Dp1_analysis

January 30, 2021

[66]:
```python
import numpy as np
import math
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from collections import Counter, OrderedDict
import copy
import pandas as pd
import time
import datetime

import keras.backend as K
from keras.models import Sequential, load_model
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
np.random.seed(7)
```

[67]:
```python
Dataset = r'../15minute_data_newyork/15minute_data_newyork.csv'
fulldata = pd.read_csv(Dataset)
```

1 Data Preprocessing

[68]:
```python
data=fulldata[['dataid','local_15min','grid']]
sorteddata=data.sort_values(by = ['dataid', 'local_15min'])
ids=sorteddata['dataid'].unique().tolist()
```

[69]:
```python
housing_data = []
def convertDate(d):
    d = pd.to_datetime(d[:-3])
    return d

for i in range(len(ids)):
    housing_data.append(sorteddata.loc[sorteddata.dataid==ids[i]])
    housing_data[i] = housing_data[i].reset_index().drop(columns=['index'])
    housing_data[i]['local_15min'] = housing_data[i]['local_15min'].
 ↪apply(convertDate)
    #Convert datetimes to ints for faster ploting
    housing_data[i]['15min_ints'] =  housing_data[i]['local_15min'].map(mdates.
 ↪date2num)
```

[70]:
```
def create_dataset(dataset, look_back=1, look_ahead=None):
        "function for creating dataset for model, X being the known data, and Y␣
 ↪being target data"
        if look_ahead is None:
                look_ahead = look_back
        dataX, dataY = [], []
        for i in range(len(dataset)-2*look_back):
                dataX.append(dataset[i:(i+look_back), 0])
                if look_ahead == 0:
                        dataY.append(dataset[i + look_back, 0])
                else:
                        dataY.append(dataset[(i+look_back):
 ↪(i+look_back+look_ahead), 0])

        return np.array(dataX), np.array(dataY)
```

[71]:
```
#set updataframe = housing_data[0]['grid']
dataframe = housing_data[0]['grid']
dataset = np.matrix(dataframe.values).transpose()
dataset = dataset.astype('float32')
```

[72]:
```
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

[73]:
```
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:,:]
```

[74]:
```
# reshape into X=t and Y=t+look_back
look_back = 96 #(60mins/15min)*24 hours
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

[75]:
```
# reshape input to be [samples, time steps, features]
trainX = np.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = np.reshape(testX, (testX.shape[0], testX.shape[1], 1))
```

[76]:
```
model = load_model('../models/D_0house_model_1_25_adam.h5')
model.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_52 (LSTM)               (None, 64)                16896

_________________________________________________________________
dropout_50 (Dropout)         (None, 64)                0

_________________________________________________________________
dense_11 (Dense)             (None, 96)                6240
=================================================================
Total params: 23,136
Trainable params: 23,136
Non-trainable params: 0

_________________________________________________________________
```

2 Data Analysis

2.1 Average Day

[77]:
```
time = housing_data[0]['local_15min']
```

[78]:
```
#diff in time
print(time[0])
print(time[look_back])
```

```
2019-05-01 00:00:00
2019-05-02 00:00:00
```

[79]:
```
fulldays=len(dataframe)//look_back
print(look_back*fulldays)
```

```
17568
```

[80]:
```
grid_data=dataframe.to_numpy()
```

[81]:
```
grid_data.shape
```

[81]:
```
(17663,)
```

[82]:
```
#convert to matrix
grid_day_matrix=grid_data[:(look_back*fulldays)].reshape(-1, 96)
#avg house0 grid data
avg_house0_grid=np.mean(grid_day_matrix, axis=0)
grid_day_matrix.shape
```

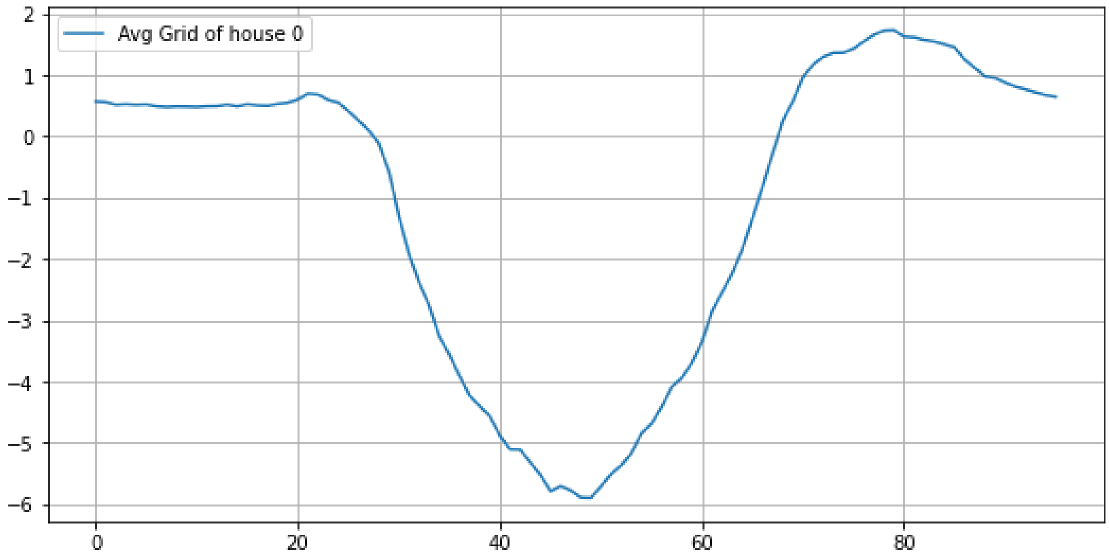
[82]:
```
(183, 96)
```

[35]:
```

```

[112]:
```
plt.figure(figsize=(10,5))
plt.plot(avg_house0_grid, label= 'Avg Grid of house 0' )
plt.grid(True)
plt.legend()
```

```
[112]: <matplotlib.legend.Legend at 0x7f82201d5dd0>
```



```
[ ]:
```

```
[84]: grid_day_matrix
```

```
[84]: array([[0.997, 0.75 , 0.608, ..., 0.219, 0.605, 0.304],
              [0.3  , 0.275, 0.296, ..., 0.409, 0.294, 0.26 ],
              [0.352, 0.355, 0.265, ..., 0.277, 0.361, 0.376],
              ...,
              [0.654, 0.512, 0.856, ..., 0.244, 0.155, 0.162],
              [0.315, 0.514, 0.429, ..., 0.238, 0.2  , 0.178],
              [0.217, 0.452, 0.43 , ..., 0.601, 0.551, 0.75 ]])
```

```
[85]: #predict on new housing data
house0X, house0Y = create_dataset(dataset, look_back)
house0X = house0X.reshape(house0X.shape[0], house0X.shape[1],1)
house0Predict = model.predict(house0X)

house0Predict = scaler.inverse_transform(house0Predict)
house0Y = scaler.inverse_transform(house0Y)
```

```
[86]: ttrain0Score = math.sqrt(mean_squared_error(house0Y, house0Predict))
print('Train Score: %.2f RMSE' % (ttrain0Score))
```

```
Train Score: 1.77 RMSE
```

```
[87]: house0Predict[::look_back]
```

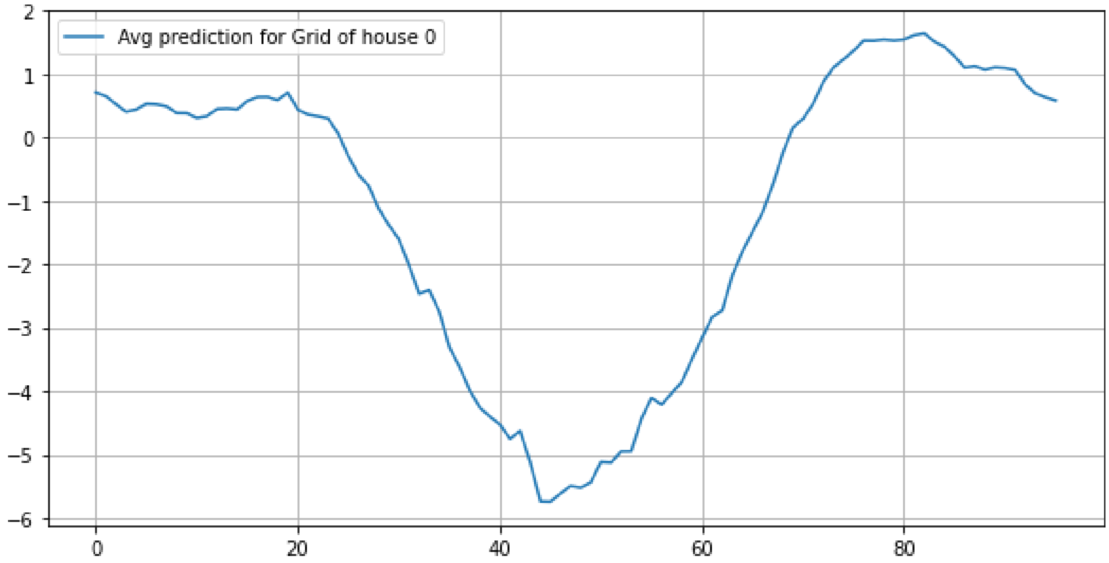
[87]:
```
array([[0.42874315, 0.3905604 , 0.28844008, ..., 1.2967987 , 1.214559  ,
         1.1457767 ],
       [0.24368382, 0.2659657 , 0.22332677, ..., 0.2980481 , 0.31262827,
         0.2763251 ],
       [0.32992408, 0.35608467, 0.29763645, ..., 0.34640476, 0.35295388,
         0.27326968],
       ...,
       [0.49467278, 0.6075746 , 0.61636394, ..., 0.19228101, 0.29056203,
         0.28807133],
       [0.45104128, 0.3699378 , 0.11540293, ..., 1.1096615 , 0.9274906 ,
         0.704601  ],
       [0.24014139, 0.2760538 , 0.20586629, ..., 0.61330044, 0.59082025,
         0.52454853]], dtype=float32)
```

[88]:
```
pred_house0_matrix=house0Predict[::look_back]
```

[89]:
```
pred_house0_grid=np.mean(pred_house0_matrix, axis=0)
```

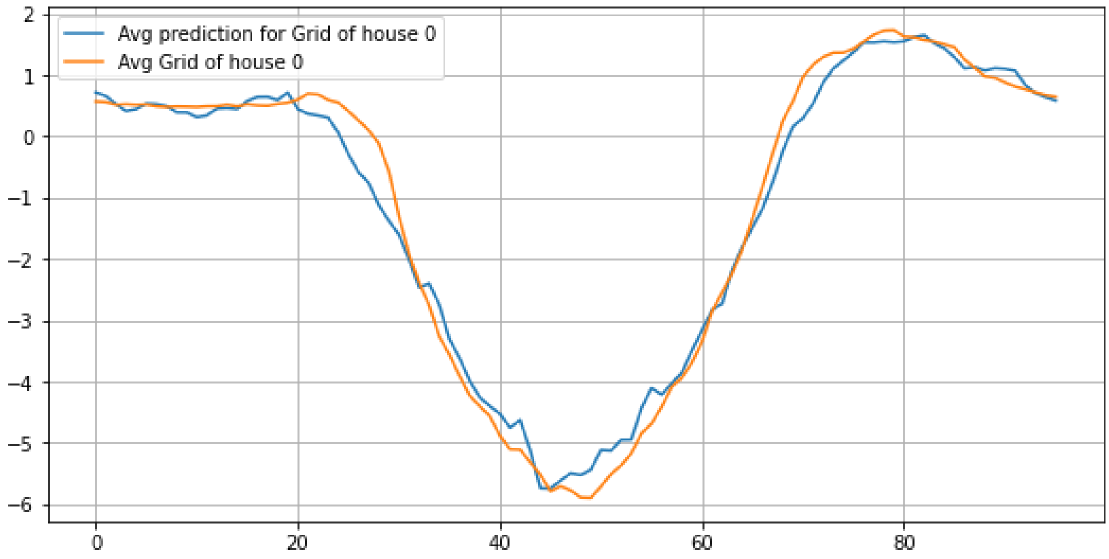
[113]:
```
plt.figure(figsize=(10,5))
plt.plot(pred_house0_grid, label= 'Avg prediction for Grid of house 0' )
plt.grid(True)
plt.legend()
```

[113]:
```
<matplotlib.legend.Legend at 0x7f8222d21090>
```



[110]:
```
plt.figure(figsize=(10,5))
plt.plot(pred_house0_grid , label= 'Avg prediction for Grid of house 0')
plt.plot(avg_house0_grid, label= 'Avg Grid of house 0' )
plt.grid(True)
plt.legend()
```

[110]:
```
<matplotlib.legend.Legend at 0x7f8222e26bd0>
```



[92]:
```
ttrain0Score = math.sqrt(mean_squared_error(pred_house0_grid, avg_house0_grid))
print('Train Score: %.2f RMSE' % (ttrain0Score))
```

```
Train Score: 0.32 RMSE
```

[ ]:
```

```

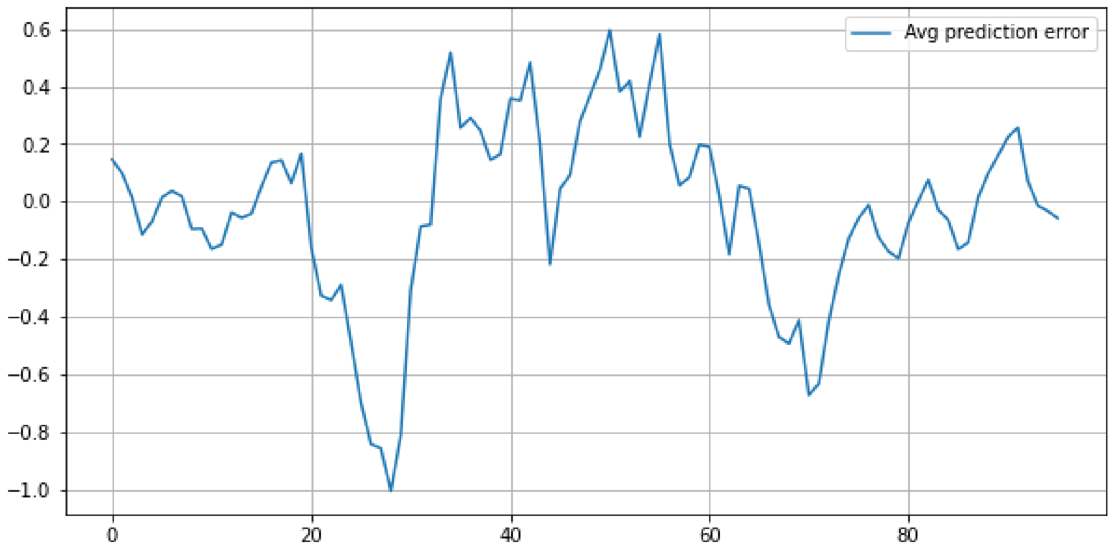
2.2 Error of model

https://fairyonice.github.io/Measure-the-uncertainty-in-deep-learning-models-using-dropout.html

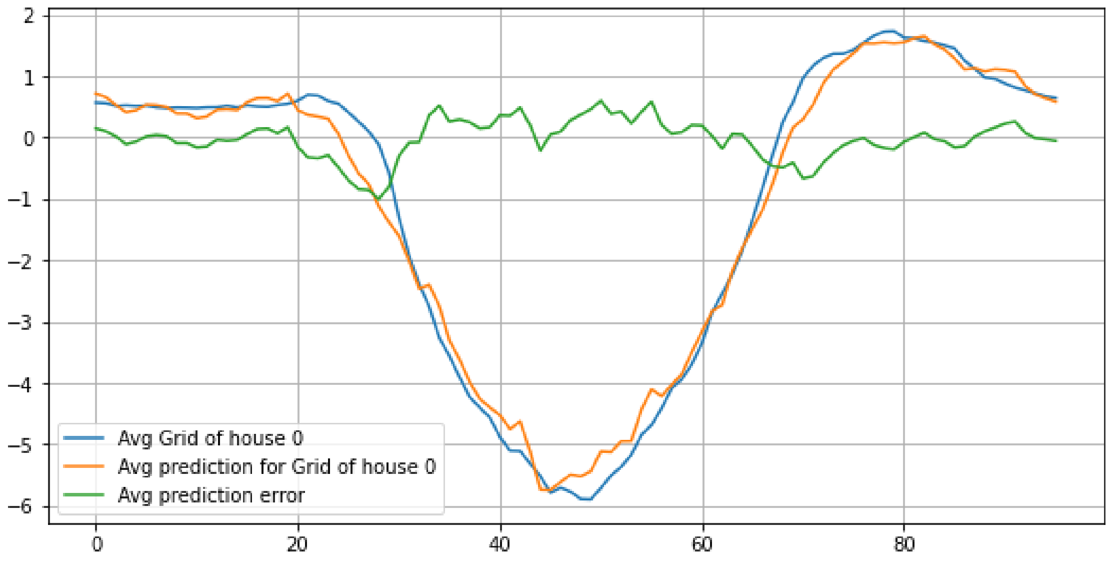
[93]:
```
error = pred_house0_grid-avg_house0_grid
```

[108]:
```
plt.figure(figsizeplt.figure(figsize=(10,5))=(10,5))
plt.plot(error, label= 'Avg prediction error')
plt.grid(True)
plt.legend()
```

[108]:
```
<matplotlib.legend.Legend at 0x7f822007cf50>
```



[109]:
```
plt.figure(figsize=(10,5))
plt.plot(avg_house0_grid, label= 'Avg Grid of house 0' )
plt.plot(pred_house0_grid, label= 'Avg prediction for Grid of house 0' )
plt.plot(error, label= 'Avg prediction error')
plt.legend()
plt.grid(True)
```



2.3 Uncertainty of model

https://fairyonice.github.io/Measure-the-uncertainty-in-deep-learning-models-using-dropout.html

[ ]:

```
#Set up X
trainX
```