Weather Forecasting

April 8, 2024

1 Assignment : 2

1.1 Author : Dev , Sakshi

Importing all the modules

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
```

Reading the csv file

[2]:
```
df = pd.read_csv("/Users/devmarwah/Documents/MSBA assignments/Advanced Machine␣
↪Learning/jena_climate_2009_2016.csv")
```

Data Exploration

Displaying the head of our data to have a look at it :

[3]:
```
df.head()
```

[3]:
```
              Date Time  p (mbar)  T (degC)  Tpot (K)  Tdew (degC)  rh (%)  \
0  01.01.2009 00:10:00    996.52     -8.02    265.40        -8.90    93.3
1  01.01.2009 00:20:00    996.57     -8.41    265.01        -9.28    93.4
2  01.01.2009 00:30:00    996.53     -8.51    264.91        -9.31    93.9
3  01.01.2009 00:40:00    996.51     -8.31    265.12        -9.07    94.2
4  01.01.2009 00:50:00    996.51     -8.27    265.15        -9.04    94.1

   VPmax (mbar)  VPact (mbar)  VPdef (mbar)  sh (g/kg)  H2OC (mmol/mol)  \
0          3.33          3.11          0.22       1.94             3.12
1          3.23          3.02          0.21       1.89             3.03
2          3.21          3.01          0.20       1.88             3.02
3          3.26          3.07          0.19       1.92             3.08
4          3.27          3.08          0.19       1.92             3.09

   rho (g/m**3)  wv (m/s)  max. wv (m/s)  wd (deg)
0       1307.75      1.03           1.75     152.3
1       1309.80      0.72           1.50     136.1
2       1310.24      0.19           0.63     171.6
```

```
3        1309.19       0.34              0.50      198.0
4        1309.00       0.32              0.63      214.3
```

Checking shape of our data

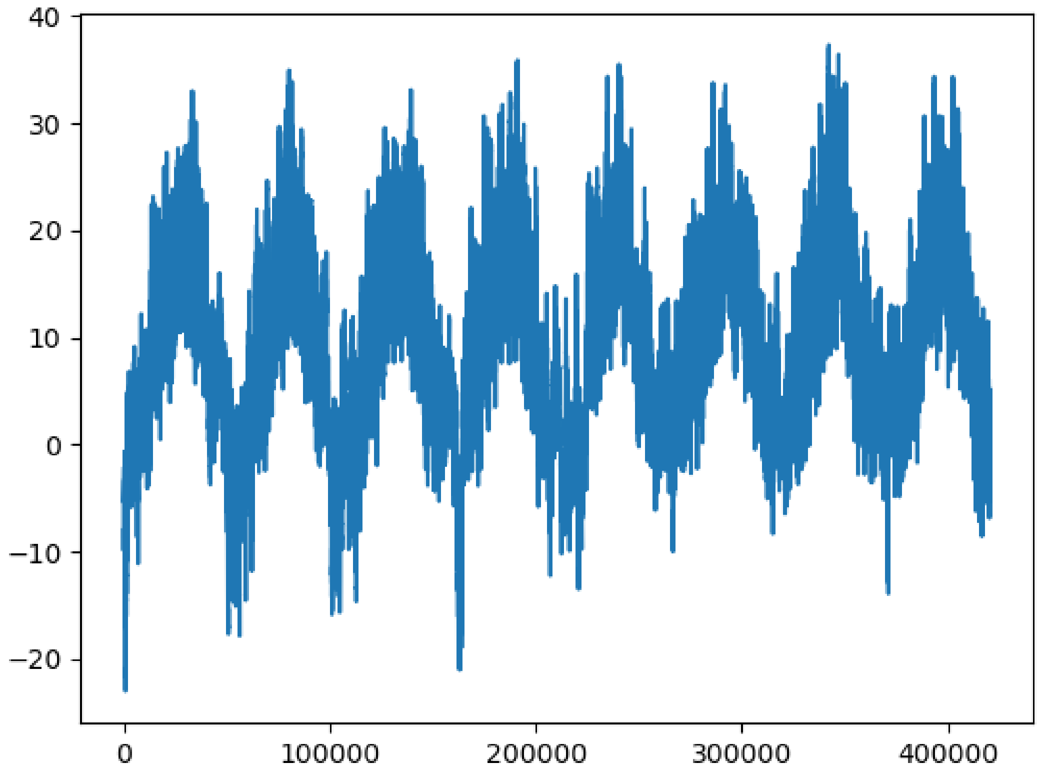
[4]:
```python
df.shape
```

[4]:
```
(420451, 15)
```

Plotting temperature

[5]:
```python
plt.plot(range(420451),df.iloc[:,2])
```

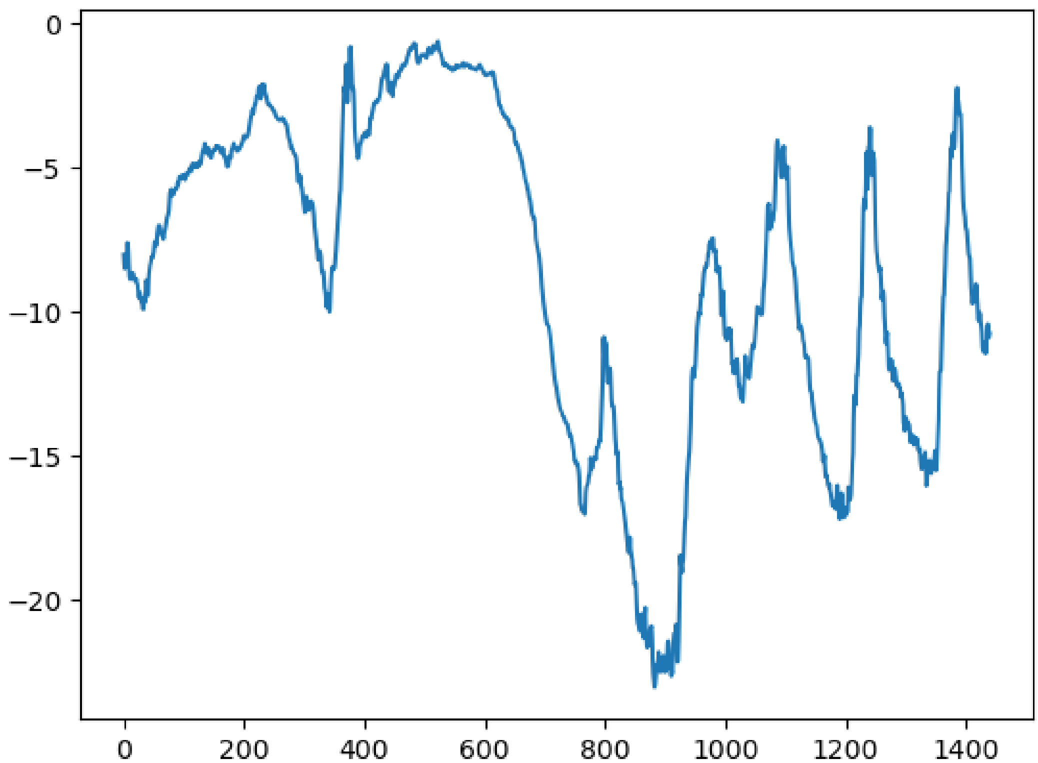
[5]:
```
[<matplotlib.lines.Line2D at 0x144aba190>]
```



Plotting the temperature data for only first 10 days :

[6]:
```python
plt.plot(range(1440),df.iloc[0:1440,2])
```

[6]:
```
[<matplotlib.lines.Line2D at 0x15042ce50>]
```



Printing the size of training, validation and test samples that we will be using :

[7]:
```
n_train = int(0.5*len(df))
n_val = int(0.25*len(df))
n_test = int(n_train-n_val)
print("Train samples : ",n_train)
print("Validation samples : ",n_val)
print("Test samples : ",n_test)
```

```
Train samples :  210225
Validation samples :  105112
Test samples :  105113
```

Data Preparation

[8]:
```
# Storing standard devidation and mean for furthur use  and normalizing data:
dfs = df.drop('Date Time',axis=1).to_numpy()
mean = dfs[:n_train].mean(axis=0)
dfs -=mean
std = dfs[:n_train].std(axis=0)
dfs /= std
print(std[1])
print(mean[1])
```

```
8.770983608349352
8.825903294089667
```

[9]:
```
temperature = dfs[:,1]b
```

Model Construction

Diving data into training, validation and test dataset

[10]:
```
sampling_rate = 6
sequence_length = 120
delay = sampling_rate * (sequence_length + 24 - 1)
batch_size = 256
```

Converting data frame to array, discarding date-time and converting values to float

[16]:
```
dfs=dfs.astype('float32')
temperature = temperature.astype('float32')
```

[23]:
```
from tensorflow import keras
Train = keras.utils.timeseries_dataset_from_array(
    dfs[:-delay],
    targets = temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length = sequence_length,
    batch_size=batch_size,
    start_index=0,
    shuffle=True,
    end_index=n_train
    )
Validation = keras.utils.timeseries_dataset_from_array(
    dfs[:-delay],
    targets = temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length = sequence_length,
    batch_size=batch_size,
    start_index=n_train,
    shuffle=True,
    end_index=n_train+n_val
    )
Test = keras.utils.timeseries_dataset_from_array(
    dfs[:-delay],
    targets = temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length = sequence_length,
    batch_size=batch_size,
    start_index=n_train+n_val,
    shuffle=True
    )
```

Inspecting the output of our Train dataset :

[24]:
```
for samples,targets in Train :
    print("Sample shape : ",samples.shape)
    print("Target shape :",targets.shape)
    break
```

```
Sample shape :  (256, 120, 14)
Target shape : (256,)
```

Making a simple dense network model to check performance :

[47]:
```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(sequence_length, dfs.shape[-1]))
x = layers.Reshape((sequence_length * dfs.shape[-1],))(inputs)
x = layers.Flatten()(x)
x = layers.Dense(16, activation="relu")(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("/Users/devmarwah/Documents/MSBA␣
 ↪assignments/Advanced Machine Learning/Weather Forecasting/jena_dense.keras",
                                    save_best_only=True)
]

model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(Train,
                    epochs=10,
                    validation_data=Validation,
                    callbacks=callbacks)
model = keras.models.load_model("/Users/devmarwah/Documents/MSBA assignments/
 ↪Advanced Machine Learning/Weather Forecasting/jena_dense.keras")
print(f"Test MAE: {model.evaluate(Test)[1]:.2f}")
```

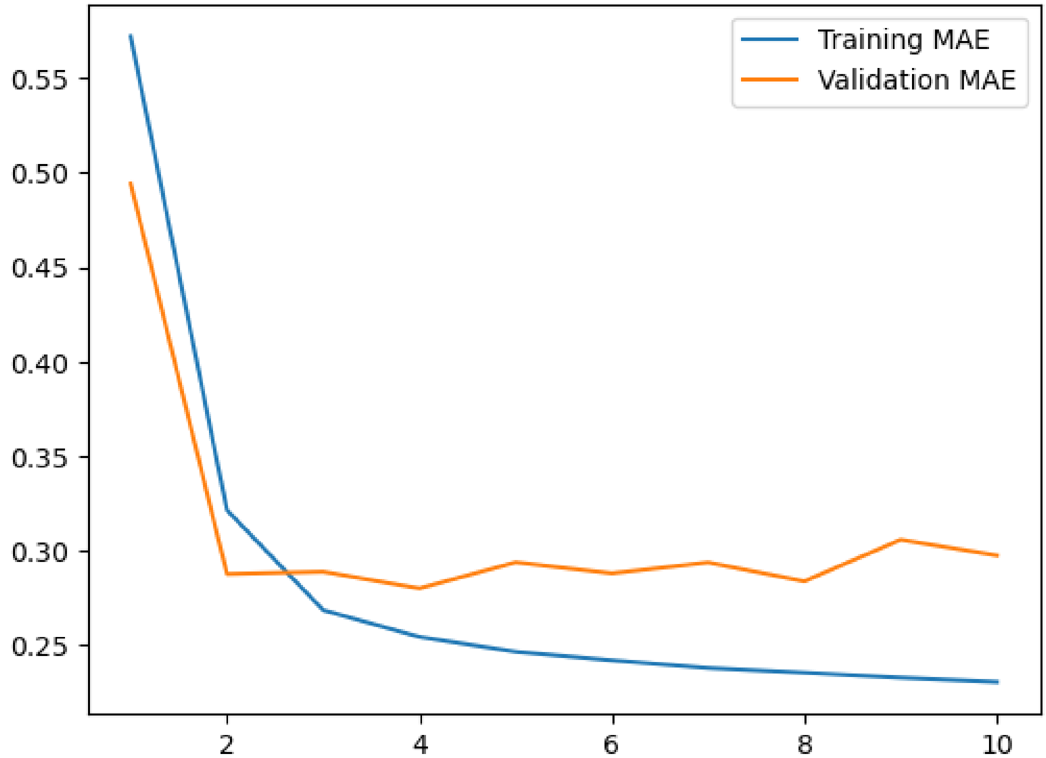
```
Epoch 1/10
819/819              8s 9ms/step -
loss: 1.4598 - mae: 0.7240 - val_loss: 0.3780 - val_mae: 0.4943
Epoch 2/10
819/819              8s 10ms/step -
loss: 0.2071 - mae: 0.3503 - val_loss: 0.1352 - val_mae: 0.2876
Epoch 3/10
819/819              8s 9ms/step -
loss: 0.1219 - mae: 0.2744 - val_loss: 0.1343 - val_mae: 0.2888
Epoch 4/10
819/819              8s 9ms/step -
loss: 0.1071 - mae: 0.2572 - val_loss: 0.1276 - val_mae: 0.2801
Epoch 5/10
```

```
819/819               8s 9ms/step -
loss: 0.1000 - mae: 0.2486 - val_loss: 0.1386 - val_mae: 0.2938
Epoch 6/10
819/819               7s 9ms/step -
loss: 0.0958 - mae: 0.2436 - val_loss: 0.1341 - val_mae: 0.2880
Epoch 7/10
819/819               8s 10ms/step -
loss: 0.0924 - mae: 0.2390 - val_loss: 0.1405 - val_mae: 0.2937
Epoch 8/10
819/819               7s 9ms/step -
loss: 0.0901 - mae: 0.2368 - val_loss: 0.1309 - val_mae: 0.2838
Epoch 9/10
819/819               8s 10ms/step -
loss: 0.0877 - mae: 0.2333 - val_loss: 0.1517 - val_mae: 0.3058
Epoch 10/10
819/819               8s 10ms/step -
loss: 0.0861 - mae: 0.2314 - val_loss: 0.1436 - val_mae: 0.2975
405/405               3s 6ms/step -
loss: 0.1476 - mae: 0.3039
Test MAE: 0.30
```

Plotting the results:

[50]:
```
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss,label="Training MAE")
plt.plot(epochs,val_loss,label="Validation MAE")
plt.legend()
```

[50]:
```
<matplotlib.legend.Legend at 0x156aa83d0>
```



Trying 1D convolution method now. :

[54]:
```
inputs = keras.Input(shape=(sequence_length, dfs.shape[-1]))
x = layers.Conv1D(8, 24, activation="relu")(inputs)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(8, 12, activation="relu")(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(8, 6, activation="relu")(x)
x = layers.GlobalAveragePooling1D()(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
callbacks = [
            keras.callbacks.ModelCheckpoint("/Users/devmarwah/Documents/MSBA␣
 ↪assignments/Advanced Machine Learning/Weather Forecasting/jena_conv.keras",
                                            save_best_only=True)
        ]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(Train,
                            epochs=10,
                            validation_data=Validation,
                            callbacks=callbacks)
```

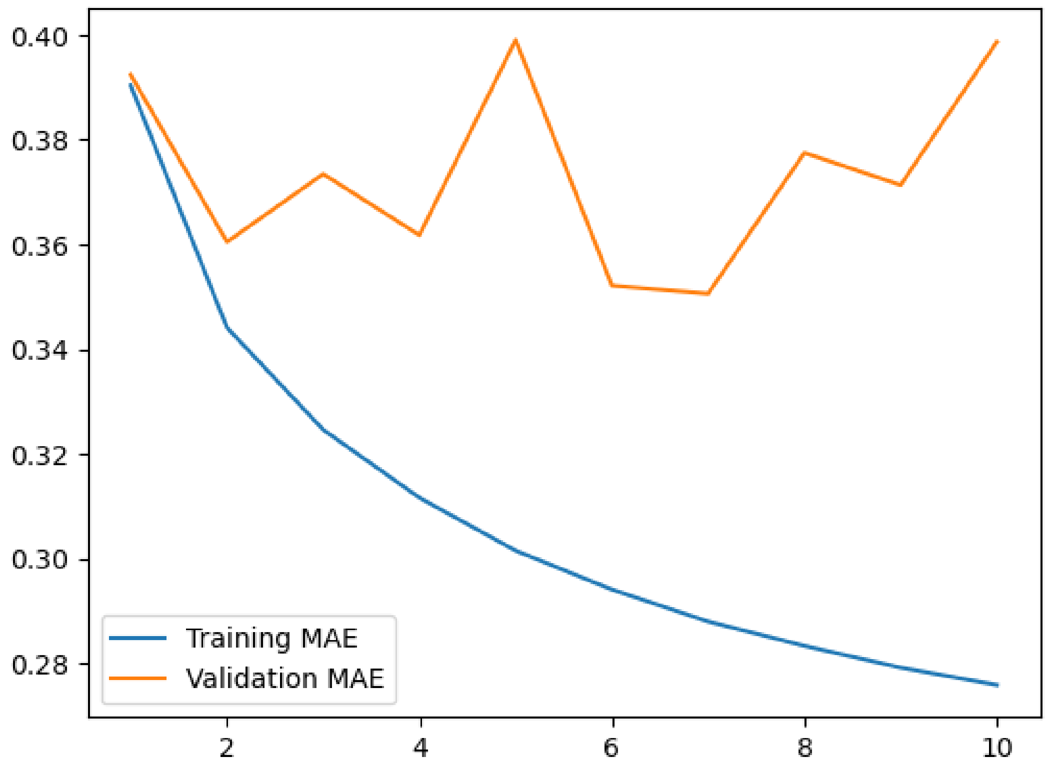
```
model = keras.models.load_model("/Users/devmarwah/Documents/MSBA assignments/
↪Advanced Machine Learning/Weather Forecasting/jena_conv.keras")
print(f"Test MAE: {model.evaluate(Test)[1]:.2f}")
```

```
Epoch 1/10
819/819              14s 16ms/step -
loss: 0.2943 - mae: 0.4259 - val_loss: 0.2469 - val_mae: 0.3924
Epoch 2/10
819/819              13s 16ms/step -
loss: 0.1948 - mae: 0.3501 - val_loss: 0.2110 - val_mae: 0.3604
Epoch 3/10
819/819              14s 17ms/step -
loss: 0.1721 - mae: 0.3282 - val_loss: 0.2262 - val_mae: 0.3734
Epoch 4/10
819/819              14s 17ms/step -
loss: 0.1586 - mae: 0.3147 - val_loss: 0.2110 - val_mae: 0.3617
Epoch 5/10
819/819              15s 18ms/step -
loss: 0.1474 - mae: 0.3031 - val_loss: 0.2594 - val_mae: 0.3990
Epoch 6/10
819/819              15s 18ms/step -
loss: 0.1408 - mae: 0.2960 - val_loss: 0.1999 - val_mae: 0.3521
Epoch 7/10
819/819              15s 18ms/step -
loss: 0.1344 - mae: 0.2895 - val_loss: 0.1994 - val_mae: 0.3506
Epoch 8/10
819/819              15s 19ms/step -
loss: 0.1299 - mae: 0.2845 - val_loss: 0.2295 - val_mae: 0.3775
Epoch 9/10
819/819              15s 19ms/step -
loss: 0.1259 - mae: 0.2803 - val_loss: 0.2242 - val_mae: 0.3713
Epoch 10/10
819/819              15s 19ms/step -
loss: 0.1225 - mae: 0.2769 - val_loss: 0.2557 - val_mae: 0.3986
405/405              4s 9ms/step -
loss: 0.2342 - mae: 0.3832
Test MAE: 0.38
```

Plotting the results. :

[55]:
```
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss,label="Training MAE")
plt.plot(epochs,val_loss,label="Validation MAE")
plt.legend()
```

[55]: <matplotlib.legend.Legend at 0x28e1d5810>



Constructing a simple recurrent neural network using LSTM model :

[62]:
```
inputs = keras.Input(shape=(sequence_length,dfs.shape[-1]))
x = layers.LSTM(16)(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs,outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("Users/devmarwah/Documents/MSBA assignments/
 ↪Advanced Machine Learning/Weather Forecasting/jena_lstm.keras",
                                    save_best_only=True)
]

model.compile(optimizer="rmsprop",loss = "mse",metrics=["mae"])

history=model.fit(Train,
                 epochs=10,
                 validation_data=Validation,
                 callbacks=callbacks)
```

```
model=keras.models.load_model("Users/devmarwah/Documents/MSBA assignments/
↪Advanced Machine Learning/Weather Forecasting/jena_lstm.keras")
```

```
Epoch 1/10
819/819              22s 26ms/step -
loss: 0.2250 - mae: 0.3494 - val_loss: 0.1148 - val_mae: 0.2631
Epoch 2/10
819/819              22s 26ms/step -
loss: 0.1193 - mae: 0.2701 - val_loss: 0.1145 - val_mae: 0.2625
Epoch 3/10
819/819              22s 27ms/step -
loss: 0.1099 - mae: 0.2598 - val_loss: 0.1171 - val_mae: 0.2652
Epoch 4/10
819/819              23s 28ms/step -
loss: 0.1045 - mae: 0.2539 - val_loss: 0.1193 - val_mae: 0.2672
Epoch 5/10
819/819              22s 27ms/step -
loss: 0.0999 - mae: 0.2485 - val_loss: 0.1247 - val_mae: 0.2740
Epoch 6/10
819/819              23s 28ms/step -
loss: 0.0957 - mae: 0.2436 - val_loss: 0.1271 - val_mae: 0.2769
Epoch 7/10
819/819              23s 28ms/step -
loss: 0.0922 - mae: 0.2390 - val_loss: 0.1278 - val_mae: 0.2776
Epoch 8/10
819/819              22s 27ms/step -
loss: 0.0887 - mae: 0.2343 - val_loss: 0.1276 - val_mae: 0.2776
Epoch 9/10
819/819              23s 28ms/step -
loss: 0.0855 - mae: 0.2299 - val_loss: 0.1366 - val_mae: 0.2863
Epoch 10/10
819/819              22s 27ms/step -
loss: 0.0828 - mae: 0.2262 - val_loss: 0.1446 - val_mae: 0.2940
```

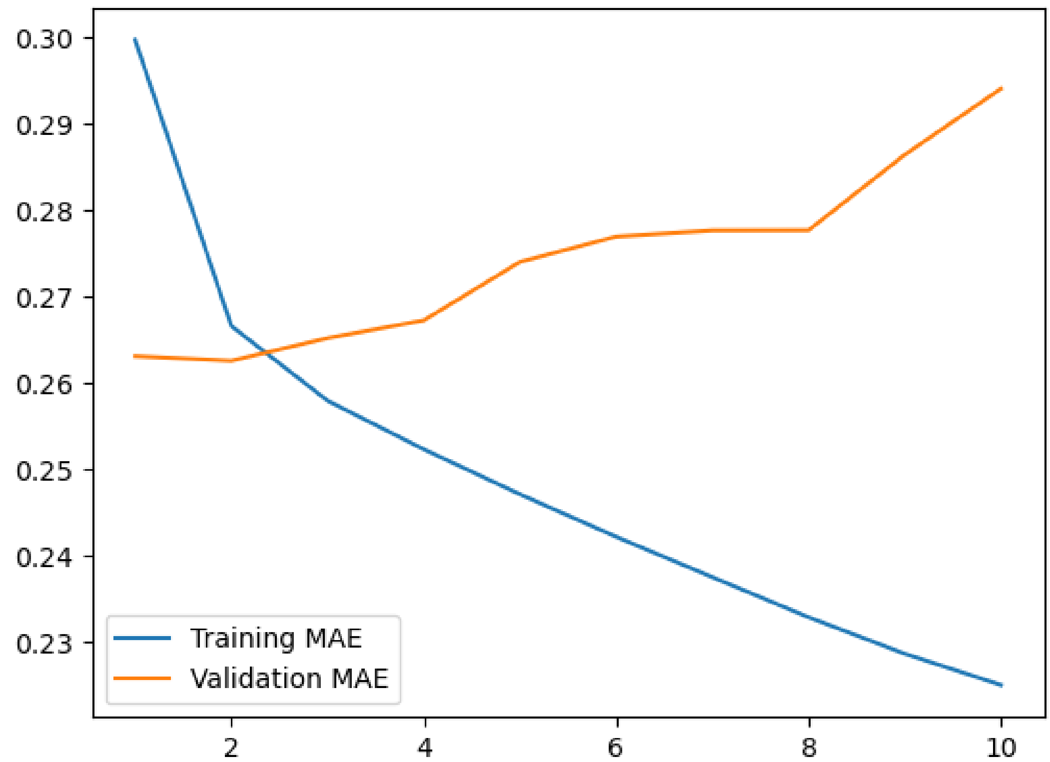
[65]:
```
print("Test MAE: ",round(model.evaluate(Test)[1],2))
```

```
405/405              4s 10ms/step -
loss: 0.1252 - mae: 0.2753
Test MAE:  0.28
```

Plotting results of LSTM model :

[66]:
```
loss = history.history['mae']
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss,label="Training MAE")
plt.plot(epochs,val_loss,label="Validation MAE")
plt.legend()
```

[66]: <matplotlib.legend.Legend at 0x28fa48f90>



Hence, we can conclude that recurrent neural networks work best on a time - series problem

1.2 Improving Forecasting

Now we will use different model architecures to acheive the best possible accuracy in our model.

Constructing a RNN with only one layer of GRU but increased units to check initial accuracy.

[69]:
```
inputs = keras.Input(shape=(sequence_length, dfs.shape[-1]))
x = layers.GRU(32, recurrent_dropout=0.25)(inputs)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
callbacks = [
            keras.callbacks.ModelCheckpoint("/Users/devmarwah/Documents/MSBA␣
 ↪assignments/Advanced Machine Learning/Weather Forecasting/jena_gru_dropout.
 ↪keras",
                                                save_best_only=True)
        ]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
```

```
history = model.fit(Train,
                            epochs=10,
                            validation_data=Validation,
                            callbacks=callbacks)
model = keras.models.load_model("Users/devmarwah/Documents/MSBA assignments/
 ↪Advanced Machine Learning/Weather Forecasting/jena_gru_dropout.keras")
```

```
Epoch 1/10
819/819              57s 68ms/step -
loss: 0.2557 - mae: 0.3812 - val_loss: 0.1187 - val_mae: 0.2693
Epoch 2/10
819/819              61s 75ms/step -
loss: 0.1643 - mae: 0.3157 - val_loss: 0.1137 - val_mae: 0.2630
Epoch 3/10
819/819              51s 62ms/step -
loss: 0.1602 - mae: 0.3113 - val_loss: 0.1144 - val_mae: 0.2640
Epoch 4/10
819/819              57s 70ms/step -
loss: 0.1569 - mae: 0.3085 - val_loss: 0.1117 - val_mae: 0.2601
Epoch 5/10
819/819              53s 65ms/step -
loss: 0.1528 - mae: 0.3043 - val_loss: 0.1126 - val_mae: 0.2614
Epoch 6/10
819/819              50s 61ms/step -
loss: 0.1497 - mae: 0.3016 - val_loss: 0.1118 - val_mae: 0.2606
Epoch 7/10
819/819              53s 65ms/step -
loss: 0.1454 - mae: 0.2972 - val_loss: 0.1137 - val_mae: 0.2627
Epoch 8/10
819/819              51s 62ms/step -
loss: 0.1421 - mae: 0.2942 - val_loss: 0.1172 - val_mae: 0.2672
Epoch 9/10
819/819              51s 62ms/step -
loss: 0.1398 - mae: 0.2912 - val_loss: 0.1121 - val_mae: 0.2604
Epoch 10/10
819/819              56s 68ms/step -
loss: 0.1365 - mae: 0.2883 - val_loss: 0.1147 - val_mae: 0.2649
```

[71]:
```
model.evaluate(Test)
```

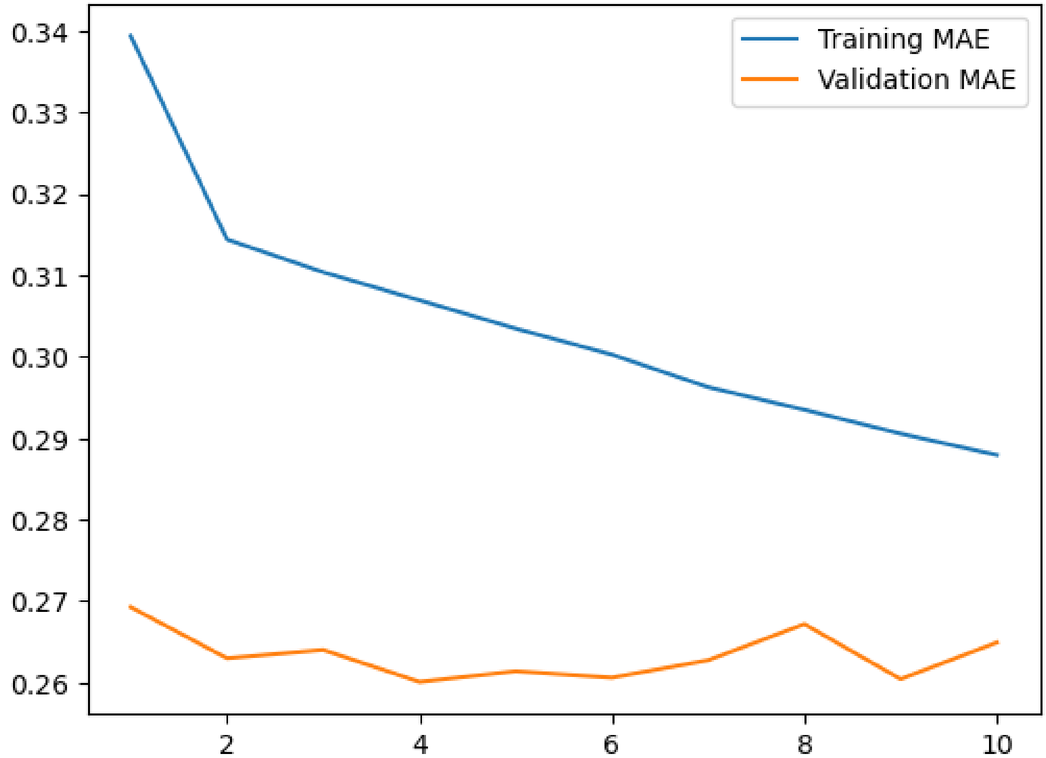
```
405/405              9s 21ms/step -
loss: 0.1277 - mae: 0.2792
```

[71]: [0.1279725730419159, 0.279487669467926]

[72]:
```
loss = history.history['mae']
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
```

```
plt.figure()
plt.plot(epochs, loss,label="Training MAE")
plt.plot(epochs,val_loss,label="Validation MAE")
plt.legend()
```

[72]: <matplotlib.legend.Legend at 0x2d5158f90>



Now, we will take measures to improve our accuracy.

We are using following points to increase our model's efficiency :

- Adjusting the number of units in each recurrent layer in the stacked setup.
- Using layer_lstm() instead of layer_gru().
- Using a combination of 1d_convnets and RNN.

Final Model Construction

[89]:
```
inputs = keras.Input(shape=(sequence_length,dfs.shape[-1]))
# Using incresased units and applying lstm instead of gru
x=layers.LSTM(32,recurrent_dropout=0.5,return_sequences=True)(inputs)
# Adding more layers to turn it into a stacked model
x=layers.LSTM(32,recurrent_dropout=0.5,return_sequences=True)(x)
# Applying 1D convolution
x = layers.Conv1D(8, 24, activation="relu")(x)
```

```
x = layers.MaxPooling1D(2)(x)
x = layers.GlobalAveragePooling1D()(x)
# We are also using dropout layer to regularize our results
x=layers.Dropout(0.5)(x)
outputs=layers.Dense(1)(x)
model=keras.Model(inputs,outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("/Users/devmarwah/Documents/MSBA␣
 ↪assignments/Advanced Machine Learning/Weather Forecasting/jena_stacked_final.
 ↪keras",
                                    save_best_only = True)
]

model.compile(optimizer="rmsprop",loss="mse",metrics=["mae"])

history=model.fit(Train,
                  epochs=10,
                  validation_data=Validation,
                  callbacks=callbacks)
```

```
Epoch 1/10
819/819              108s 131ms/step -
loss: 0.4698 - mae: 0.5320 - val_loss: 0.2734 - val_mae: 0.4102
Epoch 2/10
819/819              107s 131ms/step -
loss: 0.3609 - mae: 0.4599 - val_loss: 0.2372 - val_mae: 0.3828
Epoch 3/10
819/819              107s 130ms/step -
loss: 0.3285 - mae: 0.4324 - val_loss: 0.2543 - val_mae: 0.4015
Epoch 4/10
819/819              106s 130ms/step -
loss: 0.3057 - mae: 0.4149 - val_loss: 0.2474 - val_mae: 0.3969
Epoch 5/10
819/819              107s 130ms/step -
loss: 0.2961 - mae: 0.4056 - val_loss: 0.2600 - val_mae: 0.4028
Epoch 6/10
819/819              106s 130ms/step -
loss: 0.2894 - mae: 0.3987 - val_loss: 0.2493 - val_mae: 0.3951
Epoch 7/10
819/819              105s 128ms/step -
loss: 0.2827 - mae: 0.3929 - val_loss: 0.2592 - val_mae: 0.4054
Epoch 8/10
819/819              106s 129ms/step -
loss: 0.2792 - mae: 0.3894 - val_loss: 0.2539 - val_mae: 0.3993
Epoch 9/10
819/819              106s 129ms/step -
loss: 0.2722 - mae: 0.3841 - val_loss: 0.2698 - val_mae: 0.4143
```

```
Epoch 10/10
819/819              107s 130ms/step -
loss: 0.2702 - mae: 0.3807 - val_loss: 0.2670 - val_mae: 0.4088
```

[95]:
```
model=keras.models.load_model("/Users/devmarwah/Documents/MSBA assignments/
  ↪Advanced Machine Learning/Weather Forecasting/jena_stacked_final.keras")
model.evaluate(Test)[1]
```

```
405/405              17s 41ms/step -
loss: 0.2792 - mae: 0.4163
```

[95]:
```
0.41434764862060547
```

Hence, our final model can predict temperature with a very low mean absolute error of 0.42