ANN_Model_UMass_power

November 22, 2020

1 Setup

[58]:
```
#importing required libraries
%load_ext tensorboard
import tensorflow as tf
import keras as kr
import numpy as np
import pandas as pd
import datetime
import statistics as st
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler

from sklearn.model_selection import train_test_split

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras import regularizers
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
from tensorflow.keras import regularizers
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

[59]:
```
#Importing the dataset

data_df = pd.read_csv("https://raw.githubusercontent.com/A-Wadhwani/
 ↪ME597-Project/main/Datasets/Umass_total_data2.csv")
```

```
data_df.head(10)
```

[59]:
```
   Unnamed: 0  Year  ...  UMASS Amherst - Recreation Center null        Date
0           0  2017  ...                                1.095000  2017-02-03
1           1  2017  ...                                1.453818  2017-02-04
2           2  2017  ...                                0.818667  2017-02-05
3           3  2017  ...                                1.250591  2017-02-06
4           4  2017  ...                                0.712076  2017-02-08
5           5  2017  ...                                0.251333  2017-02-10
6           6  2017  ...                                0.006076  2017-02-11
7           7  2017  ...                                0.008955  2017-02-13
8           8  2017  ...                                0.117985  2017-02-14
9           9  2017  ...                                0.230424  2017-02-15

[10 rows x 22 columns]
```

[60]:
```
data_df.dtypes
```

[60]:
```
Unnamed: 0                                  int64
Year                                        int64
Month                                       int64
Day                                         int64
DHI                                       float64
DNI Max                                     int64
DNI Min                                     int64
DNI                                       float64
GHI                                       float64
Clearsky DHI                              float64
Clearsky DNI                              float64
Clearsky GHI                              float64
Wind Speed                                float64
Precipitable Water                        float64
Wind Direction                            float64
Relative Humidity                         float64
Temperature                               float64
Pressure                                  float64
UMASS Amherst - Computer Science null     float64
UMASS Amherst - Fine Arts Center null     float64
UMASS Amherst - Recreation Center null    float64
Date                                       object
dtype: object
```

2 Data Sorting and scaling

[61]:
```
#Splitting dataset into X and Y variables
```

```
X_data = data_df[['DHI','DNI Max', 'DNI Min', 'DHI', 'GHI', 'Clearsky DNI',␣
 ↪'Clearsky DHI', 'Clearsky GHI', 'Wind Speed', 'Precipitable Water', 'Wind␣
 ↪Direction', 'Relative Humidity', 'Temperature', 'Pressure']]
Y_data = data_df[['UMASS Amherst - Fine Arts Center null']]
```

[62]:
```
#Scaling the data
scaler1 = RobustScaler()
scaler2 = RobustScaler()

X_data_scaled = scaler1.fit_transform(X_data)
Y_data_scaled = scaler2.fit_transform(Y_data)
```

[63]:
```
#Splitting the X and Y variables into train and test datasets
X_train, X_test, Y_train, Y_test = train_test_split(X_data_scaled,␣
 ↪Y_data_scaled, test_size=0.25)
```

3 Developing ANN model

[64]:
```
#creating optimizer method
optimizer1 = tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.8, beta_2=0.
 ↪999, epsilon=1e-8)
```

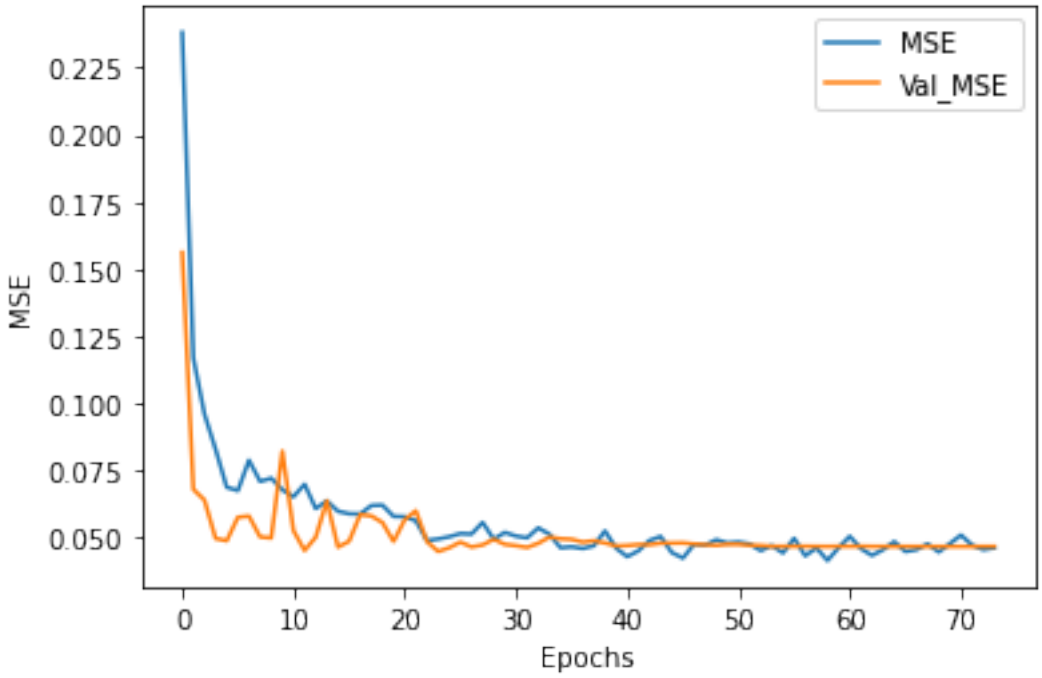
[ ]:
```
# Creating and training the ANN model
model = Sequential()
model.add(Dense(512, input_dim = X_data.shape[1], activation='relu',␣
 ↪kernel_initializer = 'random_normal'))
model.add(Dense(256, activation='relu', kernel_initializer = 'random_normal', ␣
 ↪activity_regularizer=regularizers.l1(3.5e-4)))
model.add(Dropout(0.3))
model.add(Dense(128, activation='relu', kernel_initializer =␣
 ↪'random_normal'))#,  activity_regularizer=regularizers.l1(3.5e-4)))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu', kernel_initializer = 'random_normal'))#,␣
 ↪ activity_regularizer=regularizers.l1(3.5e-4)))
model.add(Dropout(0.3))
model.add(Dense(32, activation='relu', kernel_initializer = 'random_normal'))#,␣
 ↪ activity_regularizer=regularizers.l1(3.5e-4)))
model.add(Dropout(0.3))
model.add(Dense(16, activation='relu', kernel_initializer = 'random_normal'))#,␣
 ↪ activity_regularizer=regularizers.l1(3.5e-4)))
model.add(Dropout(0.3))
model.add(Dense(1, activation = 'linear'))
model.compile(loss='mae', optimizer = optimizer1, metrics =␣
 ↪['mean_squared_error','mae'])
```

```
monitor = EarlyStopping(monitor ='val_loss', min_delta= 1e-3, patience = 50,␣
 ↪verbose =1, restore_best_weights=True  )
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=10,␣
 ↪verbose = 1)

history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs␣
 ↪= 200, verbose =1, callbacks = [monitor, reduce_lr], batch_size = 15)
```

[71]:
```
plt.plot(history.history['mean_squared_error'],label='MSE')
plt.plot(history.history['val_mean_squared_error'],label='Val_MSE')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
```

[71]: <matplotlib.legend.Legend at 0x7f8ef24cb438>



4 Analyzing effectiveness of model

[68]:
```
y_f_result = model.predict(X_test)
y_result = scaler2.inverse_transform(y_f_result)
y_actual = scaler2.inverse_transform(Y_test)

compare = pd.DataFrame()
compare['Expected'] = y_actual.reshape(1,-1)[0]
```

```
compare['Result'] = y_result.reshape(1,-1)[0]
compare['Difference'] = compare['Expected'] - compare['Result']
compare['Percentage Error'] = 100 * compare['Difference']/compare['Expected']

#Print out percentile descriptions of model accuracy
compare['Percentage Error'].describe(percentiles=[0.001, 0.01, 0.05, 0.25, 0.5,␣
 ↪0.75, 0.95, 0.99, 0.999])
```

[68]:
```
count       237.000000
mean       -619.459929
std        7126.134721
min     -102799.326682
0.1%     -87698.845617
1%         -389.511748
5%         -190.542751
25%         -19.831126
50%          -1.003695
75%           5.929017
95%          24.726111
99%          45.281149
99.9%        54.807533
max          57.101084
Name: Percentage Error, dtype: float64
```

[69]:
```
plt.scatter(compare['Expected'], compare['Result'])
plt.show()
```