W16_LSTM_MultipleHouses-Copy3

January 6, 2021

1 settings

[1]:
```
#settings:
show_every = 10
train_for = 100
learningrate = 1e-3
stationary = True
window_size = 168
houses = [28,37,40,42,105,115,56,51,58,70,99,100]
reset_scheduler_after_n_epochs = 10
```

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)
    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:, 2]
    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_consumption_'+str(house_nr))
    return df

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

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))):
```

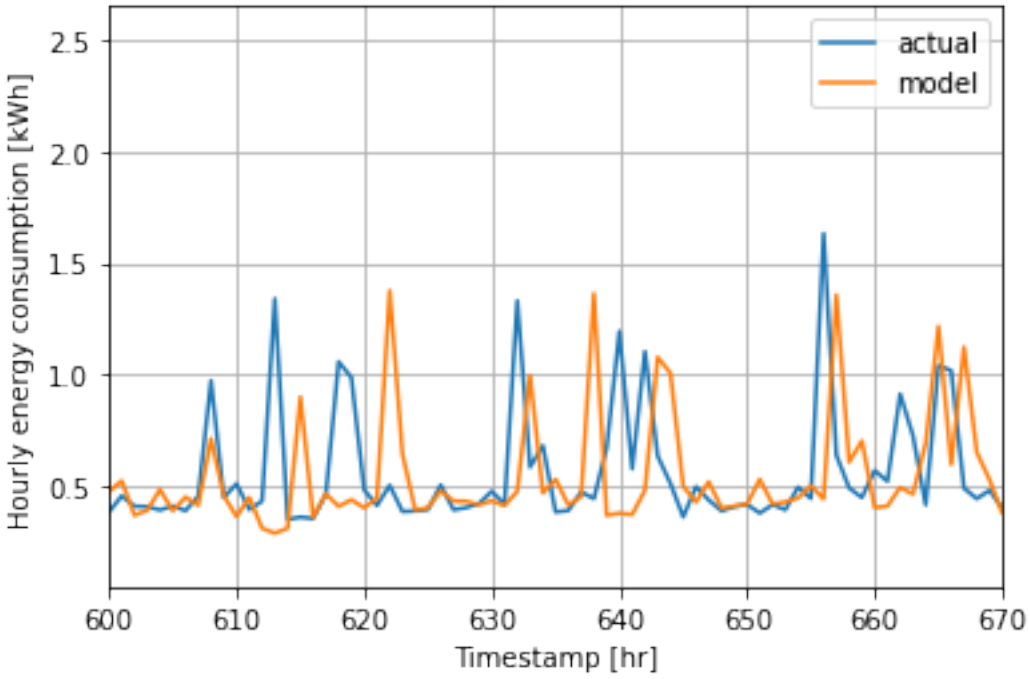
```
    house_number = houses[i]
    """
    Data loading...
    """
    #load the data:
    lstm_df = load_LSTM_data(house_number)
    #Splits de data in train valid test:
    train_LSTM, valid_LSTM, test_LSTM = LSTM_split_df(lstm_df)
    #scale de data:
    train_LSTM, valid_LSTM, test_LSTM = LSTM_data_scaler(train_LSTM,␣
↪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=3)

    target = train_y_t_LSTM.view(-1)
    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 [26:56<00:00, 134.74s/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_y_LSTM)
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_consumption_house100.png")
```



[10]:
```
LSTM_stats
```

[10]:

	28	37	40	42	105	\
MAE_train	0.304081	0.361599	0.543803	0.660888	0.421393	
MSE_train	0.223198	0.349084	0.636283	0.935168	0.415169	
MAPE_train	737578.173828	92.319578	110.153961	137.077689	86.078709	
R2_train	-0.548248	-0.220787	-0.523547	-0.677594	-0.269215	
MAE_valid	0.316353	0.366274	0.567483	0.654034	0.519571	
MSE_valid	0.231723	0.328013	0.733946	0.959151	0.592665	
MAPE_valid	59.629118	81.759286	103.843820	141.408527	106.784856	
R2_valid	-0.630764	-0.534540	-0.484853	-0.595461	-0.370193	
MAE_test	0.331176	0.460525	0.700231	0.885254	0.638122	
MSE_test	0.242042	0.454958	1.101537	1.515933	0.747861	
MAPE_test	44.525623	78.344220	96.169794	160.420716	115.053535	
R2_test	-0.691572	-0.121274	-0.547713	-0.707997	-0.315575	

	115	56	51	58	70	\
MAE_train	0.548735	0.460348	0.479198	0.329860	0.444680	
MSE_train	0.631129	0.452308	0.535653	0.284219	0.501735	
MAPE_train	77.937841	100.468576	62.509751	72.682667	175.369346	
R2_train	-0.452791	-0.368807	-0.252808	-0.492207	-0.433141	
MAE_valid	0.561724	0.491674	0.472459	0.322100	0.406039	
MSE_valid	0.670640	0.526493	0.509428	0.260471	0.462153	
MAPE_valid	54.706538	107.594669	70.810193	62.589997	158.870780	
R2_valid	-0.556034	-0.733385	-0.413572	-0.583872	-0.768303	

```
MAE_test     0.623233    0.760898   0.577062   0.360232    0.542130
MSE_test     0.731378    1.227516   0.645956   0.301267    0.546912
MAPE_test   47.555918   98.493338  56.218642  46.101463  232.787442
R2_test     -0.719935   -0.902796  -0.483920  -0.687853   -0.638910

                  99         100
MAE_train    0.410518   0.273658
MSE_train    0.376182   0.202634
MAPE_train  85.904634  62.965268
R2_train    -0.586206  -0.301477
MAE_valid    0.377046   0.235816
MSE_valid    0.334642   0.160481
MAPE_valid  76.726520  46.266380
R2_valid    -0.430590  -0.528951
MAE_test     0.475322   0.185048
MSE_test     0.495298   0.132141
MAPE_test   61.555725  24.180204
R2_test     -0.773346  -0.886871
```

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

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

4 summarize stats with mean

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

[13]:
```
MAE_train         0.436564
MSE_train         0.461897
MAPE_train    61553.470154
R2_train         -0.427236
MAE_valid         0.440881
MSE_valid         0.480817
MAPE_valid       89.249224
R2_valid         -0.552543
MAE_test          0.544936
MSE_test          0.678567
MAPE_test        88.450552
R2_test          -0.623147
dtype: float64
```