W16_LSTM_MultipleHouses_production

February 5, 2021

1 settings

[1]:
```
#settings:
train_for = 100
learningrate = 1e-3
window_size = 168
houses = [28,37,40,42,105,115,56,51,58,70,99,100]
reset_scheduler_after_n_epochs = 100
```

2 Initialization

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

from tqdm import tqdm

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

from torch.utils.data import DataLoader, TensorDataset
```

[4]:
```
#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
# 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)

#Scaler objects
scaler_X = StandardScaler()
scaler_y = StandardScaler()
```

Make all functions:

[5]:
```
class lstm(nn.Module):
    def __init__(self, feature_size=3, 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
        y = y + X[:,-1,-1:]             # make the output stationary
        return y.view(-1)               # like always

def init_lstm():
    model = lstm().to(device)
    #use multiple GPU's:
    #model = nn.DataParallel(model) #use multiple GPU'S
    return model
model = init_lstm()
```

[6]:
```
#GENERAL FUNCTIONS:
def det(tensor):
    """
    Zet de tensor om van een tensor naar numpy op de CPU.
    """
```

```
    return tensor.cpu().detach().numpy()

def calculate_metrics_for_model(output,target):
    """
    Calculates all the desired evaluation metrics for the model.
    """
    yhat = scaler_y.inverse_transform(det(output))
    y = scaler_y.inverse_transform(det(target))
    actual, pred = np.array(y), np.array(yhat)

    mae = mean_absolute_error(yhat, y)
    mse = mese(yhat, y)
    mape = np.mean(np.abs((actual - pred) / actual)) * 100
    r2 = r2_score(yhat, y)
    return [mae, mse, mape, r2]

def dim3(dft, window=7, gap=24):
    dft = pd.DataFrame(dft)
    #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]
    return X.astype(np.float32), y.astype(np.float32)


#LSTM specific functions:
def load_LSTM_data(house_nr):
    """
    Loads the Data for the lstm.
    and returns this.
    """
    house_nr = str(house_nr)
    if len(house_nr)==1:
        house_nr = "00"+str(house_nr)
    if len(house_nr)==2:
        house_nr = "0"+str(house_nr)
    df = pd.read_pickle('/home/18005152/notebooks/zero/Data:/testDataFrames/
 ↪TEST/DeepLearning_production_'+str(house_nr)+"_2")
    return df

def LSTM_split_df(df):
    """
    Splits the dataframe in train test validate parts.
    """
    trdf = df.loc['2019-01':'2019-08']
```

```python
    vadf = df.loc['2019-09':'2019-11']
    tedf = df.loc['2019-12']
    return trdf, vadf, tedf

def LSTM_data_scaler(df1,df2,df3):
    dftr = df1.copy()
    dfva = df2.copy()
    dfte = df3.copy()

    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:])

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

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

    df1 = dftr
    df2 = dfva
    df3 = dfte

    return df1,df2,df3

def LSTM_create_tensors(n1,n2,n3,n4,n5,n6):
    train_X_t = torch.from_numpy(np.array(n1))
    train_y_t = torch.from_numpy(np.array(n2))

    valid_X_t = torch.from_numpy(np.array(n3)).to(device)
    valid_y_t = torch.from_numpy(np.array(n4)).to(device)

    test_X_t = torch.from_numpy(np.array(n5)).to(device)
    test_y_t = torch.from_numpy(np.array(n6)).to(device)
    return train_X_t, train_y_t, valid_X_t, valid_y_t, test_X_t, test_y_t

def train_LSTM_Nepoch(data, N):
    """
    Train the LSTM for one epoch.
    """
    model = init_lstm()
    model.train()
```

```
    length_loader = len(data)
    scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer,␣
↪max_lr=learningrate, steps_per_epoch=length_loader, epochs=train_for)
    for i in range(N):
        for X, y in data:
            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)
            #save train loss scores correctly (not the last one)

            loss.backward()
            optimizer.step()
            scheduler.step()

        if i % reset_scheduler_after_n_epochs == 0:
            scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer,␣
↪max_lr=learningrate, steps_per_epoch=len(data),␣
↪epochs=reset_scheduler_after_n_epochs)
            pass
    return model

def validate_LSTM(data, target):
    """
    Validate the LSTM on unseen data and give back the evaluation metrics.
    """
    model.eval()
    optimizer.zero_grad()

    outputV = model(data)
    return calculate_metrics_for_model(outputV,target)
```

3 Main loop:

[7]:
```
#stats savelist:
LSTM_stats = pd.DataFrame()

#Training parameters:
optimizer = optim.Adam(model.parameters(), lr=learningrate)
criterion = nn.SmoothL1Loss()
```

```
#Learning loop:
for i in tqdm(range(len(houses))):
    house_number = houses[i]
    """
    Data loading...
    """
    #load the data:
    lstm_df = load_LSTM_data(house_number)
    #Splits de data in train valid test:
    train_LSTM, valid_LSTM, test_LSTM = LSTM_split_df(lstm_df)
    #scale de data:
    train_LSTM, valid_LSTM, test_LSTM = LSTM_data_scaler(train_LSTM,␣
 ↪valid_LSTM, test_LSTM)
    #doe een moving window van 3 eroverheen:
    train_x_LSTM, train_y_LSTM = dim3(train_LSTM, window_size)
    valid_x_LSTM, valid_y_LSTM = dim3(valid_LSTM, window_size)
    test_x_LSTM, test_y_LSTM = dim3(test_LSTM, window_size)
    #maak tensors van de data:
    train_X_t_LSTM, train_y_t_LSTM, valid_X_t_LSTM, valid_y_t_LSTM,␣
 ↪test_X_t_LSTM, test_y_t_LSTM = LSTM_create_tensors(train_x_LSTM,␣
 ↪train_y_LSTM, valid_x_LSTM, valid_y_LSTM, test_x_LSTM, test_y_LSTM)

    """
    Training
    """

    train_ds = TensorDataset(train_X_t_LSTM, train_y_t_LSTM)
    train_dl = DataLoader(train_ds, batch_size=64, num_workers=4)

    target = train_y_t_LSTM.view(-1)
    model = train_LSTM_Nepoch(train_dl,train_for)

    model.eval()
    output = model(train_X_t_LSTM.to(device))
    y = train_y_t_LSTM.to(device)
    LSTM_train_stats = calculate_metrics_for_model(output, y)


    """
    Evaluation
    """
    dataV = valid_X_t_LSTM; targetV = valid_y_t_LSTM.view(-1);
    LSTM_valid_stats = validate_LSTM(dataV, targetV)


    """
    Testing
```

```
    """
    dataTest = test_X_t_LSTM; targetTest = test_y_t_LSTM.view(-1);
    LSTM_test_stats = validate_LSTM(dataTest, targetTest)


    """
    Save metrics
    """
    index =␣
↪["MAE_train","MSE_train","MAPE_train","R2_train","MAE_valid","MSE_valid","MAPE_valid","R2_v
    New_Stats = pd.DataFrame(LSTM_train_stats+LSTM_valid_stats+LSTM_test_stats,␣
↪index=index, columns=[str(house_number)])
    LSTM_stats = pd.concat([LSTM_stats,New_Stats],axis=1)
```

```
100%|          | 12/12 [28:51<00:00, 144.30s/it]
```

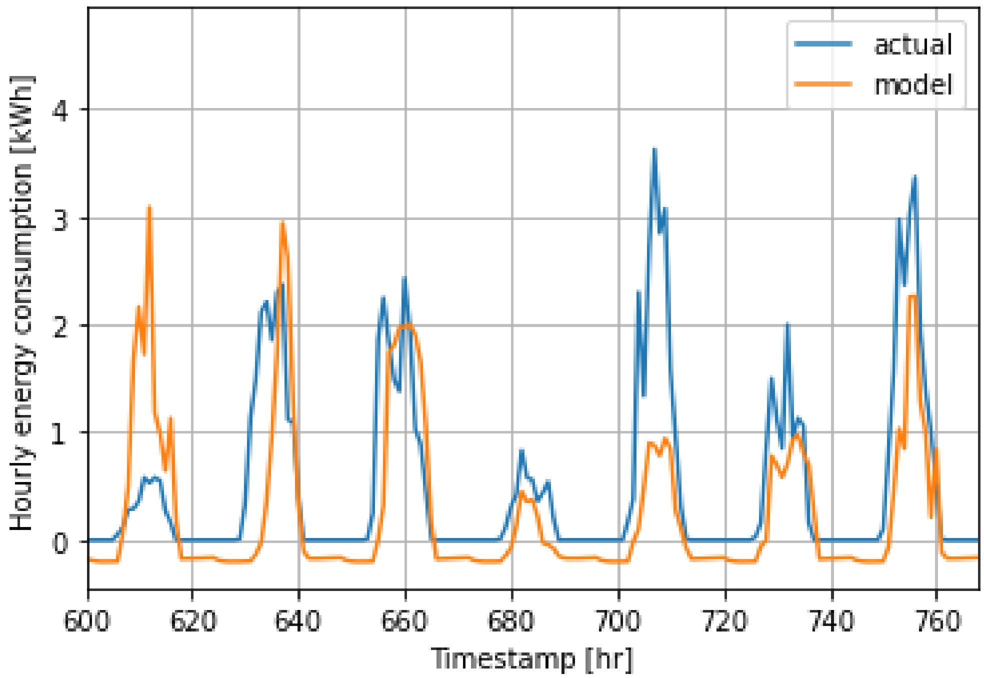
[8]:

```
dataV = valid_X_t_LSTM; targetV = valid_y_t_LSTM.view(-1);
model.eval()
optimizer.zero_grad()

outputV = det(model(dataV))
```

[9]:

```
yhat = scaler_y.inverse_transform(outputV)
y = scaler_y.inverse_transform(valid_LSTM.production)
plt.plot(y, label='actual')
plt.plot(yhat, label="model")
plt.xlabel("Timestamp [hr]")
plt.ylabel("Hourly energy consumption [kWh]")
plt.grid()
plt.legend()
plt.xlim([600, 768])
plt.savefig("LSTM_production_house100.png",dpi=1000)
np.save("LSTM_production.npy",yhat)
```



[10]:
```
LSTM_stats
```

[10]:

	28	37	40	42 \
MAE_train	5.732209e-01	5.944607e-01	5.541975e-01	5.571116e-01
MSE_train	9.416228e-01	1.089541e+00	9.119200e-01	9.079280e-01
MAPE_train	6.080952e+07	9.239711e+07	2.401183e+07	1.090915e+09
R2_train	5.579621e-01	5.834972e-01	5.728501e-01	5.944482e-01
MAE_valid	3.438545e-01	3.255121e-01	3.144279e-01	3.307727e-01
MSE_valid	3.812945e-01	3.979626e-01	3.431515e-01	3.677724e-01
MAPE_valid	7.494831e+07	9.237308e+07	2.762433e+07	1.239988e+09
R2_valid	4.247574e-01	4.883732e-01	4.651857e-01	4.801756e-01
MAE_test	1.608626e-01	1.363117e-01	1.341137e-01	1.466906e-01
MSE_test	7.633180e-02	7.549204e-02	5.910935e-02	6.960032e-02
MAPE_test	9.181282e+07	9.211347e+07	3.032906e+07	1.262943e+09
R2_test	1.302575e-01	2.874385e-01	2.390219e-01	2.874656e-01

	105	115	56	51 \
MAE_train	5.564283e-01	5.841301e-01	5.721691e-01	5.691409e-01
MSE_train	8.263488e-01	9.326700e-01	8.943994e-01	9.575666e-01
MAPE_train	1.262608e+08	1.486480e+08	6.472692e+07	4.417703e+07
R2_train	6.002533e-01	5.734216e-01	5.805541e-01	5.837264e-01
MAE_valid	3.401336e-01	3.575751e-01	3.338887e-01	3.345654e-01
MSE_valid	3.213599e-01	3.829600e-01	3.241198e-01	3.887597e-01
MAPE_valid	1.492633e+08	1.687134e+08	7.972223e+07	5.254020e+07
R2_valid	4.628276e-01	4.417533e-01	4.613628e-01	4.641143e-01

```
MAE_test    1.748021e-01  1.767713e-01  1.751964e-01  1.454448e-01
MSE_test    6.137542e-02  7.526226e-02  7.163199e-02  7.435241e-02
MAPE_test   1.671597e+08  1.903488e+08  9.856806e+07  5.410114e+07
R2_test     1.489692e-01  1.716417e-01  1.286159e-01  2.394950e-01

                      58            70            99           100
MAE_train   5.587426e-01  5.621280e-01  6.106745e-01  6.387776e-01
MSE_train   8.539765e-01  9.409116e-01  9.317442e-01  9.733203e-01
MAPE_train  3.415980e+08  7.708542e+07  1.557542e+08  6.776722e+09
R2_train    5.922873e-01  5.735736e-01  5.694865e-01  5.653950e-01
MAE_valid   3.301230e-01  3.302307e-01  4.001200e-01  4.199025e-01
MSE_valid   3.147831e-01  3.870770e-01  3.998013e-01  4.055640e-01
MAPE_valid  4.416022e+08  8.298409e+07  1.939993e+08  8.659301e+09
R2_valid    4.606352e-01  4.493234e-01  4.180665e-01  4.214881e-01
MAE_test    1.782957e-01  1.398596e-01  2.308274e-01  2.445045e-01
MSE_test    7.205137e-02  7.534026e-02  9.044568e-02  9.482445e-02
MAPE_test   5.365684e+08  8.235493e+07  2.275054e+08  1.043638e+10
R2_test     1.471753e-01  1.981772e-01 -9.368681e-03 -1.697221e-01
```

[11]:
```
LSTM_stats.to_pickle("LSTM_statistics")
```

[12]:
```
pd.read_pickle("LSTM_statistics").to_excel("LSTM_production.xlsx")
```

4 summarize stats with mean

[13]:
```
(LSTM_stats).mean(axis=1)
```

[13]:
```
MAE_train     5.775985e-01
MSE_train     9.301624e-01
MAPE_train    7.502588e+08
R2_train      5.789546e-01
MAE_valid     3.467588e-01
MSE_valid     3.678838e-01
MAPE_valid    9.385883e+08
R2_valid      4.531719e-01
MAE_test      1.703067e-01
MSE_test      7.465145e-02
MAPE_test     1.105849e+09
R2_test       1.499306e-01
dtype: float64
```