W15_LSTMProductionSimpleV03

January 12, 2021

[1]:
```python
version = "02"
```

1 TO DO:

2 Simpler form of the LSTM (from Week 14)

Jefry el Bhwash 16095065 (Created v01 and onward)

Niels van Drunen (joined from 03)

Levy Duivenvoorden (joined @03, left @05)

Using: - DL Lectures: 6.3 RNN Stationary - W14_Simple_LSTM_Consumption - W13_LSTM_Start_shaping - W15_LSTMConsumptionSimpleV06

Data: - House 28

Versions:

nr	Date	Changes
06	15/12/'20	

3 Initialization

[2]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


from IPython.display import display, HTML
import time
```

[3]:
```python
import random
#Neural Network imports
```

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error as mese
from sklearn.metrics import r2_score
```

[4]:
```python
#cuda imports
ngpu = torch.cuda.device_count() # number of available gpus
device = torch.device("cuda:0") 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

#Random Seed
random.seed(1337)
torch.manual_seed(1337)

def det(tensor):
    return tensor.detach().cpu().numpy()
```

4 Data Preparation

[5]:
```python
df = pd.read_pickle('consumptionOf28')
df['hour'] = df.index.hour
display(df.head(2))
```

	consumption	production	hour
2018-12-31 23:00:00	0.573	0.0	23
2019-01-01 00:00:00	0.435	0.0	0

4.1 Data Split

[6]:
```python
dft = df.filter(['hour','production'])
display(dft.head(24))
```

	hour	production
2018-12-31 23:00:00	23	0.00
2019-01-01 00:00:00	0	0.00
2019-01-01 01:00:00	1	0.00

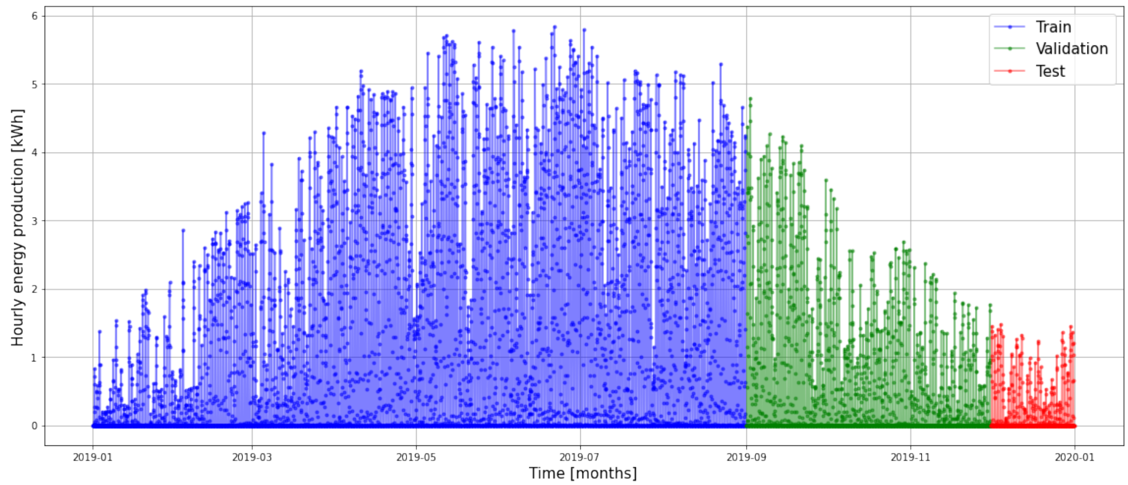
```
2019-01-01 02:00:00     2         0.00
2019-01-01 03:00:00     3         0.00
2019-01-01 04:00:00     4         0.00
2019-01-01 05:00:00     5         0.00
2019-01-01 06:00:00     6         0.00
2019-01-01 07:00:00     7         0.00
2019-01-01 08:00:00     8         0.01
2019-01-01 09:00:00     9         0.05
2019-01-01 10:00:00    10         0.19
2019-01-01 11:00:00    11         0.60
2019-01-01 12:00:00    12         0.71
2019-01-01 13:00:00    13         0.83
2019-01-01 14:00:00    14         0.27
2019-01-01 15:00:00    15         0.02
2019-01-01 16:00:00    16         0.00
2019-01-01 17:00:00    17         0.00
2019-01-01 18:00:00    18         0.00
2019-01-01 19:00:00    19         0.00
2019-01-01 20:00:00    20         0.00
2019-01-01 21:00:00    21         0.00
2019-01-01 22:00:00    22         0.00
```

[7]:

```
trdf = dft.loc['2019-01':'2019-08']
vadf = dft.loc['2019-09':'2019-11']
tedf = dft.loc['2019-12':]
```

[8]:

```
plt.figure(figsize=[16,7])
plt.plot(trdf.index, trdf.production, '.-', alpha=0.5, c='b', label='Train')
plt.plot(vadf.index, vadf.production, '.-', alpha=0.5, c='g',␣
 ↪label='Validation')
plt.plot(tedf.index, tedf.production, '.-', alpha=0.5, c='r', label='Test')
plt.xlabel('Time [months]', fontsize=15)
plt.ylabel('Hourly energy production [kWh]', fontsize=15)
plt.grid()
plt.tight_layout()
plt.legend(loc='upper right', fontsize=15)
plt.savefig('Production_year_split.png',dpi=1200)
```



[9]:
```
trdf.head()
```

[9]:

	hour	production
2019-01-01 00:00:00	0	0.0
2019-01-01 01:00:00	1	0.0
2019-01-01 02:00:00	2	0.0
2019-01-01 03:00:00	3	0.0
2019-01-01 04:00:00	4	0.0

4.1.1 Scaling

[10]:
```
dftr = trdf.copy()
dfva = vadf.copy()
dfte = tedf.copy()

scaler_X = StandardScaler()
scaler_y = StandardScaler()

scaler_X.fit(dftr.iloc[:,:-1])
scaler_y.fit(dftr.iloc[:,-1:])

#train
dftr.iloc[:,:-1] = scaler_X.transform(dftr.iloc[:,:-1])
dftr.iloc[:,-1:] = scaler_y.transform(dftr.iloc[:,-1:])
display(dftr.head())

#Valid
dfva.iloc[:,:-1] = scaler_X.transform(dfva.iloc[:,:-1])
dfva.iloc[:,-1:] = scaler_y.transform(dfva.iloc[:,-1:])
```

```
display(dfva.head())

#Test
dfte.iloc[:,:-1] = scaler_X.transform(dfte.iloc[:,:-1])
dfte.iloc[:,-1:] = scaler_y.transform(dfte.iloc[:,-1:])
display(dfte.head())
```

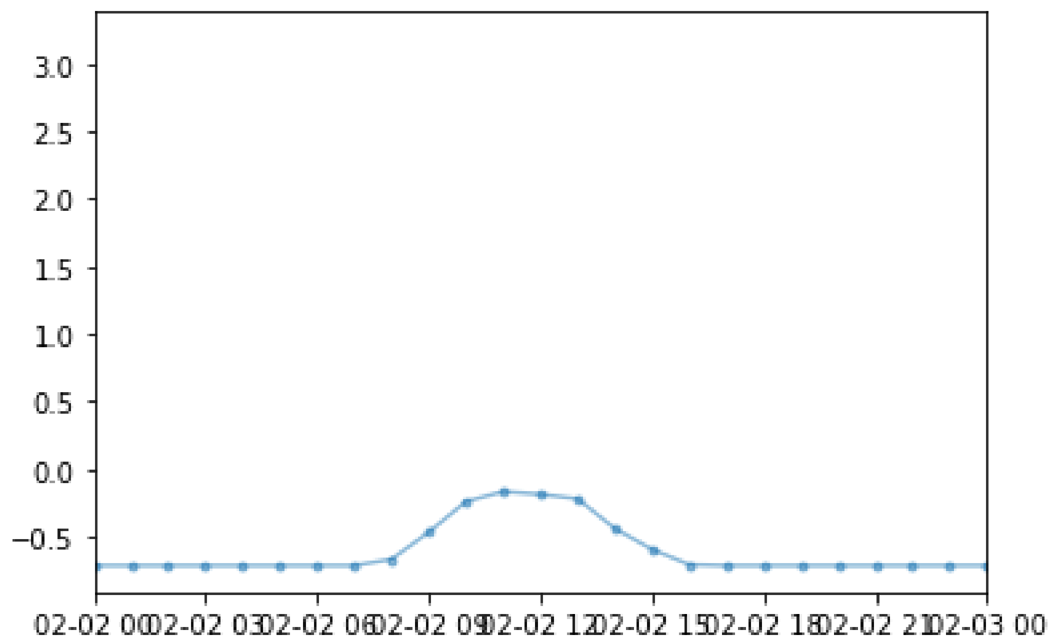
```
                         hour  production
2019-01-01 00:00:00 -1.661325   -0.712953
2019-01-01 01:00:00 -1.516862   -0.712953
2019-01-01 02:00:00 -1.372399   -0.712953
2019-01-01 03:00:00 -1.227936   -0.712953
2019-01-01 04:00:00 -1.083473   -0.712953

                         hour  production
2019-09-01 00:00:00 -1.661325   -0.712953
2019-09-01 01:00:00 -1.516862   -0.712953
2019-09-01 02:00:00 -1.372399   -0.712953
2019-09-01 03:00:00 -1.227936   -0.712953
2019-09-01 04:00:00 -1.083473   -0.712953

                         hour  production
2019-12-01 00:00:00 -1.661325   -0.712953
2019-12-01 01:00:00 -1.516862   -0.712953
2019-12-01 02:00:00 -1.372399   -0.712953
2019-12-01 03:00:00 -1.227936   -0.712953
2019-12-01 04:00:00 -1.083473   -0.712953
```

[11]:
```
dftr.head()
```

[11]:
```
                         hour  production
2019-01-01 00:00:00 -1.661325   -0.712953
2019-01-01 01:00:00 -1.516862   -0.712953
2019-01-01 02:00:00 -1.372399   -0.712953
2019-01-01 03:00:00 -1.227936   -0.712953
2019-01-01 04:00:00 -1.083473   -0.712953
```

[12]:
```
plt.plot(dftr.index, dftr.production, '.-', alpha=0.5)
plt.plot(dfva.index, dfva.production, '.-', alpha=0.5)
plt.plot(dfte.index, dfte.production, '.-', alpha=0.5)
plt.xlim(['2019-02-02','2019-02-03' ])
```

[12]:
```
(array(17929.), array(17930.))
```



4.2 3 dimensional for train data

batch | sequence | features

[13]:
```
def dim3Tensor(dft, window=7):
    #Get time shifted values and apply a moving window
    X = np.concatenate([ dft[i:i+window].to_numpy().reshape(1, window, dft.
 ↪shape[1]) for i in range(len(dft)-window-24) ], axis=0)

    #Get the target value (which is the next one in the sequence)
    y = dft.to_numpy()[window + 24:, -1]
    print(f"X_shape: {X.shape}")
    print(f"y_shape: {y.shape}")
    return X, y
```

[14]:
```
wsize = 7*24

dftr = dftr.astype(np.float32)
dfva = dfva.astype(np.float32)
dfte = dfte.astype(np.float32)

train_X, train_y = dim3Tensor(dftr, wsize)
valid_X, valid_y = dim3Tensor(dfva, wsize)
test_X, test_y = dim3Tensor(dfte, wsize)
```

```
X_shape: (5640, 168, 2)
y_shape: (5640,)
X_shape: (1992, 168, 2)
y_shape: (1992,)
X_shape: (551, 168, 2)
y_shape: (551,)
```

[15]:
```
display(train_X[0])
display(train_y[0])
#print(f"---\n{'Went OK' if train_y[0]== train_X[1][6] else '!!!Went NOK!!!'}")
```

```
array([[-1.6613247 , -0.7129529 ],
       [-1.5168618 , -0.7129529 ],
       [-1.3723987 , -0.7129529 ],
       [-1.2279357 , -0.7129529 ],
       [-1.0834727 , -0.7129529 ],
       [-0.93900967, -0.7129529 ],
       [-0.7945466 , -0.7129529 ],
       [-0.6500836 , -0.7129529 ],
       [-0.5056206 , -0.7062752 ],
       [-0.36115757, -0.67956454],
       [-0.21669453, -0.5860773 ],
       [-0.07223151, -0.31229302],
       [ 0.07223151, -0.23883872],
       [ 0.21669453, -0.15870674],
       [ 0.36115757, -0.53265595],
       [ 0.5056206 , -0.69959754],
       [ 0.6500836 , -0.7129529 ],
       [ 0.7945466 , -0.7129529 ],
       [ 0.93900967, -0.7129529 ],
       [ 1.0834727 , -0.7129529 ],
       [ 1.2279357 , -0.7129529 ],
       [ 1.3723987 , -0.7129529 ],
       [ 1.5168618 , -0.7129529 ],
       [ 1.6613247 , -0.7129529 ],
       [-1.6613247 , -0.7129529 ],
       [-1.5168618 , -0.7129529 ],
       [-1.3723987 , -0.7129529 ],
       [-1.2279357 , -0.7129529 ],
       [-1.0834727 , -0.7129529 ],
       [-0.93900967, -0.7129529 ],
       [-0.7945466 , -0.7129529 ],
       [-0.6500836 , -0.7129529 ],
       [-0.5056206 , -0.6595316 ],
       [-0.36115757, -0.5393336 ],
       [-0.21669453, -0.392425  ],
       [-0.07223151, -0.37906966],
       [ 0.07223151, -0.32564834],
```

```
[ 0.21669453, -0.44584632],
[ 0.36115757, -0.54601127],
[ 0.5056206 , -0.6862422 ],
[ 0.6500836 , -0.7129529 ],
[ 0.7945466 , -0.7129529 ],
[ 0.93900967, -0.7129529 ],
[ 1.0834727 , -0.7129529 ],
[ 1.2279357 , -0.7129529 ],
[ 1.3723987 , -0.7129529 ],
[ 1.5168618 , -0.7129529 ],
[ 1.6613247 , -0.7129529 ],
[-1.6613247 , -0.7129529 ],
[-1.5168618 , -0.7129529 ],
[-1.3723987 , -0.7129529 ],
[-1.2279357 , -0.7129529 ],
[-1.0834727 , -0.7129529 ],
[-0.93900967, -0.7129529 ],
[-0.7945466 , -0.7129529 ],
[-0.6500836 , -0.7129529 ],
[-0.5056206 , -0.6328209 ],
[-0.36115757, -0.392425  ],
[-0.21669453, -0.29226002],
[-0.07223151,  0.20188713],
[ 0.07223151, -0.32564834],
[ 0.21669453, -0.11864075],
[ 0.36115757, -0.11864075],
[ 0.5056206 , -0.6528539 ],
[ 0.6500836 , -0.7129529 ],
[ 0.7945466 , -0.7129529 ],
[ 0.93900967, -0.7129529 ],
[ 1.0834727 , -0.7129529 ],
[ 1.2279357 , -0.7129529 ],
[ 1.3723987 , -0.7129529 ],
[ 1.5168618 , -0.7129529 ],
[ 1.6613247 , -0.7129529 ],
[-1.6613247 , -0.7129529 ],
[-1.5168618 , -0.7129529 ],
[-1.3723987 , -0.7129529 ],
[-1.2279357 , -0.7129529 ],
[-1.0834727 , -0.7129529 ],
[-0.93900967, -0.7129529 ],
[-0.7945466 , -0.7129529 ],
[-0.6500836 , -0.7129529 ],
[-0.5056206 , -0.7129529 ],
[-0.36115757, -0.6929199 ],
[-0.21669453, -0.6461762 ],
[-0.07223151, -0.6061103 ],
[ 0.07223151, -0.5793996 ],
```

```
[ 0.21669453, -0.6394986 ],
[ 0.36115757, -0.6862422 ],
[ 0.5056206 , -0.7129529 ],
[ 0.6500836 , -0.7129529 ],
[ 0.7945466 , -0.7129529 ],
[ 0.93900967, -0.7129529 ],
[ 1.0834727 , -0.7129529 ],
[ 1.2279357 , -0.7129529 ],
[ 1.3723987 , -0.7129529 ],
[ 1.5168618 , -0.7129529 ],
[ 1.6613247 , -0.7129529 ],
[-1.6613247 , -0.7129529 ],
[-1.5168618 , -0.7129529 ],
[-1.3723987 , -0.7129529 ],
[-1.2279357 , -0.7129529 ],
[-1.0834727 , -0.7129529 ],
[-0.93900967, -0.7129529 ],
[-0.7945466 , -0.7129529 ],
[-0.6500836 , -0.7129529 ],
[-0.5056206 , -0.7129529 ],
[-0.36115757, -0.6662092 ],
[-0.21669453, -0.6061103 ],
[-0.07223151, -0.5793996 ],
[ 0.07223151, -0.5860773 ],
[ 0.21669453, -0.5994326 ],
[ 0.36115757, -0.6662092 ],
[ 0.5056206 , -0.7129529 ],
[ 0.6500836 , -0.7129529 ],
[ 0.7945466 , -0.7129529 ],
[ 0.93900967, -0.7129529 ],
[ 1.0834727 , -0.7129529 ],
[ 1.2279357 , -0.7129529 ],
[ 1.3723987 , -0.7129529 ],
[ 1.5168618 , -0.7129529 ],
[ 1.6613247 , -0.7129529 ],
[-1.6613247 , -0.7129529 ],
[-1.5168618 , -0.7129529 ],
[-1.3723987 , -0.7129529 ],
[-1.2279357 , -0.7129529 ],
[-1.0834727 , -0.7129529 ],
[-0.93900967, -0.7129529 ],
[-0.7945466 , -0.7129529 ],
[-0.6500836 , -0.7129529 ],
[-0.5056206 , -0.7129529 ],
[-0.36115757, -0.6595316 ],
[-0.21669453, -0.5860773 ],
[-0.07223151, -0.53265595],
[ 0.07223151, -0.57272196],
```

```
       [ 0.21669453, -0.6328209 ],
       [ 0.36115757, -0.6929199 ],
       [ 0.5056206 , -0.7129529 ],
       [ 0.6500836 , -0.7129529 ],
       [ 0.7945466 , -0.7129529 ],
       [ 0.93900967, -0.7129529 ],
       [ 1.0834727 , -0.7129529 ],
       [ 1.2279357 , -0.7129529 ],
       [ 1.3723987 , -0.7129529 ],
       [ 1.5168618 , -0.7129529 ],
       [ 1.6613247 , -0.7129529 ],
       [-1.6613247 , -0.7129529 ],
       [-1.5168618 , -0.7129529 ],
       [-1.3723987 , -0.7129529 ],
       [-1.2279357 , -0.7129529 ],
       [-1.0834727 , -0.7129529 ],
       [-0.93900967, -0.7129529 ],
       [-0.7945466 , -0.7129529 ],
       [-0.6500836 , -0.7129529 ],
       [-0.5056206 , -0.6929199 ],
       [-0.36115757, -0.5793996 ],
       [-0.21669453, -0.45920163],
       [-0.07223151, -0.35903668],
       [ 0.07223151, -0.45920163],
       [ 0.21669453, -0.5793996 ],
       [ 0.36115757, -0.6461762 ],
       [ 0.5056206 , -0.7062752 ],
       [ 0.6500836 , -0.7129529 ],
       [ 0.7945466 , -0.7129529 ],
       [ 0.93900967, -0.7129529 ],
       [ 1.0834727 , -0.7129529 ],
       [ 1.2279357 , -0.7129529 ],
       [ 1.3723987 , -0.7129529 ],
       [ 1.5168618 , -0.7129529 ],
       [ 1.6613247 , -0.7129529 ]], dtype=float32)
```

```
-0.7129529
```

[16]:
```python
train_X.shape
```

[16]:
```
(5640, 168, 2)
```

5 Tensors

[17]:
```
train_X_t = torch.from_numpy(np.array(train_X)).to(device)
train_y_t = torch.from_numpy(np.array(train_y)).to(device)

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

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

5.0.1 Dataloaders

[18]:
```
train_ds = TensorDataset(torch.tensor(train_X), torch.tensor(train_y))
train_dl = DataLoader(train_ds, batch_size=64, num_workers=3)
```

[19]:
```
train_y_t.shape
```

[19]:
```
torch.Size([5640])
```

5.0.2 Dataloader

6 LSTM Class

[20]:
```
class lstm(nn.Module):
    def __init__(self, features=1 ,hidden_state_size = 100):
        super().__init__()
        self.hidden_state_size = hidden_state_size
        self.lstm1 = nn.LSTM(features, self.hidden_state_size,␣
↪batch_first=True) #3 changed to 1
        self.lstm2 = nn.LSTM(self.hidden_state_size, self.hidden_state_size,␣
↪batch_first=True) #3 changed to 1
        self.linear2 = nn.Linear(self.hidden_state_size, 1)
#         self.dropout = nn.Dropout(0.08)

    def forward(self, X): #tensor X
        h0, _ = self.lstm1( X )           # h shaped (batch, sequence,␣
↪hidden_layer) #_hidden state saved, rest trashed
        h, _ = self.lstm2( h0 )           # h shaped (batch, sequence,␣
↪hidden_layer) #_hidden state saved, rest trashed
#         h = self.dropout(h1)
        h = h[:,-1, :]                    # only need the output for the last␣
↪sequence
```

```
        y = self.linear2(h)              # make a prediction
#          y = y + X[:,-1,-1:]             # make the output stationary #␣
↪enorme datalek? of miljoenen idee
        return y.view(-1)                # like always
```

[21]:
```
model = lstm(2).to(device)
```

7 LSTM training

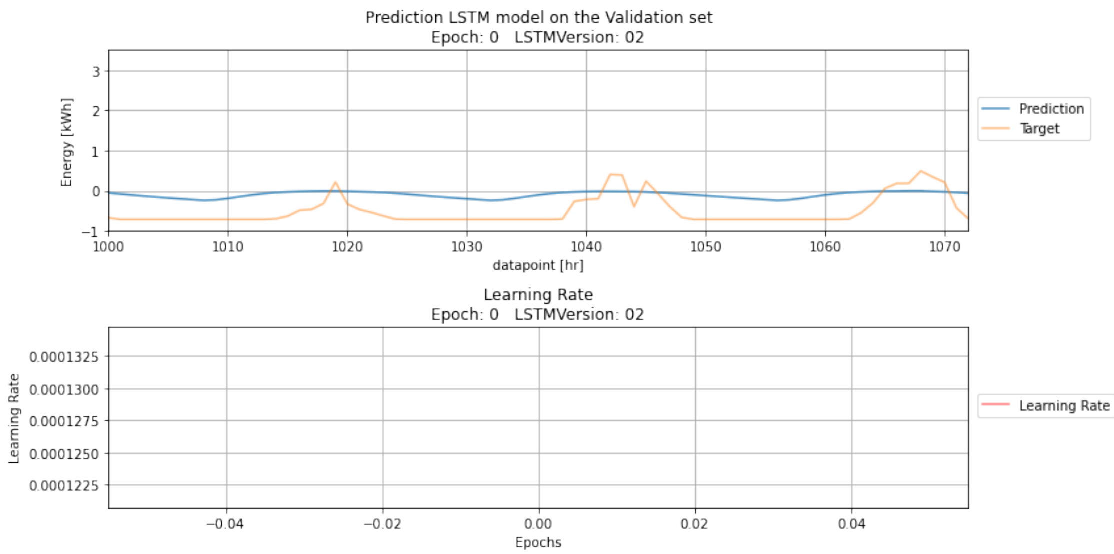
[22]:
```
device
```

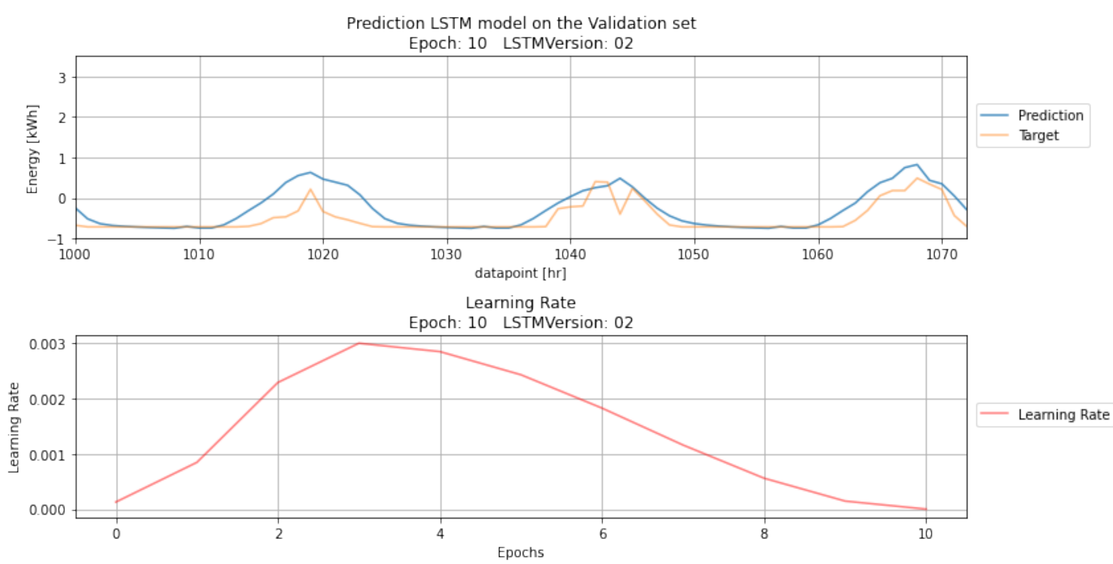
[22]:
```
device(type='cuda', index=0)
```

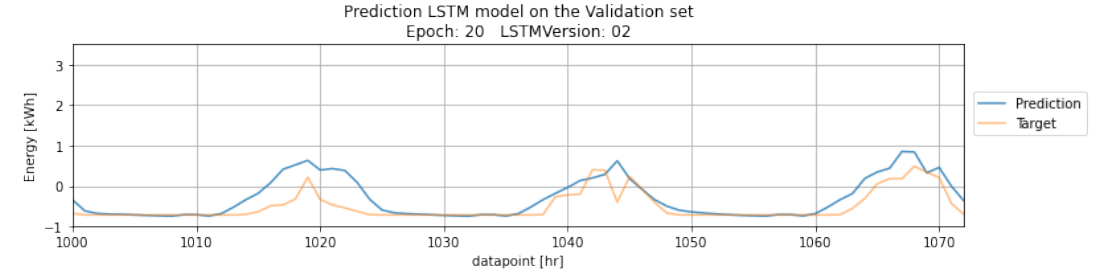
[23]:
```
train_for = 100 + 1 #amount of epochs
show_every = 10
reset_scheduler_after_n_epochs = 10 #10
max_lr = 3e-3
# weight_decay = 0.5

#visualization parameters
alijst = []; blijst = []; lrlijst = []

#Training parameters:
optimizer = optim.Adam(model.parameters(), lr=max_lr)
# , weight_decay=weight_decay
criterion = nn.SmoothL1Loss()
scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr=max_lr,␣
↪steps_per_epoch=len(train_dl), epochs=train_for)


#Learning loop:
for i in (range(train_for)):
    model.train()
    stime = time.time()
    for X, y in train_dl:
        X, y = X.to(device), y.to(device)
        """
        Training
        """
        optimizer.zero_grad()
        output = model(X)

        #bereken de loss over de output en update de parameters:
        loss = criterion(output, y)
```

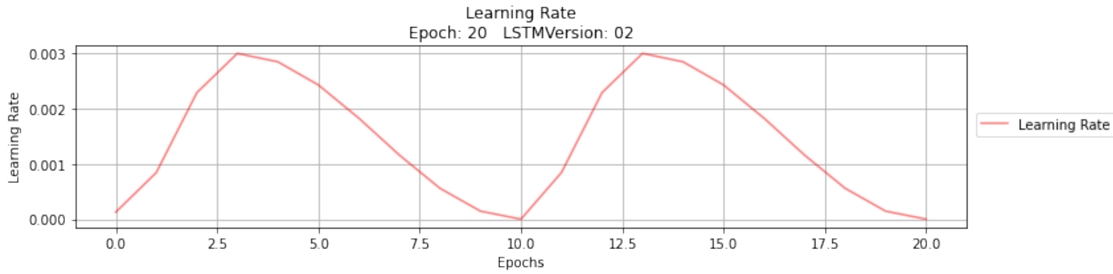
```
        lossT = mese(det(output), det(y))
        loss.backward()
        optimizer.step()
        scheduler.step()

    """
    Evaluation
    """
    model.eval()
    optimizer.zero_grad()

    dataV = valid_X_t;
    targetV = valid_y_t.view(-1);

    outputV = model(dataV)

    #bereken de loss over de output en update de parameters:
    lossV = mese(det(outputV), det(targetV))

    alijst.append(lossT)
    blijst.append(lossV)
    lrlijst.append(scheduler.get_last_lr())
    etime = time.time()
    if i%show_every == 0:
        plt.figure(figsize=(12,6))
        def plotting():

            plt.subplot(2,1,1)
            plt.plot(det(outputV), alpha=0.8, label="Prediction")
            plt.plot(det(targetV), alpha=0.5, label = "Target")
            plt.title(f'Prediction LSTM model on the Validation set\nEpoch: {i}␣
↪  LSTMVersion: {version}')
            plt.xlabel("datapoint [hr]")
            plt.ylabel("Energy [kWh]")
            plt.xlim([1000, 1072])
            plt.ylim([-1, 3.5])
            plt.grid()
            plt.legend(loc=(1.01, 0.5))

            plt.subplot(2,1,2)
            plt.plot([i for i in range(0,len(lrlijst))], lrlijst, alpha=0.5,␣
↪label="Learning Rate", c='r')
            plt.title(f'Learning Rate\nEpoch: {i}   LSTMVersion: {version}' )
            plt.xlabel("Epochs")
            plt.ylabel("Learning Rate")
            plt.grid()
            plt.legend(loc=(1.01, 0.5))
```

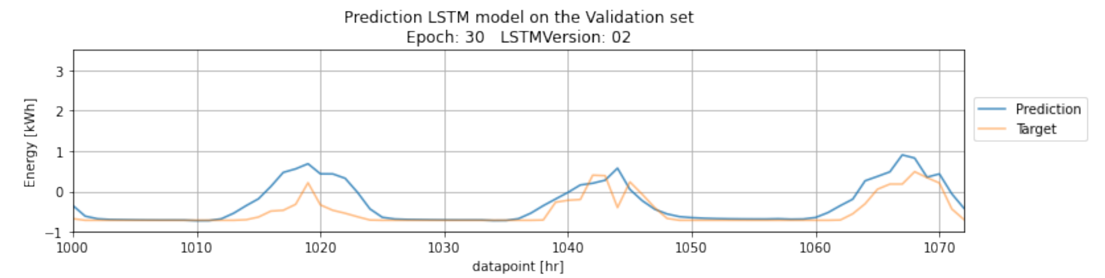
```python
            plt.tight_layout()
            plt.show()
        plotting()
    # reset scheduler after n epochs so we can have endless scheduler cycles
    if i % reset_scheduler_after_n_epochs == 0:
        scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr=max_lr, steps_per_epoch=len(train_dl), epochs=reset_scheduler_after_n_epochs)
        pass
```

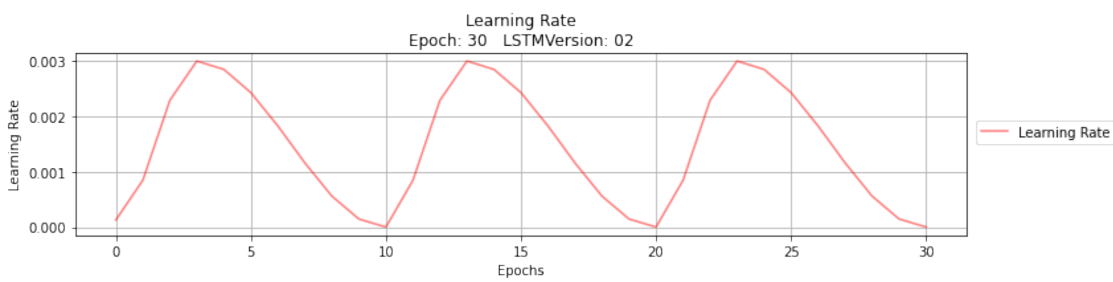


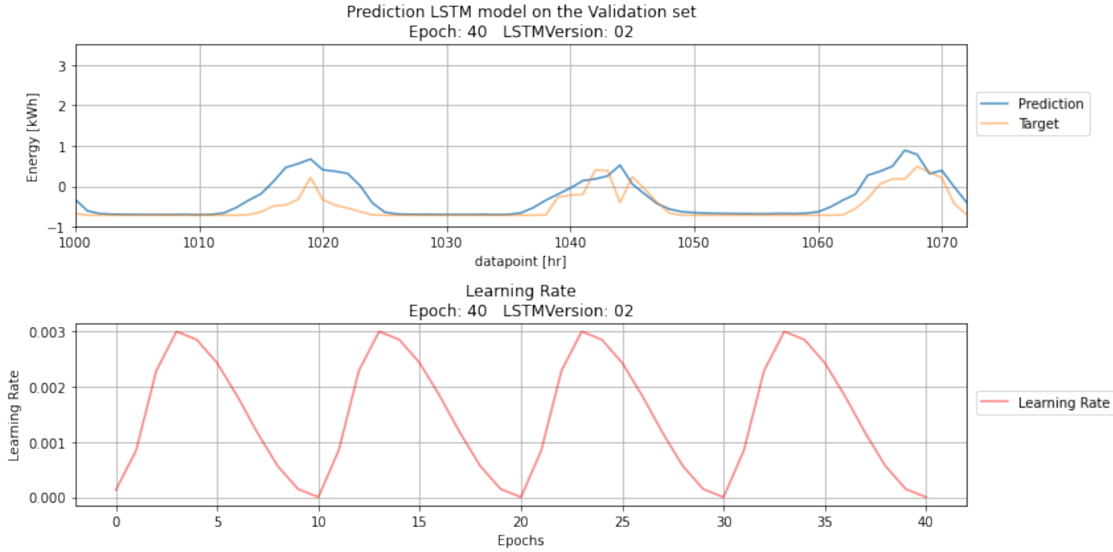


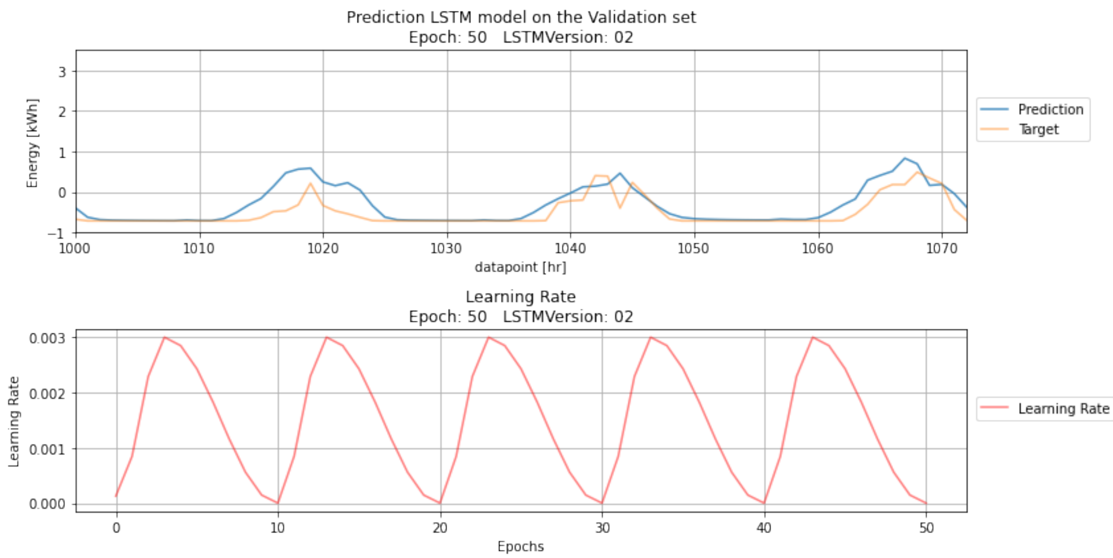

Prediction LSTM model on the Validation set
Epoch: 20 LSTMVersion: 02
Energy [kWh]
datapoint [hr]
Prediction
Target

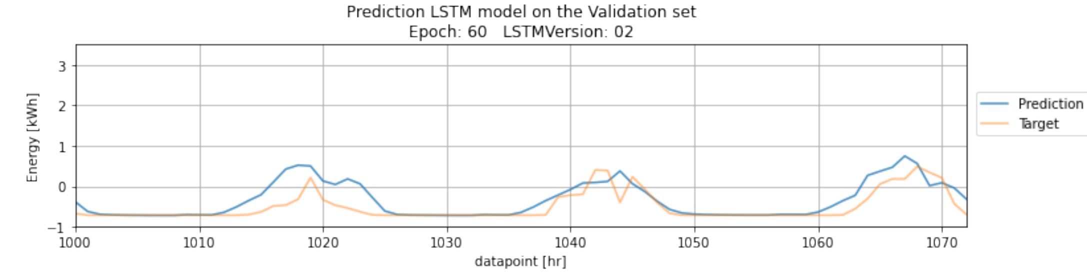

Learning Rate
Epoch: 20 LSTMVersion: 02
Learning Rate
Epochs
Learning Rate

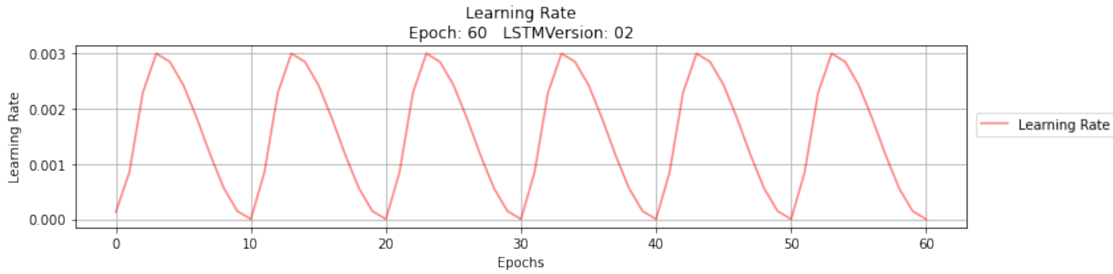

Prediction LSTM model on the Validation set
Epoch: 30 LSTMVersion: 02
Energy [kWh]
datapoint [hr]
Prediction
Target

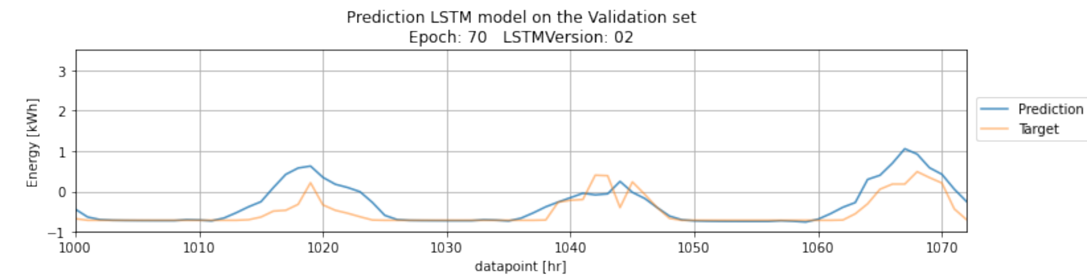

Learning Rate
Epoch: 30 LSTMVersion: 02
Learning Rate
Epochs
Learning Rate

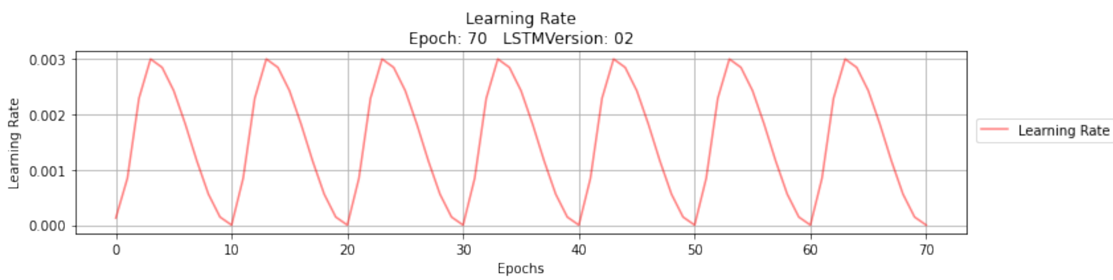

Prediction LSTM model on the Validation set
Epoch: 40 LSTMVersion: 02
Energy [kWh]
datapoint [hr]
Prediction
Target
Learning Rate
Epoch: 40 LSTMVersion: 02
Learning Rate
Epochs
Learning Rate

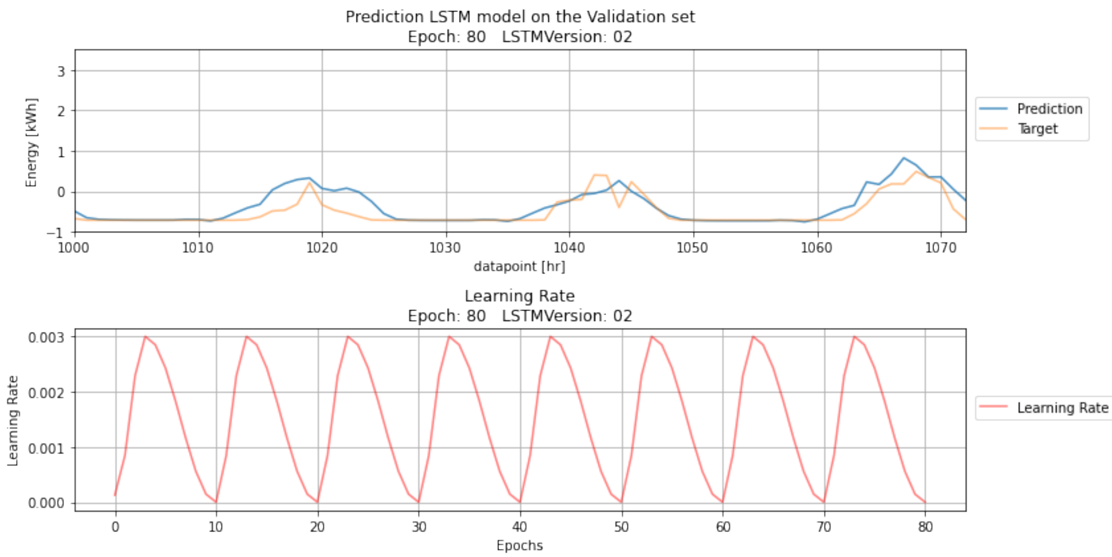

Prediction LSTM model on the Validation set
Epoch: 50 LSTMVersion: 02
Energy [kWh]
datapoint [hr]
Prediction
Target
Learning Rate
Epoch: 50 LSTMVersion: 02
Learning Rate
Epochs
Learning Rate

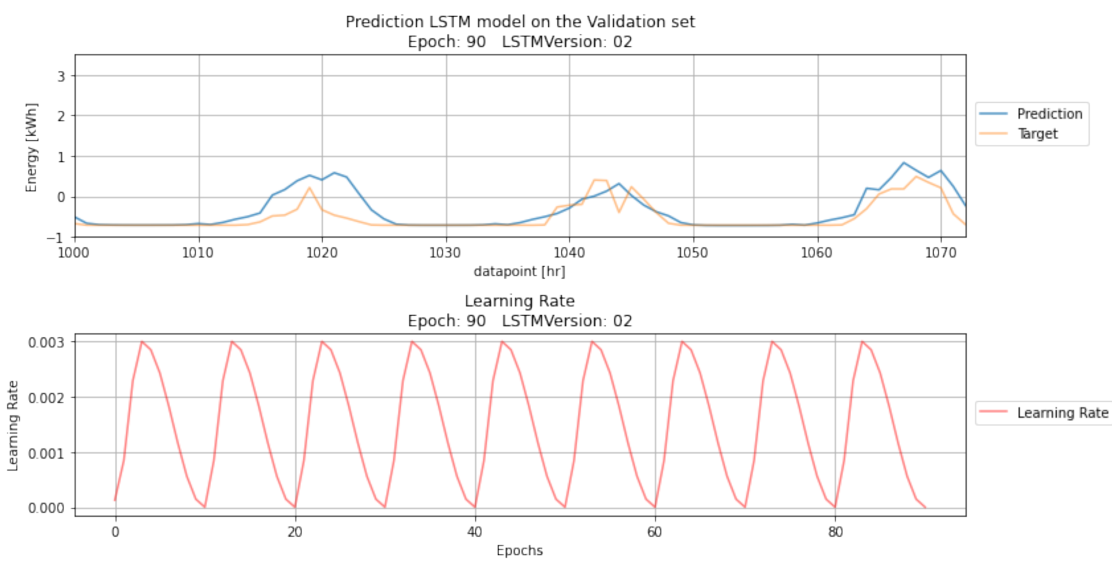

Prediction LSTM model on the Validation set
Epoch: 60 LSTMVersion: 02
Energy [kWh]
datapoint [hr]
Prediction
Target

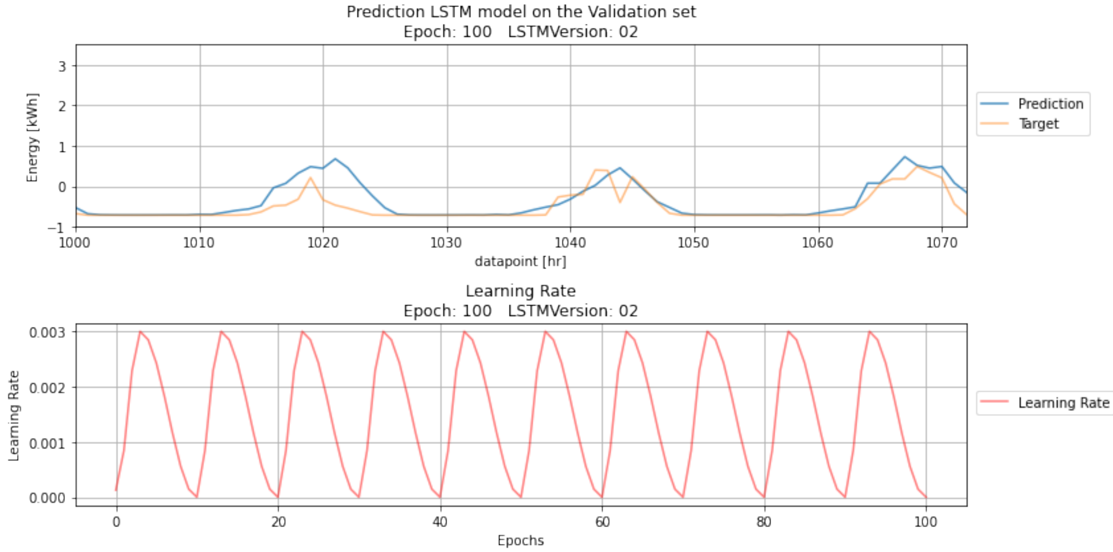

Learning Rate
Epoch: 60 LSTMVersion: 02
Learning Rate
Epochs
Learning Rate


Prediction LSTM model on the Validation set
Epoch: 70 LSTMVersion: 02
Energy [kWh]
datapoint [hr]
Prediction
Target


Learning Rate
Epoch: 70 LSTMVersion: 02
Learning Rate
Epochs
Learning Rate

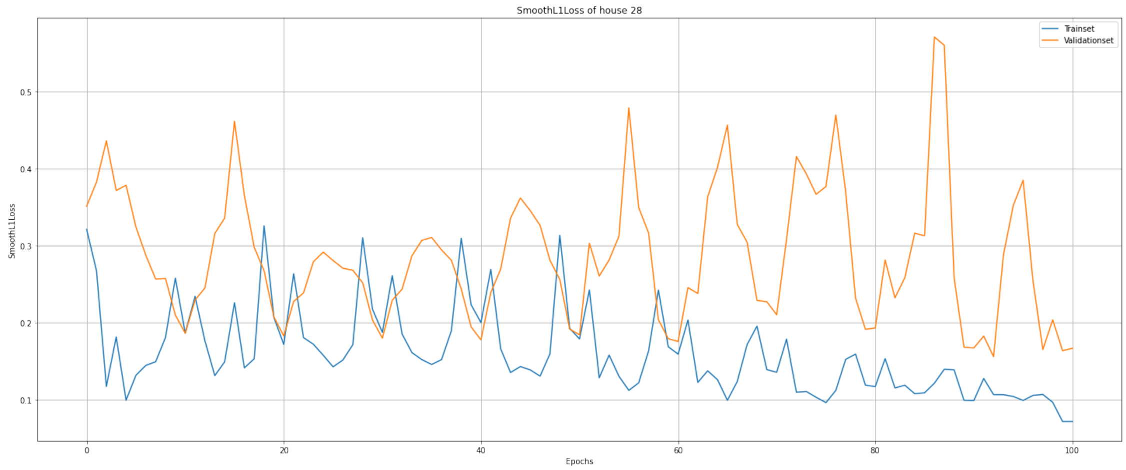

Prediction LSTM model on the Validation set
Epoch: 80 LSTMVersion: 02
Energy [kWh]
datapoint [hr]
Prediction
Target
Learning Rate
Epoch: 80 LSTMVersion: 02
Learning Rate
Epochs
Learning Rate


Prediction LSTM model on the Validation set
Epoch: 90 LSTMVersion: 02
Energy [kWh]
datapoint [hr]
Prediction
Target
Learning Rate
Epoch: 90 LSTMVersion: 02
Learning Rate
Epochs
Learning Rate

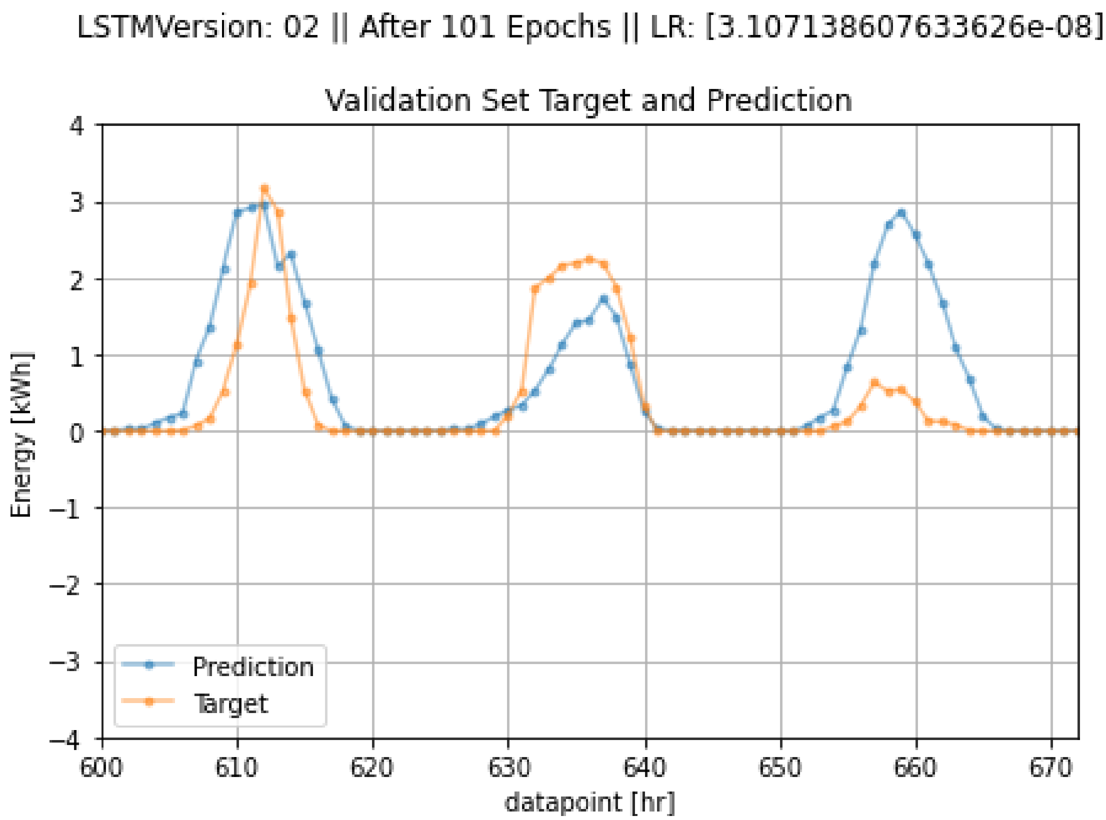


8 Visualizing the results

[24]:
```
%matplotlib inline

plt.figure(figsize = [25,10])
plt.plot([i for i in range(0,len(alijst))],alijst,label="Trainset")
plt.plot([i for i in range(0,len(blijst))],blijst,label="Validationset")

plt.title('SmoothL1Loss of house 28' )
plt.xlabel("Epochs")
plt.ylabel("SmoothL1Loss") # SmoothL1Loss
plt.legend()
plt.grid()
plt.show()
```

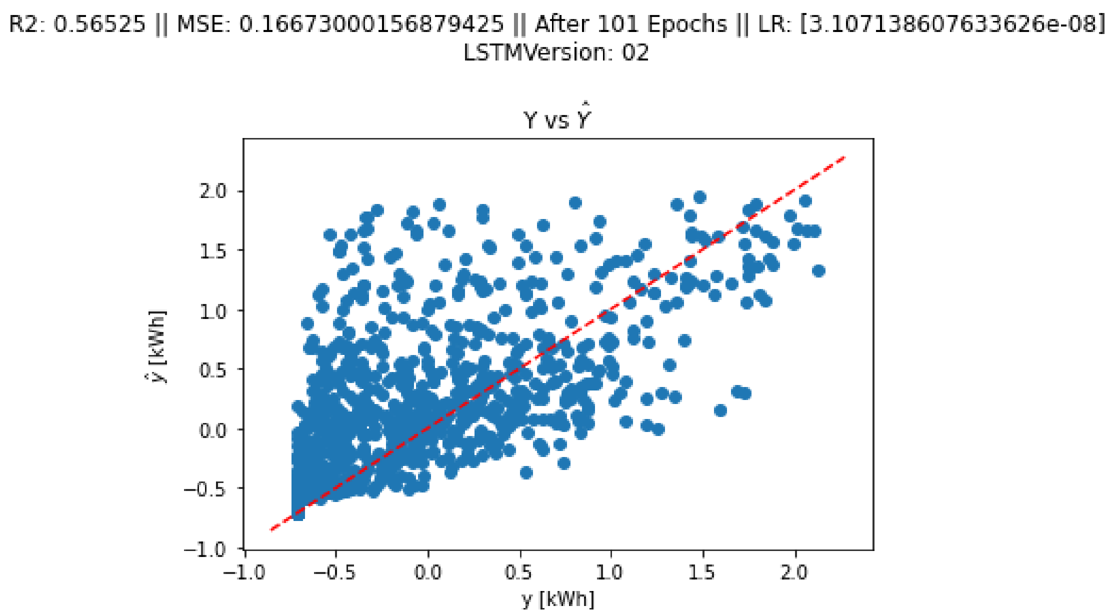


[25]:
```
plt.plot(scaler_y.inverse_transform(det(outputV)), '.-', alpha=0.5,␣
 ↪label="Prediction")
plt.plot(scaler_y.inverse_transform(det(targetV)), '.-', alpha=0.5, label =␣
 ↪"Target")
plt.tight_layout()
plt.title(f'LSTMVersion: {version} || After {train_for} Epochs || LR:␣
 ↪{lrlijst[-1]}\n\nValidation Set Target and Prediction')
plt.xlim([600, 672])
plt.ylim([-4,4])
plt.legend()
plt.xlabel("datapoint [hr]")
plt.ylabel("Energy [kWh]")
plt.grid()
plt.show()


print(f"Area under prediction: \t{det(outputV).sum()}\nArea under Target:␣
 ↪\t{det(targetV).sum()}\nDifference: {det(outputV).sum()-det(targetV).sum()}␣
 ↪")
```




```
Area under prediction:  -548.8289184570312
Area under Target:      -801.3095703125
Difference: 252.48065185546875
```

[26]:
```
y = det(valid_y_t)
yhat = det(model(valid_X_t))

r2 = round(r2_score(yhat, y),5)
lossV = round(mese(yhat, y),5)

plt.scatter(y, yhat)
plt.plot(plt.xlim(), plt.xlim(), ls="--", c='r', label="$y$=$\hat{y}$")
plt.title(f'R2: {r2} || MSE: {lossV} || After {train_for} Epochs || LR:␣
↪{lrlijst[-1]}\nLSTMVersion: {version}\n\nY vs $\hatY$')
plt.xlabel("y [kWh]")
plt.ylabel("$\haty$ [kWh]")
plt.show()
```



[27]:
```
%%html
<style>
table {float:left}
</style>
```

```
<IPython.core.display.HTML object>
```

[ ]: