W15_LSTMConsumptionSimpleV14

January 12, 2021

[1]:
```
version = "14"


stationary = True
scaling = True
resetting = True

hardcode = False
train_for = 1000 #2001
learningrate = 1e-3 #1e-3
```

1 TO DO:

- Revisit train/validate/split
- Create dummy variables of the hour and day
- Add features
  - isPeak
  - Mean of previous Day
  - Mean of previous Week

2 Simpler form of the LSTM (from Week 14)

Jefry el Bhwash 16095065 (Created v01 and onward)

Niels van Drunen (joined from 03m left@05 for the production variant)

Levy Duivenvoorden (joined @03, left @05)

Using: - DL Lectures: 6.3 RNN Stationary - W14_Simple_LSTM_Consumption - W13_LSTM_Start_shaping - Help session of mr Vuurens

Data: - House 28

Versions:

nr	Date	Changes
01	10/12/'20	First draft, does not work
02	11/12/'20	Added validation (valid set, no actual tests yet) and visualization
03	14/12/'20	Added this notebook to the server, added Levy and Niels, to try and understand LSTM's better as a group
04	14/12/'20	Added Validation and visualization during training
05	15/12/'20	Quick look at the test set to see if something weird is going on
06	15/12/'20	Removed Test, because that's what validation is for.Changed print while training. Added visualizations on the bottom and made the usage of stationary variable on the top of this notebook
07	Skipped	NvD added his own beautiful version of 07 to the server
08	15/12/'20	Scaled the dataframes going in
09	16/12/'20	Changed where Scaling happens
10	16/12/'20	Changed dim3 function
11	17/12/'20	Added multiple features, dataloader
12	18/12/'20	Inverse scaling fixed, added learningrate schedular + resetting, better training visualization
13		Tried it witout stationary
14	06/01/'21	Trained on significantly less epochs

3 Initialization

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

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

[3]:
```
import random
#Neural Network imports
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
from sklearn.metrics import r2_score
```

[4]:
```
#cuda imports
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

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

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

4 Data Preparation

[5]:
```
df = pd.read_pickle('consumptionOf28').drop(['production'], axis=1)
#adding hour feature
df['hour'] = df.index.hour
df['weekday'] = df.index.dayofweek
#df['day'] = df.index.day


#AUTOMATE THIS
cols = ['hour', 'weekday', 'consumption']
df = df[cols]
display(df.head(2))
```

	hour	weekday	consumption
2018-12-31 23:00:00	23	0	0.573

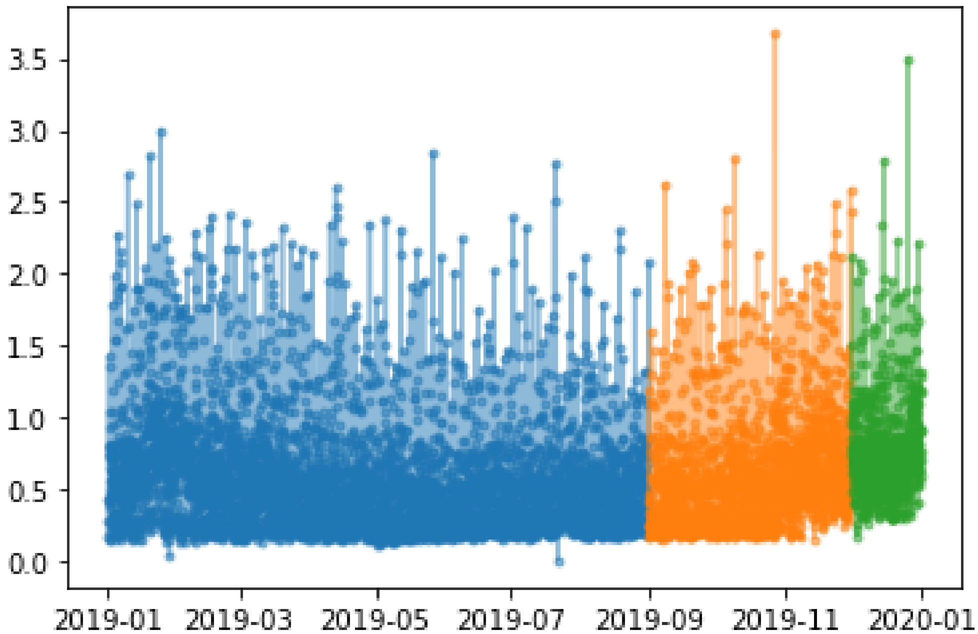
```
2019-01-01 00:00:00     0          1          0.435
```

4.1 Data Split

[6]:
```
# Split
trdf = df.loc['2019-01':'2019-08']
vadf = df.loc['2019-09':'2019-11']
tedf = df.loc['2019-12':]
```

[7]:
```
plt.plot(trdf.index, trdf.consumption, '.-', alpha=0.5)
plt.plot(vadf.index, vadf.consumption, '.-', alpha=0.5)
plt.plot(tedf.index, tedf.consumption, '.-', alpha=0.5)
```

[7]:
```
[<matplotlib.lines.Line2D at 0x7f5fffcca3c8>]
```



4.2 Scaling

[8]:
```
dftr = trdf.copy()
dfva = vadf.copy()
dfte = tedf.copy()
if scaling:
    print('Scaling: On')

    scaler_X = StandardScaler()
```

```python
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:])
print('Train')
display(dftr.head())

#Valid
dfva.iloc[:,:-1] = scaler_X.transform(dfva.iloc[:,:-1])
dfva.iloc[:,-1:] = scaler_y.transform(dfva.iloc[:,-1:])
print(f'\nValidation')
display(dfva.head())

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

```
Scaling: On
Train
```

	hour	weekday	consumption
2019-01-01 00:00:00	-1.661325	-1.005184	-0.305184
2019-01-01 01:00:00	-1.516862	-1.005184	-0.739154
2019-01-01 02:00:00	-1.372399	-1.005184	-0.999535
2019-01-01 03:00:00	-1.227936	-1.005184	2.293371
2019-01-01 04:00:00	-1.083473	-1.005184	-0.473512

```
Validation
```

	hour	weekday	consumption
2019-09-01 00:00:00	-1.661325	1.507776	-1.002165
2019-09-01 01:00:00	-1.516862	1.507776	-0.999535
2019-09-01 02:00:00	-1.372399	1.507776	-0.802276
2019-09-01 03:00:00	-1.227936	1.507776	-0.313075
2019-09-01 04:00:00	-1.083473	1.507776	1.675294

```
Test
```

	hour	weekday	consumption
2019-12-01 00:00:00	-1.661325	1.507776	0.802095
2019-12-01 01:00:00	-1.516862	1.507776	0.441769
2019-12-01 02:00:00	-1.372399	1.507776	0.483851

```
2019-12-01 03:00:00 -1.227936  1.507776     0.660069
2019-12-01 04:00:00 -1.083473  1.507776     1.186092
```

4.3 3 dimensional

batch | sequence | features

[9]:
```
def dim3(dft, window=7, gap=24):
    #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-gap) ], axis=0)

    #Get the target value (which is the next one in the sequence)
    y = dft.to_numpy()[window + gap:, -1]
    print(f"X_shape: {X.shape}")
    print(f"y_shape: {y.shape}")
    return X.astype(np.float32), y.astype(np.float32)
```

[10]:
```
wsize = 168
gsize = 24

print("Train")
train_X, train_y = dim3(dftr, wsize, gsize)
print("\nValid")
valid_X, valid_y = dim3(dfva, wsize, gsize)
print("\nTest")
test_X, test_y = dim3(dfte, wsize, gsize)
```

```
Train
X_shape: (5640, 168, 3)
y_shape: (5640,)

Valid
X_shape: (1992, 168, 3)
y_shape: (1992,)

Test
X_shape: (551, 168, 3)
y_shape: (551,)
```

[11]:
```
display(train_X[0][:10])
print("...")
display(train_y[0])
```

```
array([[-1.6613247 , -1.005184  , -0.30518422],
       [-1.5168618 , -1.005184  , -0.7391535 ],
       [-1.3723987 , -1.005184  , -0.9995351 ],
```

```
       [-1.2279357 , -1.005184  ,  2.2933714 ],
       [-1.0834727 , -1.005184  , -0.4735117 ],
       [-0.93900967, -1.005184  ,  0.64954823],
       [-0.7945466 , -1.005184  ,  0.49700147],
       [-0.6500836 , -1.005184  ,  1.2781461 ],
       [-0.5056206 , -1.005184  , -0.71285236],
       [-0.36115757, -1.005184  , -1.0626578 ]], dtype=float32)
```

…

```
-0.85487866
```

5 Tensors and Dataloader

Dataloader for train

Tensor for valid and test

[12]:
```
train_ds = TensorDataset(torch.tensor(train_X), torch.tensor(train_y))
train_dl = DataLoader(train_ds, batch_size=64, num_workers=3)

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)
```

6 LSTM Class

[13]:
```
class lstm(nn.Module):
    def __init__(self, feature_size=1, hidden_state_size = 100):
        super().__init__()
        self.hidden_state_size = hidden_state_size
        self.lstm1 = nn.LSTM(feature_size, self.hidden_state_size,␣
 ↪batch_first=True)
        self.linear2 = nn.Linear(self.hidden_state_size, 1)

    def forward(self, X): #tensor X
        h, _ = self.lstm1( X )          # h shaped (batch, sequence,␣
 ↪hidden_layer)
        h = h[:,-1, :]                  # only need the output for the last␣
 ↪sequence

        y = self.linear2(h)             # make a prediction
        if stationary:
            y = y + X[:,-1,-1:]         # make the output stationary
```

```
        return y.view(-1)                    # like always
```

6.0.1 LSTM Object

[14]:
```
model = lstm(df.shape[1]).to(device)
```

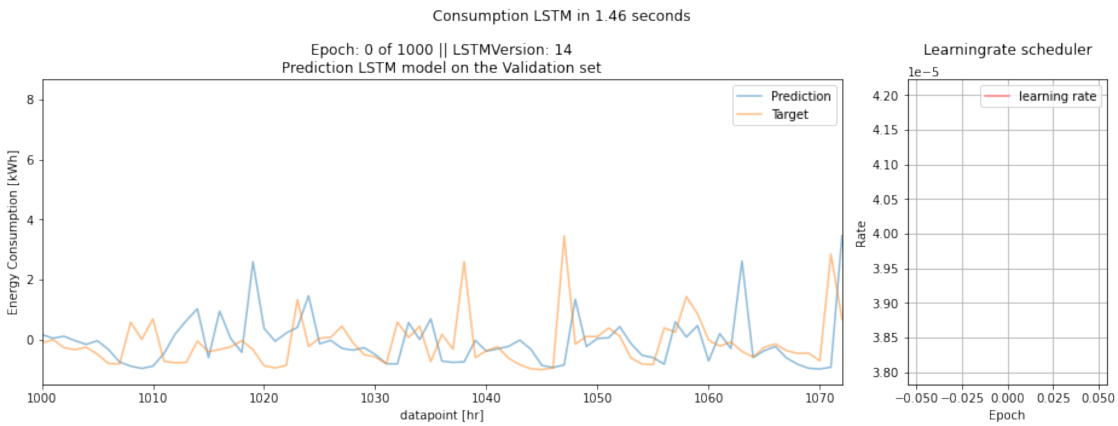
7 LSTM training

[15]:
```
stime = time.time()
show_every = 100
reset_scheduler_after_n_epochs = 100
if hardcode:
    train_for = 2001
    learningrate = 1e-3

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

#Training parameters:
optimizer = optim.AdamW(model.parameters(), lr=learningrate)
scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr=learningrate,␣
 ↪steps_per_epoch=len(train_dl), epochs=train_for)
#criterion = nn.SmoothL1Loss()
criterion = nn.L1Loss()
ttime = time.time()
#Learning loop:
for i in (range(train_for)):
    """
    Training
    """
    model.train()

    for X, y in train_dl:
        X, y = X.to(device), y.to(device)

        #train LSTM and reshape output:
        optimizer.zero_grad()
        output = model(X)

        #bereken de loss over de output en update de parameters:
        loss = criterion(output, y)
        lossT = mean_squared_error(det(output), det(y))
        loss.backward()
```

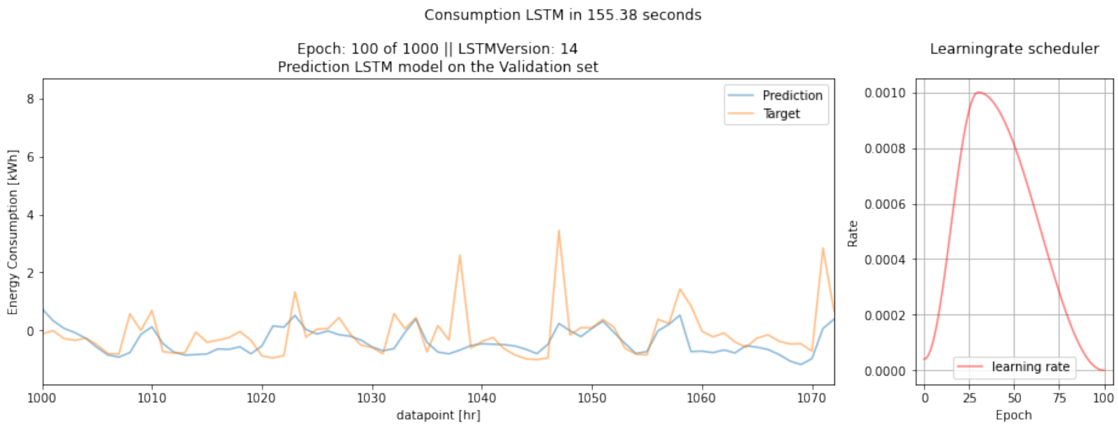
```
        optimizer.step()
        scheduler.step()
    """
    Evaluation
    """
    model.eval()
    optimizer.zero_grad()

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

    outputV = model(dataV)

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

    alijst.append(lossT)
    blijst.append(lossV)
    lrlijst.append(scheduler.get_last_lr())

    #Plotting the prediction on the validation set and plotting the learningrate
    if i%show_every == 0:
        vtime = time.time()
        fig = plt.figure(figsize=(13,5))
        spec = gridspec.GridSpec(ncols=2, nrows=1,width_ratios=[4, 1])

        display(HTML(f'<h3 style="text-align:center"> {i} </h3>'))
        fig.suptitle(f"Consumption LSTM in {round(vtime-ttime, 2)} seconds")

        ax0 = fig.add_subplot(spec[0],title=f'Epoch: {i} of {train_for} ||␣
↪LSTMVersion: {version}\nPrediction LSTM model on the Validation set',
                              xlabel='datapoint [hr]', ylabel='Energy␣
↪Consumption [kWh]', xlim=[1000,1072])
        ax0.plot(det(outputV), alpha=0.5, label="Prediction")
        ax0.plot(det(targetV), alpha=0.5, label = "Target")
        ax0.legend()

        ax1 = fig.add_subplot(spec[1], title=f'Learningrate scheduler\n',
                              xlabel= 'Epoch', ylabel='Rate' )
        ax1.plot([i for i in range(0,len(lrlijst))], lrlijst, alpha=0.5,␣
↪label='learning rate', c='r')
        ax1.grid()
        ax1.legend()#loc=(1.01, 0.5))

        plt.tight_layout()
        plt.show()
        ttime = time.time()
```

```
    if resetting:
        if i % reset_scheduler_after_n_epochs == 0:
            scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer,␣
↪max_lr=learningrate, steps_per_epoch=len(train_dl),␣
↪epochs=reset_scheduler_after_n_epochs)
            pass
etime = time.time()
```

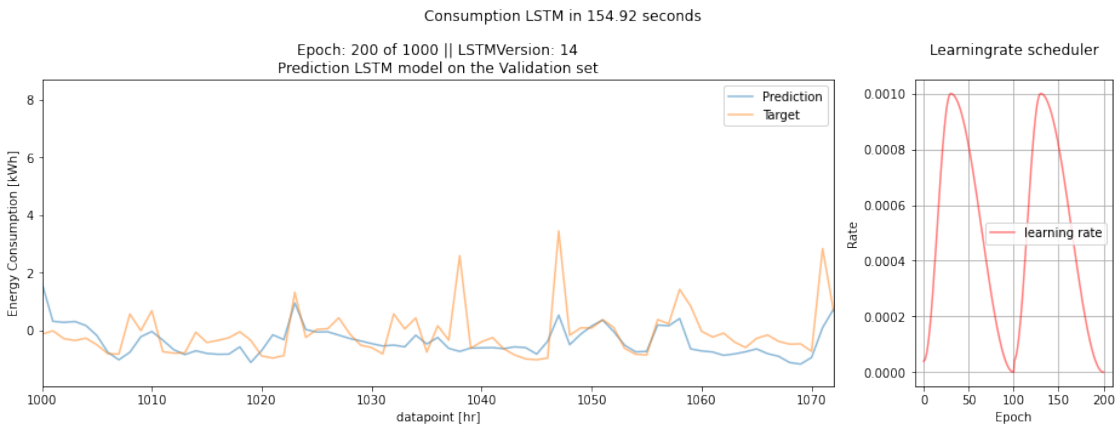
```
<IPython.core.display.HTML object>
```



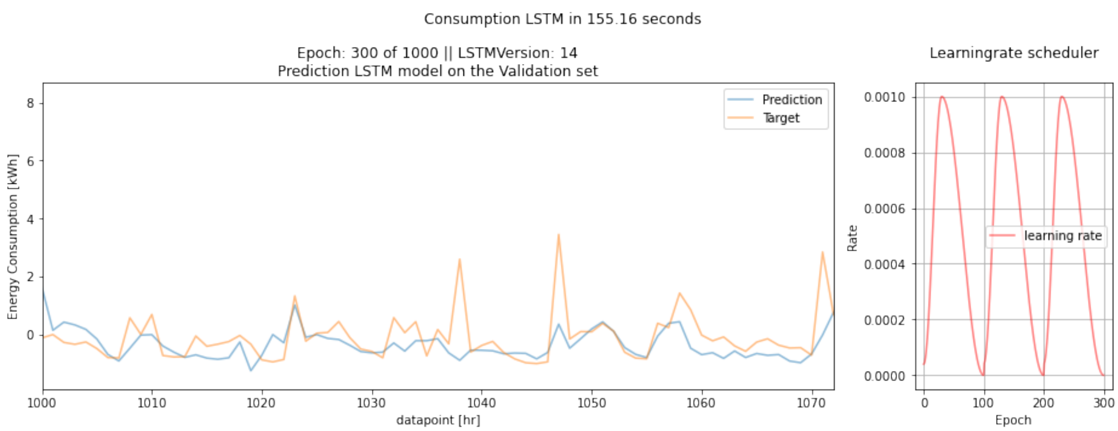

```
<IPython.core.display.HTML object>
```



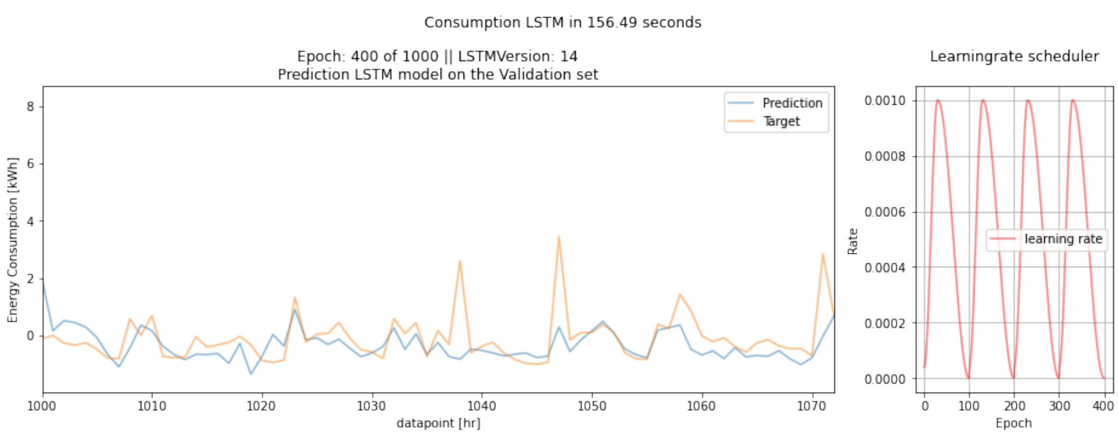

```
<IPython.core.display.HTML object>
```



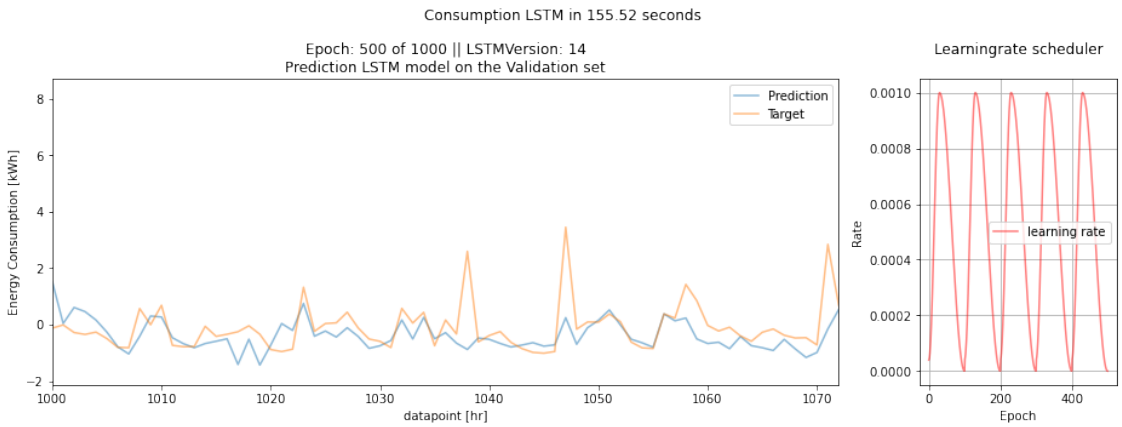
<IPython.core.display.HTML object>



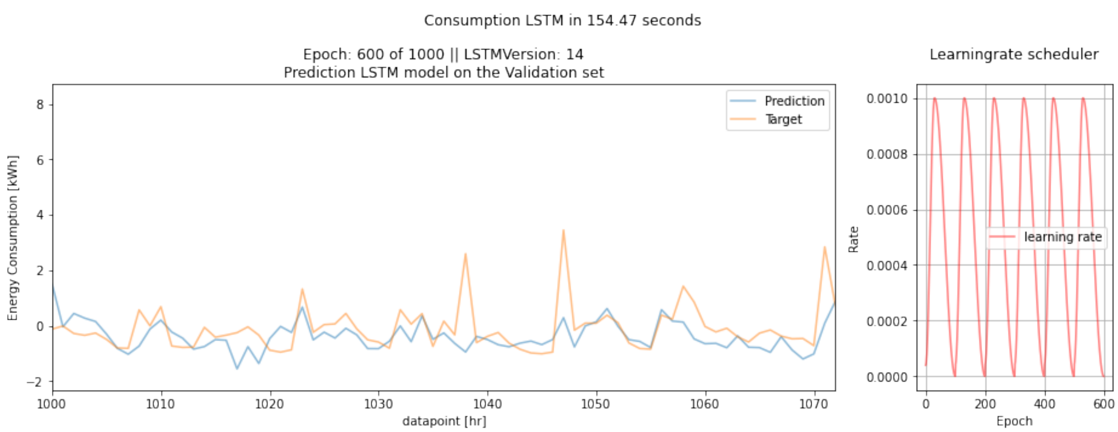
<IPython.core.display.HTML object>



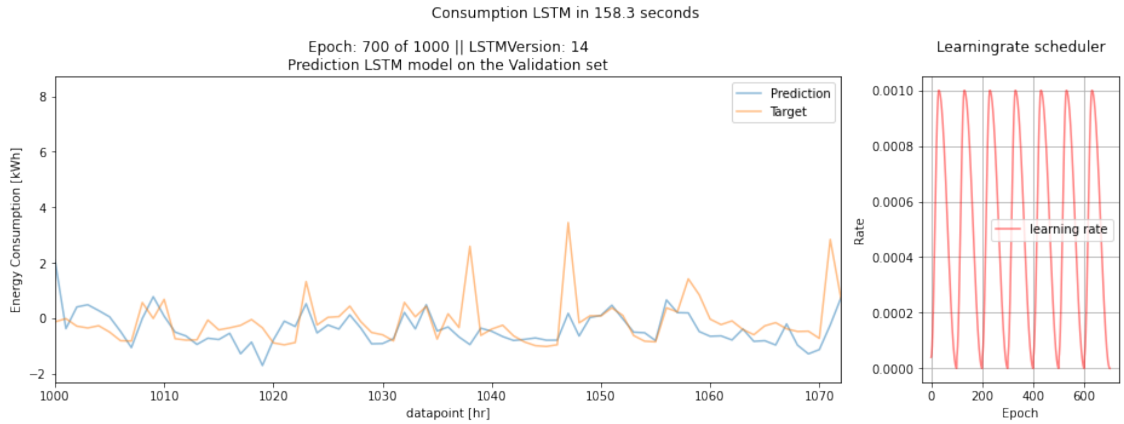
<IPython.core.display.HTML object>



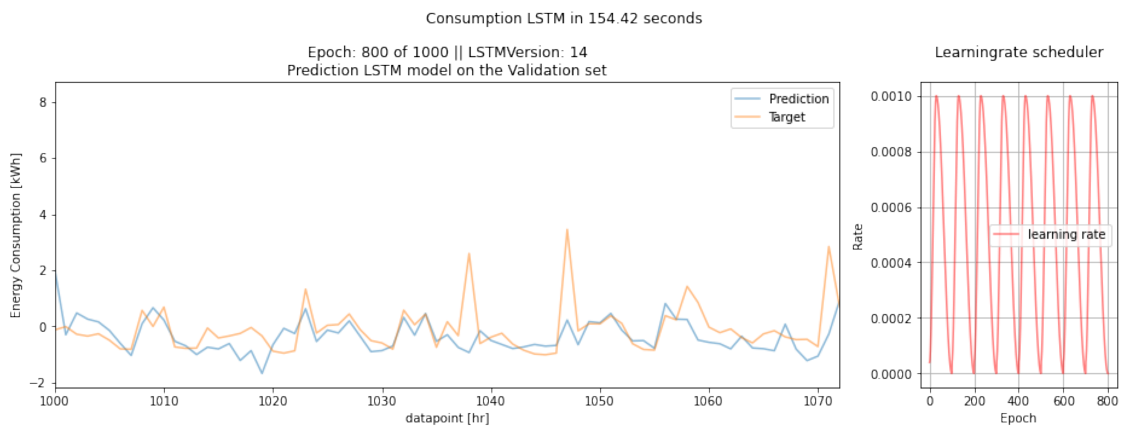
<IPython.core.display.HTML object>



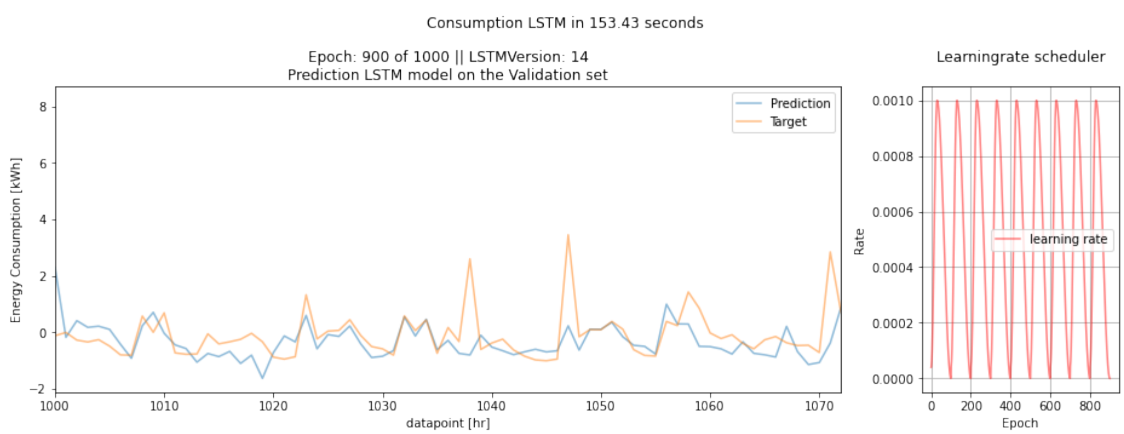
<IPython.core.display.HTML object>



<IPython.core.display.HTML object>



<IPython.core.display.HTML object>

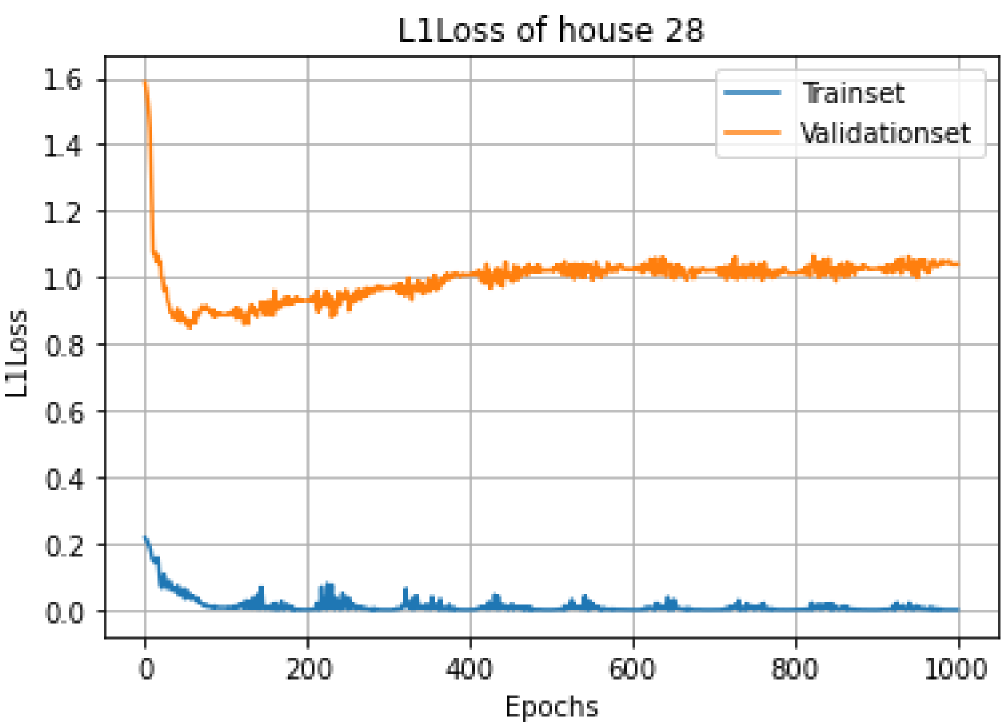


[16]:
```
print(f"Training Time for {train_for} epochs: {etime-stime}seconds")
```

```
Training Time for 1000 epochs: 1553.231360912323seconds
```

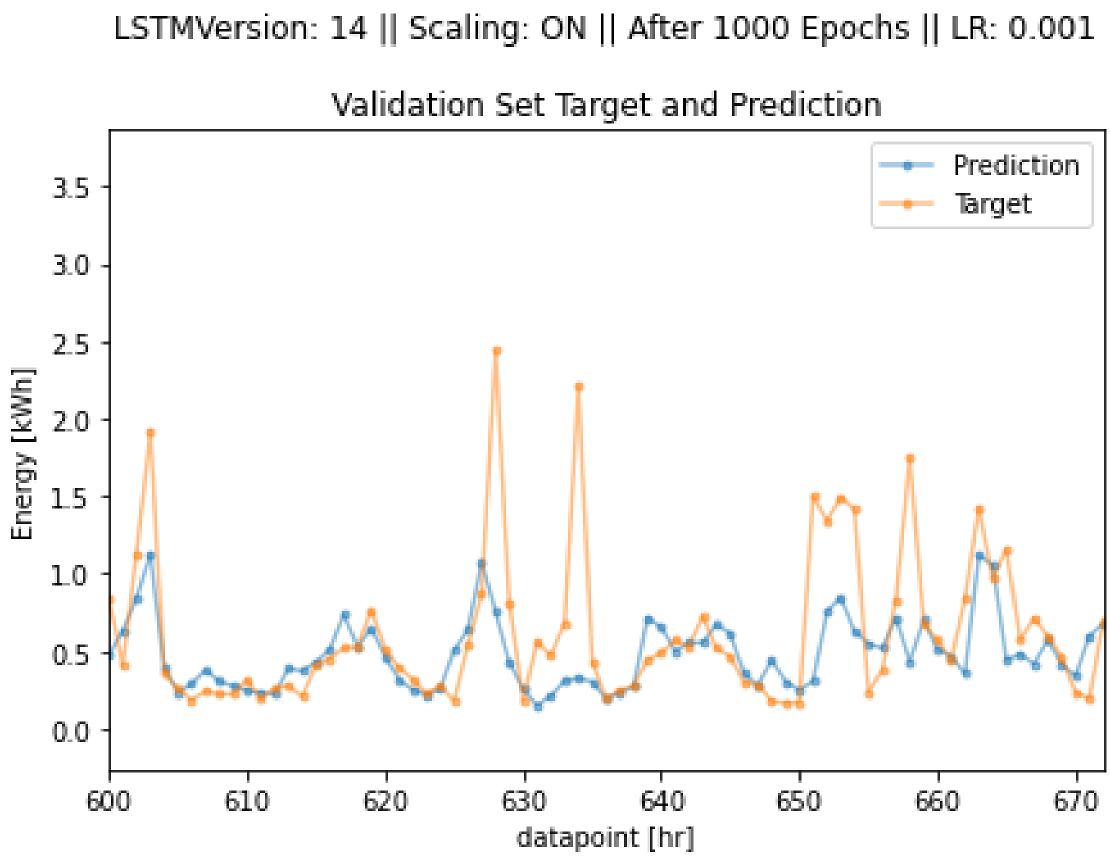
8 Visualizing the results

[17]:
```
model.eval()

%matplotlib inline
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(f'{str(criterion)[:-2]} of house 28' )
plt.xlabel("Epochs")
plt.ylabel(f"{str(criterion)[:-2]}") # Lossfunction
plt.legend()
plt.grid()
plt.show()
```



[18]:

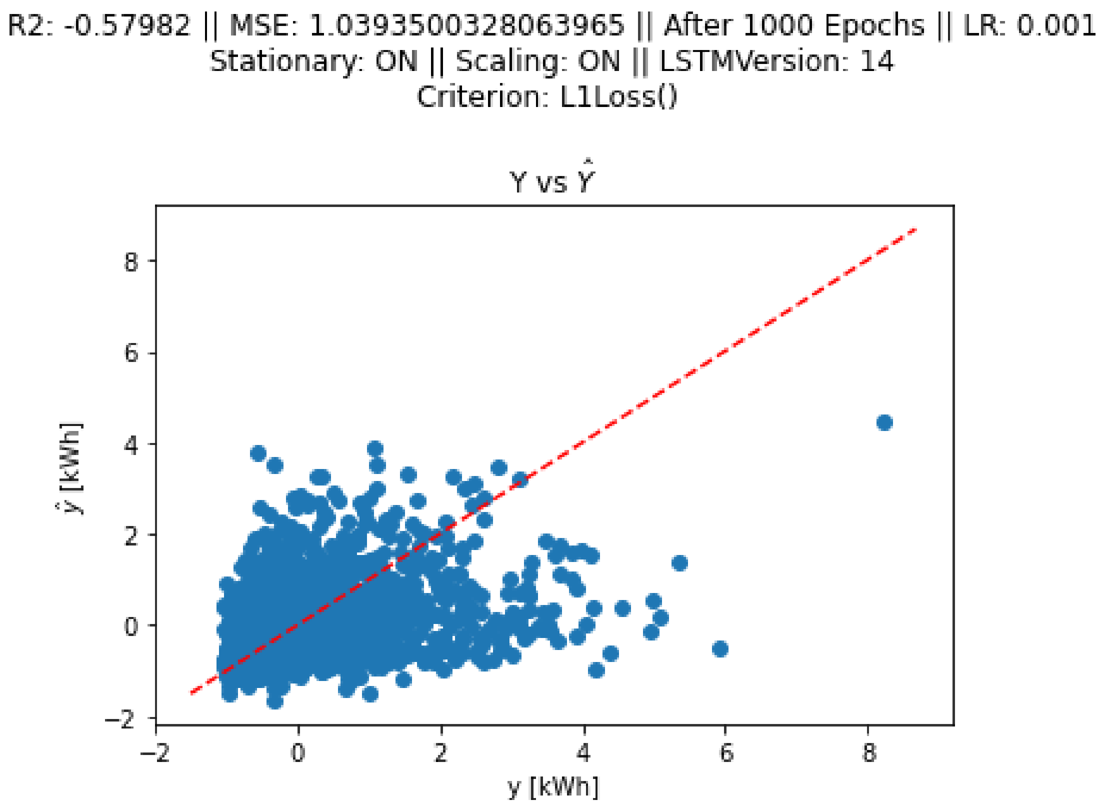
```
predictedvalue = det(outputV)
targetvalue =det(targetV)

if scaling:
    predictedvalue = scaler_y.inverse_transform(det(outputV))
    targetvalue = scaler_y.inverse_transform(det(targetV))



plt.plot(predictedvalue, '.-', alpha=0.5, label="Prediction")
plt.plot(targetvalue, '.-', alpha=0.5, label = "Target")
plt.tight_layout()
plt.title(f'LSTMVersion: {version} || Scaling: {"ON" if scaling else "OFF"} ||␣
↪After {train_for} Epochs || LR: {learningrate}\n\nValidation Set Target and␣
↪Prediction')
plt.xlim([600, 672])
#plt.ylim([0,2])
plt.legend()
plt.xlabel("datapoint [hr]")
plt.ylabel("Energy [kWh]")
plt.show()


print(f"Area under prediction: \t{predictedvalue.sum()}\nArea under Target:␣
↪\t{targetvalue.sum()}\nDifference: \t\t{predictedvalue.sum()-targetvalue.
↪sum()} ")
print(f"Difference percentage: \t{round((predictedvalue.sum()-targetvalue.
↪sum())/(targetvalue.sum())*100, 2)}%")
```




```
Area under prediction:  1107.65380859375
Area under Target:      1196.030029296875
Difference:             -88.376220703125
Difference percentage:  -7.39%
```

[19]:

```
y = det(valid_y_t)
yhat = det(model(valid_X_t))

r2 = round(r2_score(yhat, y),5)
lossV = round(mean_squared_error(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:␣
 ↪{learningrate}\nStationary: {"ON" if stationary else "OFF"} || Scaling:␣
 ↪{"ON" if scaling else "OFF"} || LSTMVersion: {version}\nCriterion:␣
 ↪{criterion} \n\nY vs $\hatY$')
plt.xlabel("y [kWh]")
plt.ylabel("$\haty$ [kWh]")
plt.show()
```



[ ]:

```
%%javascript
Jupyter.notebook.session.delete()
```

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