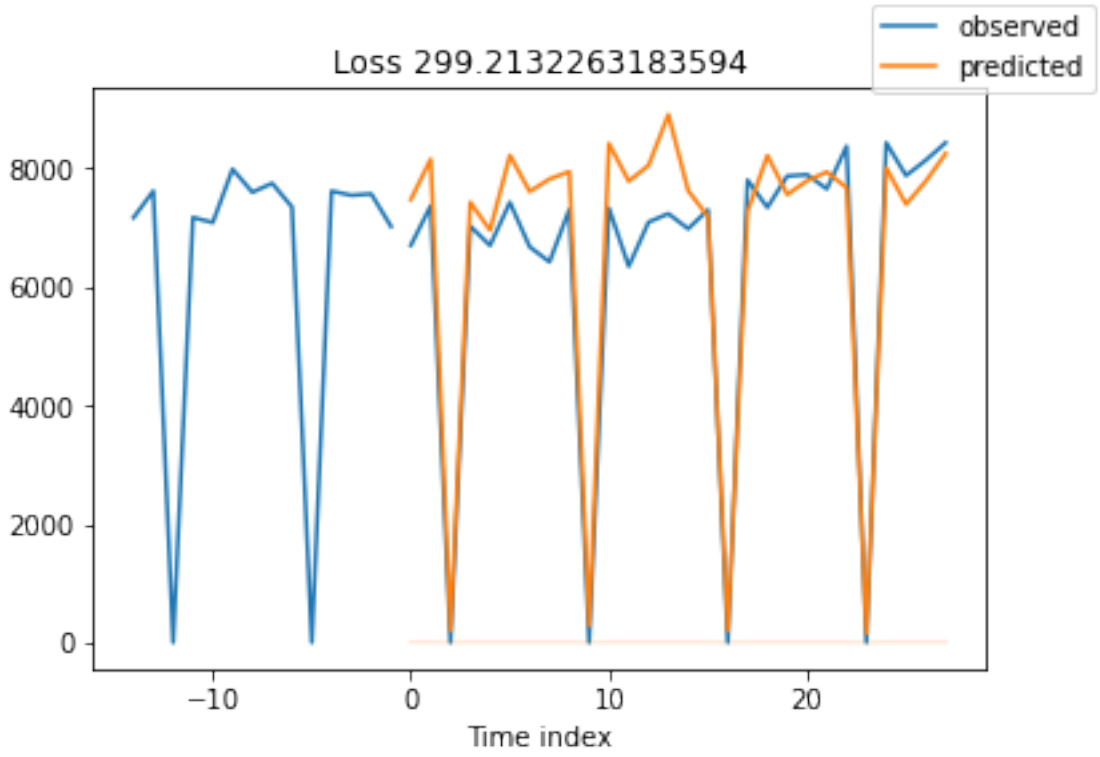

observed
predicted
Loss 290.2919921875
10000
8000
6000
4000
2000
0
−10
0
10
20
Time index

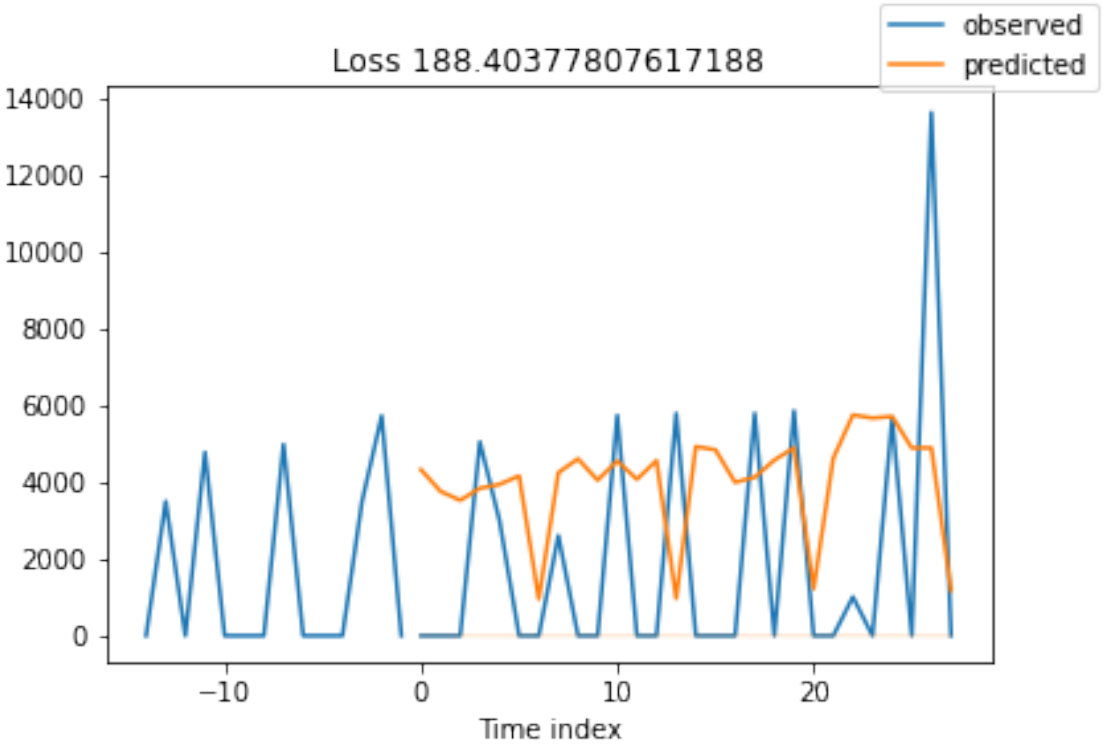

observed
predicted
Loss 118.1273193359375
1400
1200
1000
800
600
400
200
0
−10
0
10
20
Time index

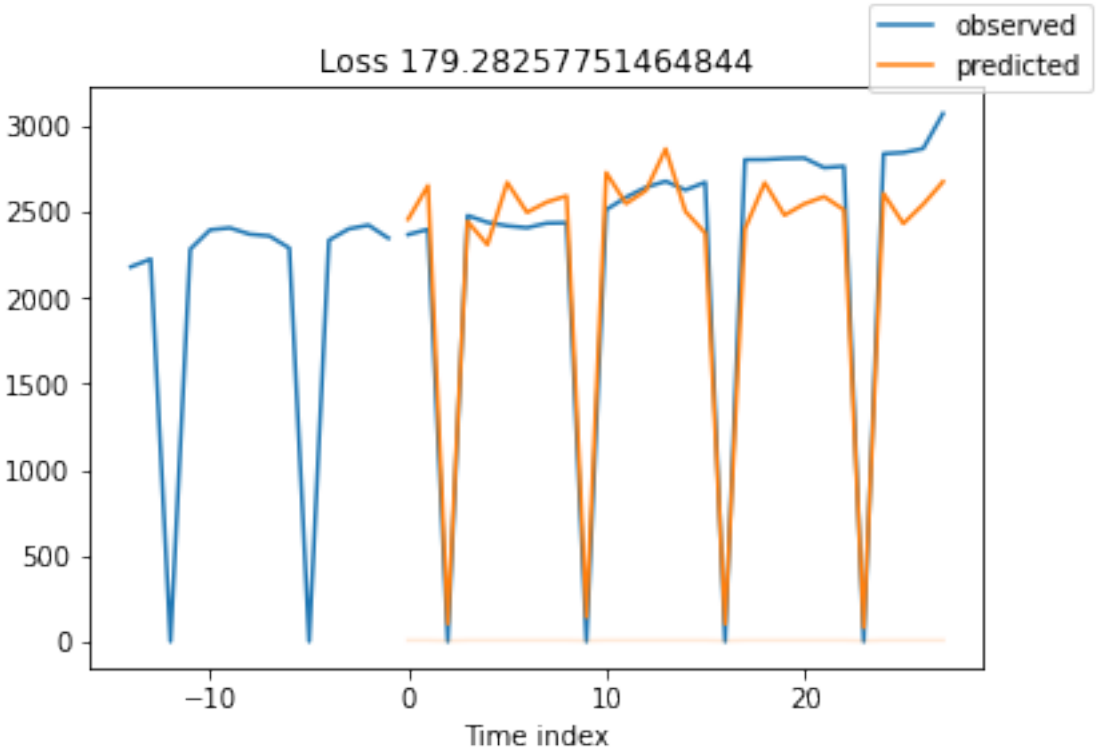

observed
predicted
Loss 175.15171813964844
3000
2500
2000
1500
1000
500
0
−10
0
10
20
Time index

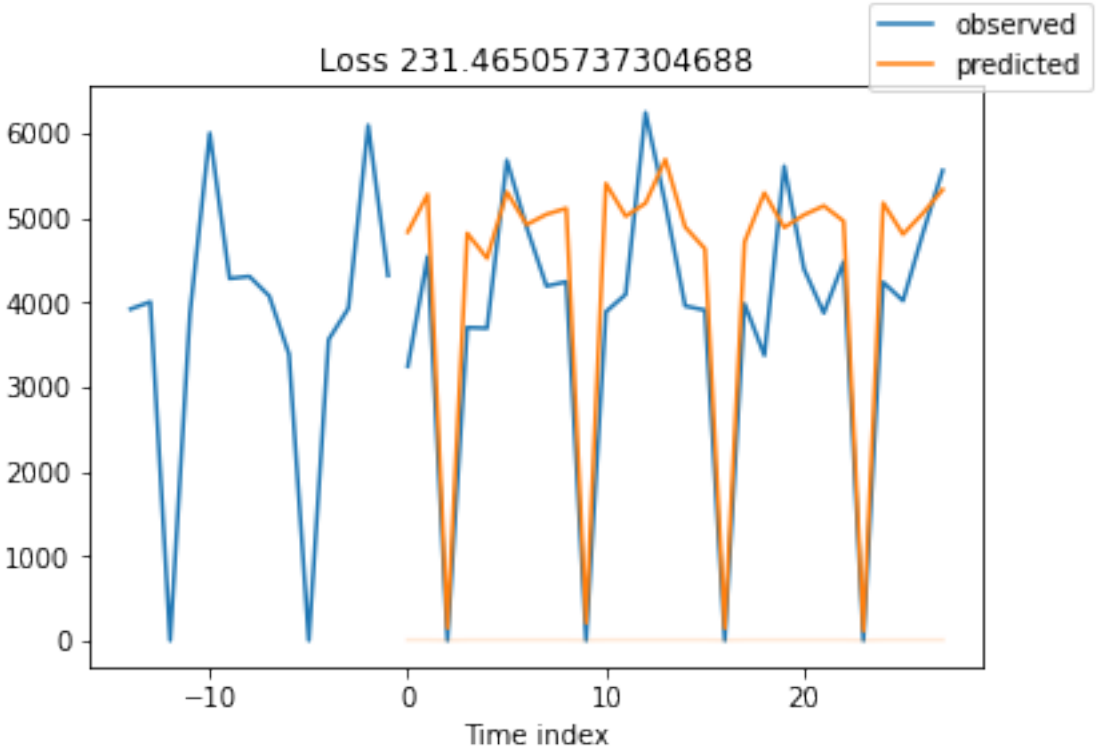

observed
predicted
Loss 190.2730255126953
3500
3000
2500
2000
1500
1000
500
0
−10
0
10
20
Time index

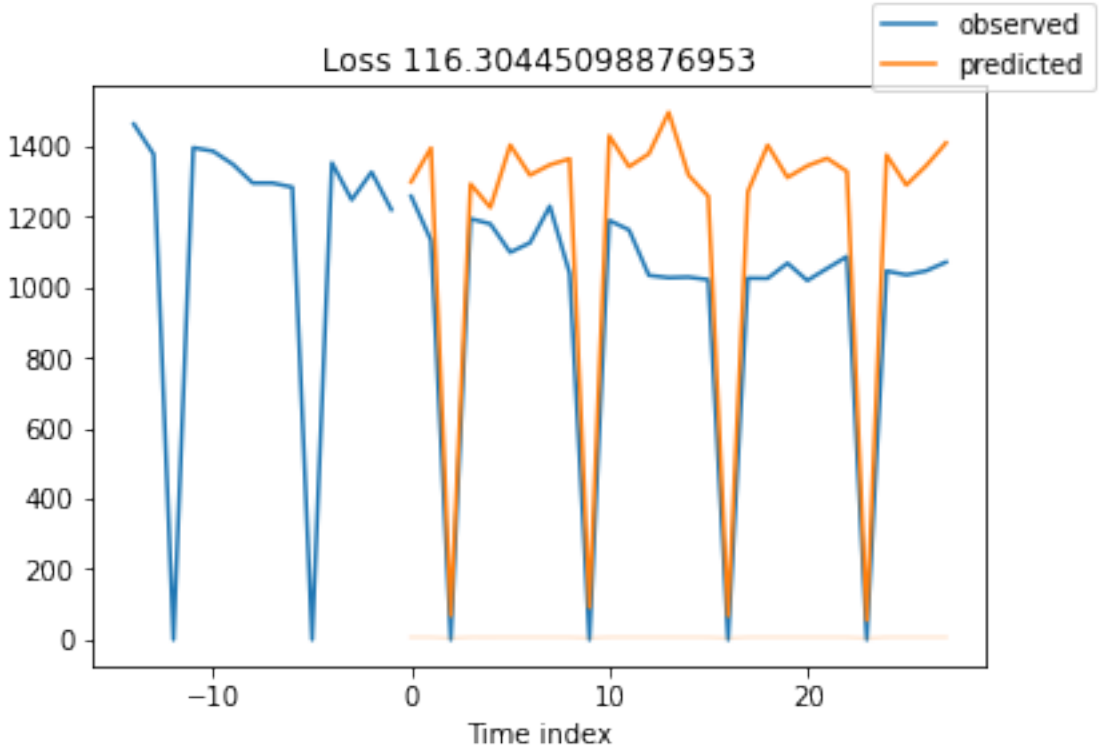

observed
predicted
Loss 210.10353088378906
4000
3000
2000
1000
0
−10
0
10
20
Time index

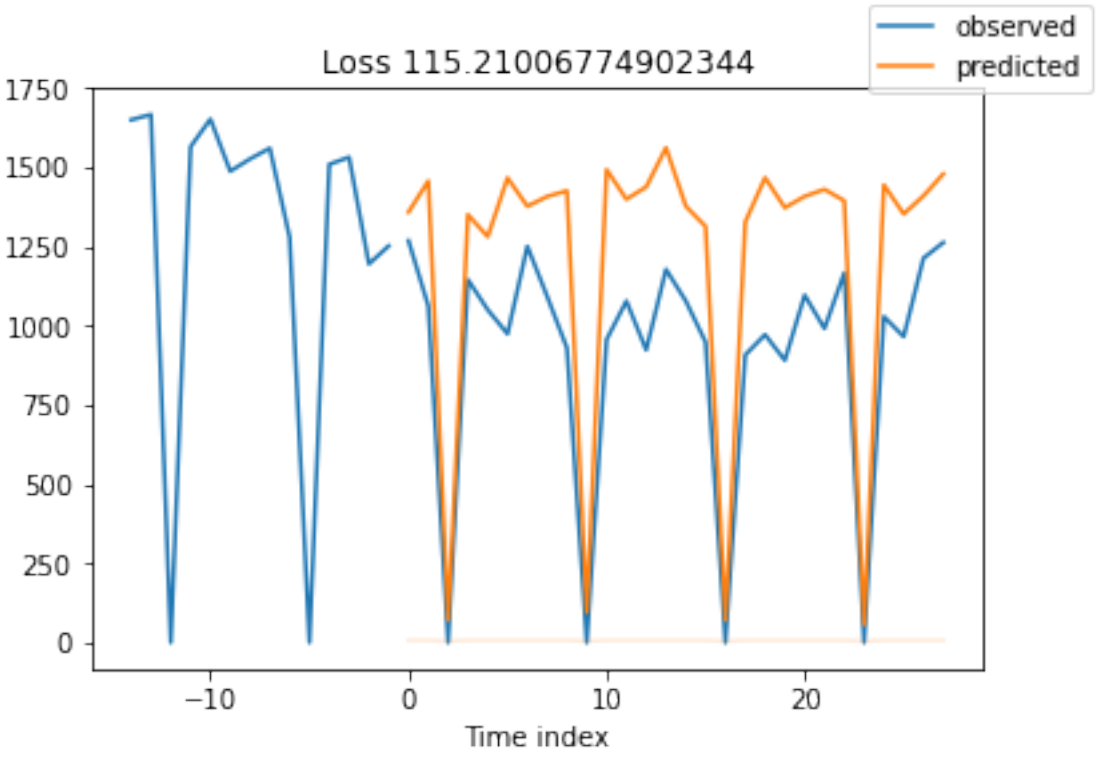

observed
predicted
Loss 299.2132263183594
8000
6000
4000
2000
0
−10
0
10
20
Time index

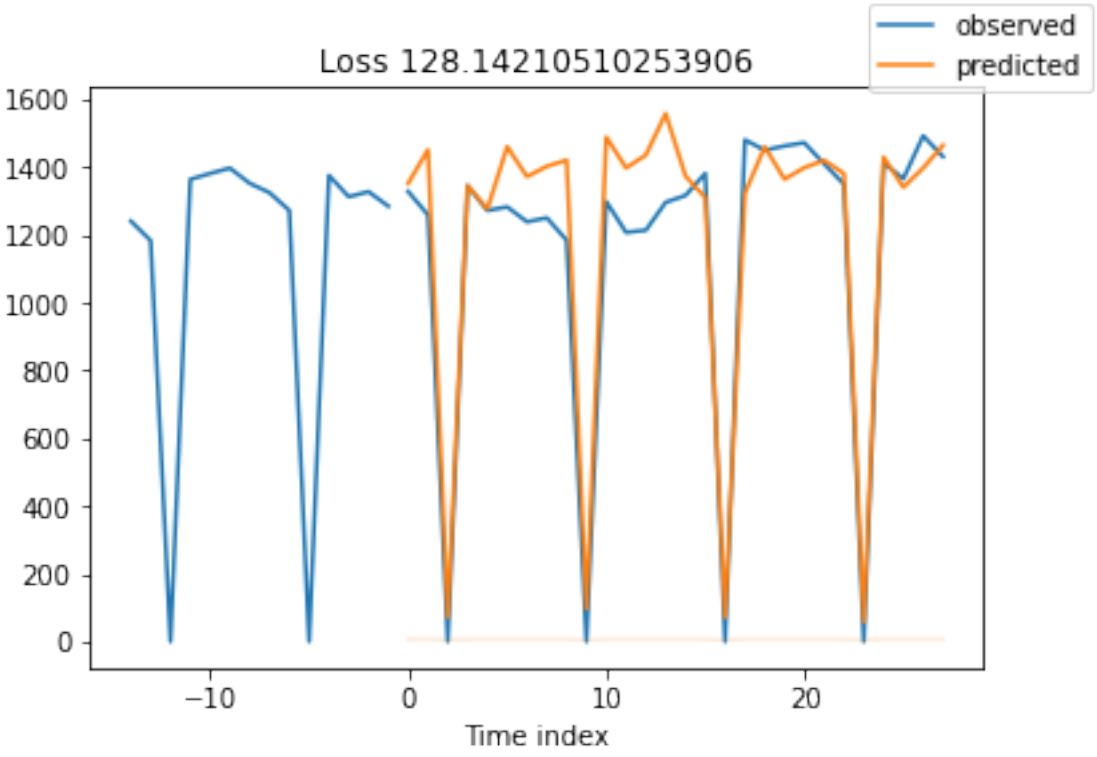

observed
predicted
Loss 188.40377807617188
14000
12000
10000
8000
6000
4000
2000
0
−10
0
10
20
Time index

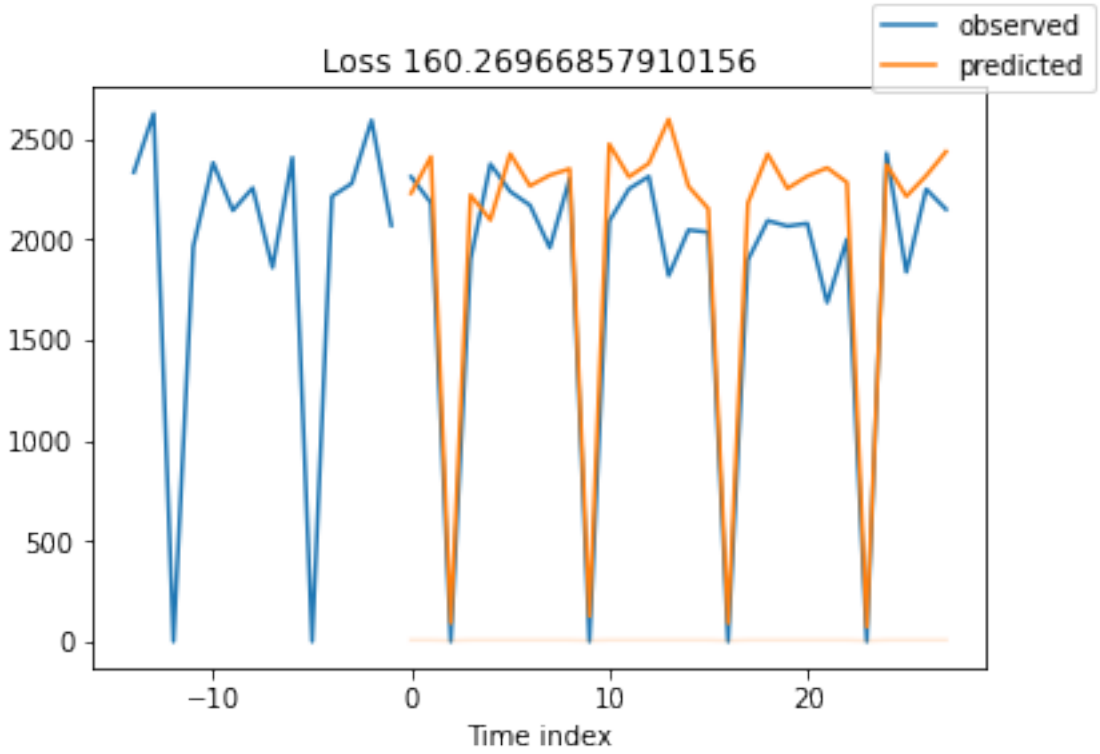

observed
predicted
Loss 179.28257751464844
3000
2500
2000
1500
1000
500
0
−10
0
10
20
Time index

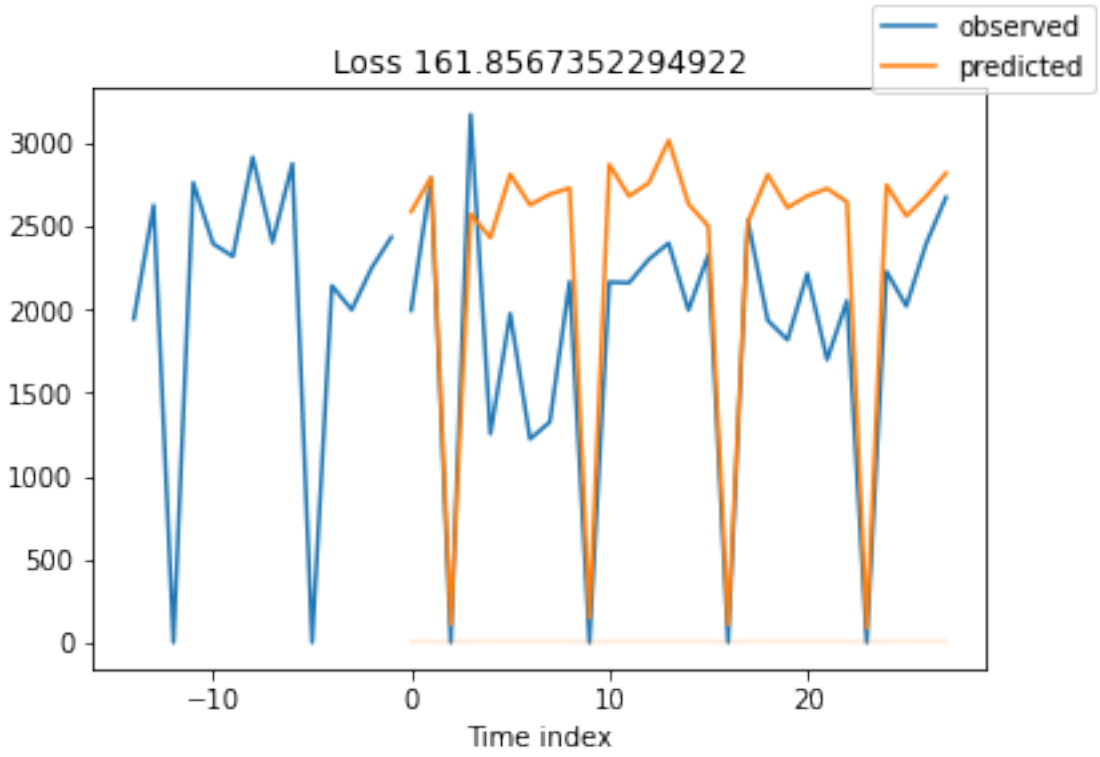

observed
predicted
Loss 231.46505737304688
6000
5000
4000
3000
2000
1000
0
−10
0
10
20
Time index

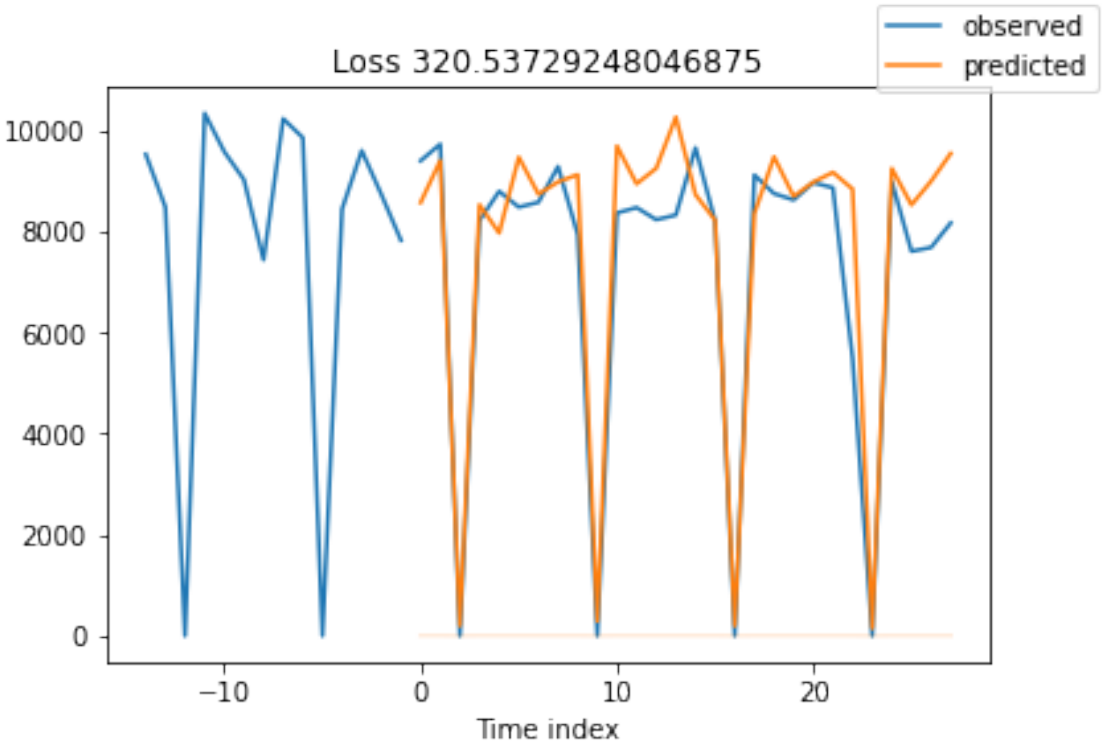

Loss 116.30445098876953
observed
predicted
1400
1200
1000
800
600
400
200
0
−10
0
10
20
Time index

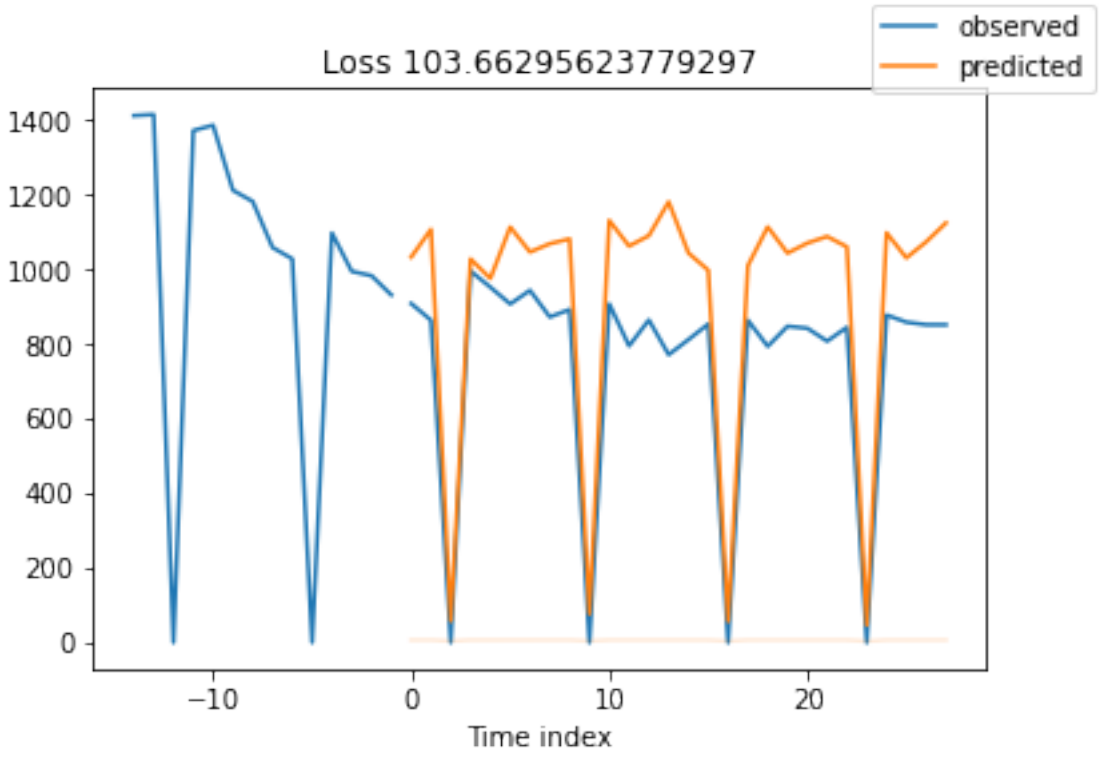

observed
predicted
Loss 115.21006774902344
1750
1500
1250
1000
750
500
250
0
−10
0
10
20
Time index

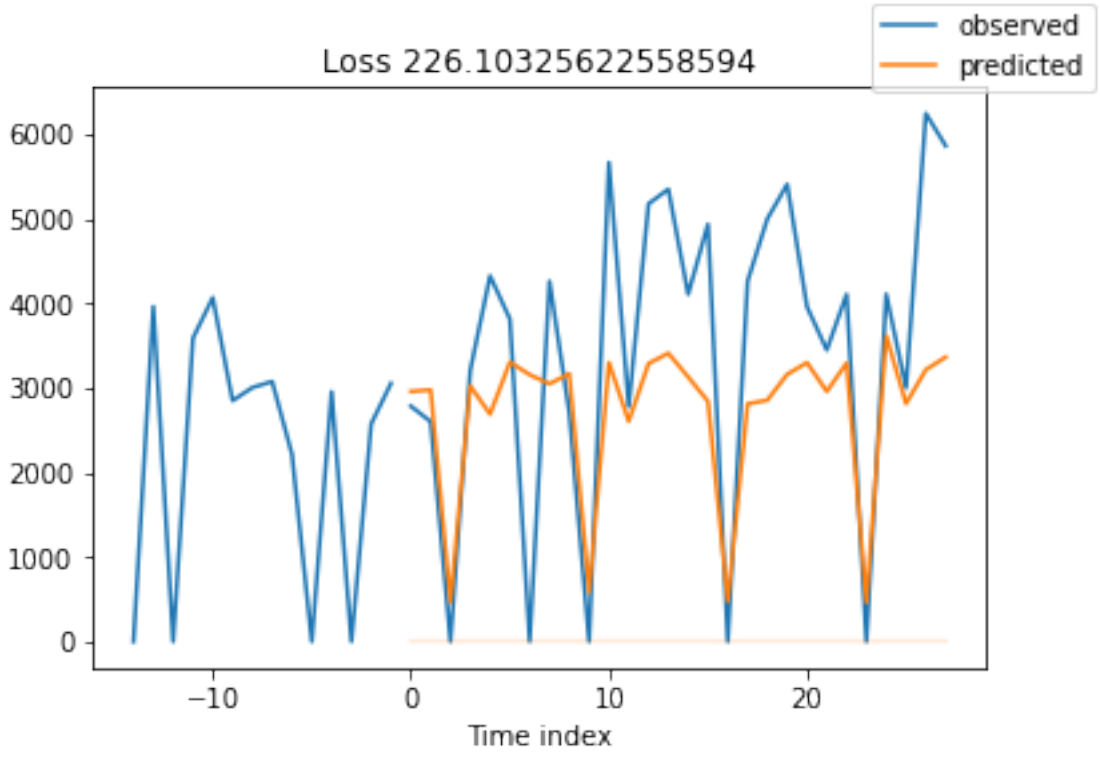

observed
predicted
Loss 128.14210510253906
1600
1400
1200
1000
800
600
400
200
0
−10
0
10
20
Time index

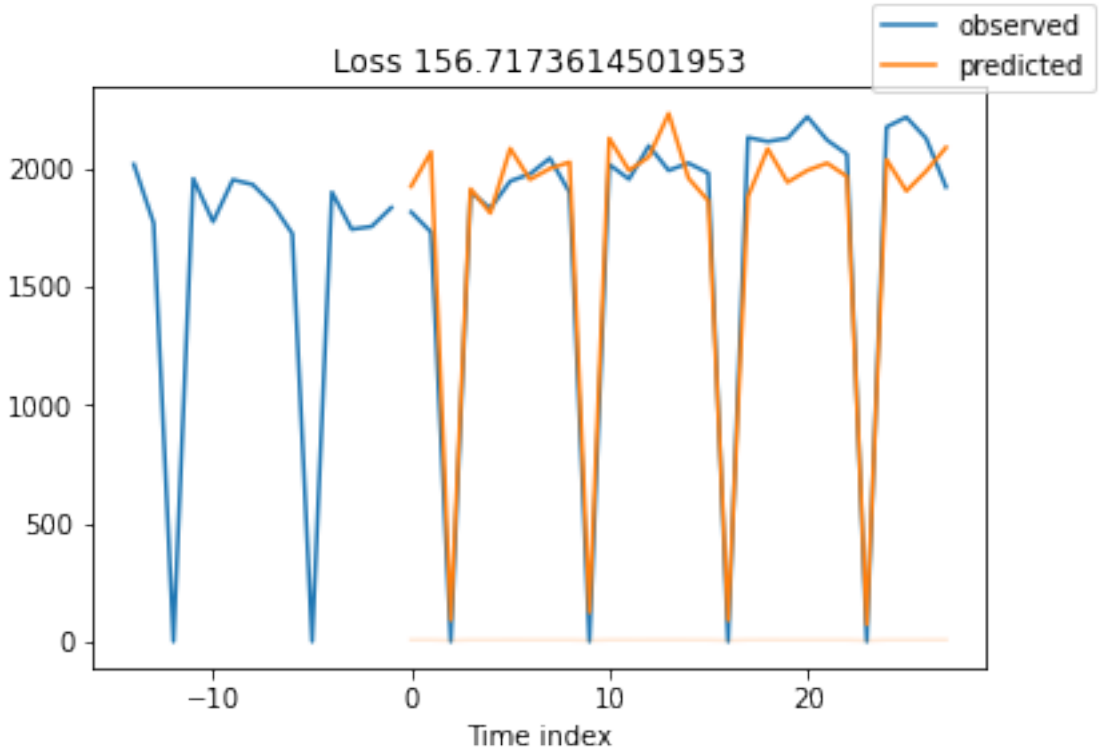

observed
predicted
Loss 160.26966857910156
2500
2000
1500
1000
500
0
−10
0
10
20
Time index

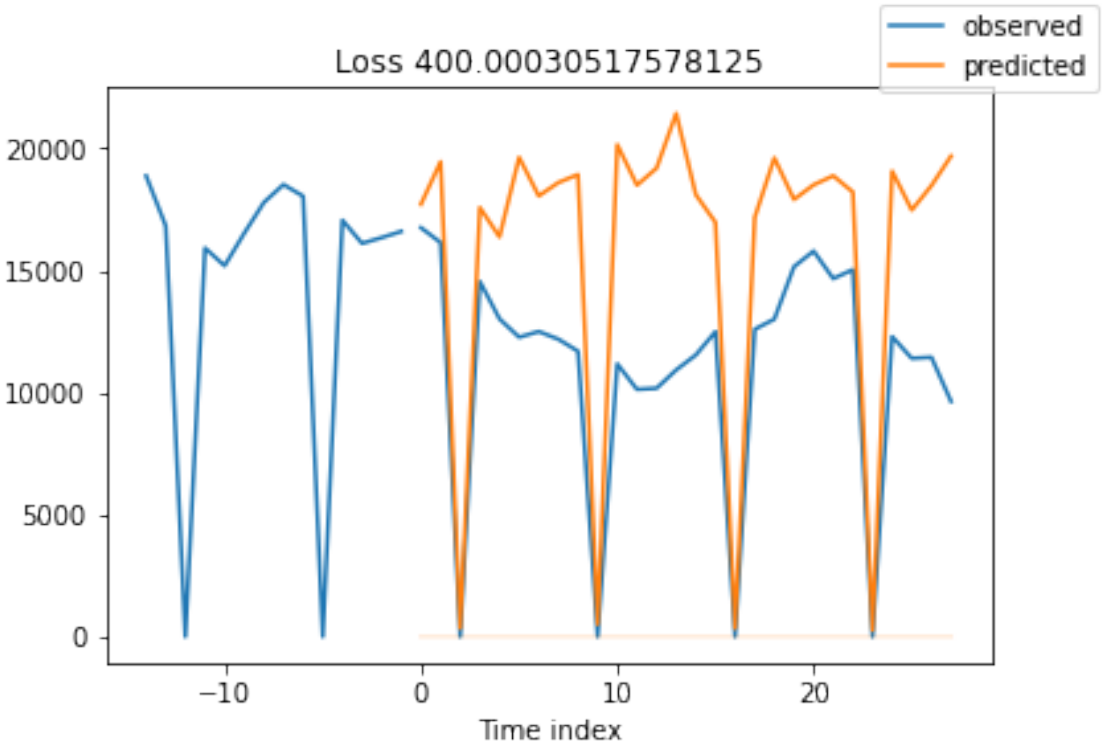

observed
predicted
Loss 161.8567352294922
3000
2500
2000
1500
1000
500
0
−10
0
10
20
Time index

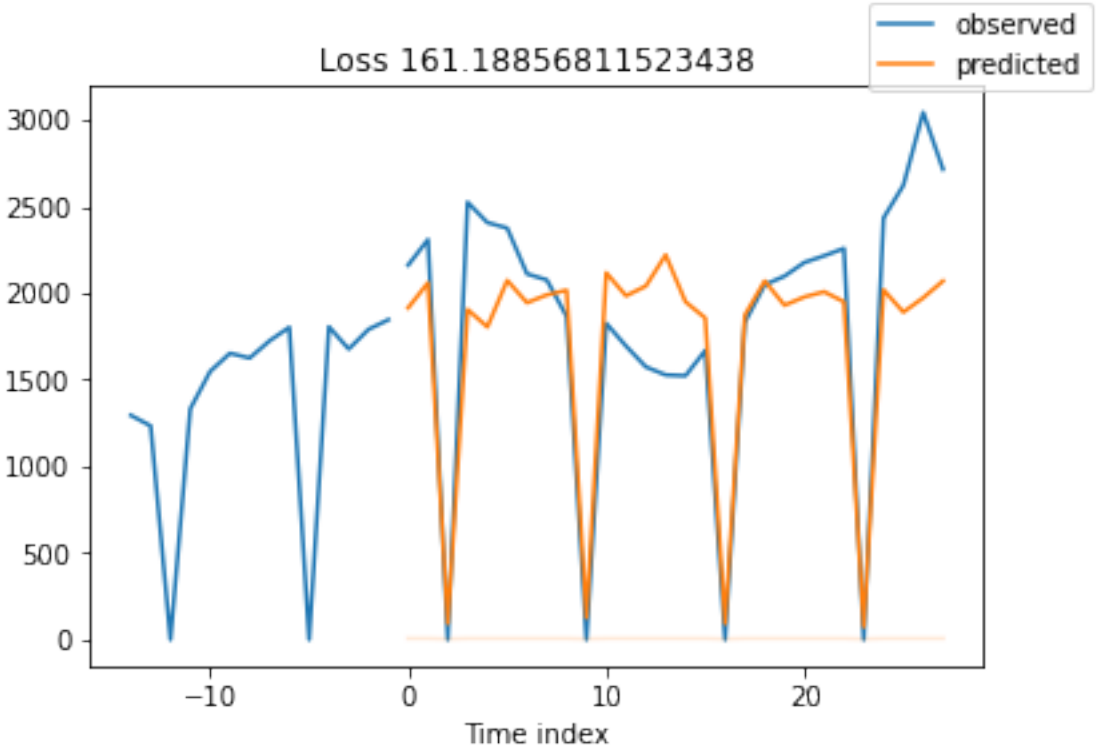

observed
predicted
Loss 320.53729248046875
10000
8000
6000
4000
2000
0
−10
0
10
20
Time index

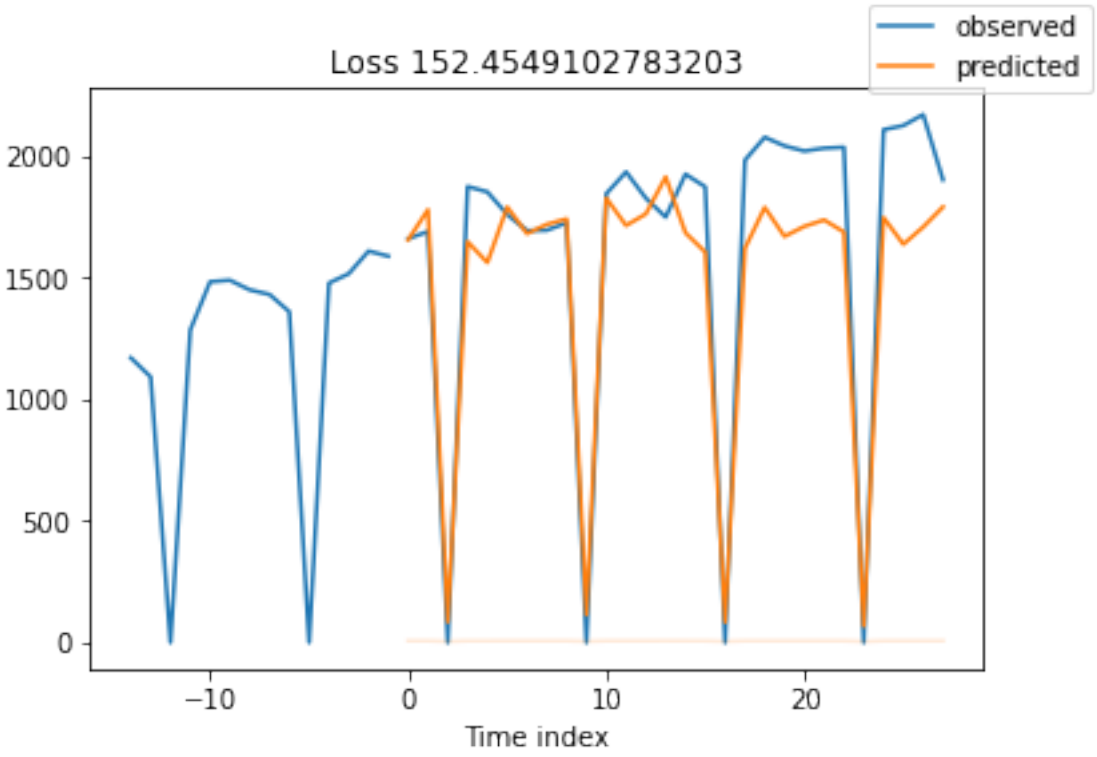

observed
predicted
Loss 103.66295623779297
1400
1200
1000
800
600
400
200
0
−10
0
10
20
Time index

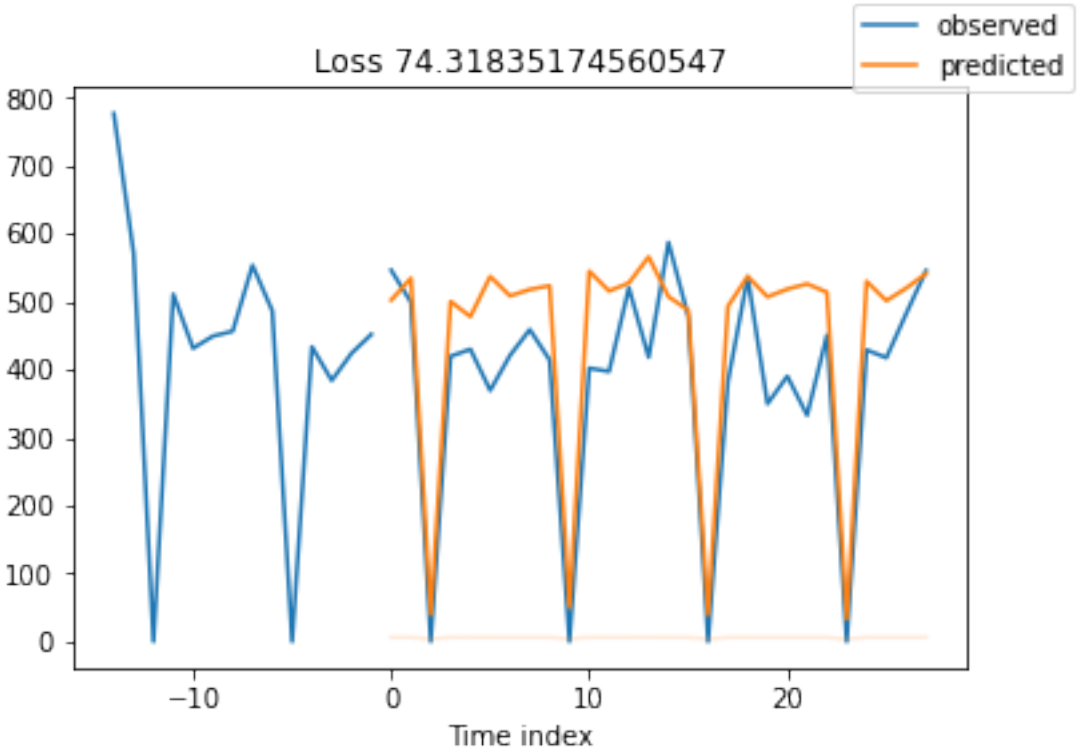

observed
predicted
Loss 226.10325622558594
6000
5000
4000
3000
2000
1000
0
−10
0
10
20
Time index

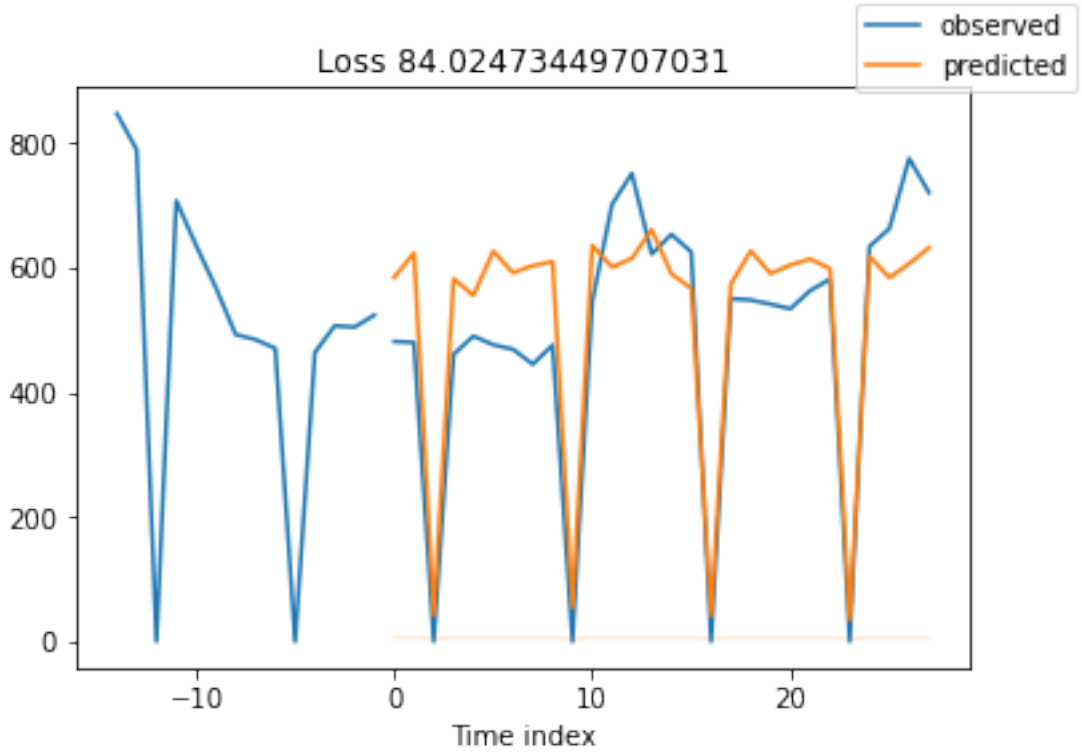

Loss 156.7173614501953
observed
predicted
2000
1500
1000
500
0
−10
0
10
20
Time index

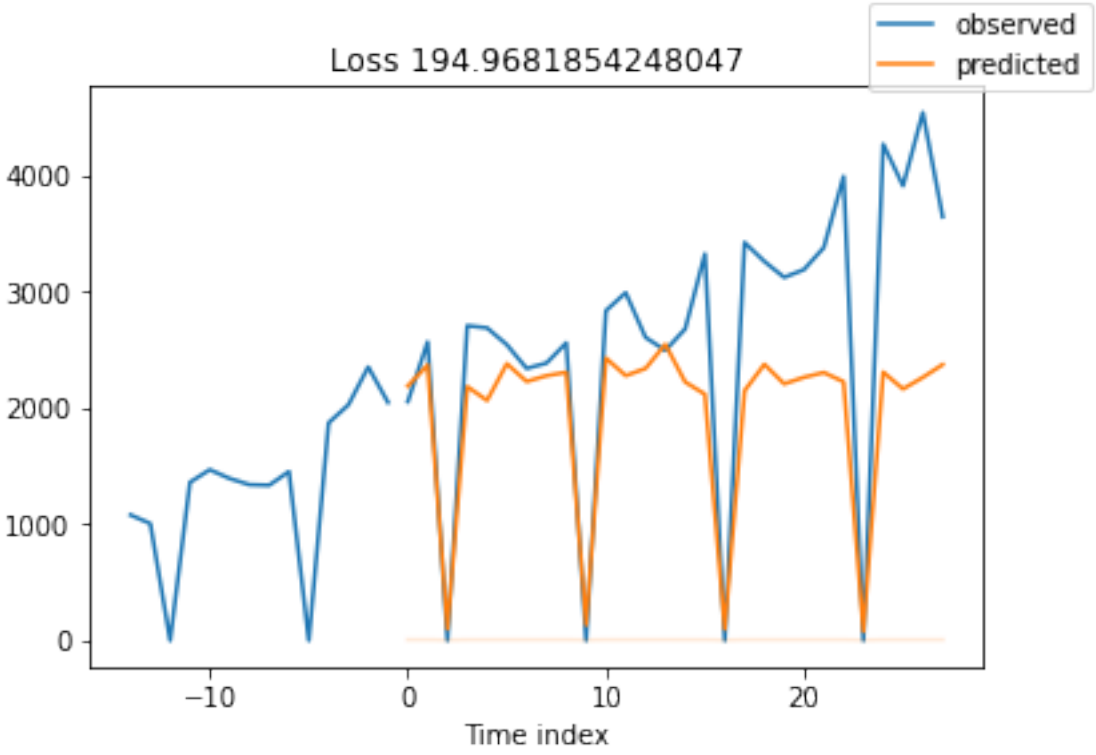

Loss 400.00030517578125
observed
predicted
20000
15000
10000
5000
0
−10
0
10
20
Time index

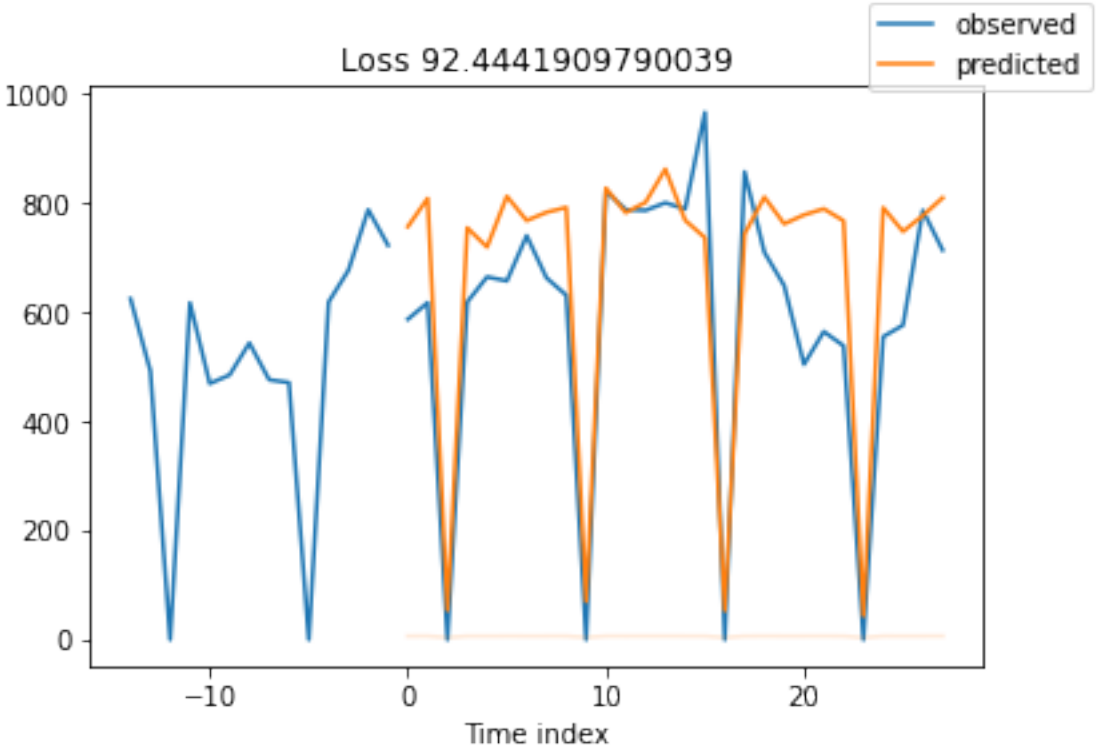

observed
predicted
Loss 161.18856811523438
3000
2500
2000
1500
1000
500
0
−10
0
10
20
Time index

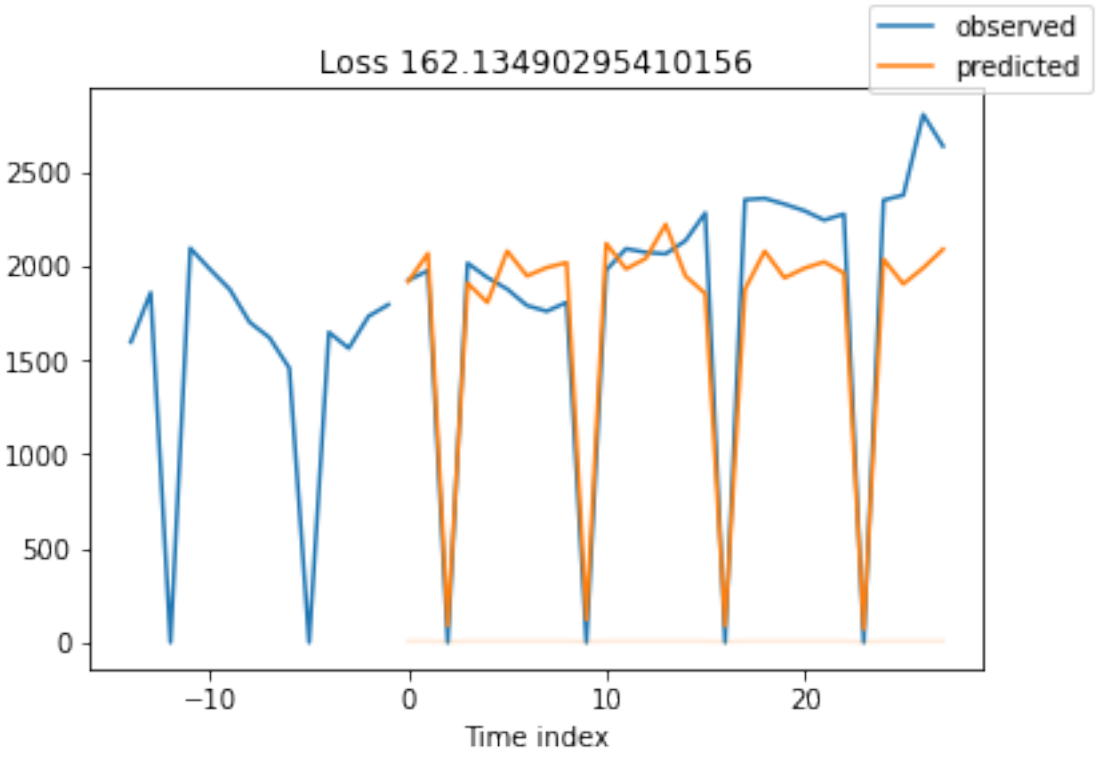

observed
predicted
Loss 152.4549102783203
2000
1500
1000
500
0
−10
0
10
20
Time index

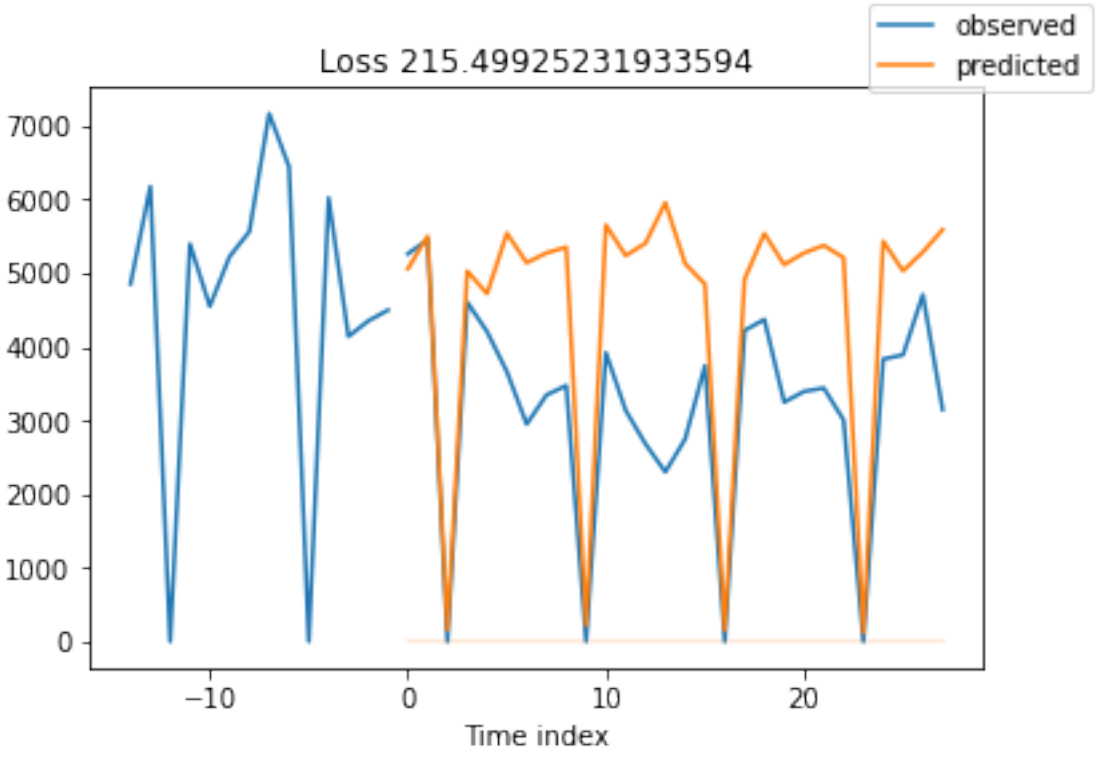

observed
predicted
Loss 74.31835174560547
800
700
600
500
400
300
200
100
0
−10
0
10
20
Time index

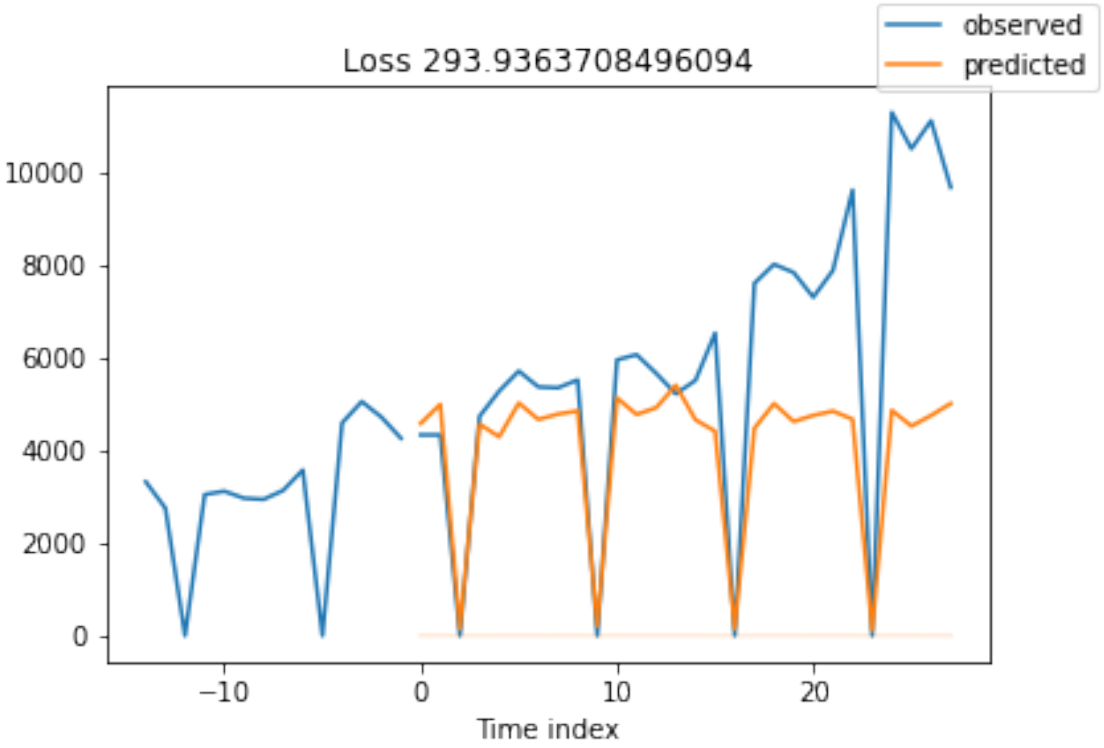

observed
predicted
Loss 84.02473449707031
800
600
400
200
0
−10
0
10
20
Time index

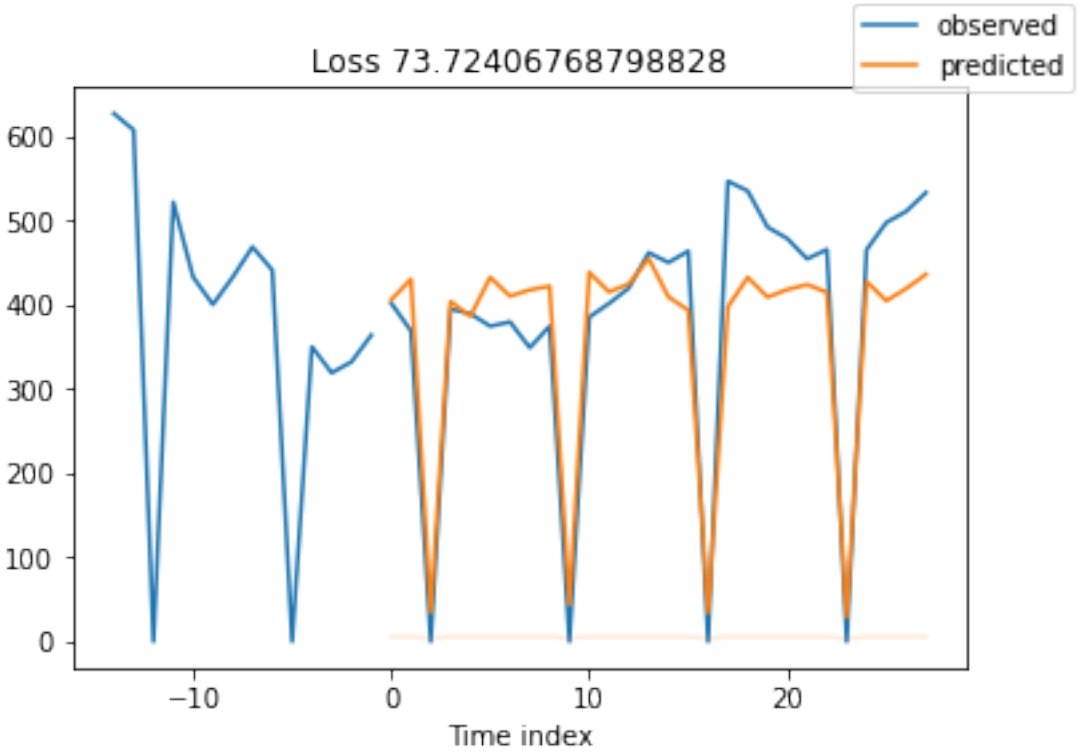

observed
predicted
Loss 194.9681854248047
4000
3000
2000
1000
0
−10
0
10
20
Time index

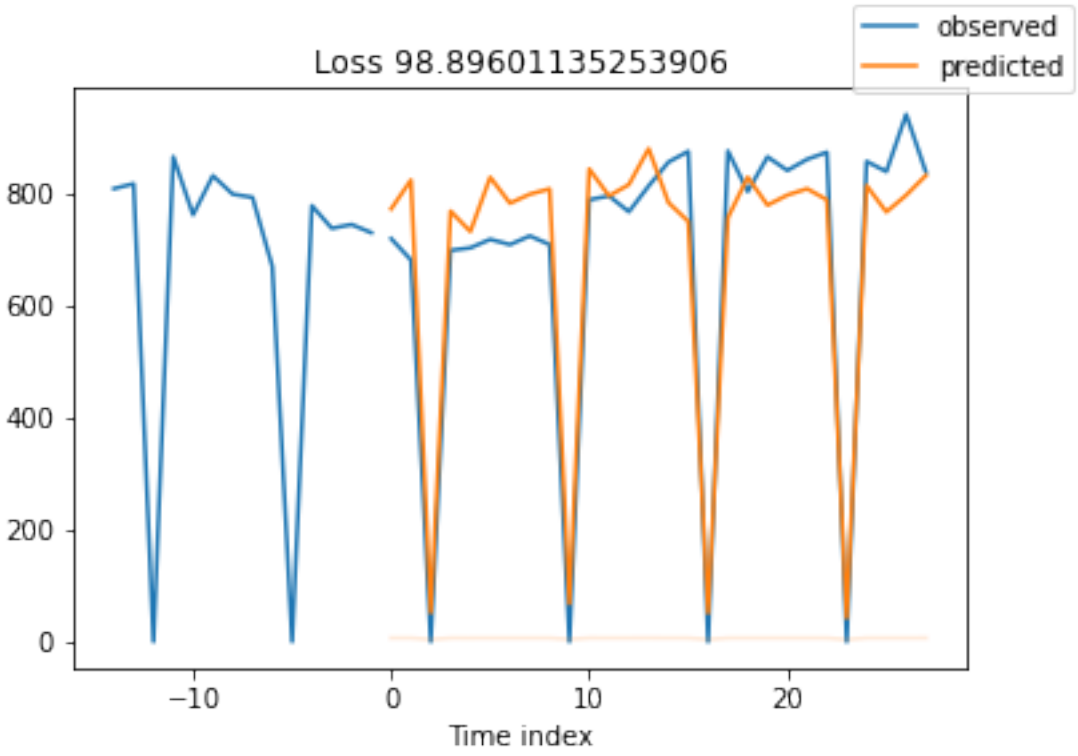

observed
predicted
Loss 92.4441909790039
1000
800
600
400
200
0
−10
0
10
20
Time index

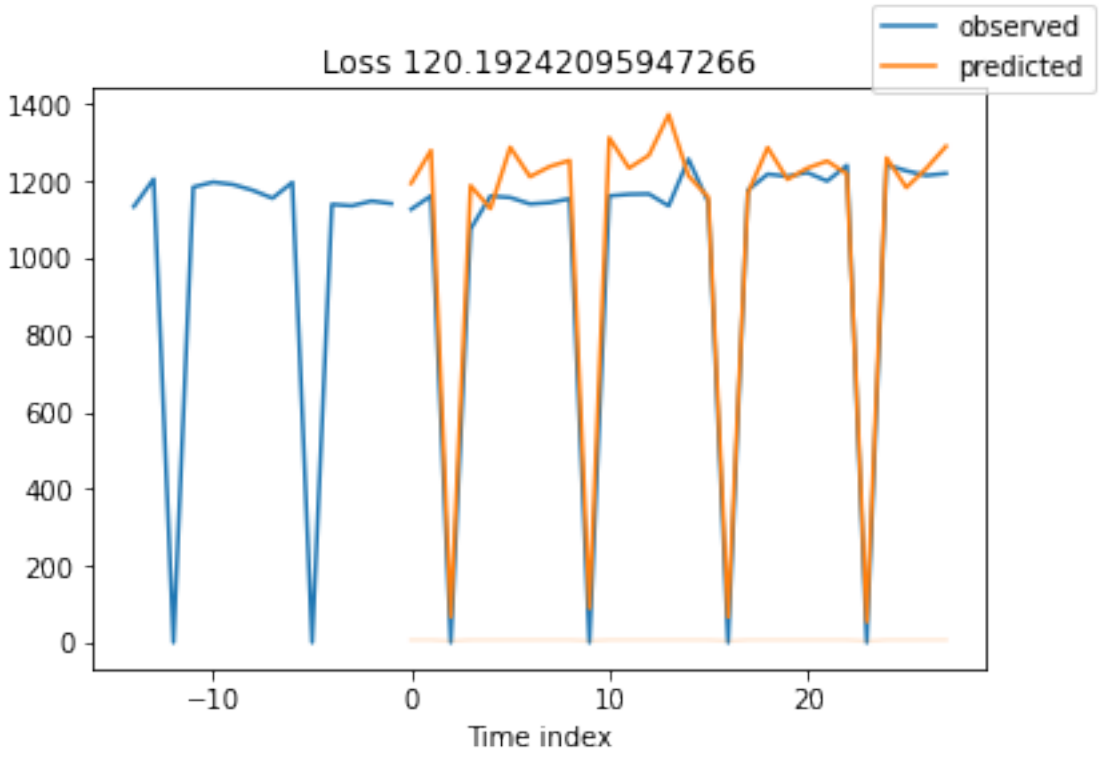

observed
predicted
Loss 162.13490295410156
2500
2000
1500
1000
500
0
−10
0
10
20
Time index

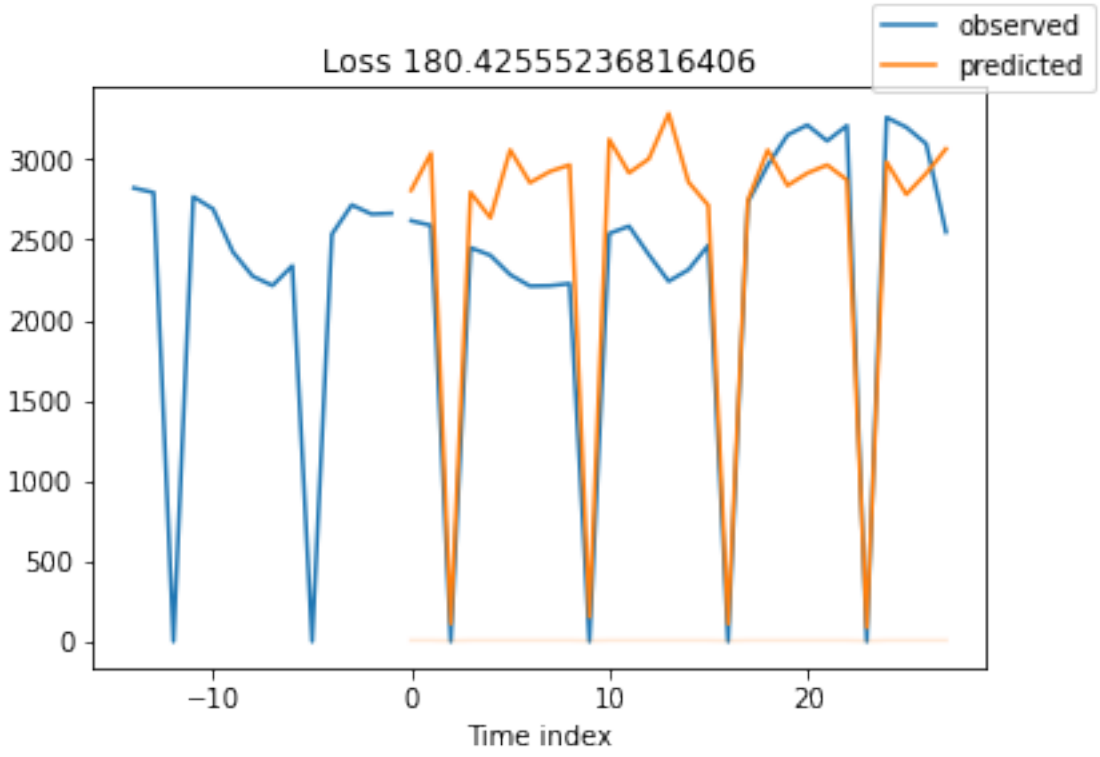

observed
predicted
Loss 215.49925231933594
7000
6000
5000
4000
3000
2000
1000
0
−10
0
10
20
Time index

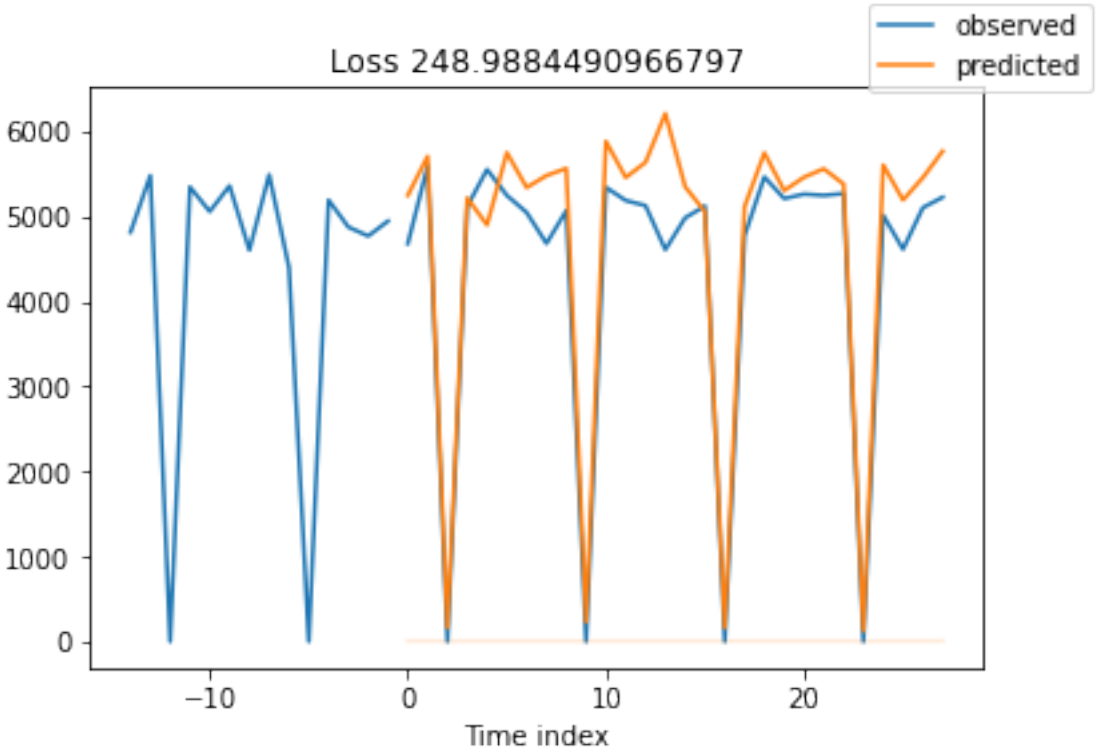

observed
predicted
Loss 293.9363708496094
10000
8000
6000
4000
2000
0
−10
0
10
20
Time index

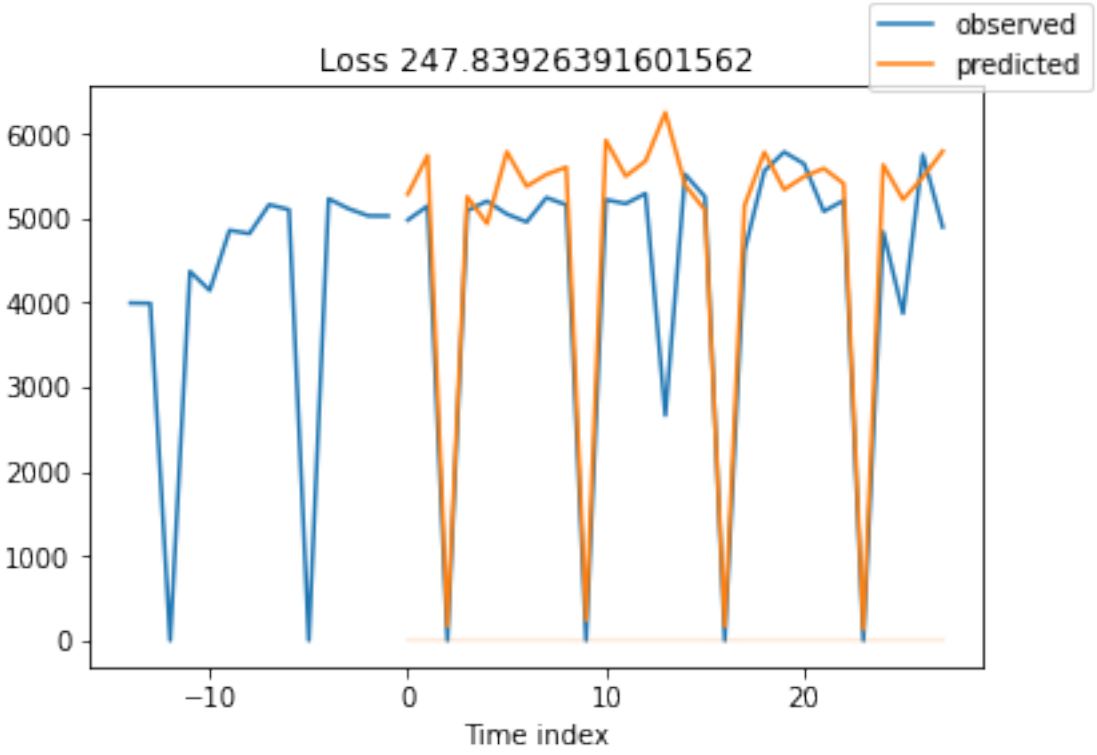

observed
predicted
Loss 73.72406768798828
600
500
400
300
200
100
0
−10
0
10
20
Time index

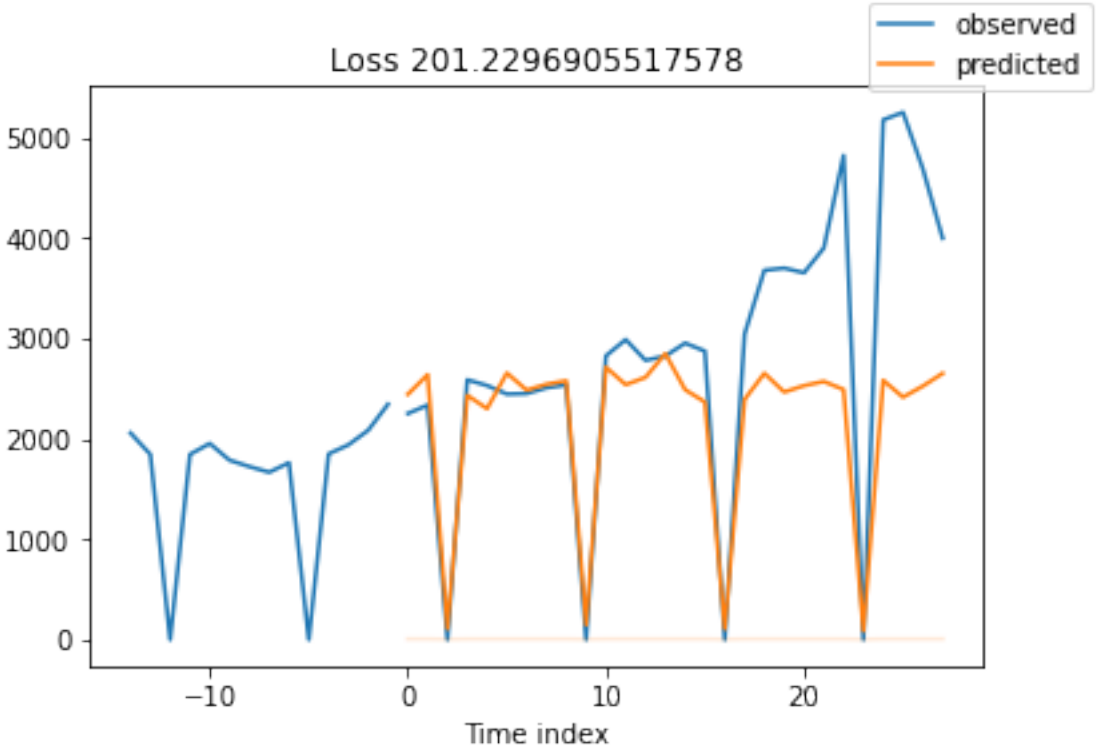

Loss 98.89601135253906
observed
predicted
800
600
400
200
0
−10
0
10
20
Time index

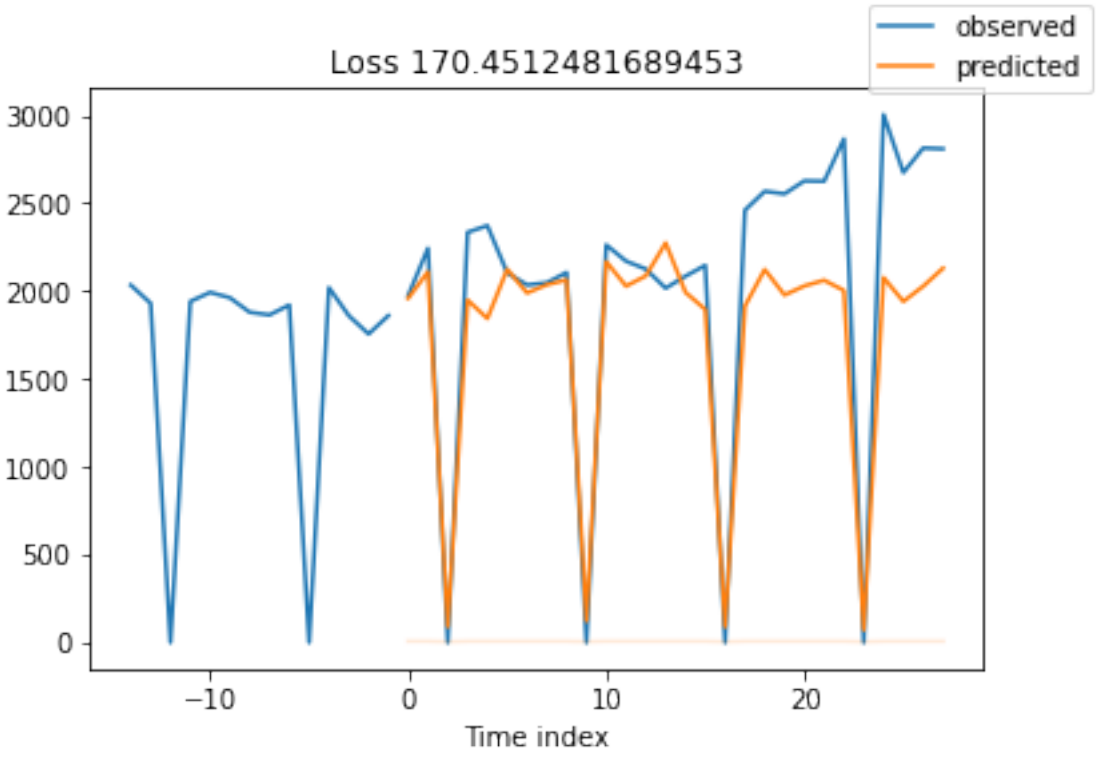

observed
predicted
Loss 120.19242095947266
1400
1200
1000
800
600
400
200
0
−10
0
10
20
Time index

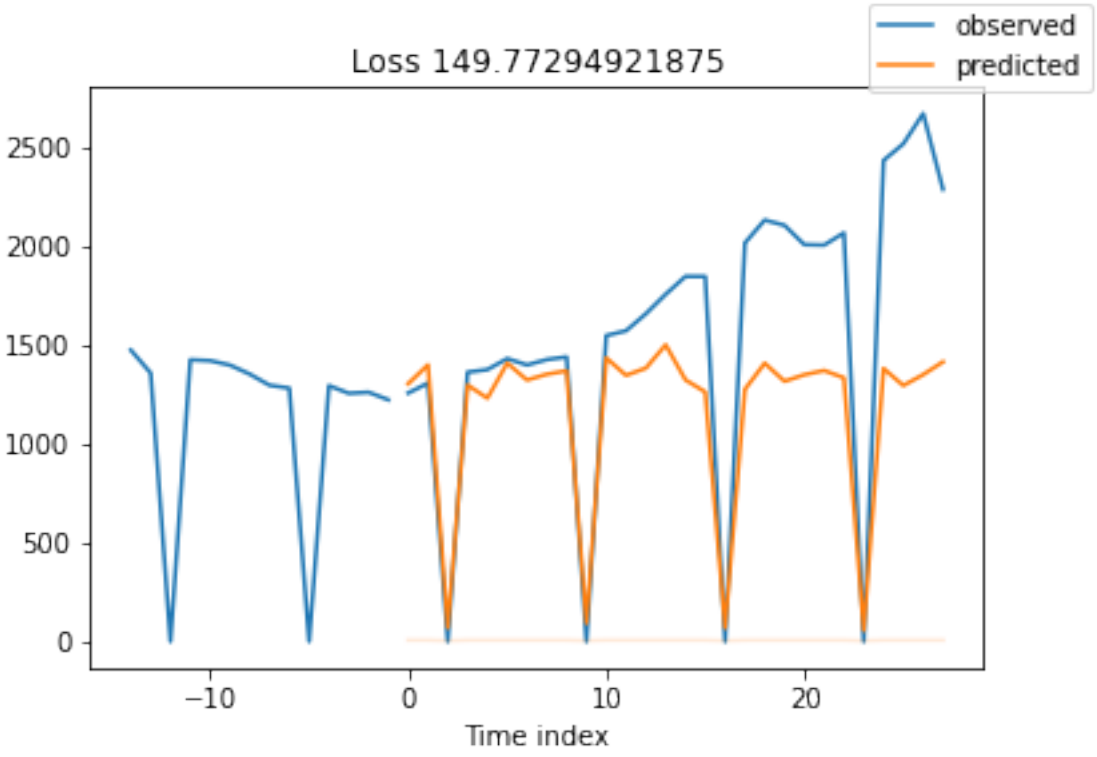

observed
predicted
Loss 180.42555236816406
3000
2500
2000
1500
1000
500
0
−10
0
10
20
Time index


observed
predicted
Loss 248.9884490966797
6000
5000
4000
3000
2000
1000
0
−10
0
10
20
Time index


observed
predicted
Loss 247.83926391601562
6000
5000
4000
3000
2000
1000
0
−10
0
10
20
Time index


observed
predicted
Loss 201.2296905517578
5000
4000
3000
2000
1000
0
−10
0
10
20
Time index


Loss 170.4512481689453
observed
predicted
3000
2500
2000
1500
1000
500
0
−10
0
10
20
Time index


Loss 149.77294921875
observed
predicted
2500
2000
1500
1000
500
0
−10
0
10
20
Time index

[12]:
```
pred_dict = defaultdict(object)
t_dict = defaultdict(list)

for j in range(10):
    test=pd.DataFrame()

    for i in range(37):
        t_temp=pd.read_csv('./data/test/set_0/test_'+str(i)+'.csv')
        t_temp['series'] = int(i)
        t_temp['date']=t_temp['datadate'].apply(lambda x: pd.
 ↪to_datetime(str(x), format='%Y%m%d'))
        t_temp.drop(columns='datadate',inplace=True)
        t_temp['t_diff'] = t_temp['  _    ()'] - t_temp['  _    ()'].shift(1)
        t_temp['month'] = t_temp['date'].dt.month
        t_temp['weekday'] = t_temp['date'].dt.weekday
        t_temp.fillna(0,inplace=True)

        t_temp['ECOD'] = clf_dict[i].decision_function(t_temp[clf_col_dict[i]])

        start_time_idx=(t_temp['date'][0] - df['date'][0]).days
        for k in range(14):
            t_temp.loc[k,'time_idx'] = start_time_idx + k

        t_temp[14:42]=0
        for l in range(28):
            t_temp.loc[14+l,'time_idx'] = start_time_idx+14+l
        t_temp['time_idx']=t_temp['time_idx'].astype(int)
        t_temp['series'] = i
        test = pd.concat((test,t_temp),axis=0)
        t_dict[j].append(t_temp.loc[13,'  _    ()'])

    test[ _cols]=test[ _cols].replace(' ',np.nan)
    test[ _cols]=test[ _cols].astype(float)
    test.fillna(0,inplace=True)

    for col in test.columns:
        if col not in list(df.columns):
            test.drop(columns=col,inplace=True)

    test.reset_index(inplace=True,drop=True)

    pred_dict[j]=best_model.predict(test)
```

[13]:
```
set_dict = defaultdict(object)
for i in range(10):
```

```
    set_dict[i] = pd.DataFrame(pred_dict[i]).transpose()
```

[14]:
```
answer= pd.read_csv('answer_example_correct.csv')
```

[15]:
```
col_names=list(answer.filter(regex=' ').columns)
```

[16]:
```
a = pd.DataFrame(columns=col_names)

for i in range(10):
    for j in range(37):
        set_dict[i][j]=set_dict[i][j] - t_dict[i][j]
        for k in range(14):
            set_dict[i].loc[k,j] = set_dict[i].loc[k,j]/t_dict[i][j]
        set_dict[i].loc[14,j] = np.mean(set_dict[i].loc[21:,j]/t_dict[i][j])
```

[17]:
```
for i in range(10):
    set_dict[i].columns=col_names
    a = pd.concat((a,set_dict[i].loc[:14,:]),axis=0)
```

[18]:
```
a.reset_index(drop=True,inplace=True)
```

[19]:
```
for col in a.columns:
    for i in range(150):
        if np.isnan(answer.loc[i,col]):
            continue
        else:
            answer.loc[i,col] = a.loc[i,col]
```

[20]:
```
answer.to_csv('nhits_tweedie_ECOD.csv',index=False)
```

[ ]:
```
```