W14_LSTM_Improvements-Copy5

February 5, 2021

1 LSTM

Niels van Drunen

Improving the LSTM made in copy3 of this file made by Levy Duivenvoorden & Niels van Schaik

WEIRD STUFF: - Weird negative consumption (x200). (house 28) - LSTM learns some kind of smooth sine. - LSTM uses ENORMOUS amount of memory (500 epochs ~60GB [on cpu]) - Loss function - features??? - epoch loss over training and validating isn't very usefull...

[1]:
```
import random
import time
import torch
import torchvision
import torchvision.transforms as transforms
import numpy as np
import pandas as pd
import math
import random

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import seaborn as sns
import matplotlib.pyplot as plt

from tqdm import tqdm
from IPython.display import clear_output
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, mutual_info_classif
from sklearn.metrics import mean_absolute_error
import wandb

# hier random seeds mee geven
```

```
random.seed(1337)
torch.manual_seed(1337)

%matplotlib inline
%config InlineBackend.print_figure_kwargs={'facecolor' : "w"}

# CUDA initialisation
ngpu = torch.cuda.device_count() # number of available gpus
device = torch.device("cuda:4") if (torch.cuda.is_available() and ngpu > 0)␣
 ↪else "cpu" #cuda:0 for gpu 0, cuda:4 for gpu 5
#device = torch.device("cpu") if (torch.cuda.is_available() and ngpu > 0) else␣
 ↪"cpu" #cuda:0 for gpu 0, cuda:4 for gpu 5
torch.backends.cudnn.benchmark=True # Uses cudnn auto-tuner to find the best␣
 ↪algorithm to use for your hardware
```

2 Data

- Removed cons_T-1,2 … 168

[2]:
```
# #lees de pickles in:
# # df1 = pd.read_pickle('/home/18005152/notebooks/zero/Data:/modelData/_v01_1')
# # df2 = pd.read_pickle('W12_featureselection_h28_onlyShiftedCons2')
# df1 = pd.read_pickle('W14_dataframeConsumption')
# df2 =


# #peaks toevoegen
# tempdf = pd.DataFrame(index = df1.index)
# for i in range(-6,6+1):
#     tempdf[str(i)] = df1['consumption'].shift(periods = i, freq = 'H')
# tempdf['Threshold'] = (tempdf.mean(axis = 1, skipna = True) * 1.30) + (tempdf.
 ↪std(axis = 1, skipna = True) * 1.01)
# df1.loc[df1['consumption'] < tempdf['Threshold'], 'Is_Peak'] = 0
# df1.loc[df1['consumption'] >= tempdf['Threshold'], 'Is_Peak'] = 1

# #Maak de gehele dataframe:
# df = pd.merge(df1[['day_mean','week_mean', 'Is_Peak']], df2, left_index=True,␣
 ↪right_on=df2.index)
# df = df.drop('key_0', axis=1)
# df['Is_Peak2'] = df['Is_Peak']
# df = df.drop('Is_Peak', axis = 1)

# #laat de dataframe zien:
# df.head()
```

[3]:
```
df = pd.read_pickle('W14_dataframeConsumption')
#df['rolling_mean'] = df['consumption'].rolling(4, center=True).mean()
df = df.dropna()
#df['consumption'] = df['rolling_mean']
#df = df.drop('rolling_mean', axis=1)
# df = df.drop(['production'], axis = 1)
df = df.drop("Is_Peak2", axis=1)
df.head(5)
```

[3]:

	consumption	day_mean	week_mean	hour_0	hour_1	hour_2	\
2018-12-31 23:00:00	0.573	0.0	0.0	0	0	0	
2019-01-01 00:00:00	0.435	0.0	0.0	1	0	0	
2019-01-01 01:00:00	0.270	0.0	0.0	0	1	0	
2019-01-01 02:00:00	0.171	0.0	0.0	0	0	1	
2019-01-01 03:00:00	1.423	0.0	0.0	0	0	0	

	hour_3	hour_4	hour_5	hour_6	…	hour_14	hour_15	\
2018-12-31 23:00:00	0	0	0	0	…	0	0	
2019-01-01 00:00:00	0	0	0	0	…	0	0	
2019-01-01 01:00:00	0	0	0	0	…	0	0	
2019-01-01 02:00:00	0	0	0	0	…	0	0	
2019-01-01 03:00:00	1	0	0	0	…	0	0	

	hour_16	hour_17	hour_18	hour_19	hour_20	hour_21	\
2018-12-31 23:00:00	0	0	0	0	0	0	
2019-01-01 00:00:00	0	0	0	0	0	0	
2019-01-01 01:00:00	0	0	0	0	0	0	
2019-01-01 02:00:00	0	0	0	0	0	0	
2019-01-01 03:00:00	0	0	0	0	0	0	

	hour_22	hour_23
2018-12-31 23:00:00	0	1
2019-01-01 00:00:00	0	0
2019-01-01 01:00:00	0	0
2019-01-01 02:00:00	0	0
2019-01-01 03:00:00	0	0

[5 rows x 27 columns]

2.0.1 Features

De data is afkomstig van huis 28. In de pickle wordt de berekende consumptie geresampled naar een uur met de sum methode.

day_mean: het gemiddelde van de dag relatief T-24 - T-48

week_mean: het gemiddelde van de dagen relatief T-24 - T-168

hour_x: het x'de uur van de dag

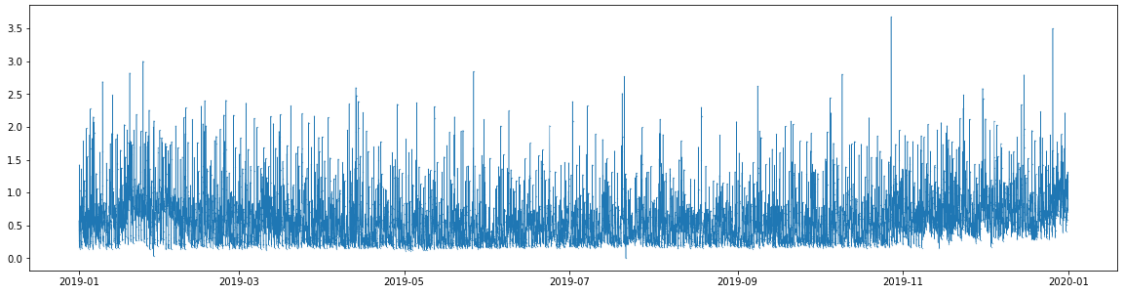
Is_Peak2: is er een piek in de data op dat moment?

2.1 Visualisatie

Geeft de dataset van één heel jaar weer. LET OP: er zitten negatieve (consumptie-)pieken in. Er wordt nog onderzocht waar deze vandaan komen.

[4]:
```
plt.figure(figsize=(20,5))
plt.plot(df.index, df.consumption, ".-", ms=0.5, lw=0.5)
```

[4]:
```
[<matplotlib.lines.Line2D at 0x7f300103d0f0>]
```



[5]:
```
# print('DataFrame Standard Deviation:')
# print(df.std())
# print()
# print('Consumption mean & Consumption Standard Deviation:')
# print(df.consumption.mean(), df.consumption.std())

# _ = plt.hist(df.consumption, bins=50) # edge color toepassen
# plt.title('Consumption Histogram')

# print()
# print('Consumption described:')
# df.describe()
```

3 Data Preparation

Train op de eerste 15 weken.

Daarna valideren op de 16e week.

en dan testen op de 17e week.

[6]:
```
#variabelen
start = 0
end_train = 24*7*15
end_valid = 24*7*16
end_test = 24*7*17

#Split
#Train
dtrain_X = df.iloc[start:end_train].drop('consumption', axis=1)
dtrain_y = df.iloc[start:end_train][['consumption']]
#Valid
dvalid_X = df.iloc[end_train:end_valid].drop('consumption', axis=1)
dvalid_y = df.iloc[end_train:end_valid][['consumption']]
#Test
dtest_X = df.iloc[end_valid:end_test].drop('consumption', axis=1)
dtest_y = df.iloc[end_valid:end_test][['consumption']]
```

3.1 Scaling

Gebruik een Standardscaler om de data individueel te scalen.

[7]:
```
#Train
scaler_xt = StandardScaler()
scaler_xt.fit(dtrain_X[dtrain_X.columns[:-25]])
train_X = pd.concat([pd.DataFrame(scaler_xt.transform(dtrain_X[dtrain_X.
 ↪columns[:-25]]), columns=dtrain_X.columns[:-25], index=dtrain_X.
 ↪index),dtrain_X[dtrain_X.columns[-25:]]], axis=1)

scaler_yt = StandardScaler()
scaler_yt.fit(dtrain_y)
train_y = pd.DataFrame(scaler_yt.transform(dtrain_y), columns=dtrain_y.columns,␣
 ↪index=dtrain_y.index)

#Valid
scaler_xv = StandardScaler()
scaler_xv.fit(dvalid_X[dvalid_X.columns[:-25]])
valid_X = pd.concat([pd.DataFrame(scaler_xv.transform(dvalid_X[dvalid_X.
 ↪columns[:-25]]), columns=dvalid_X.columns[:-25], index=dvalid_X.
 ↪index),dvalid_X[dvalid_X.columns[-25:]]], axis=1)

scaler_yv = StandardScaler()
scaler_yv.fit(dvalid_y)
valid_y = pd.DataFrame(scaler_yv.transform(dvalid_y), columns=dvalid_y.columns,␣
 ↪index=dvalid_y.index)

#Test
```

```
scaler_xtest = StandardScaler()
scaler_xtest.fit(dtest_X[dtest_X.columns[:-25]])
test_X = pd.concat([pd.DataFrame(scaler_xtest.transform(dtest_X[dtest_X.
 ↪columns[:-25]]), columns=dtest_X.columns[:-25], index=dtest_X.
 ↪index),dtest_X[dtest_X.columns[-25:]]], axis=1)

scaler_ytest = StandardScaler()
scaler_ytest.fit(dtest_y)
test_y = pd.DataFrame(scaler_ytest.transform(dtest_y), columns=dtest_y.columns,␣
 ↪index=dtest_y.index)
```

3.2 Zet de Datasets om naar Tensors

De Tensors zijn in de vorm van Doubles gezet, dit zou de LSTM helpen.

[8]:
```
#Make tensors from the numpy arrays.
train_X_t = torch.from_numpy(np.array(train_X)).to(device).double()
train_y_t = torch.from_numpy(np.array(train_y)).to(device).double()

valid_X_t = torch.from_numpy(np.array(valid_X)).to(device).double()
valid_y_t = torch.from_numpy(np.array(valid_y)).to(device).double()

test_X_t = torch.from_numpy(np.array(test_X)).to(device).double()
test_y_t = torch.from_numpy(np.array(test_y)).to(device).double()
```

4 LSTM Neural Network

4.1 Class

[9]:
```
#Parameters:
layerSize = 32
outputSize = 1
featureSize = train_X_t.shape[1]

#class maken voor NN
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.hidden_state_size = layerSize
        self.lstm1 = nn.LSTMCell(featureSize, layerSize)
        self.lstm2 = nn.LSTMCell(layerSize, layerSize)
        self.linear1 = nn.Linear(layerSize, layerSize)
        self.linear2 = nn.Linear(layerSize, outputSize)
```

```
    def forward(self, input_, future = 0):
        outputs = []
        h_t1 = torch.zeros(input_.size(1), self.hidden_state_size, dtype=torch.
↪double).to(device)
        c_t1 = torch.zeros(input_.size(1), self.hidden_state_size, dtype=torch.
↪double).to(device)
        h_t2 = torch.zeros(input_.size(1), self.hidden_state_size, dtype=torch.
↪double).to(device)
        c_t2 = torch.zeros(input_.size(1), self.hidden_state_size, dtype=torch.
↪double).to(device)

        #connect and train LSTM cells
        k = 1
        for input_t in [input_[i] for i in range(input_.shape[0]-future)]:
            h_t1, c_t1 = self.lstm1(input_t, (h_t1, c_t1))
            h_t2, c_t2 = self.lstm2(h_t1, (h_t2, c_t2))

            output = self.linear2(self.linear1(h_t2))
            outputs += [output]
            k += 1
        #prediction
        for i in range(input_.shape[0]-future,input_.shape[0]): # feed h_t, c_t␣
↪and output to itself to make future predictions
            #print(k)
            h_t1, c_t1 = self.lstm1(input_[i], (h_t1, c_t1))
            h_t2, c_t2 = self.lstm2(h_t1, (h_t2, c_t2))
            output = self.linear2(self.linear1(h_t2))
            outputs += [output]
            #print(type(outputs))
            k += 1
        outputs = torch.cat(outputs, dim=1)
        return outputs
```

4.1.1 Create class object

[10]:
```
model = Net().to(device)
model.double()
```

[10]:
```
Net(
  (lstm1): LSTMCell(26, 32)
  (lstm2): LSTMCell(32, 32)
  (linear1): Linear(in_features=32, out_features=32, bias=True)
  (linear2): Linear(in_features=32, out_features=1, bias=True)
)
```

4.2 Train LSTM

TODO - data opsplitsen in partjes van 8 - meerdere huizen inladen (10) en opslaan in tensor

[11]:
```
#without moving window:
def moving_window(data, target):
    """
    Makes an moving window from the given dataframe.
    """
    #1e tijd, 2e samples, 3e features
    temp = data.reshape(data.shape[0], 1, data.shape[1])
    return temp, target
```

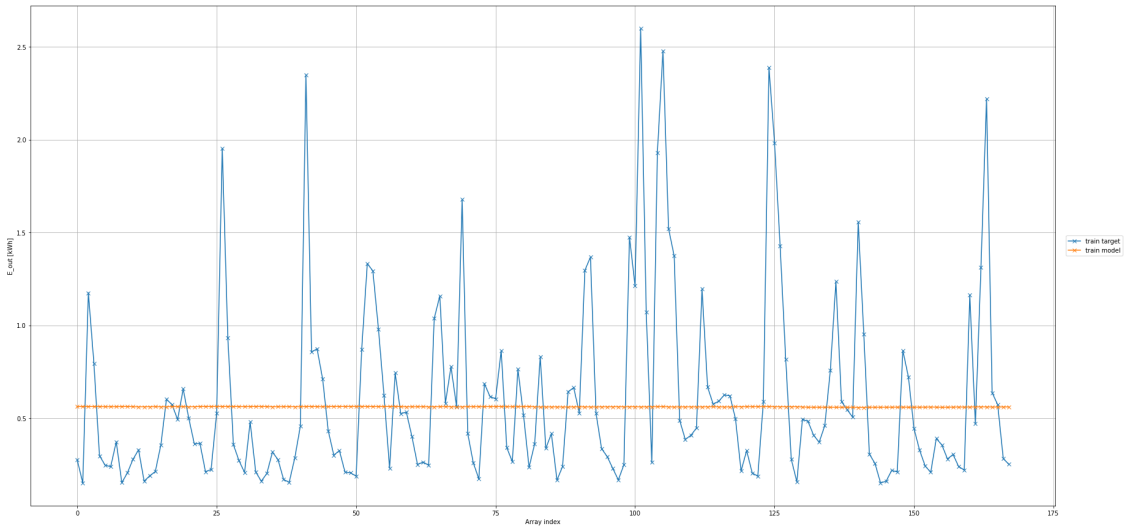
[12]:
```
#with moving window of 3:
# def moving_window(data, target):
#     """
#     Makes an moving window from the given dataframe.
#     """
#     #1e tijd, 2e samples, 3e features
#     data = data.reshape(data.shape[0], 1, data.shape[1])
#     temp = torch.zeros([data.shape[0]-3,3,data.shape[2]], dtype=torch.double).
 ↪to(device)
#     temp = temp.reshape(temp.shape[0], 3, temp.shape[2])
#     for i in range(0,len(data)-3):
#         temp[i] = torch.cat((data[i],data[i+1],data[i+2]),0)
#     return temp, target[3:]
```

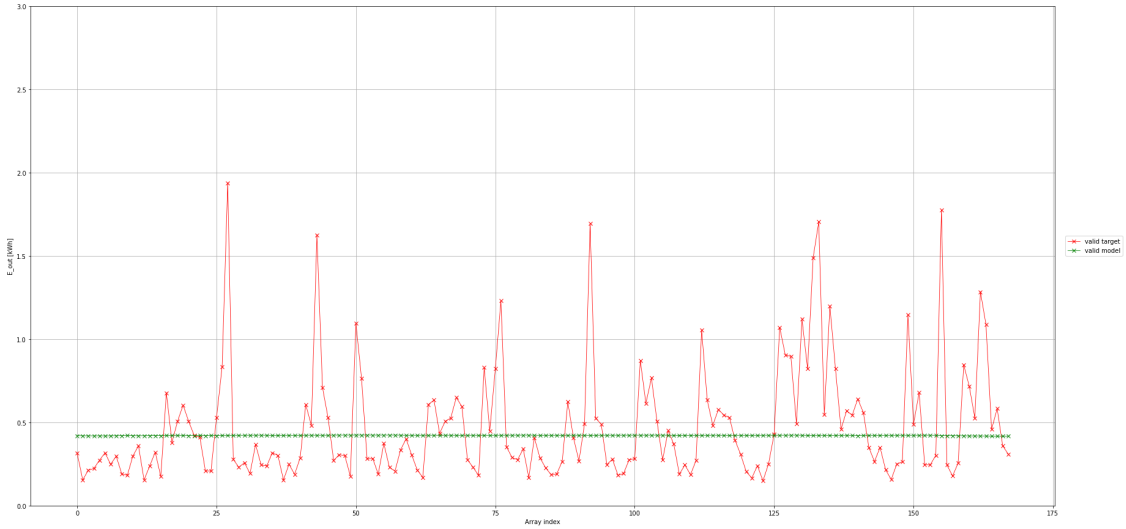
[13]:
```
#initialize:
train_for = 250 #amount of epochs
model_future = 168 #future for the model
show_every = 1
alijst = []; blijst = []; lr_lijst = []

#Training parameters:
optimizer = optim.Adam(model.parameters(), lr=3e-3)
criterion = nn.MSELoss()
#criterion = nn.SmoothL1Loss()

#Learning loop:
for i in (range(train_for)):

    """
    Training
    """
    btime = time.time()
    model.train()
```

```
    #reshape data:
    data = train_X_t; target = train_y_t
    data, target = moving_window(train_X_t, target)

    #train LSTM and reshape output:
    optimizer.zero_grad()
    output = model(data, future=model_future)
    output = output[0].reshape(output.shape[1],1)
    output = output[:,-model_future:]
    target = target[:,-model_future:]

    #bereken de loss over de output en update de parameters:
    loss = criterion(output, target)
    loss.backward()
    optimizer.step()
    etime = time.time()



    """
    Evaluation
    """
    model.eval()
    if (i%show_every) == 0:
        #bereken the MAE over the training dataset:
        yt = scaler_yt.inverse_transform(target.cpu().detach().numpy())
        yhatt = scaler_yt.inverse_transform(output.cpu().detach().numpy())
        mset = mean_squared_error(yhatt, yt)

        #concatenate de Xtrain en Xvalidatie zodat de LSTM zo veel mogelijk␣
↪data heeft.
        temp, _ = moving_window(train_X_t, target)
        temp0, _ = moving_window(valid_X_t, target)
        data = torch.cat((temp,temp0),0)
        target = torch.cat((train_y_t,valid_y_t),0)

        #doe de data in de LSTM en kijk wat eruit komt:
        output = model(data, future=model_future)
        output = output[0].reshape(output.shape[1],1)
        output = output[:,-model_future:]
        target = target[:,-model_future:]

        #bereken the MAE over the validatie dataset:
        yv = scaler_yv.inverse_transform(target.cpu().detach().numpy())
        yhatv = scaler_yv.inverse_transform(output.cpu().detach().numpy())
        msev = mean_squared_error(yhatv, yv)
```

```
        #print de loss per epoch:
        print('Epoch: %d\t MSE Train: %.3f\t MSE Valid: %.3f\t Looptime: %.3f␣
↪s' % (i,mset,msev,etime-btime))

        #voeg de losses toe aan de lijsten:
        alijst.append(loss)
        blijst.append(criterion(output,target))

    #plot every 10*show_every de resultaten:
    if i%(show_every*10) == 0:
        #show the training dataset:
        plt.subplots(figsize=(30,15))
        plt.plot(range(len(yt[-model_future:])),yt[-model_future:
↪],'x-',label="train target")
        plt.plot(range(len(yt[-model_future:])),yhatt[-model_future:
↪],'x-',label="train model")
        plt.legend()
        plt.grid()

        plt.xlabel("Array index")
        plt.ylabel("E_out [kWh]")
        plt.legend(loc=(1.01, 0.5))
        plt.show()

        #show the validation dataset:
        plt.subplots(figsize=(30,15))
        plt.plot(range(len(yv[-model_future:])),yv[-model_future:
↪],'x-',label="valid target",color='red',lw=0.8)
        plt.plot(range(len(yv[-model_future:])),yhatv[-model_future:
↪],'x-',label="valid model",color='green',lw=0.8)
        plt.legend()
        plt.grid()
        plt.ylim([0,3.0])

        plt.xlabel("Array index")
        plt.ylabel("E_out [kWh]")
        plt.legend(loc=(1.01, 0.5))
        plt.show()
```

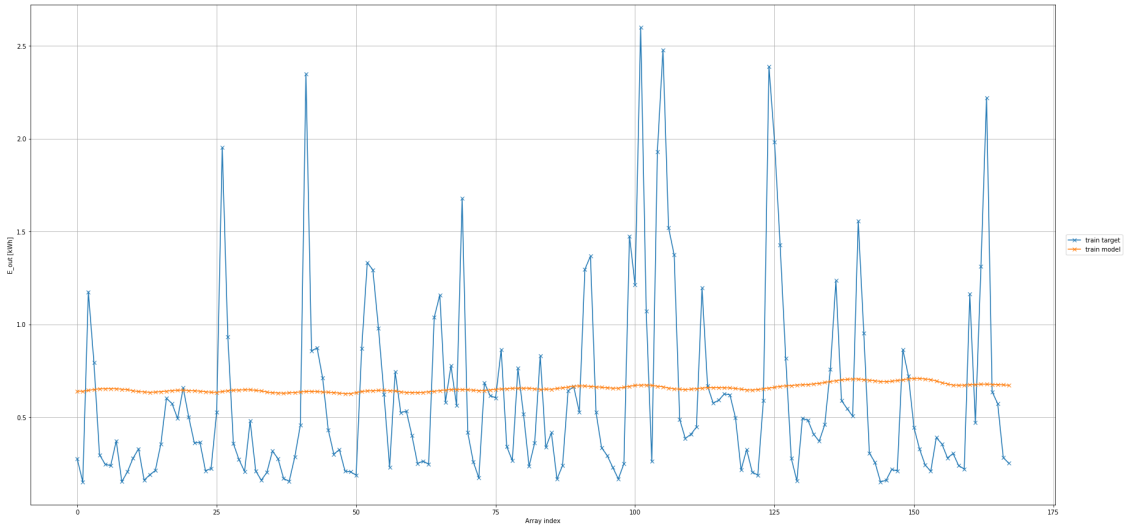
```
Epoch: 0         MSE Train: 0.185        MSE Valid: 0.123        Looptime: 2.611
s
```

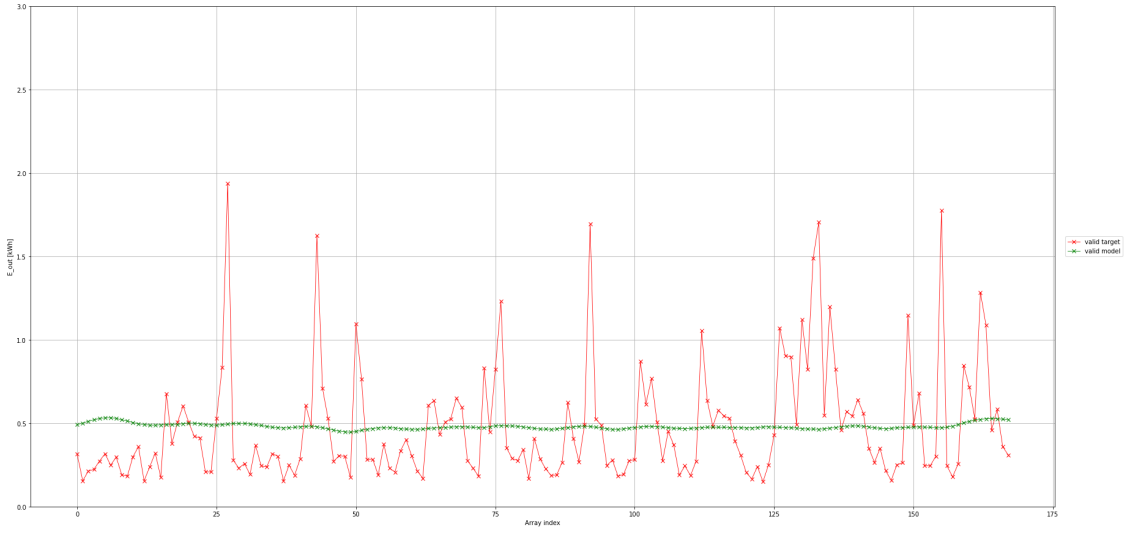





```
Epoch: 1        MSE Train: 0.181        MSE Valid: 0.121        Looptime: 2.273
s
Epoch: 2        MSE Train: 0.179        MSE Valid: 0.120        Looptime: 2.364
s
Epoch: 3        MSE Train: 0.178        MSE Valid: 0.120        Looptime: 2.300
s
Epoch: 4        MSE Train: 0.177        MSE Valid: 0.120        Looptime: 2.351
s
Epoch: 5        MSE Train: 0.177        MSE Valid: 0.120        Looptime: 2.353
s
Epoch: 6        MSE Train: 0.177        MSE Valid: 0.119        Looptime: 2.293
s
```

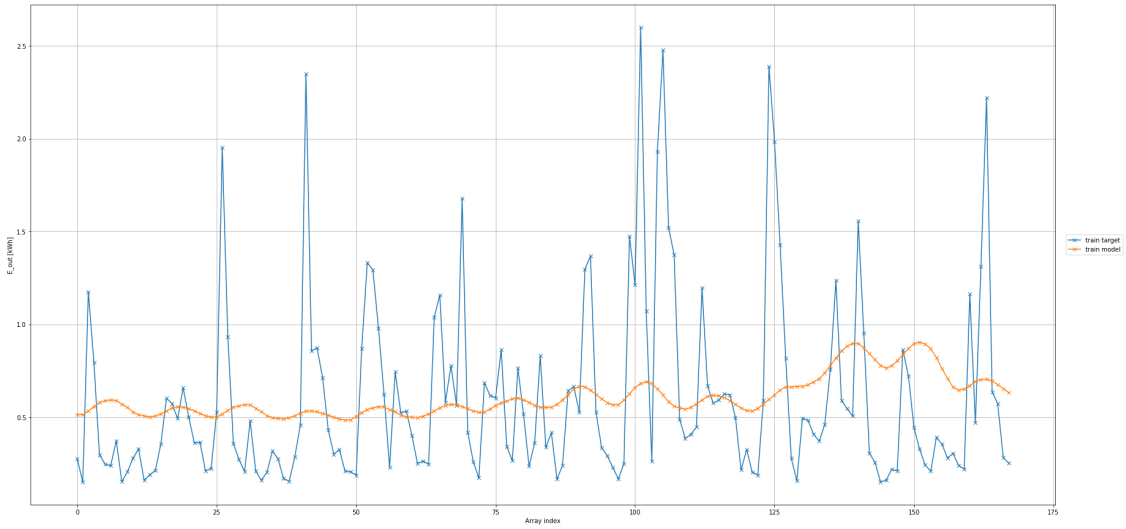
```
Epoch: 7         MSE Train: 0.177         MSE Valid: 0.119         Looptime: 2.291
s
Epoch: 8         MSE Train: 0.176         MSE Valid: 0.119         Looptime: 2.585
s
Epoch: 9         MSE Train: 0.175         MSE Valid: 0.118         Looptime: 2.415
s
Epoch: 10        MSE Train: 0.174         MSE Valid: 0.117         Looptime: 2.328
s
```

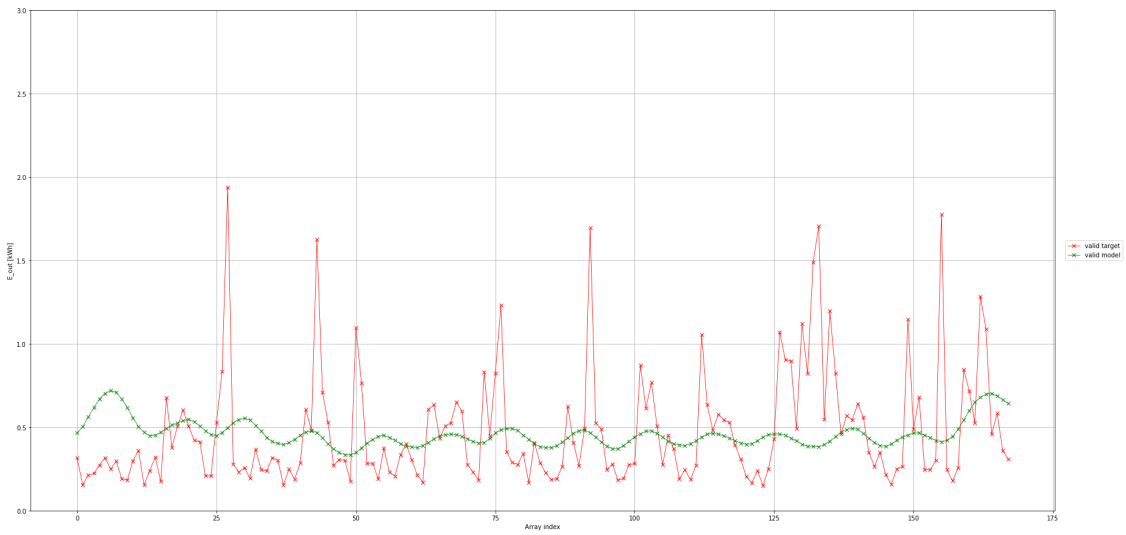




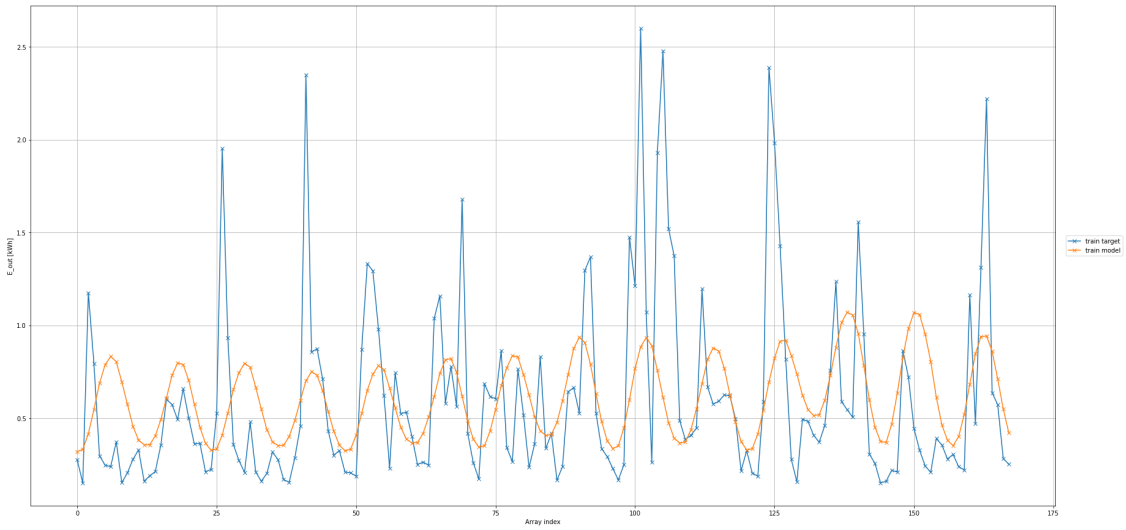

```
Epoch: 11        MSE Train: 0.174         MSE Valid: 0.117         Looptime: 2.389
s
Epoch: 12        MSE Train: 0.173         MSE Valid: 0.116         Looptime: 2.346
```

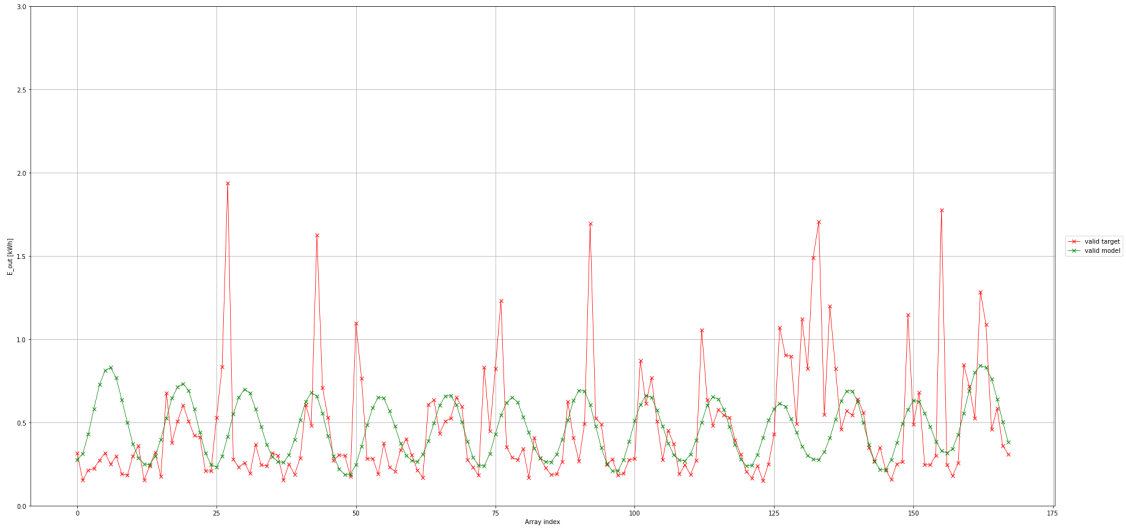
```
s
Epoch: 13        MSE Train: 0.172        MSE Valid: 0.116        Looptime: 2.648
s
Epoch: 14        MSE Train: 0.171        MSE Valid: 0.115        Looptime: 2.613
s
Epoch: 15        MSE Train: 0.170        MSE Valid: 0.115        Looptime: 2.655
s
Epoch: 16        MSE Train: 0.169        MSE Valid: 0.114        Looptime: 2.602
s
Epoch: 17        MSE Train: 0.168        MSE Valid: 0.114        Looptime: 2.881
s
Epoch: 18        MSE Train: 0.167        MSE Valid: 0.113        Looptime: 2.637
s
Epoch: 19        MSE Train: 0.166        MSE Valid: 0.112        Looptime: 2.653
s
Epoch: 20        MSE Train: 0.164        MSE Valid: 0.110        Looptime: 2.636
s
```





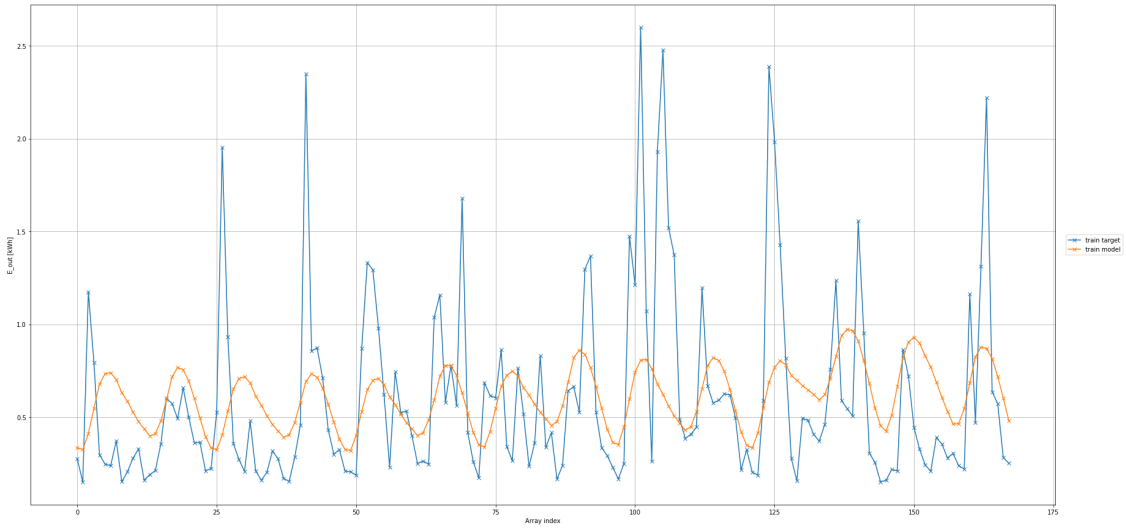

```
Epoch: 21       MSE Train: 0.162       MSE Valid: 0.109       Looptime: 2.636
s
Epoch: 22       MSE Train: 0.160       MSE Valid: 0.107       Looptime: 2.606
s
Epoch: 23       MSE Train: 0.157       MSE Valid: 0.105       Looptime: 2.414
s
Epoch: 24       MSE Train: 0.154       MSE Valid: 0.104       Looptime: 2.314
s
Epoch: 25       MSE Train: 0.152       MSE Valid: 0.102       Looptime: 2.656
s
Epoch: 26       MSE Train: 0.150       MSE Valid: 0.102       Looptime: 2.487
s
Epoch: 27       MSE Train: 0.149       MSE Valid: 0.102       Looptime: 2.408
s
Epoch: 28       MSE Train: 0.149       MSE Valid: 0.103       Looptime: 2.350
s
Epoch: 29       MSE Train: 0.150       MSE Valid: 0.102       Looptime: 2.458
s
Epoch: 30       MSE Train: 0.149       MSE Valid: 0.101       Looptime: 2.638
s
```

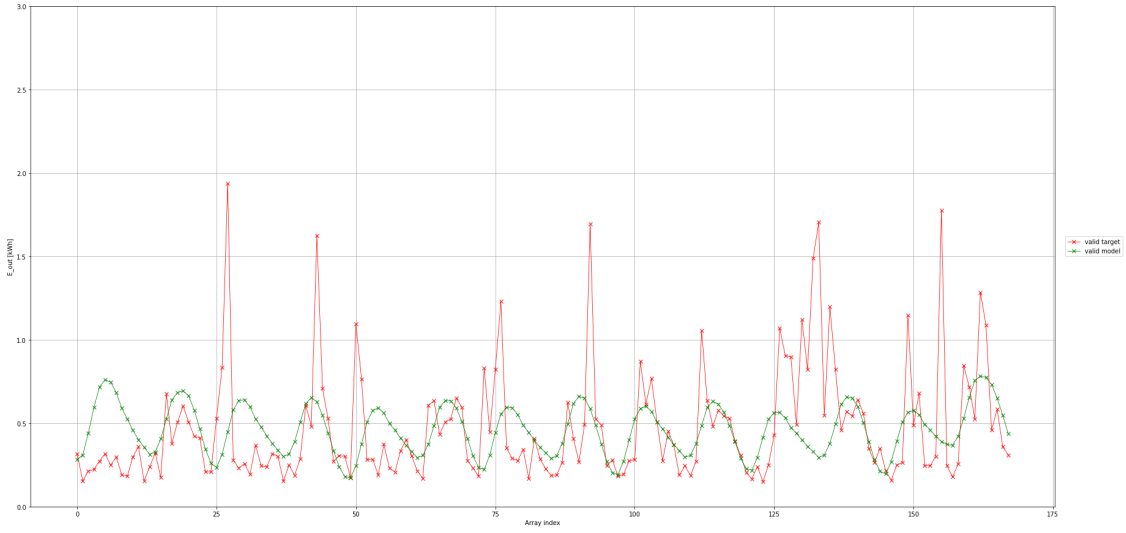





```
Epoch: 31       MSE Train: 0.147       MSE Valid: 0.100       Looptime: 2.381
s
Epoch: 32       MSE Train: 0.146       MSE Valid: 0.099       Looptime: 2.437
s
Epoch: 33       MSE Train: 0.145       MSE Valid: 0.099       Looptime: 2.343
s
Epoch: 34       MSE Train: 0.145       MSE Valid: 0.099       Looptime: 2.330
s
Epoch: 35       MSE Train: 0.145       MSE Valid: 0.099       Looptime: 2.789
s
Epoch: 36       MSE Train: 0.145       MSE Valid: 0.099       Looptime: 2.271
s
```

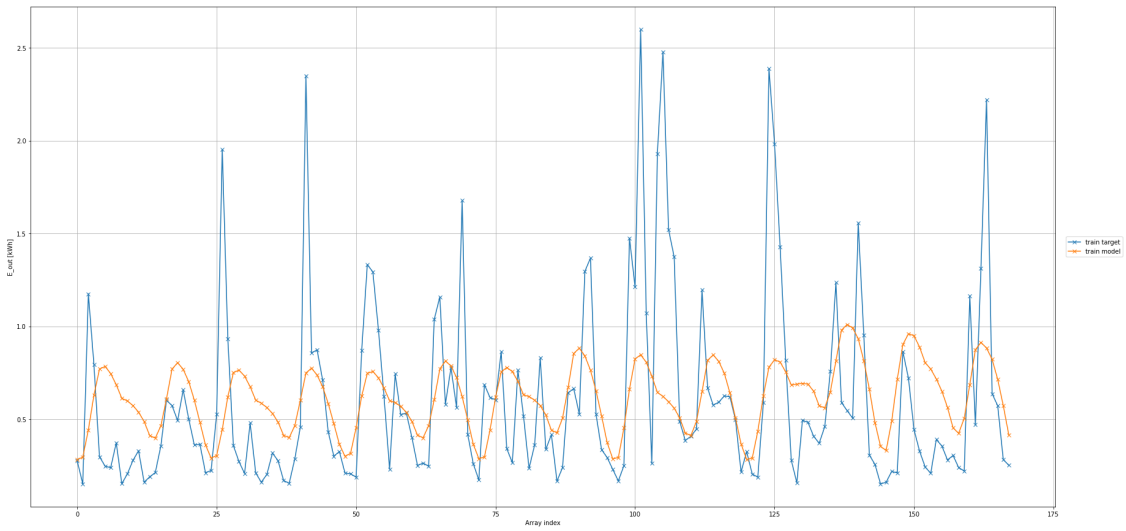
```
Epoch: 37        MSE Train: 0.145        MSE Valid: 0.099        Looptime: 2.273
s
Epoch: 38        MSE Train: 0.144        MSE Valid: 0.098        Looptime: 2.273
s
Epoch: 39        MSE Train: 0.143        MSE Valid: 0.098        Looptime: 2.427
s
Epoch: 40        MSE Train: 0.143        MSE Valid: 0.097        Looptime: 2.650
s
```

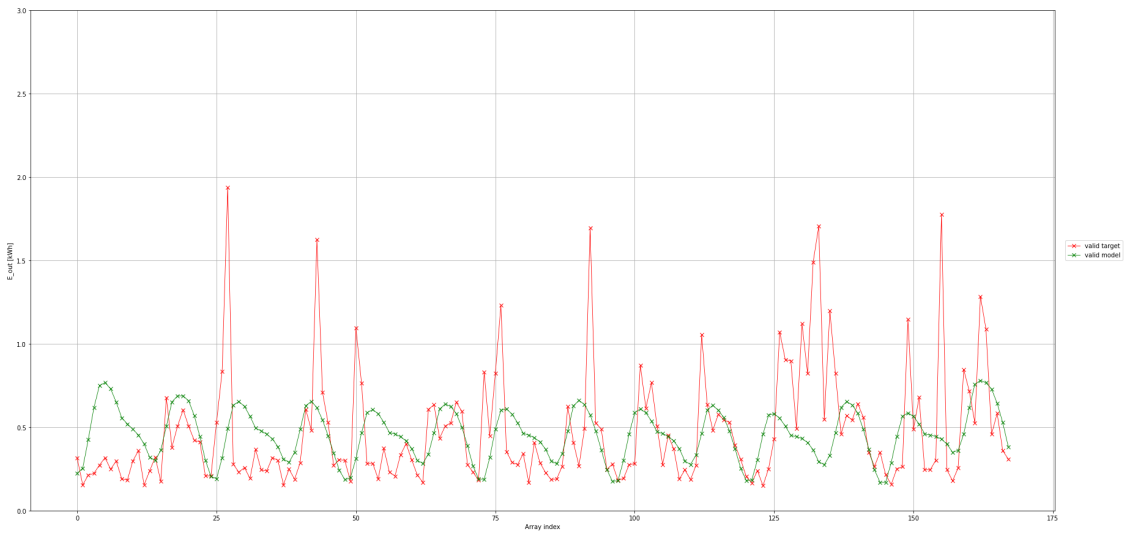




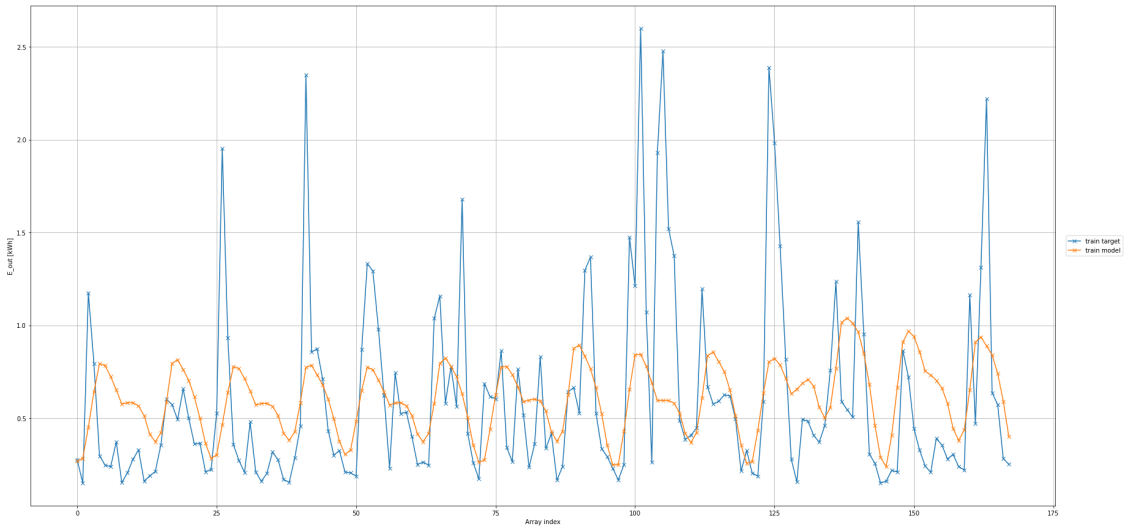

```
Epoch: 41        MSE Train: 0.142        MSE Valid: 0.097        Looptime: 2.565
s
Epoch: 42        MSE Train: 0.141        MSE Valid: 0.097        Looptime: 2.819
```

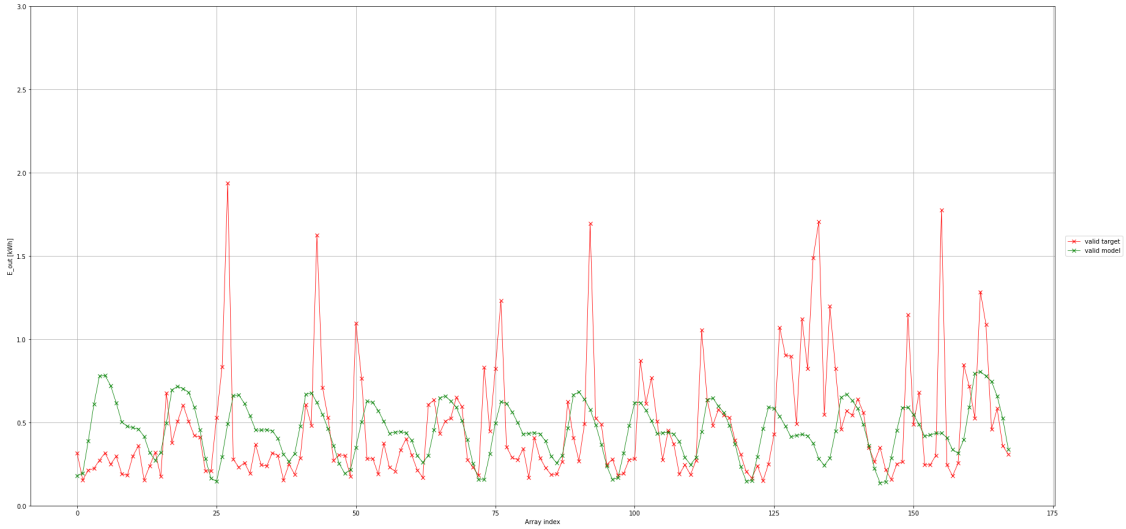
```
s
Epoch: 43        MSE Train: 0.141        MSE Valid: 0.097        Looptime: 2.718
s
Epoch: 44        MSE Train: 0.141        MSE Valid: 0.097        Looptime: 2.634
s
Epoch: 45        MSE Train: 0.141        MSE Valid: 0.097        Looptime: 2.593
s
Epoch: 46        MSE Train: 0.141        MSE Valid: 0.096        Looptime: 2.606
s
Epoch: 47        MSE Train: 0.140        MSE Valid: 0.096        Looptime: 2.603
s
Epoch: 48        MSE Train: 0.140        MSE Valid: 0.096        Looptime: 2.659
s
Epoch: 49        MSE Train: 0.140        MSE Valid: 0.096        Looptime: 2.648
s
Epoch: 50        MSE Train: 0.139        MSE Valid: 0.096        Looptime: 2.689
s
```





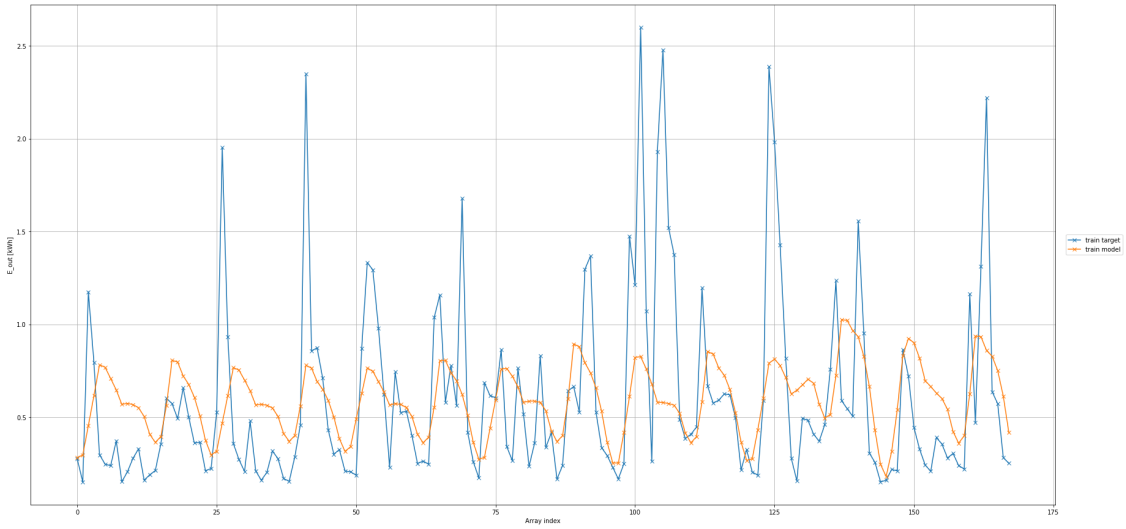

```
Epoch: 51       MSE Train: 0.139       MSE Valid: 0.095       Looptime: 2.645
s
Epoch: 52       MSE Train: 0.139       MSE Valid: 0.095       Looptime: 2.677
s
Epoch: 53       MSE Train: 0.139       MSE Valid: 0.095       Looptime: 2.572
s
Epoch: 54       MSE Train: 0.139       MSE Valid: 0.095       Looptime: 2.394
s
Epoch: 55       MSE Train: 0.139       MSE Valid: 0.095       Looptime: 2.379
s
Epoch: 56       MSE Train: 0.138       MSE Valid: 0.095       Looptime: 2.362
s
Epoch: 57       MSE Train: 0.138       MSE Valid: 0.095       Looptime: 2.380
s
Epoch: 58       MSE Train: 0.138       MSE Valid: 0.095       Looptime: 2.404
s
Epoch: 59       MSE Train: 0.138       MSE Valid: 0.095       Looptime: 2.488
s
Epoch: 60       MSE Train: 0.138       MSE Valid: 0.095       Looptime: 2.341
s
```

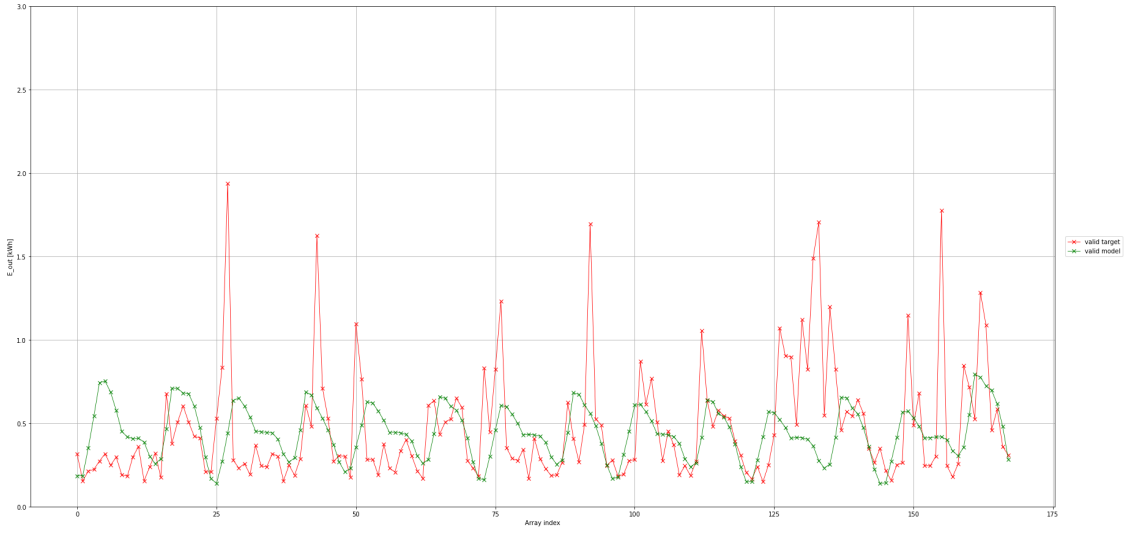





```
Epoch: 61       MSE Train: 0.138       MSE Valid: 0.095       Looptime: 2.360
s
Epoch: 62       MSE Train: 0.138       MSE Valid: 0.095       Looptime: 2.406
s
Epoch: 63       MSE Train: 0.137       MSE Valid: 0.094       Looptime: 2.498
s
Epoch: 64       MSE Train: 0.137       MSE Valid: 0.094       Looptime: 2.352
s
Epoch: 65       MSE Train: 0.137       MSE Valid: 0.094       Looptime: 2.337
s
Epoch: 66       MSE Train: 0.137       MSE Valid: 0.094       Looptime: 2.376
s
```

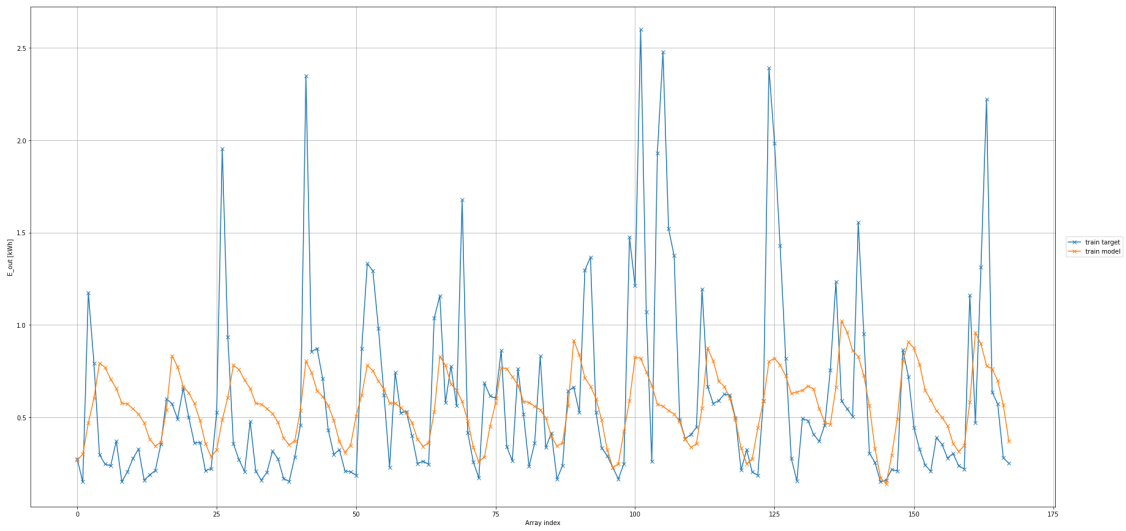
```
Epoch: 67        MSE Train: 0.137        MSE Valid: 0.094        Looptime: 2.340
s
Epoch: 68        MSE Train: 0.137        MSE Valid: 0.094        Looptime: 2.348
s
Epoch: 69        MSE Train: 0.136        MSE Valid: 0.094        Looptime: 2.461
s
Epoch: 70        MSE Train: 0.136        MSE Valid: 0.094        Looptime: 2.339
s
```

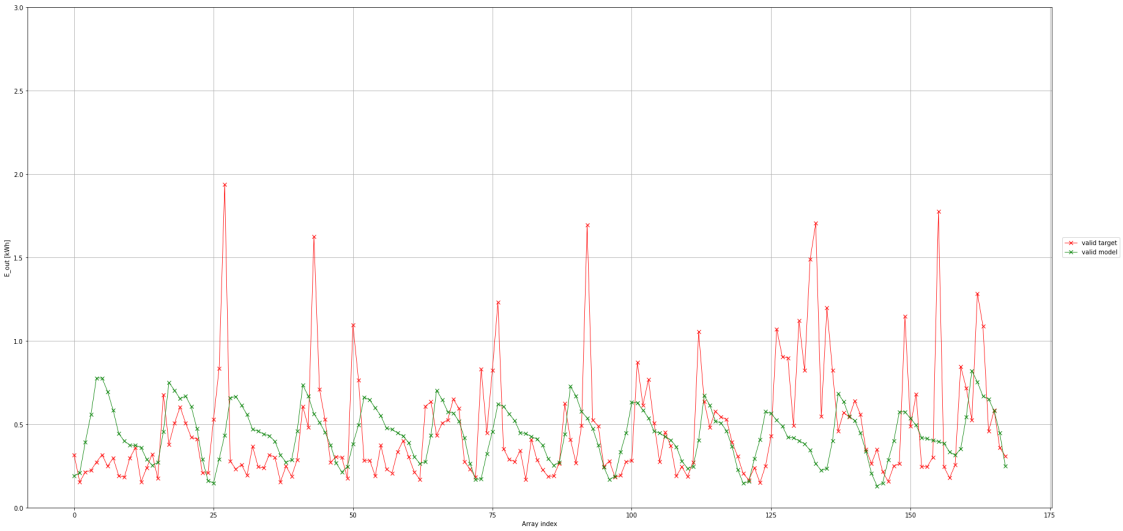




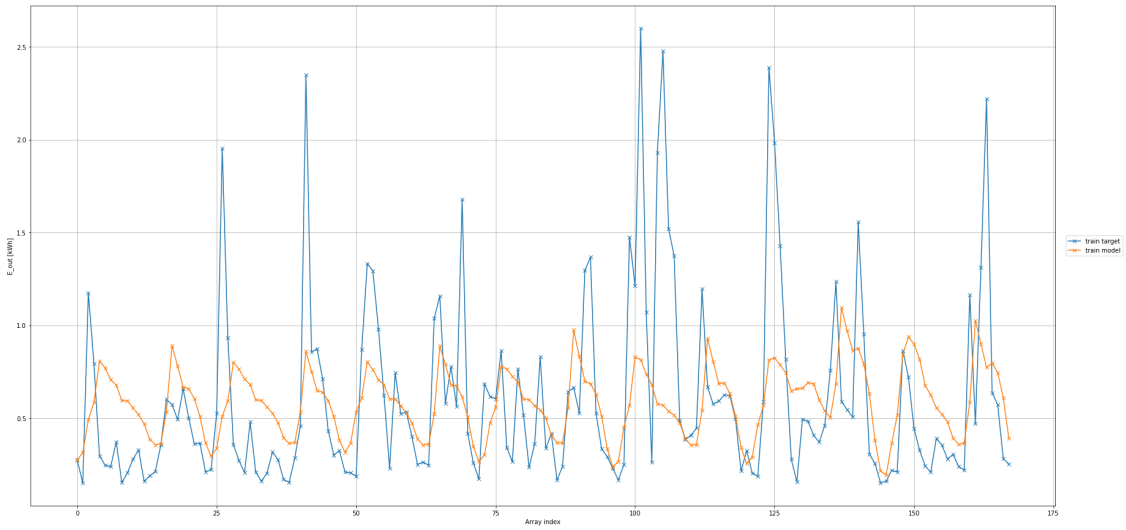

```
Epoch: 71        MSE Train: 0.136        MSE Valid: 0.094        Looptime: 2.352
s
Epoch: 72        MSE Train: 0.136        MSE Valid: 0.094        Looptime: 3.586
```

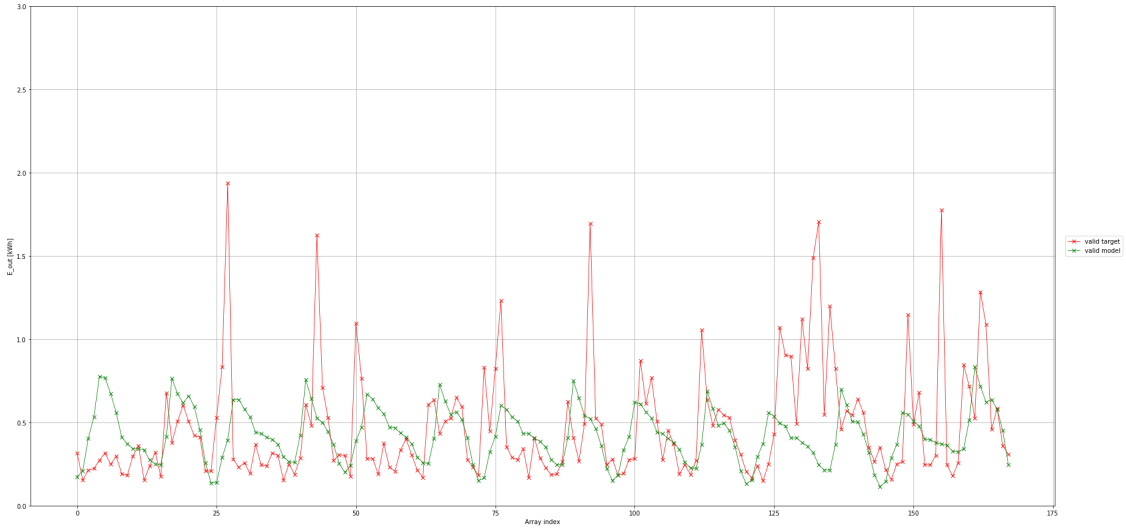
```
s
Epoch: 73        MSE Train: 0.136        MSE Valid: 0.094        Looptime: 2.293
s
Epoch: 74        MSE Train: 0.136        MSE Valid: 0.094        Looptime: 2.296
s
Epoch: 75        MSE Train: 0.135        MSE Valid: 0.093        Looptime: 2.390
s
Epoch: 76        MSE Train: 0.135        MSE Valid: 0.093        Looptime: 2.281
s
Epoch: 77        MSE Train: 0.135        MSE Valid: 0.093        Looptime: 2.288
s
Epoch: 78        MSE Train: 0.135        MSE Valid: 0.093        Looptime: 2.281
s
Epoch: 79        MSE Train: 0.135        MSE Valid: 0.093        Looptime: 2.350
s
Epoch: 80        MSE Train: 0.135        MSE Valid: 0.093        Looptime: 2.299
s
```





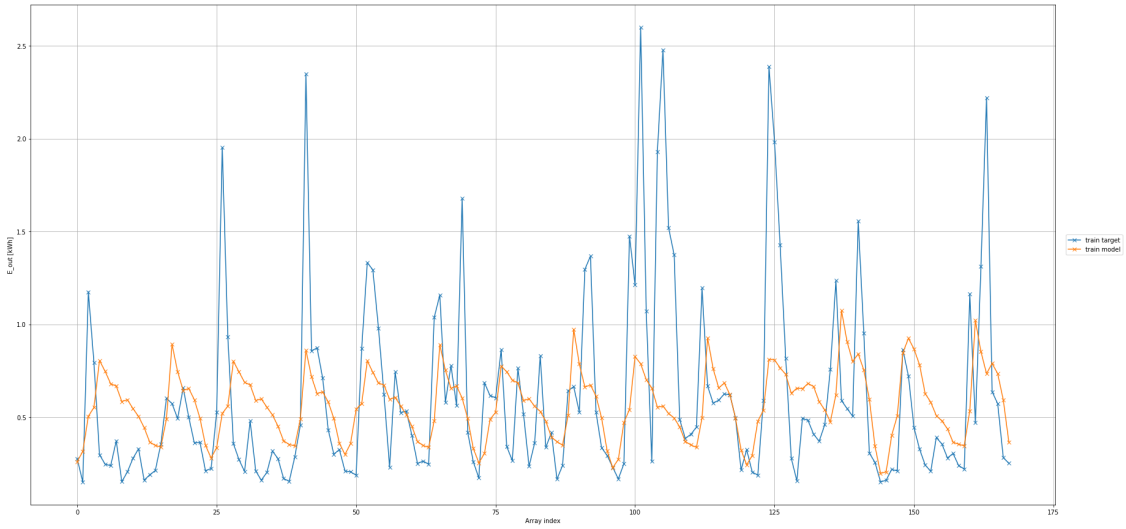

```
Epoch: 81       MSE Train: 0.135       MSE Valid: 0.093       Looptime: 2.424
s
Epoch: 82       MSE Train: 0.135       MSE Valid: 0.093       Looptime: 2.325
s
Epoch: 83       MSE Train: 0.134       MSE Valid: 0.093       Looptime: 2.288
s
Epoch: 84       MSE Train: 0.134       MSE Valid: 0.093       Looptime: 2.298
s
Epoch: 85       MSE Train: 0.134       MSE Valid: 0.093       Looptime: 2.355
s
Epoch: 86       MSE Train: 0.134       MSE Valid: 0.093       Looptime: 2.349
s
Epoch: 87       MSE Train: 0.134       MSE Valid: 0.093       Looptime: 2.382
s
Epoch: 88       MSE Train: 0.134       MSE Valid: 0.093       Looptime: 2.363
s
Epoch: 89       MSE Train: 0.134       MSE Valid: 0.093       Looptime: 2.451
s
Epoch: 90       MSE Train: 0.134       MSE Valid: 0.093       Looptime: 2.287
s
```

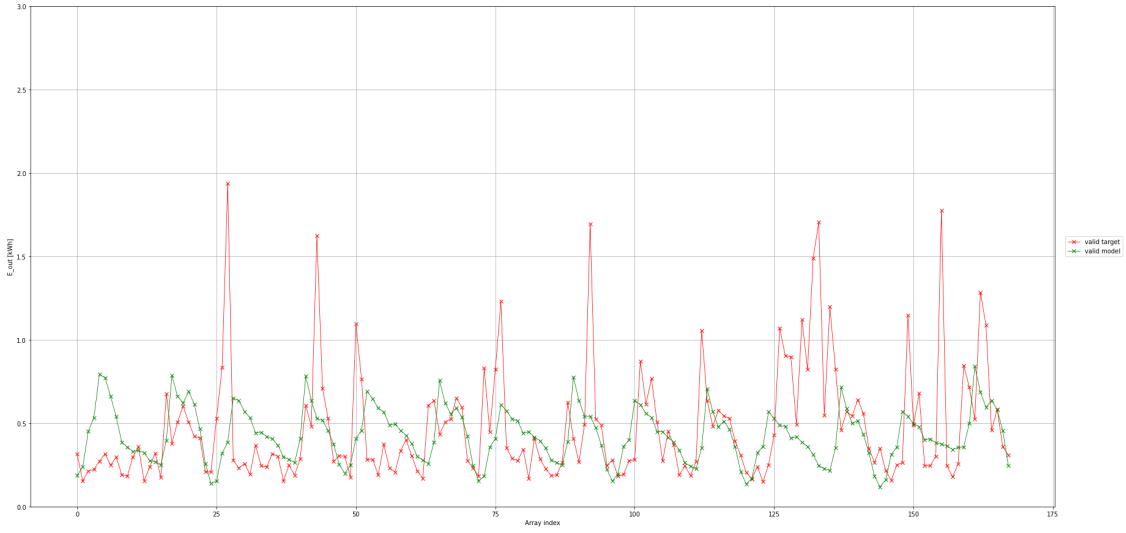





```
Epoch: 91       MSE Train: 0.134       MSE Valid: 0.093       Looptime: 2.356
s
Epoch: 92       MSE Train: 0.134       MSE Valid: 0.093       Looptime: 2.313
s
Epoch: 93       MSE Train: 0.134       MSE Valid: 0.093       Looptime: 2.622
s
Epoch: 94       MSE Train: 0.133       MSE Valid: 0.093       Looptime: 2.333
s
Epoch: 95       MSE Train: 0.133       MSE Valid: 0.092       Looptime: 2.576
s
Epoch: 96       MSE Train: 0.133       MSE Valid: 0.092       Looptime: 2.552
s
```

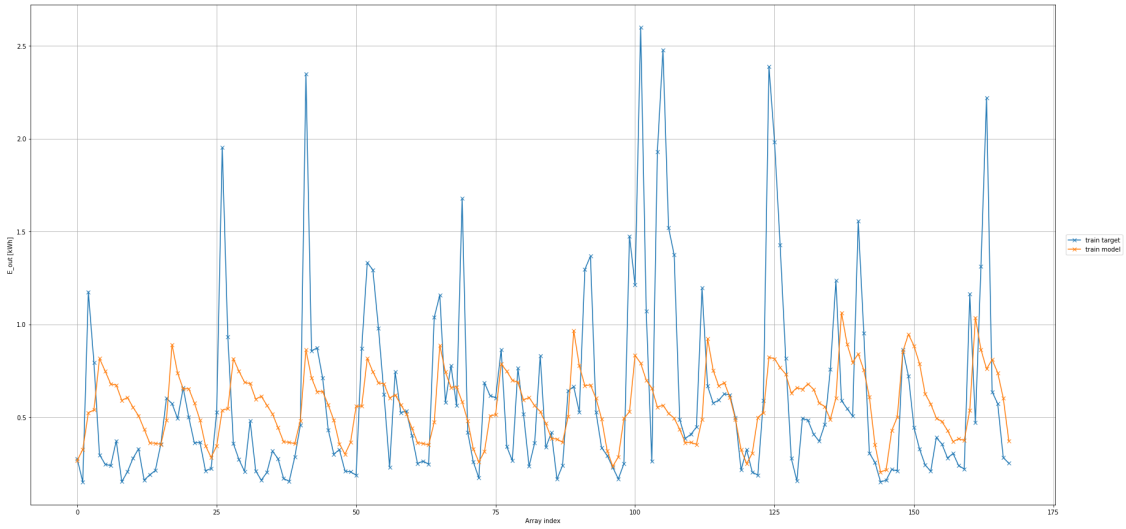
```
Epoch: 97        MSE Train: 0.133        MSE Valid: 0.092        Looptime: 2.552
s
Epoch: 98        MSE Train: 0.133        MSE Valid: 0.092        Looptime: 2.419
s
Epoch: 99        MSE Train: 0.133        MSE Valid: 0.092        Looptime: 2.399
s
Epoch: 100       MSE Train: 0.133        MSE Valid: 0.092        Looptime: 2.402
s
```

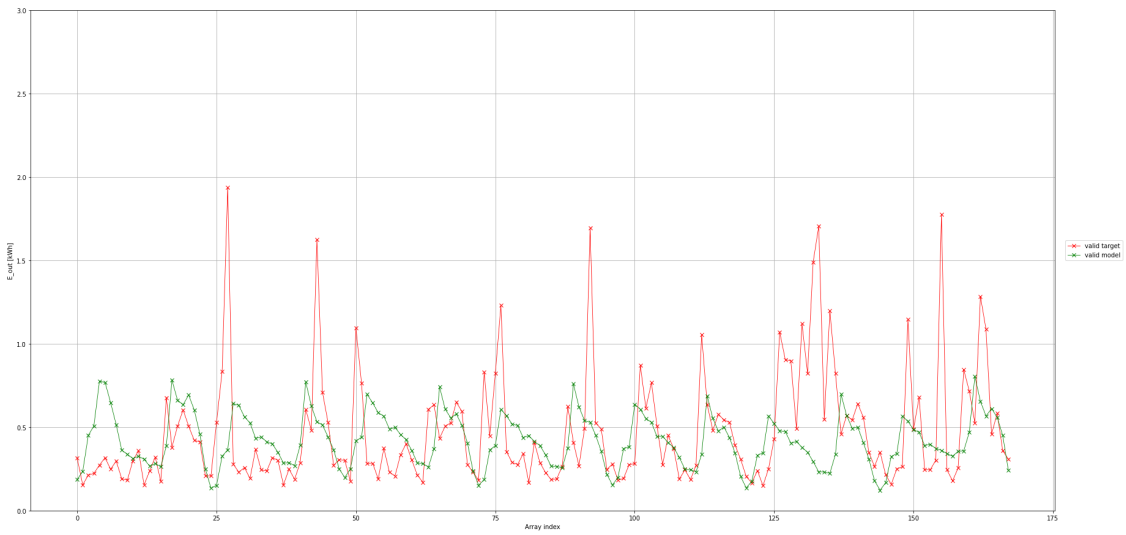




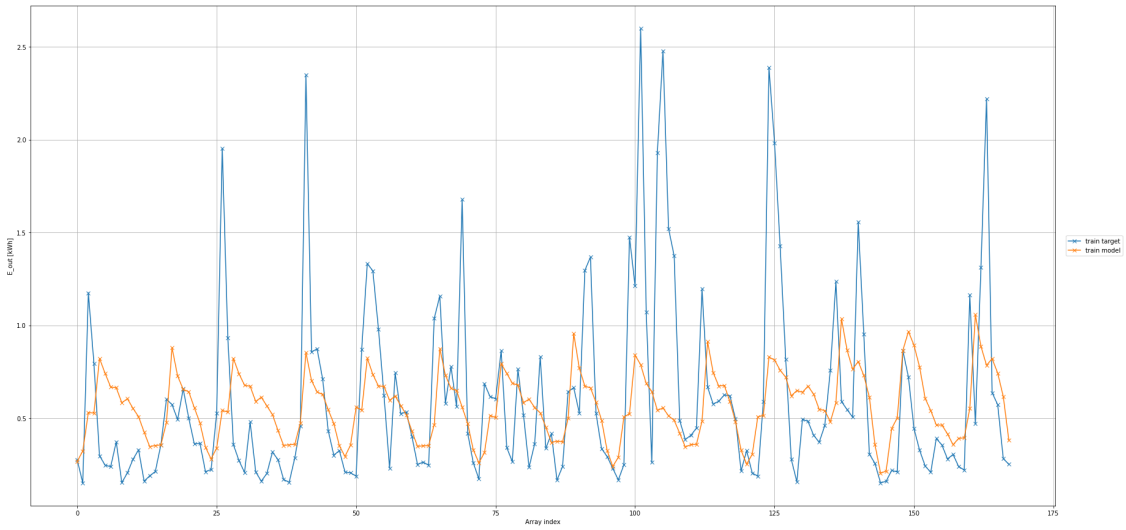

```
Epoch: 101       MSE Train: 0.133        MSE Valid: 0.092        Looptime: 2.331
s
Epoch: 102       MSE Train: 0.133        MSE Valid: 0.092        Looptime: 2.346
```

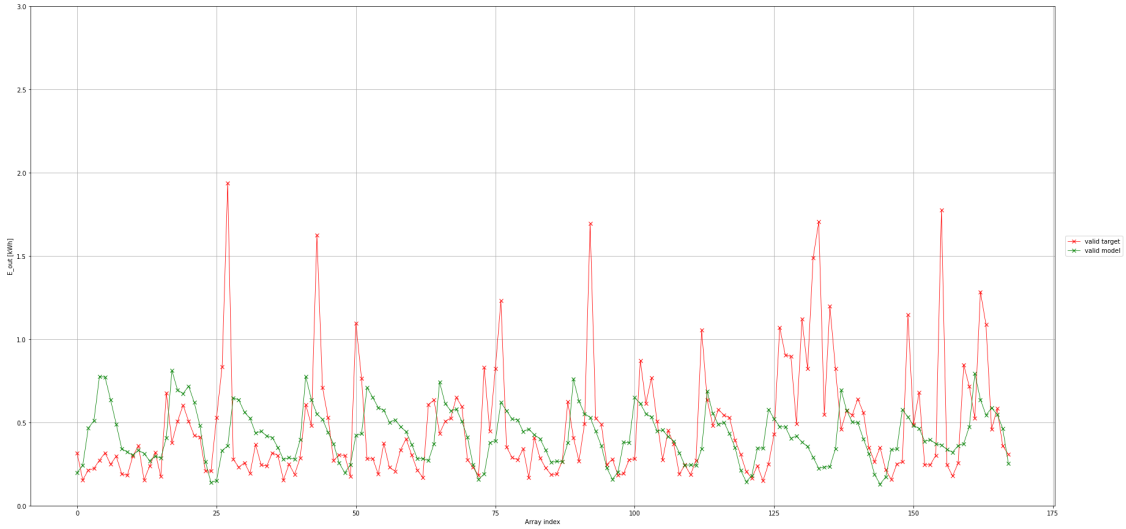
```
s
Epoch: 103       MSE Train: 0.133       MSE Valid: 0.092       Looptime: 2.381
s
Epoch: 104       MSE Train: 0.133       MSE Valid: 0.092       Looptime: 2.363
s
Epoch: 105       MSE Train: 0.133       MSE Valid: 0.092       Looptime: 2.341
s
Epoch: 106       MSE Train: 0.133       MSE Valid: 0.092       Looptime: 2.366
s
Epoch: 107       MSE Train: 0.133       MSE Valid: 0.092       Looptime: 2.473
s
Epoch: 108       MSE Train: 0.133       MSE Valid: 0.092       Looptime: 2.561
s
Epoch: 109       MSE Train: 0.133       MSE Valid: 0.092       Looptime: 2.443
s
Epoch: 110       MSE Train: 0.132       MSE Valid: 0.092       Looptime: 2.520
s
```





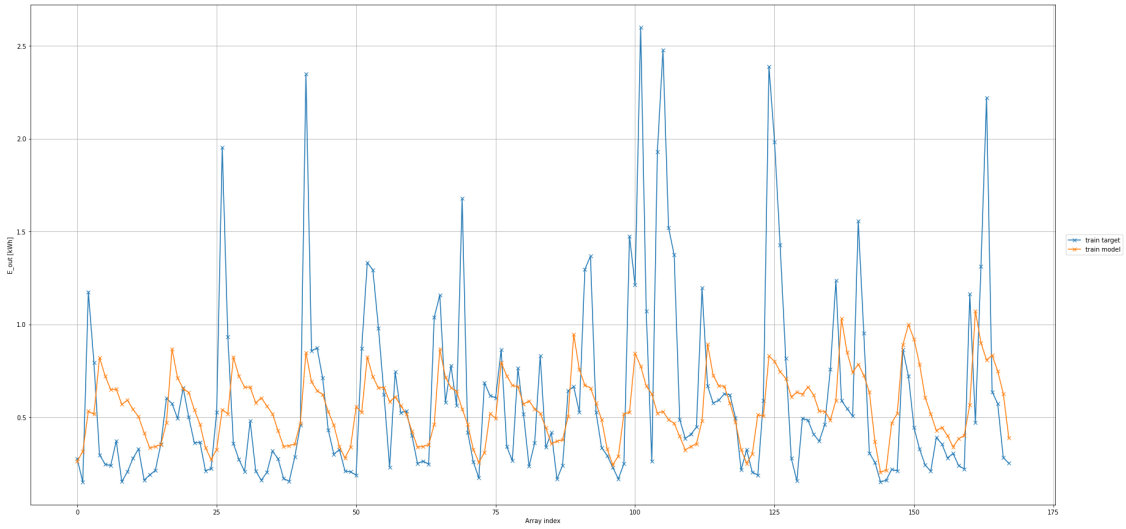

```
Epoch: 111      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.351
s
Epoch: 112      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.372
s
Epoch: 113      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.549
s
Epoch: 114      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.350
s
Epoch: 115      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.347
s
Epoch: 116      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.397
s
Epoch: 117      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.363
s
Epoch: 118      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.440
s
Epoch: 119      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.556
s
Epoch: 120      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.383
s
```

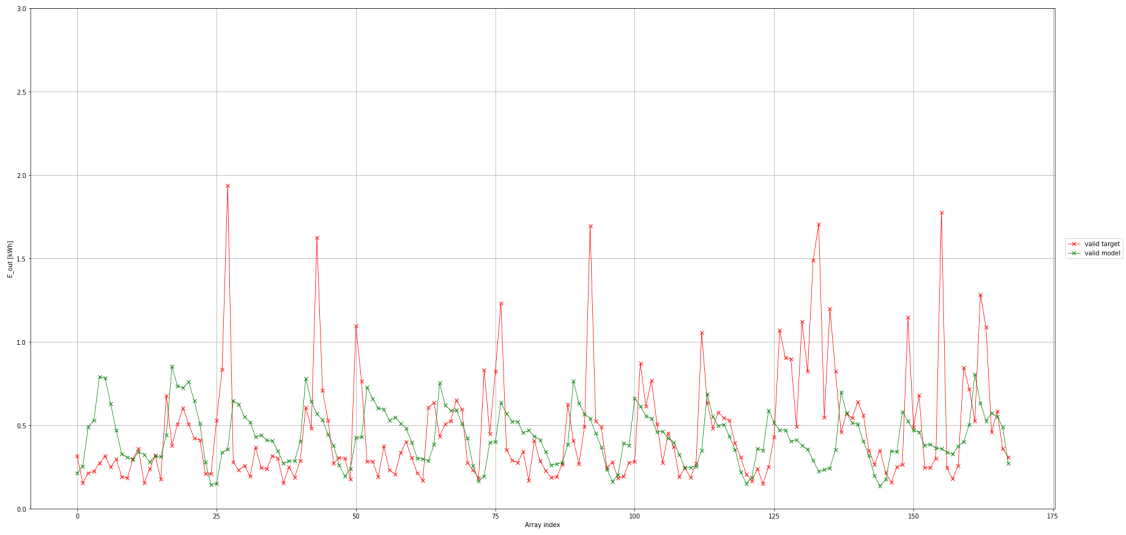





```
Epoch: 121      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.422
s
Epoch: 122      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.341
s
Epoch: 123      MSE Train: 0.132      MSE Valid: 0.092      Looptime: 2.484
s
Epoch: 124      MSE Train: 0.132      MSE Valid: 0.091      Looptime: 2.544
s
Epoch: 125      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.350
s
Epoch: 126      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.330
s
```

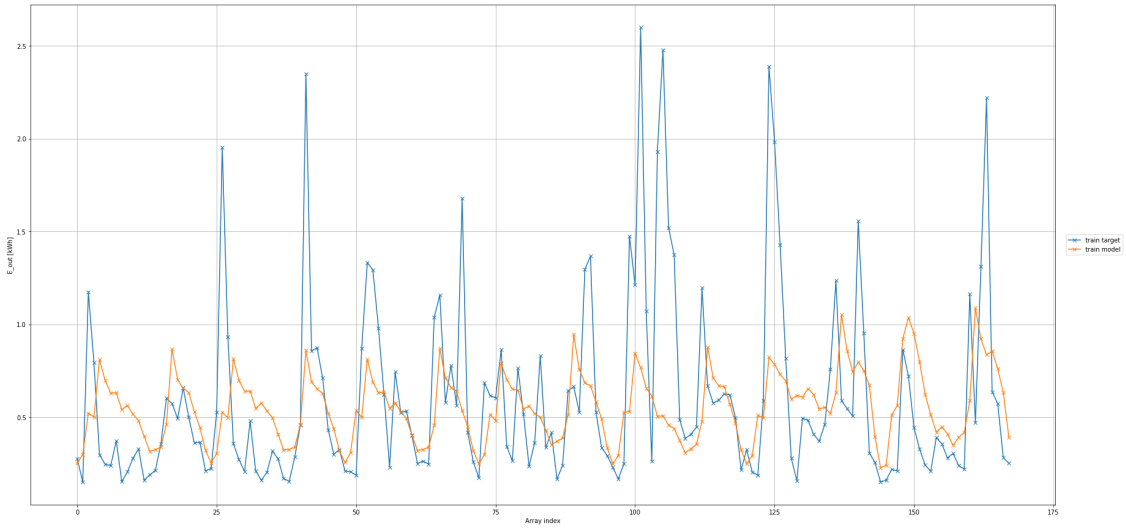
```
Epoch: 127        MSE Train: 0.131        MSE Valid: 0.091        Looptime: 2.335
s
Epoch: 128        MSE Train: 0.131        MSE Valid: 0.091        Looptime: 2.350
s
Epoch: 129        MSE Train: 0.131        MSE Valid: 0.091        Looptime: 2.547
s
Epoch: 130        MSE Train: 0.131        MSE Valid: 0.091        Looptime: 2.453
s
```

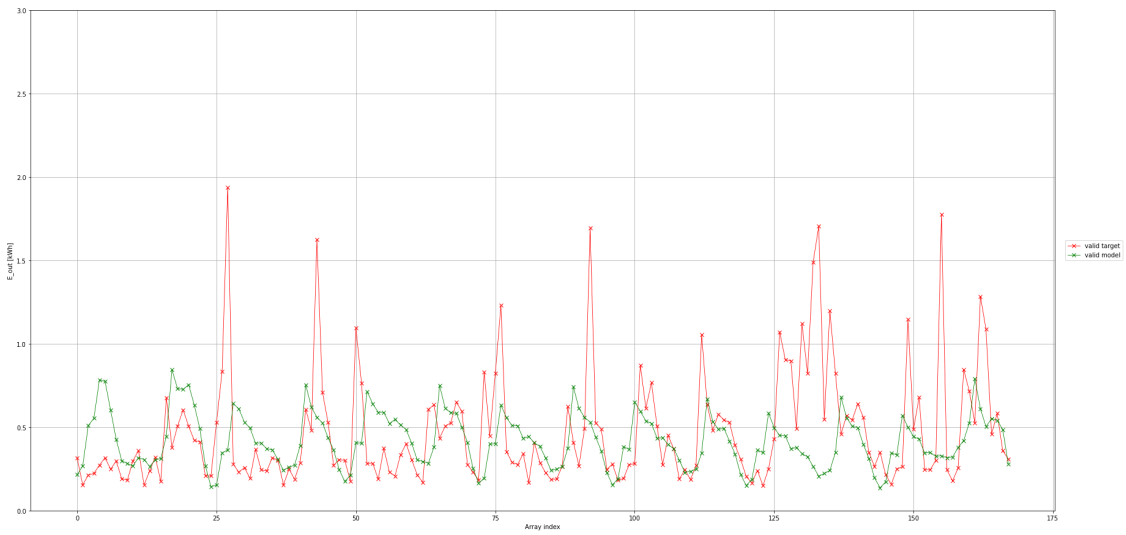




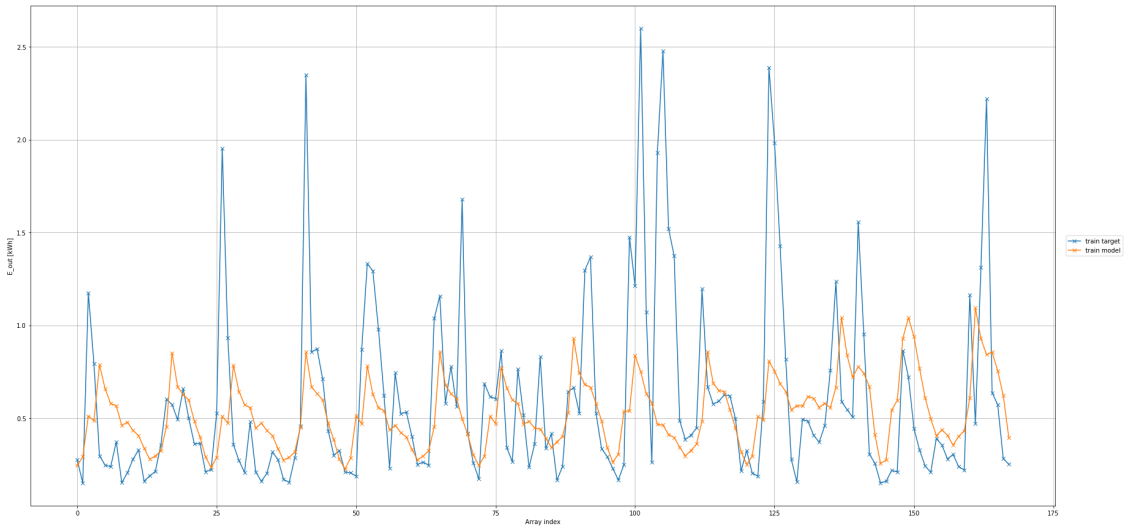

```
Epoch: 131        MSE Train: 0.131        MSE Valid: 0.092        Looptime: 2.753
s
Epoch: 132        MSE Train: 0.132        MSE Valid: 0.091        Looptime: 2.456
```

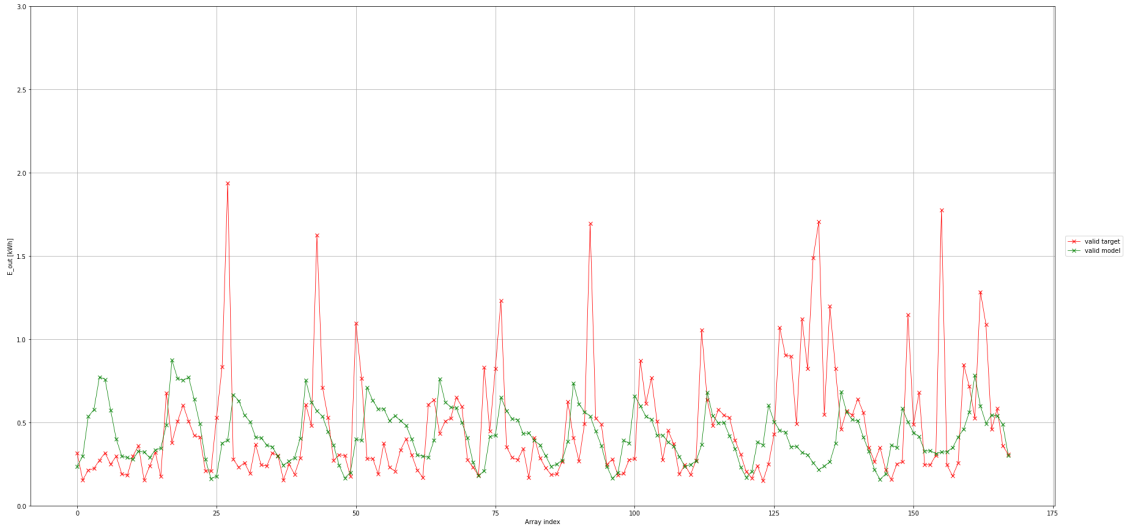
```
s
Epoch: 133      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.344
s
Epoch: 134      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.347
s
Epoch: 135      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.352
s
Epoch: 136      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.351
s
Epoch: 137      MSE Train: 0.130      MSE Valid: 0.091      Looptime: 2.513
s
Epoch: 138      MSE Train: 0.130      MSE Valid: 0.091      Looptime: 2.469
s
Epoch: 139      MSE Train: 0.130      MSE Valid: 0.091      Looptime: 2.371
s
Epoch: 140      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.424
s
```





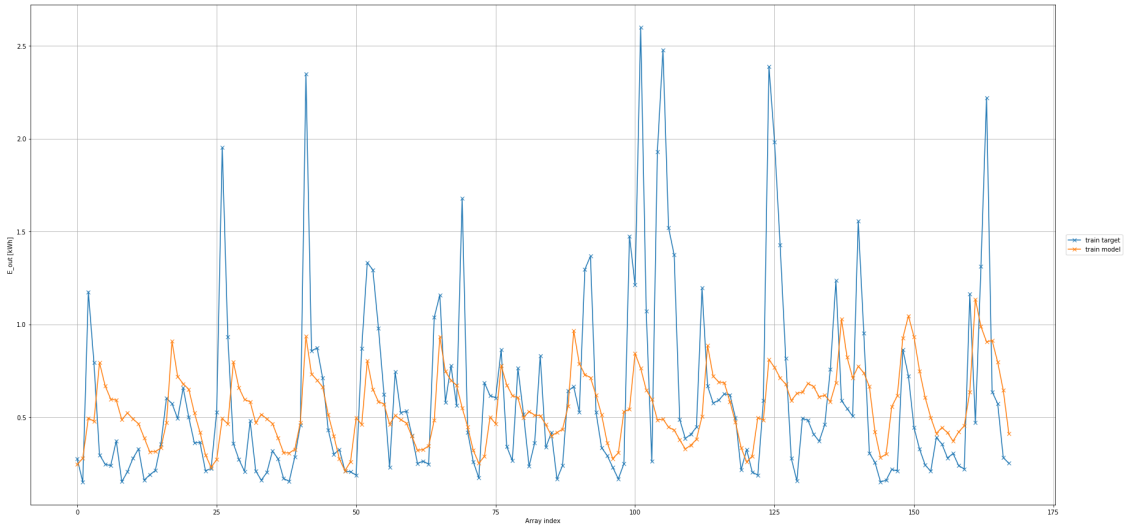

```
Epoch: 141      MSE Train: 0.130      MSE Valid: 0.091      Looptime: 2.414
s
Epoch: 142      MSE Train: 0.130      MSE Valid: 0.091      Looptime: 2.405
s
Epoch: 143      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.355
s
Epoch: 144      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.672
s
Epoch: 145      MSE Train: 0.129      MSE Valid: 0.090      Looptime: 3.269
s
Epoch: 146      MSE Train: 0.129      MSE Valid: 0.090      Looptime: 2.939
s
Epoch: 147      MSE Train: 0.129      MSE Valid: 0.090      Looptime: 6.635
s
Epoch: 148      MSE Train: 0.129      MSE Valid: 0.090      Looptime: 2.797
s
Epoch: 149      MSE Train: 0.129      MSE Valid: 0.091      Looptime: 2.918
s
Epoch: 150      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 3.005
s
```

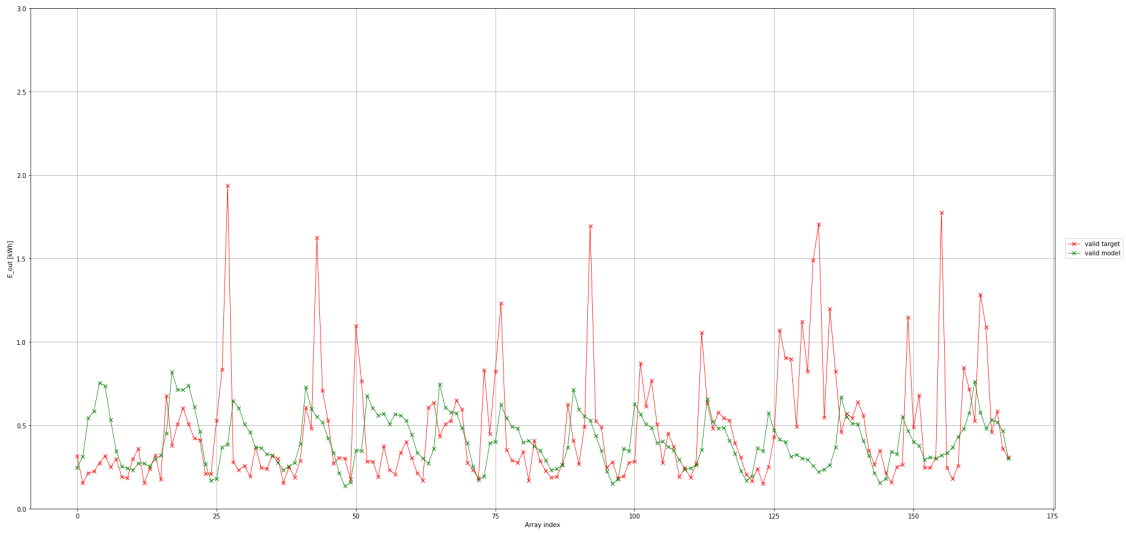





```
Epoch: 151      MSE Train: 0.129      MSE Valid: 0.090      Looptime: 2.822
s
Epoch: 152      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.839
s
Epoch: 153      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.837
s
Epoch: 154      MSE Train: 0.129      MSE Valid: 0.090      Looptime: 2.843
s
Epoch: 155      MSE Train: 0.129      MSE Valid: 0.090      Looptime: 2.828
s
Epoch: 156      MSE Train: 0.129      MSE Valid: 0.090      Looptime: 2.915
s
```

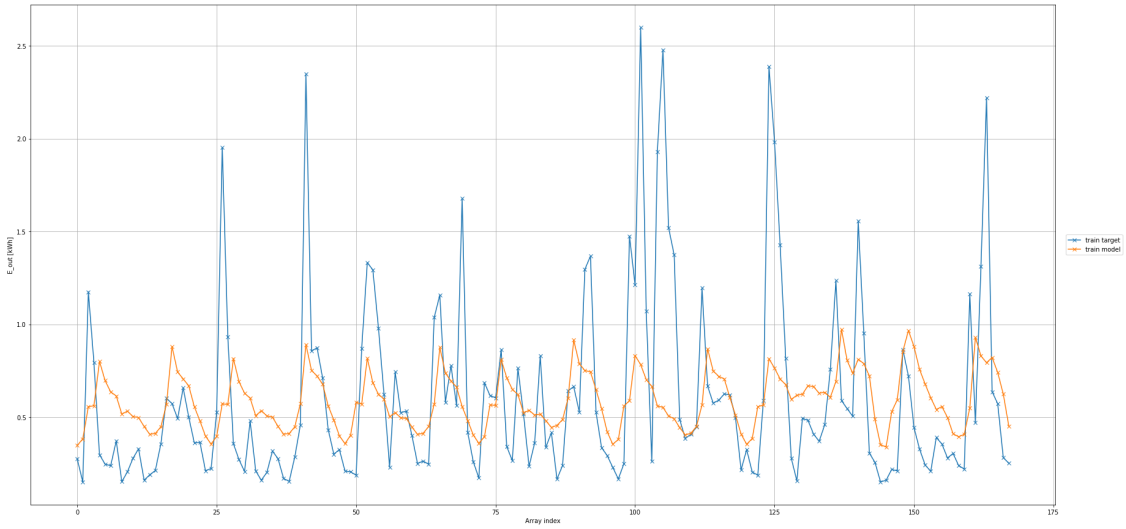
Epoch: 157 MSE Train: 0.129 MSE Valid: 0.089 Looptime: 2.875
s
Epoch: 158 MSE Train: 0.128 MSE Valid: 0.090 Looptime: 2.840
s
Epoch: 159 MSE Train: 0.129 MSE Valid: 0.089 Looptime: 2.854
s
Epoch: 160 MSE Train: 0.127 MSE Valid: 0.089 Looptime: 2.821
s

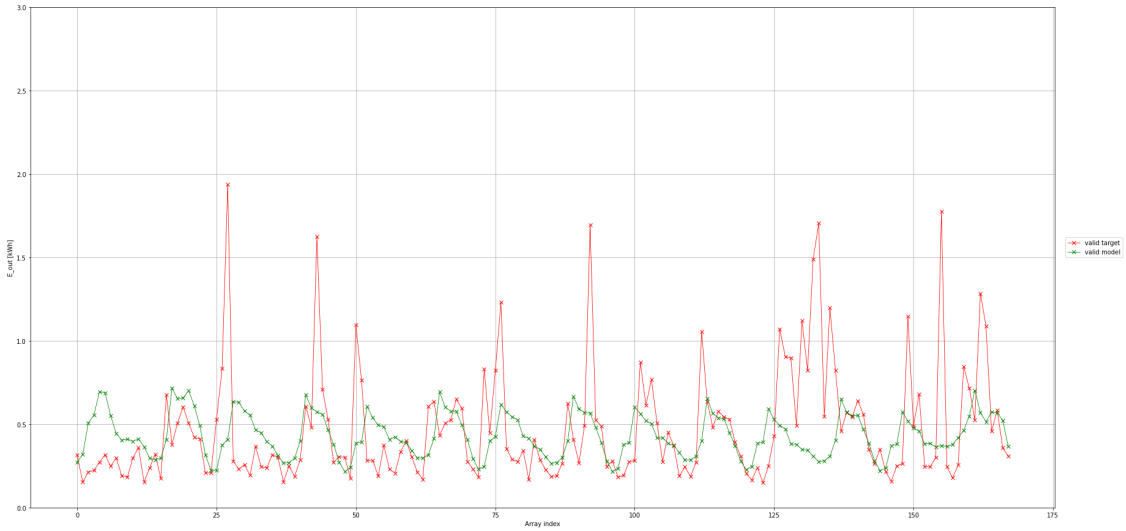




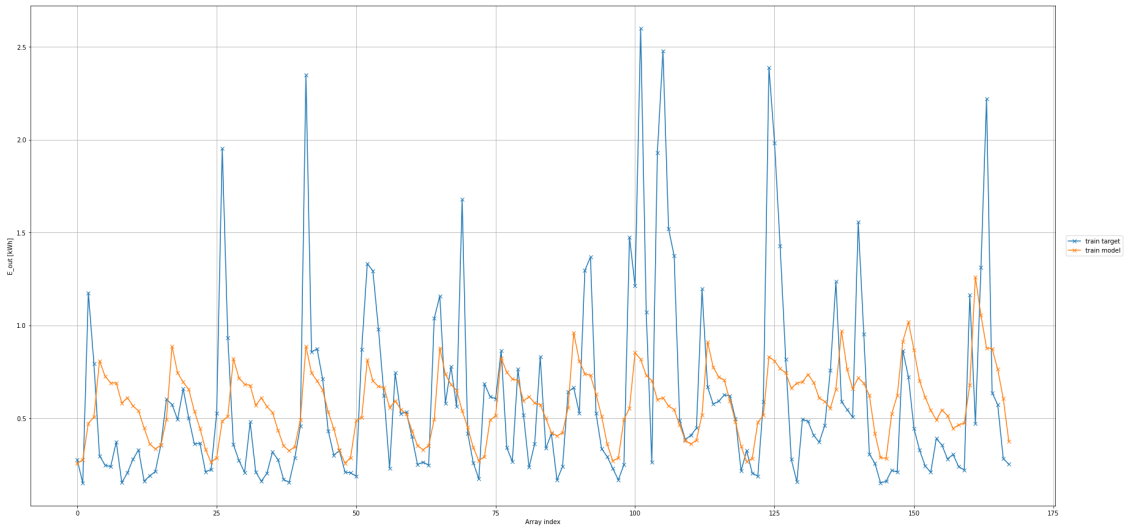
Epoch: 161 MSE Train: 0.128 MSE Valid: 0.093 Looptime: 2.857
s
Epoch: 162 MSE Train: 0.133 MSE Valid: 0.096 Looptime: 2.996

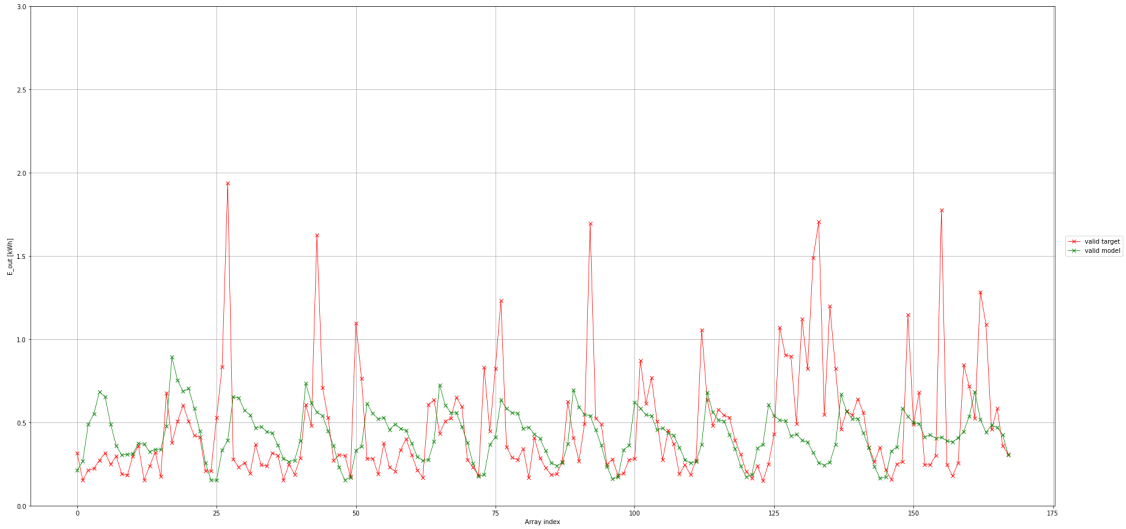
```
s
Epoch: 163      MSE Train: 0.138      MSE Valid: 0.095      Looptime: 2.465
s
Epoch: 164      MSE Train: 0.137      MSE Valid: 0.094      Looptime: 2.457
s
Epoch: 165      MSE Train: 0.136      MSE Valid: 0.094      Looptime: 2.459
s
Epoch: 166      MSE Train: 0.137      MSE Valid: 0.093      Looptime: 2.426
s
Epoch: 167      MSE Train: 0.135      MSE Valid: 0.093      Looptime: 2.328
s
Epoch: 168      MSE Train: 0.135      MSE Valid: 0.093      Looptime: 2.326
s
Epoch: 169      MSE Train: 0.134      MSE Valid: 0.092      Looptime: 2.318
s
Epoch: 170      MSE Train: 0.134      MSE Valid: 0.092      Looptime: 2.341
s
```





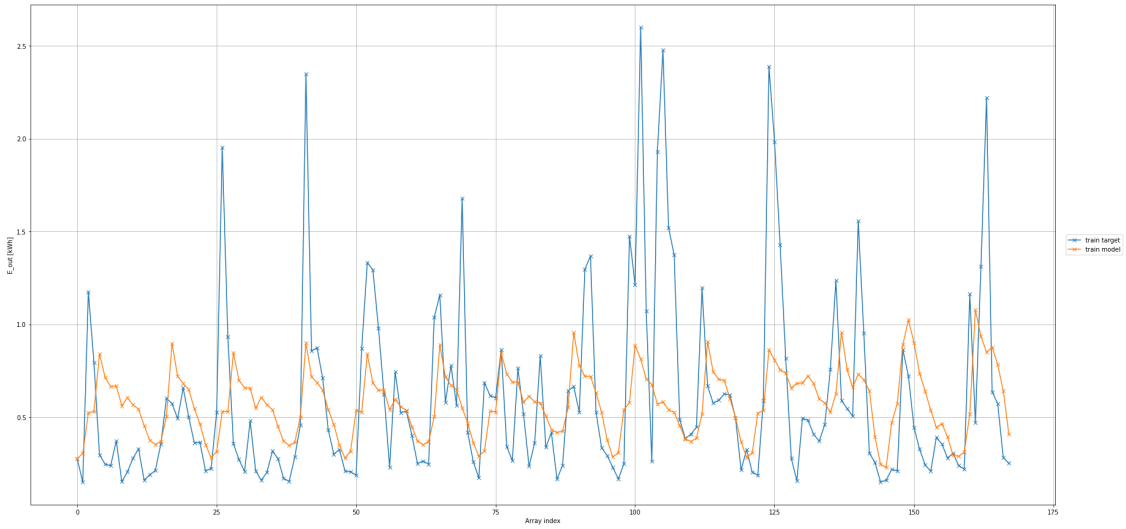

```
Epoch: 171      MSE Train: 0.133      MSE Valid: 0.092      Looptime: 2.345
s
Epoch: 172      MSE Train: 0.133      MSE Valid: 0.091      Looptime: 2.328
s
Epoch: 173      MSE Train: 0.132      MSE Valid: 0.091      Looptime: 2.330
s
Epoch: 174      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.429
s
Epoch: 175      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.334
s
Epoch: 176      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.332
s
Epoch: 177      MSE Train: 0.132      MSE Valid: 0.091      Looptime: 2.498
s
Epoch: 178      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.426
s
Epoch: 179      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.450
s
Epoch: 180      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.336
s
```

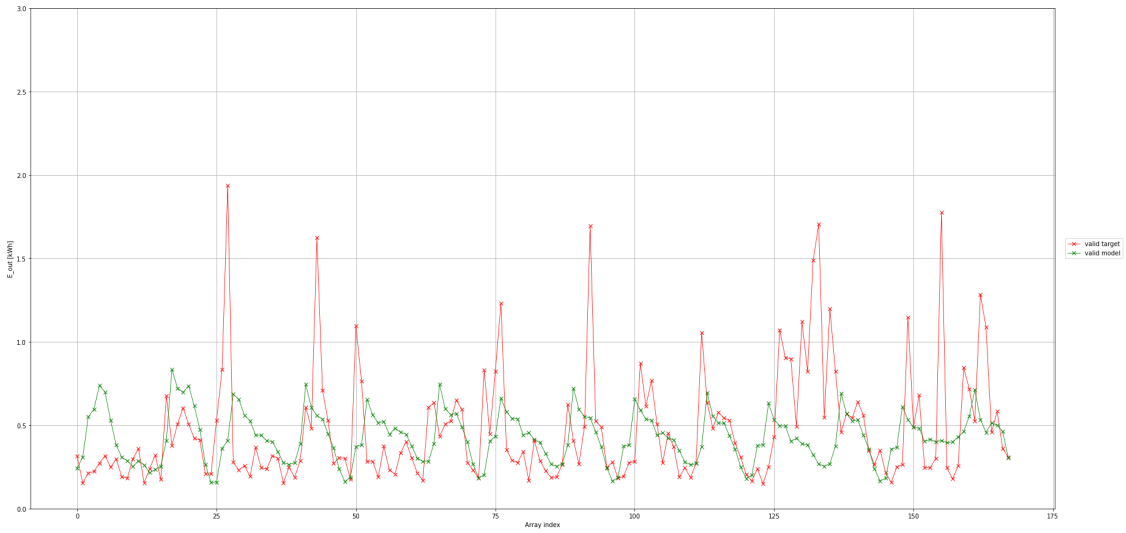





```
Epoch: 181      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.309
s
Epoch: 182      MSE Train: 0.131      MSE Valid: 0.091      Looptime: 2.340
s
Epoch: 183      MSE Train: 0.131      MSE Valid: 0.090      Looptime: 2.348
s
Epoch: 184      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.470
s
Epoch: 185      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.419
s
Epoch: 186      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.404
s
```

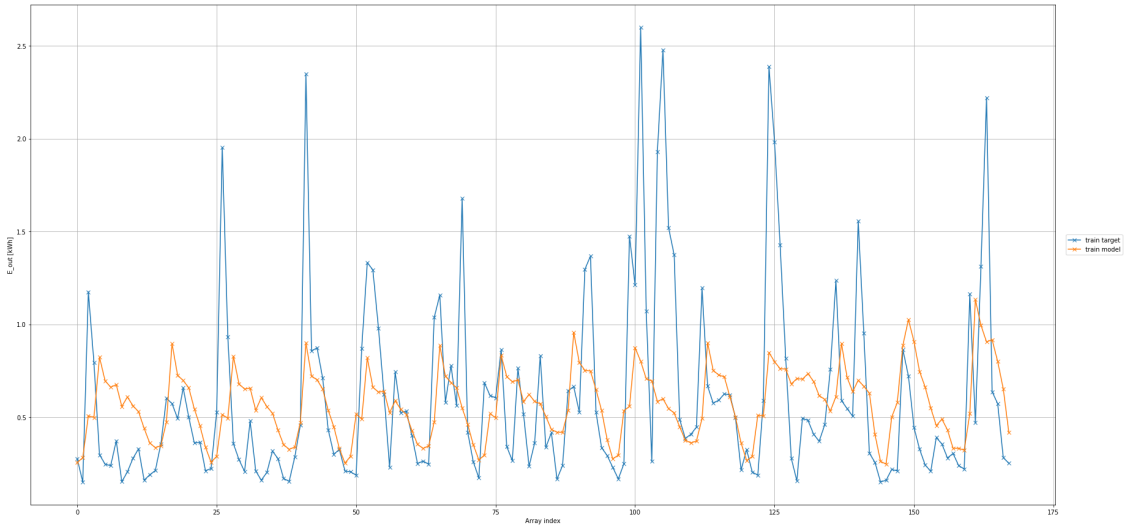
```
Epoch: 187      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.540
s
Epoch: 188      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.649
s
Epoch: 189      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.711
s
Epoch: 190      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.690
s
```

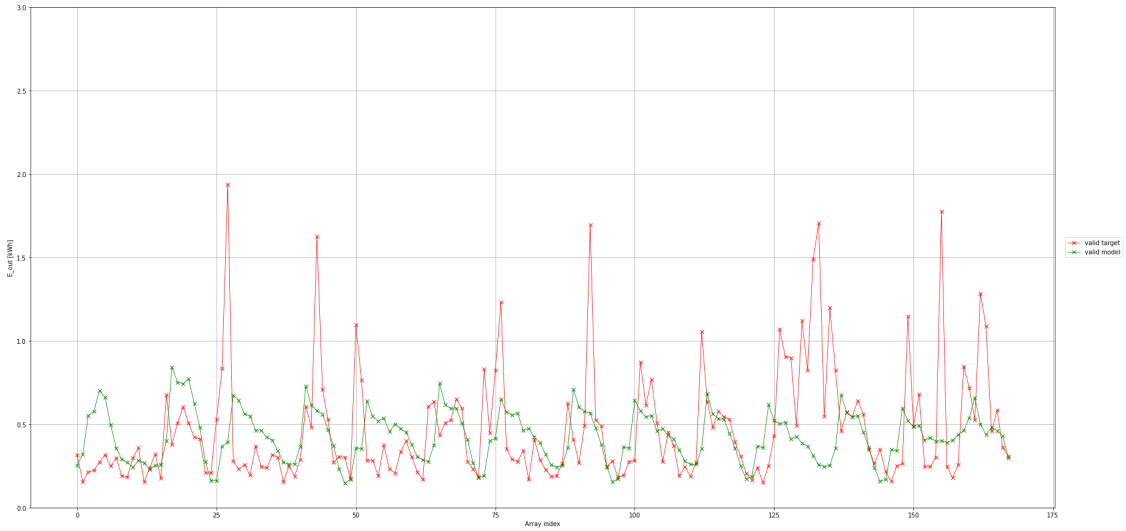




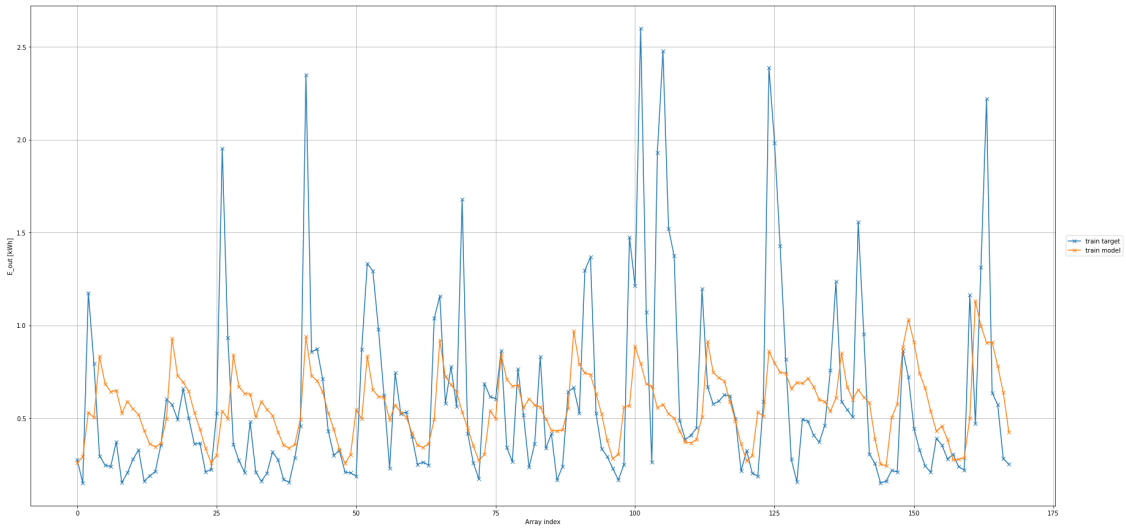

```
Epoch: 191      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.712
s
Epoch: 192      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.700
```

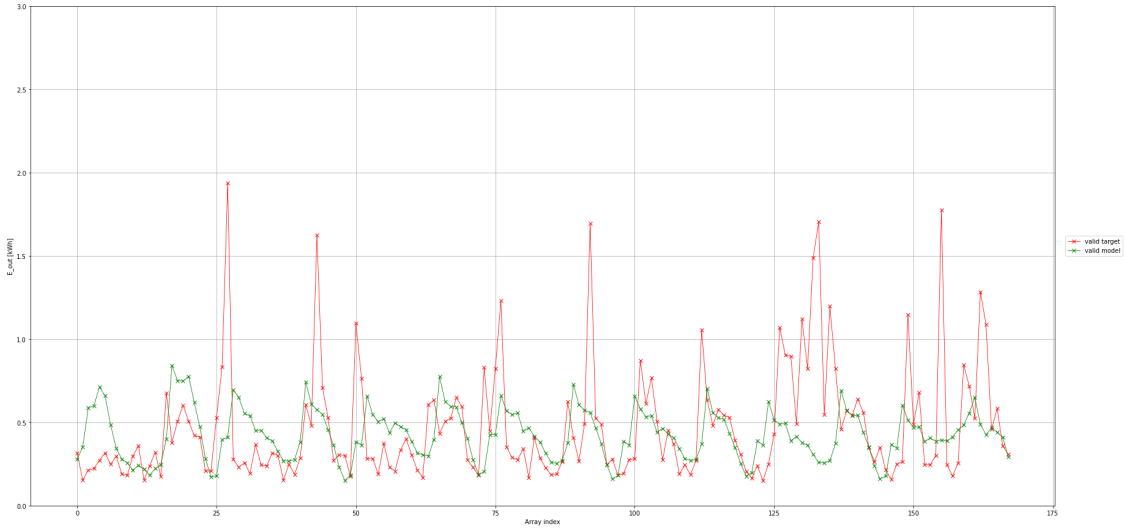
```
s
Epoch: 193      MSE Train: 0.130      MSE Valid: 0.090      Looptime: 2.683
s
Epoch: 194      MSE Train: 0.129      MSE Valid: 0.090      Looptime: 2.739
s
Epoch: 195      MSE Train: 0.129      MSE Valid: 0.090      Looptime: 2.713
s
Epoch: 196      MSE Train: 0.129      MSE Valid: 0.090      Looptime: 2.717
s
Epoch: 197      MSE Train: 0.129      MSE Valid: 0.090      Looptime: 2.705
s
Epoch: 198      MSE Train: 0.129      MSE Valid: 0.089      Looptime: 2.694
s
Epoch: 199      MSE Train: 0.129      MSE Valid: 0.089      Looptime: 2.653
s
Epoch: 200      MSE Train: 0.129      MSE Valid: 0.089      Looptime: 2.642
s
```





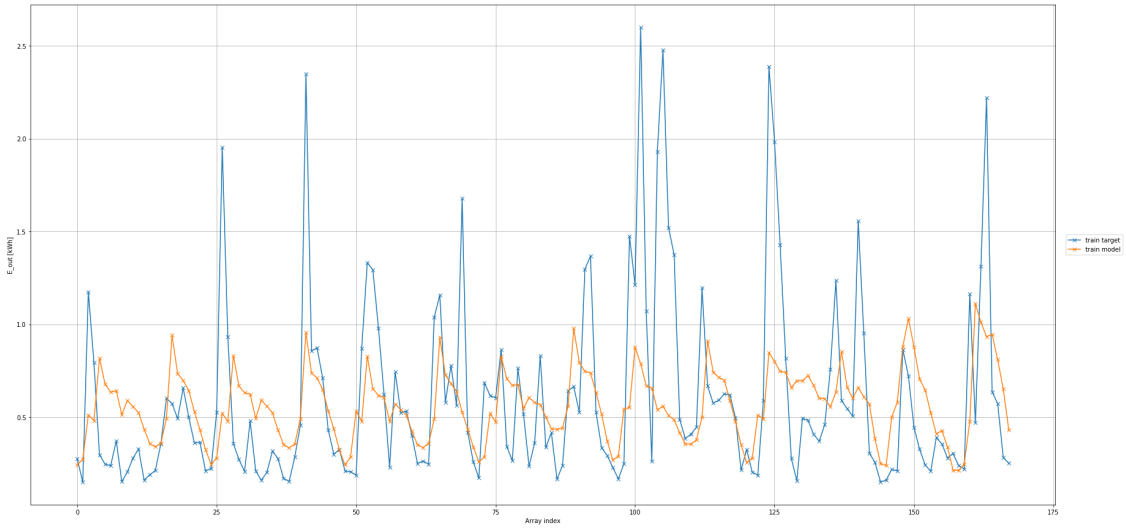

```
Epoch: 201      MSE Train: 0.129      MSE Valid: 0.089      Looptime: 2.610
s
Epoch: 202      MSE Train: 0.129      MSE Valid: 0.089      Looptime: 2.379
s
Epoch: 203      MSE Train: 0.128      MSE Valid: 0.089      Looptime: 2.382
s
Epoch: 204      MSE Train: 0.128      MSE Valid: 0.089      Looptime: 2.385
s
Epoch: 205      MSE Train: 0.128      MSE Valid: 0.089      Looptime: 2.374
s
Epoch: 206      MSE Train: 0.128      MSE Valid: 0.089      Looptime: 2.453
s
Epoch: 207      MSE Train: 0.128      MSE Valid: 0.089      Looptime: 2.460
s
Epoch: 208      MSE Train: 0.128      MSE Valid: 0.089      Looptime: 2.408
s
Epoch: 209      MSE Train: 0.128      MSE Valid: 0.089      Looptime: 2.478
s
Epoch: 210      MSE Train: 0.128      MSE Valid: 0.089      Looptime: 2.461
s
```

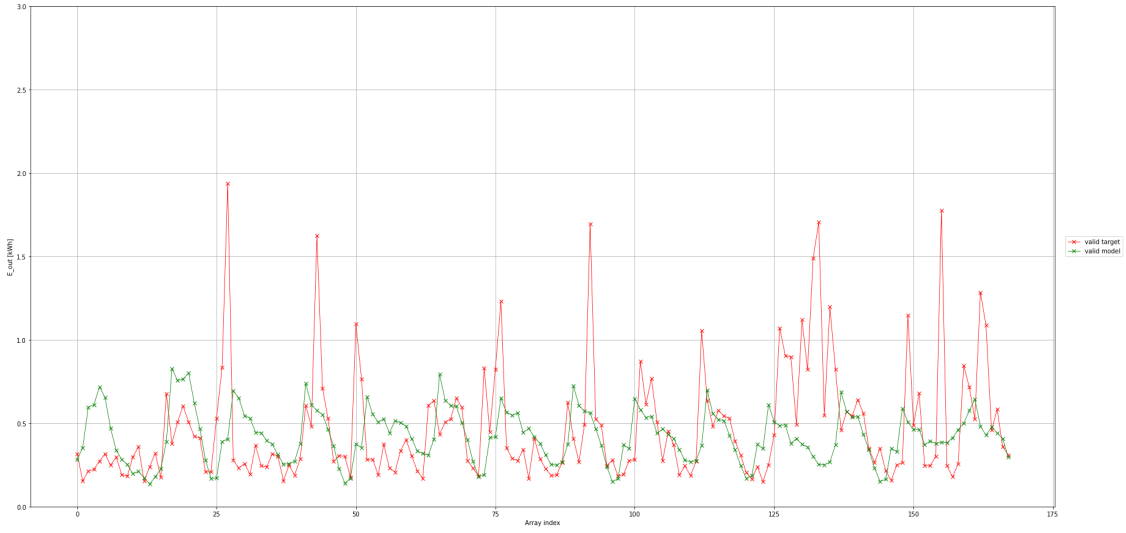





```
Epoch: 211      MSE Train: 0.128      MSE Valid: 0.089      Looptime: 2.403
s
Epoch: 212      MSE Train: 0.128      MSE Valid: 0.089      Looptime: 2.392
s
Epoch: 213      MSE Train: 0.128      MSE Valid: 0.089      Looptime: 2.397
s
Epoch: 214      MSE Train: 0.127      MSE Valid: 0.089      Looptime: 2.586
s
Epoch: 215      MSE Train: 0.127      MSE Valid: 0.089      Looptime: 2.369
s
Epoch: 216      MSE Train: 0.127      MSE Valid: 0.089      Looptime: 2.377
s
```

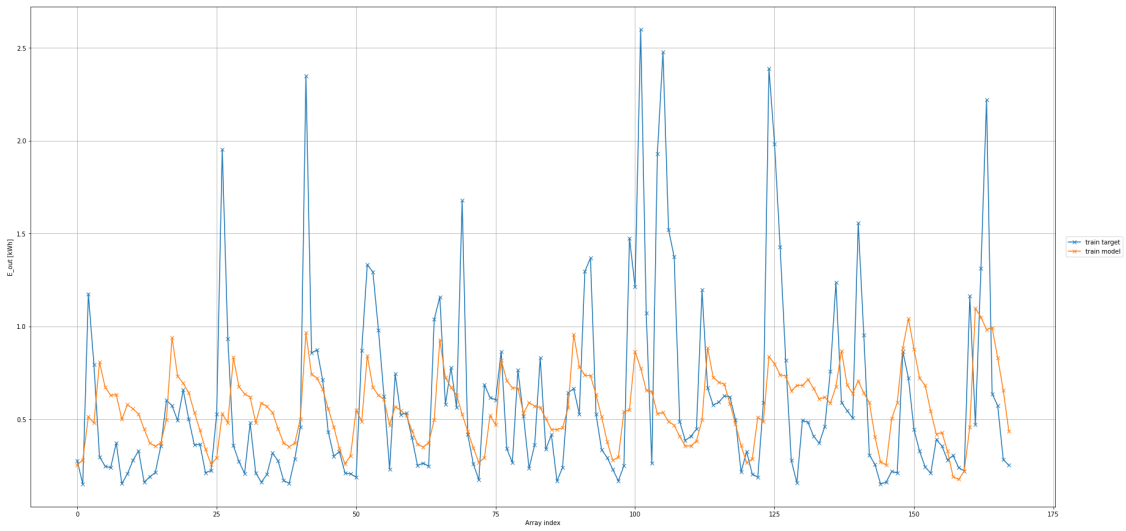
```
Epoch: 217      MSE Train: 0.127      MSE Valid: 0.088      Looptime: 2.356
s
Epoch: 218      MSE Train: 0.127      MSE Valid: 0.088      Looptime: 2.430
s
Epoch: 219      MSE Train: 0.127      MSE Valid: 0.088      Looptime: 2.369
s
Epoch: 220      MSE Train: 0.127      MSE Valid: 0.088      Looptime: 2.353
s
```

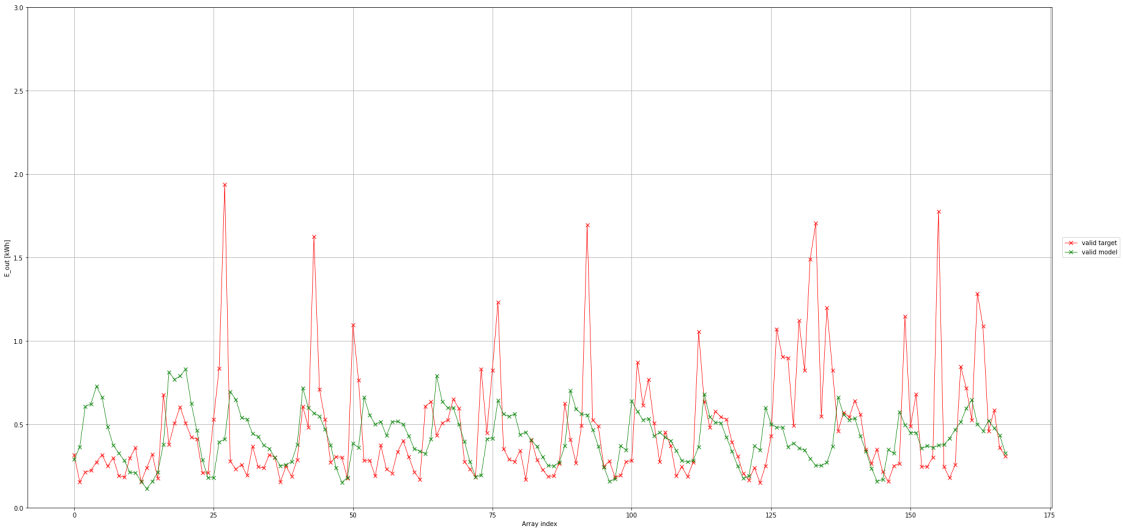




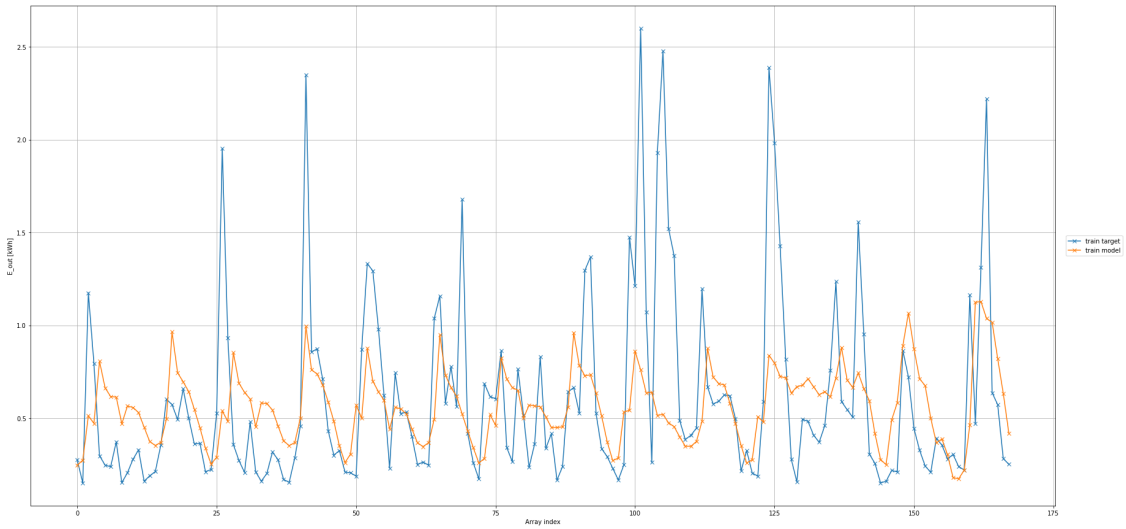

```
Epoch: 221      MSE Train: 0.127      MSE Valid: 0.088      Looptime: 2.383
s
Epoch: 222      MSE Train: 0.127      MSE Valid: 0.088      Looptime: 2.398
```

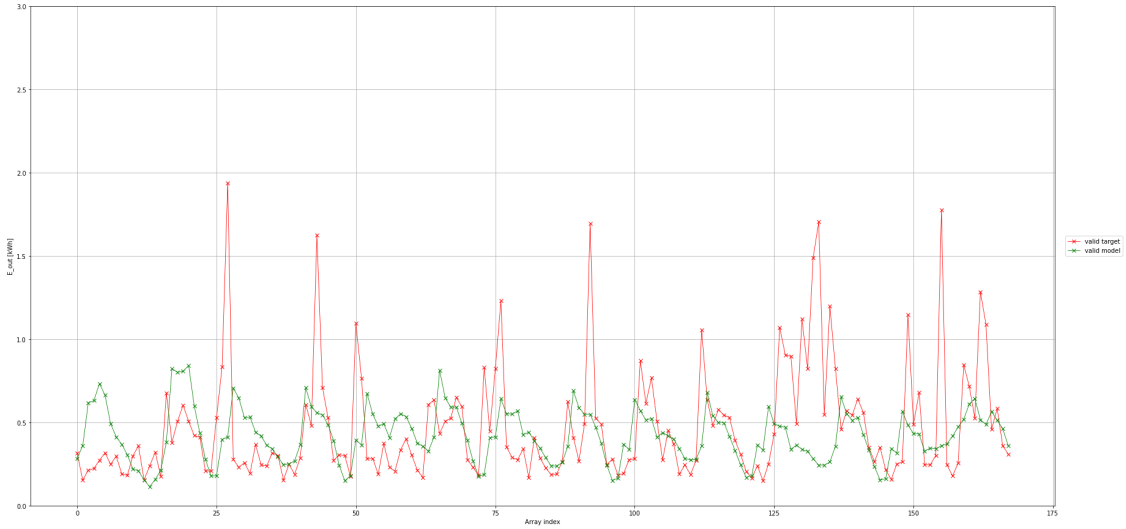
```
s
Epoch: 223      MSE Train: 0.127      MSE Valid: 0.088      Looptime: 2.521
s
Epoch: 224      MSE Train: 0.126      MSE Valid: 0.088      Looptime: 2.453
s
Epoch: 225      MSE Train: 0.126      MSE Valid: 0.088      Looptime: 2.624
s
Epoch: 226      MSE Train: 0.126      MSE Valid: 0.088      Looptime: 2.373
s
Epoch: 227      MSE Train: 0.126      MSE Valid: 0.088      Looptime: 2.355
s
Epoch: 228      MSE Train: 0.126      MSE Valid: 0.088      Looptime: 2.375
s
Epoch: 229      MSE Train: 0.126      MSE Valid: 0.088      Looptime: 2.570
s
Epoch: 230      MSE Train: 0.126      MSE Valid: 0.088      Looptime: 2.433
s
```






```
Epoch: 231      MSE Train: 0.126      MSE Valid: 0.087      Looptime: 2.445
s
Epoch: 232      MSE Train: 0.125      MSE Valid: 0.087      Looptime: 2.440
s
Epoch: 233      MSE Train: 0.125      MSE Valid: 0.087      Looptime: 2.560
s
Epoch: 234      MSE Train: 0.125      MSE Valid: 0.087      Looptime: 2.580
s
Epoch: 235      MSE Train: 0.125      MSE Valid: 0.087      Looptime: 2.394
s
Epoch: 236      MSE Train: 0.125      MSE Valid: 0.087      Looptime: 2.408
s
Epoch: 237      MSE Train: 0.125      MSE Valid: 0.087      Looptime: 2.377
s
Epoch: 238      MSE Train: 0.124      MSE Valid: 0.087      Looptime: 2.412
s
Epoch: 239      MSE Train: 0.124      MSE Valid: 0.087      Looptime: 2.490
s
Epoch: 240      MSE Train: 0.124      MSE Valid: 0.087      Looptime: 2.393
s
```





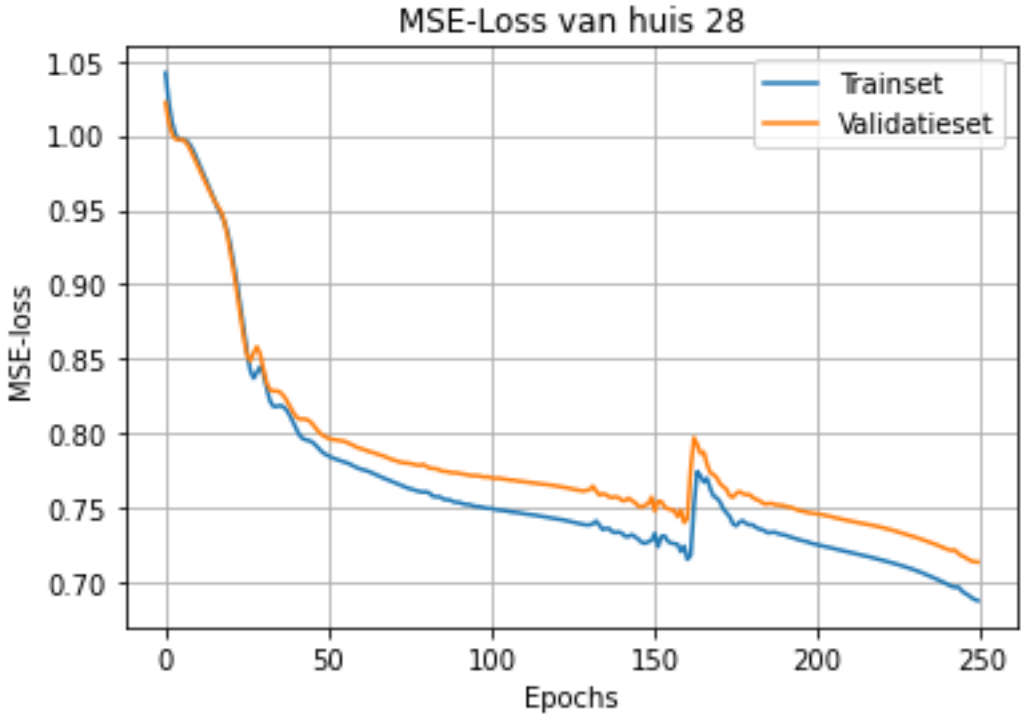

```
Epoch: 241      MSE Train: 0.124      MSE Valid: 0.086      Looptime: 2.371
s
Epoch: 242      MSE Train: 0.124      MSE Valid: 0.087      Looptime: 2.388
s
Epoch: 243      MSE Train: 0.124      MSE Valid: 0.086      Looptime: 2.487
s
Epoch: 244      MSE Train: 0.123      MSE Valid: 0.086      Looptime: 2.484
s
Epoch: 245      MSE Train: 0.123      MSE Valid: 0.086      Looptime: 2.470
s
Epoch: 246      MSE Train: 0.123      MSE Valid: 0.086      Looptime: 2.401
s
```

```
Epoch: 247       MSE Train: 0.122        MSE Valid: 0.086        Looptime: 2.472
s
Epoch: 248       MSE Train: 0.122        MSE Valid: 0.086        Looptime: 2.421
s
Epoch: 249       MSE Train: 0.122        MSE Valid: 0.086        Looptime: 2.451
s
```

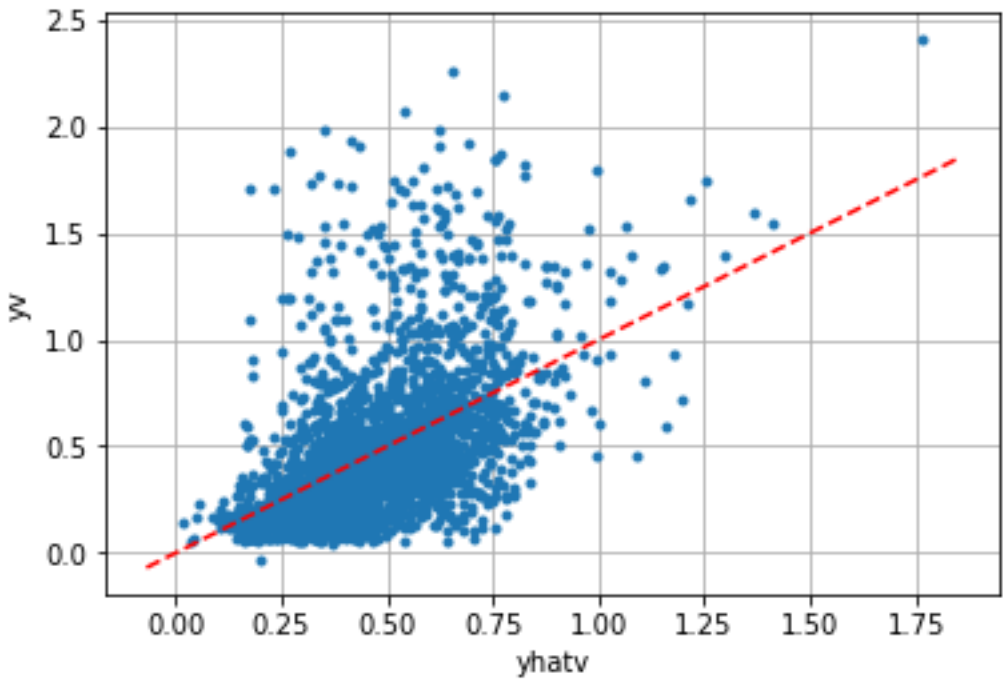
4.3 evaluation

Laat de loss over de Validatieset en de trainset zien per opoch

[14]:
```
%matplotlib inline
plt.plot([i*show_every for i in range(0,len(alijst))],alijst,label="Trainset")
plt.plot([i*show_every for i in range(0,len(blijst))],blijst,␣
 ↪label="Validatieset")

plt.title('MSE-Loss van huis 28' )
plt.xlabel("Epochs")
plt.ylabel("MSE-loss") # MSELoss
plt.legend()
#plt.xlim([150,250])
#plt.ylim([0.5,0.8])
plt.grid()
plt.show()
```



[15]:
```
plt.plot(yhatv, yv,  '.')
plt.plot(plt.xlim(), plt.xlim(), ls="--", c='r', label="$y$=$\hat{y}$")
plt.xlabel('yhatv')
plt.ylabel('yv')
plt.grid()
plt.show()
```



[16]:
```
y = yv
yhat = yhatv

r2 = r2_score(yhat, y)
rmse = np.sqrt(mean_squared_error(yhat, y))
actual, pred = np.array(y), np.array(yhat)
mape = np.mean(np.abs((actual - pred) / actual)) * 100
mae = mean_absolute_error(yhat, y)

print('R\u00b2: \t%.2f \nRMSE: \t%.2f \nMAPE: \t%.2f \nMAE: \t%.2f' %␣
 ↪(r2,rmse,mape,mae))
```

```
R²:     -1.43
RMSE:   0.29
MAPE:   62.39
MAE:    0.20
```

[ ]: