Weather_timeseries_bchennu

July 21, 2024

1 Deep learning for timeseries

1.1 Weather Forecasting Using Time Series

---

[1]:
```
!wget https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip
!unzip jena_climate_2009_2016.csv.zip
```

```
--2024-07-21 14:18:28--  https://s3.amazonaws.com/keras-
datasets/jena_climate_2009_2016.csv.zip
Resolving s3.amazonaws.com (s3.amazonaws.com)… 52.216.206.133, 3.5.16.5,
16.182.64.216, …
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.216.206.133|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 13565642 (13M) [application/zip]
Saving to: ‘jena_climate_2009_2016.csv.zip’

jena_climate_2009_2 100%[===================>]  12.94M  6.78MB/s    in 1.9s

2024-07-21 14:18:31 (6.78 MB/s) - ‘jena_climate_2009_2016.csv.zip’ saved
[13565642/13565642]

Archive:  jena_climate_2009_2016.csv.zip
  inflating: jena_climate_2009_2016.csv
  inflating: __MACOSX/._jena_climate_2009_2016.csv
```

Inspecting the data of the Jena weather dataset

[2]:
```
import os
fname = os.path.join("jena_climate_2009_2016.csv")

with open(fname) as f:
    data = f.read()

lines = data.split("\n")
header = lines[0].split(",")
lines = lines[1:]
```

```python
print(header)
print(len(lines))
```

```
['"Date Time"', '"p (mbar)"', '"T (degC)"', '"Tpot (K)"', '"Tdew (degC)"', '"rh
(%)"', '"VPmax (mbar)"', '"VPact (mbar)"', '"VPdef (mbar)"', '"sh (g/kg)"',
'"H2OC (mmol/mol)"', '"rho (g/m**3)"', '"wv (m/s)"', '"max. wv (m/s)"', '"wd
(deg)"']
420451
```

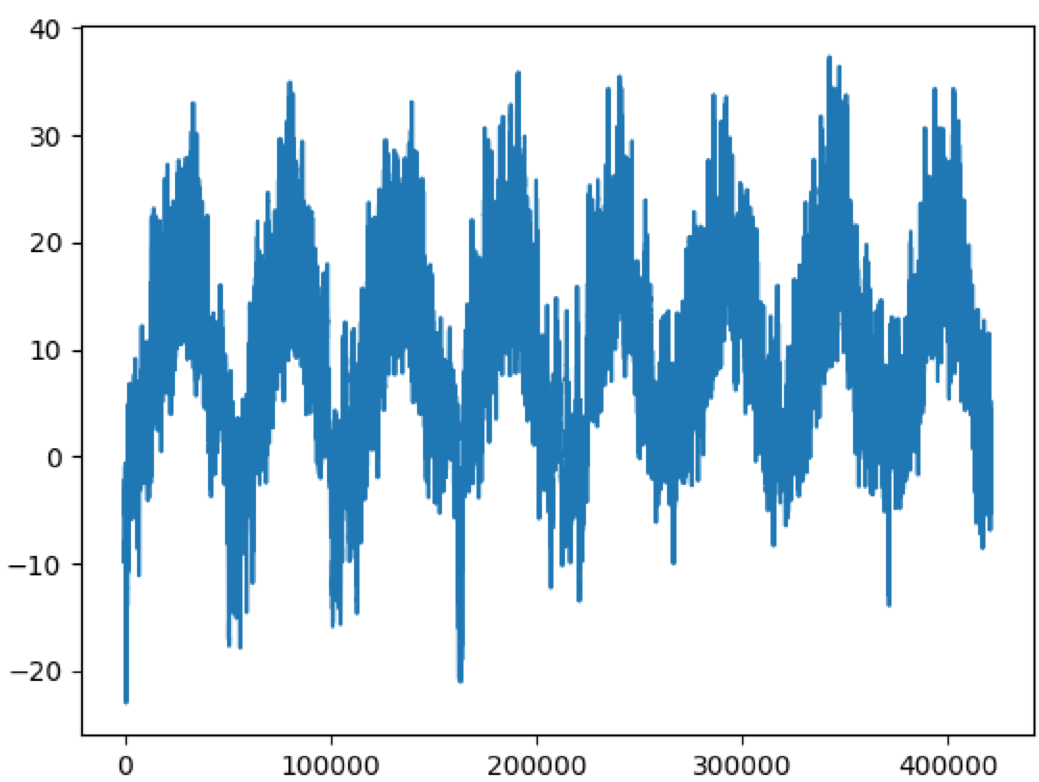
Parsing the data

[3]:
```python
import numpy as np
temperature = np.zeros((len(lines),))
raw_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(",")[1:]]
    temperature[i] = values[1]
    raw_data[i, :] = values[:]
```

Plotting the temperature timeseries

[4]:
```python
from matplotlib import pyplot as plt
plt.plot(range(len(temperature)), temperature)
```

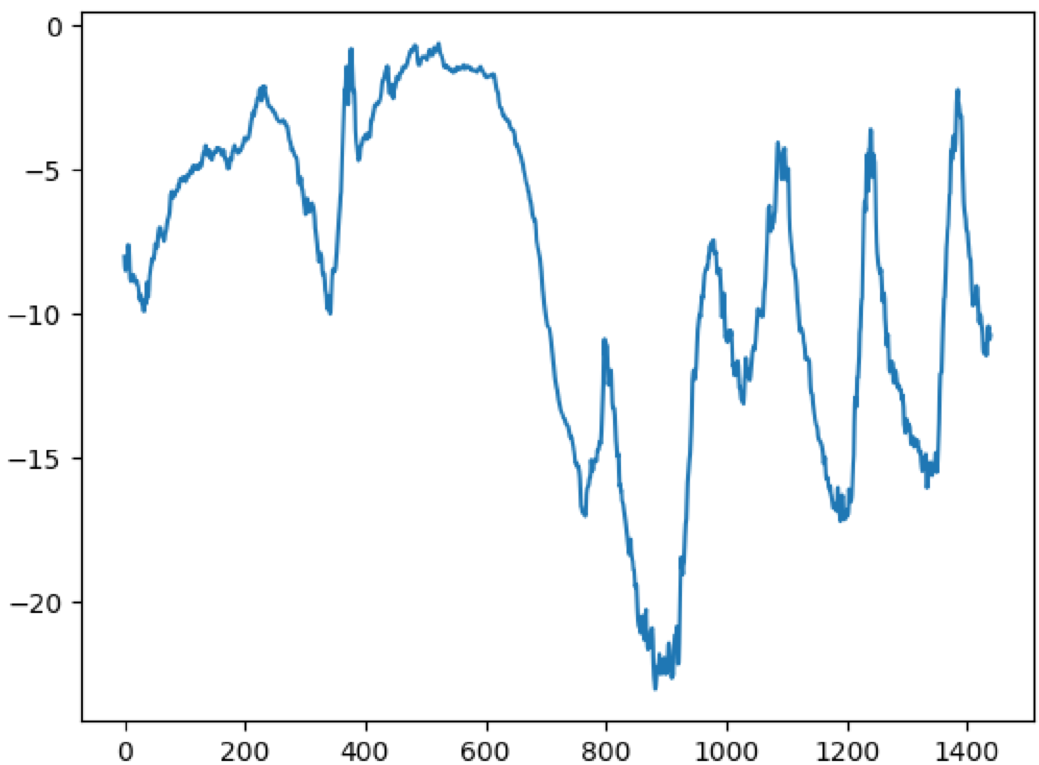
[4]:
```
[<matplotlib.lines.Line2D at 0x78e0d5f93160>]
```



Plotting the first 10 days of the temperature timeseries

[5]:
```
plt.plot(range(1440), temperature[:1440])
```

[5]:
```
[<matplotlib.lines.Line2D at 0x78e0a97e84c0>]
```



Computing the number of samples we'll use for each data split

[6]:
```
num_train_samples = int(0.5 * len(raw_data))
num_val_samples = int(0.25 * len(raw_data))
num_test_samples = len(raw_data) - num_train_samples - num_val_samples
print("num_train_samples:", num_train_samples)
print("num_val_samples:", num_val_samples)
print("num_test_samples:", num_test_samples)
```

```
num_train_samples: 210225
num_val_samples: 105112
num_test_samples: 105114
```

1.1.1 Preparing the data

Normalizing the data

[7]:
```
mean = raw_data[:num_train_samples].mean(axis=0)
raw_data -= mean
std = raw_data[:num_train_samples].std(axis=0)
raw_data /= std
```

[9]:
```
import numpy as np
from tensorflow import keras
int_sequence = np.arange(10)
dummy_dataset = keras.utils.timeseries_dataset_from_array(
    data=int_sequence[:-3],
    targets=int_sequence[3:],
    sequence_length=3,
    batch_size=2,
)

for inputs, targets in dummy_dataset:
    for i in range(inputs.shape[0]):
        print([int(x) for x in inputs[i]], int(targets[i]))
```

```
[0, 1, 2] 3
[1, 2, 3] 4
[2, 3, 4] 5
[3, 4, 5] 6
[4, 5, 6] 7
```

Instantiating datasets for training, validation, and testing

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

train_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=0,
    end_index=num_train_samples)

val_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
```

```
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples,
    end_index=num_train_samples + num_val_samples)

test_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples + num_val_samples)
```

Inspecting the output of one of our datasets

[11]:
```
for samples, targets in train_dataset:
    print("samples shape:", samples.shape)
    print("targets shape:", targets.shape)
    break
```

```
samples shape: (256, 120, 14)
targets shape: (256,)
```

1.1.2 A common-sense, non-machine-learning baseline

Computing the common-sense baseline MAE

[12]:
```
def evaluate_naive_method(dataset):
    total_abs_err = 0.
    samples_seen = 0
    for samples, targets in dataset:
        preds = samples[:, -1, 1] * std[1] + mean[1]
        total_abs_err += np.sum(np.abs(preds - targets))
        samples_seen += samples.shape[0]
    return total_abs_err / samples_seen

print(f"Validation MAE: {evaluate_naive_method(val_dataset):.2f}")
print(f"Test MAE: {evaluate_naive_method(test_dataset):.2f}")
```

```
Validation MAE: 2.44
Test MAE: 2.62
```

A Basic model with regular calculation has been performed and the validation and test MAE is as follows:

Validation MAE: 2.44 Test MAE: 2.62

1.1.3 Initial Learning Model

Training and evaluating a densely connected model

- With two dense layers and 32 units in input layer with relu activation function.
- RMSprop optimizer is chosen for training the model, offering adaptive learning rates.
- Mean Squared Error (MSE) is specified as the loss function, measuring the difference between predicted and actual values.
- Mean Absolute Error (MAE) is defined as a metric to monitor during training, providing insight into the model's performance on the validation set.

[13]:
```python
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Flatten()(inputs)
x = layers.Dense(32, activation="relu")(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_dense.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_dense.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

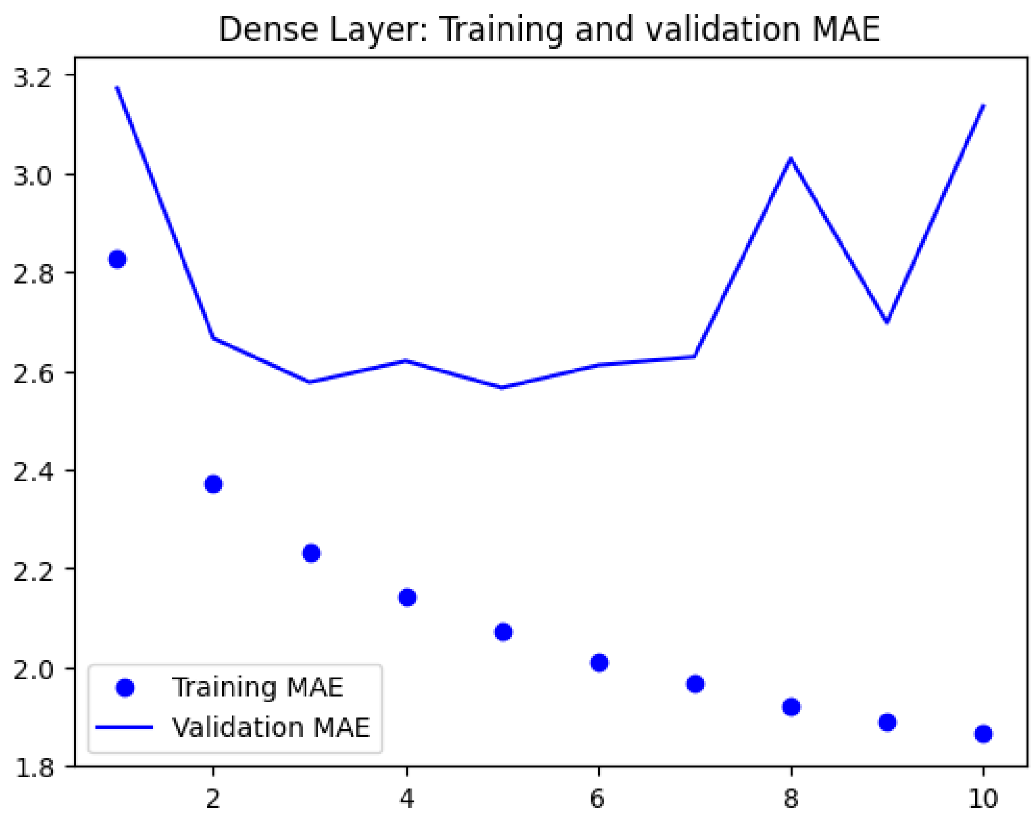
```
Epoch 1/10
819/819 [==============================] - 13s 15ms/step - loss: 13.3733 - mae:
2.8267 - val_loss: 15.7769 - val_mae: 3.1723
Epoch 2/10
819/819 [==============================] - 12s 14ms/step - loss: 9.1268 - mae:
2.3746 - val_loss: 11.4966 - val_mae: 2.6661
Epoch 3/10
819/819 [==============================] - 12s 14ms/step - loss: 8.0438 - mae:
2.2322 - val_loss: 10.7687 - val_mae: 2.5770
Epoch 4/10
819/819 [==============================] - 13s 15ms/step - loss: 7.3975 - mae:
2.1447 - val_loss: 11.0758 - val_mae: 2.6202
Epoch 5/10
819/819 [==============================] - 12s 15ms/step - loss: 6.9258 - mae:
2.0733 - val_loss: 10.7143 - val_mae: 2.5659
Epoch 6/10
```

```
819/819 [==============================] - 12s 14ms/step - loss: 6.5165 - mae:
2.0113 - val_loss: 10.9510 - val_mae: 2.6117
Epoch 7/10
819/819 [==============================] - 13s 16ms/step - loss: 6.2382 - mae:
1.9691 - val_loss: 11.1565 - val_mae: 2.6287
Epoch 8/10
819/819 [==============================] - 13s 15ms/step - loss: 5.9627 - mae:
1.9235 - val_loss: 14.6637 - val_mae: 3.0303
Epoch 9/10
819/819 [==============================] - 12s 14ms/step - loss: 5.7752 - mae:
1.8917 - val_loss: 11.7476 - val_mae: 2.6977
Epoch 10/10
819/819 [==============================] - 12s 15ms/step - loss: 5.6101 - mae:
1.8653 - val_loss: 15.4184 - val_mae: 3.1358
405/405 [==============================] - 4s 9ms/step - loss: 11.5684 - mae:
2.6811
Test MAE: 2.68
```

Obtained a test MAE of **2.68** with densely connected model

Plotting results

[14]:
```python
import matplotlib.pyplot as plt
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("Dense Layer: Training and validation MAE")
plt.legend()
plt.show()
```

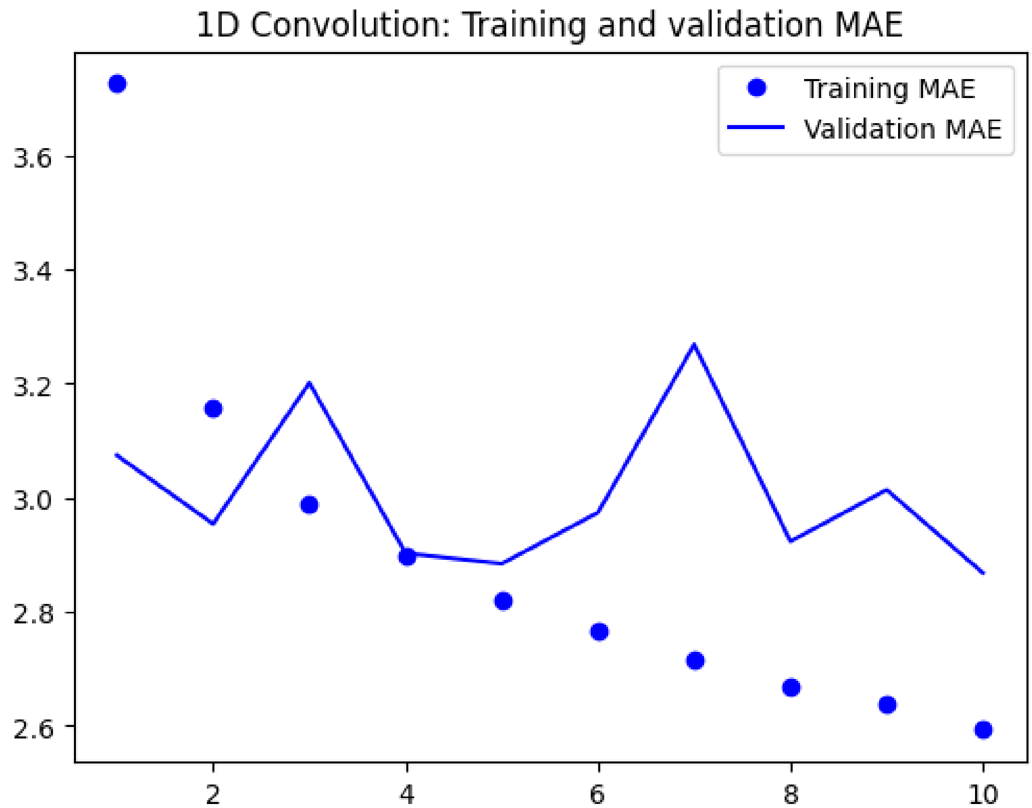


1.1.4 Let's try a 1D convolutional model

[15]:
```
inputs = keras.Input(shape=(sequence_length, raw_data.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("jena_conv.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)
```

```python
model = keras.models.load_model("jena_conv.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")

loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("1D Convolution: Training and validation MAE")
plt.legend()
plt.show()
```

```
Epoch 1/10
819/819 [==============================] - 15s 15ms/step - loss: 22.4272 - mae:
3.7258 - val_loss: 15.2770 - val_mae: 3.0743
Epoch 2/10
819/819 [==============================] - 12s 15ms/step - loss: 15.8676 - mae:
3.1564 - val_loss: 13.9672 - val_mae: 2.9542
Epoch 3/10
819/819 [==============================] - 12s 15ms/step - loss: 14.2923 - mae:
2.9895 - val_loss: 16.2133 - val_mae: 3.2013
Epoch 4/10
819/819 [==============================] - 12s 15ms/step - loss: 13.4124 - mae:
2.8983 - val_loss: 13.5296 - val_mae: 2.9028
Epoch 5/10
819/819 [==============================] - 12s 15ms/step - loss: 12.6881 - mae:
2.8194 - val_loss: 13.3112 - val_mae: 2.8845
Epoch 6/10
819/819 [==============================] - 13s 16ms/step - loss: 12.2075 - mae:
2.7656 - val_loss: 14.0751 - val_mae: 2.9740
Epoch 7/10
819/819 [==============================] - 12s 15ms/step - loss: 11.7921 - mae:
2.7170 - val_loss: 17.1871 - val_mae: 3.2688
Epoch 8/10
819/819 [==============================] - 13s 16ms/step - loss: 11.4077 - mae:
2.6705 - val_loss: 13.8652 - val_mae: 2.9235
Epoch 9/10
819/819 [==============================] - 12s 15ms/step - loss: 11.1002 - mae:
2.6374 - val_loss: 14.8257 - val_mae: 3.0139
Epoch 10/10
819/819 [==============================] - 12s 15ms/step - loss: 10.7471 - mae:
2.5931 - val_loss: 13.3510 - val_mae: 2.8683
405/405 [==============================] - 4s 9ms/step - loss: 15.1325 - mae:
3.0905
Test MAE: 3.09
```



A regular 1D convultional network yielded a test MAE of 3.09 which is more than the dense layer network means it is underperforming.

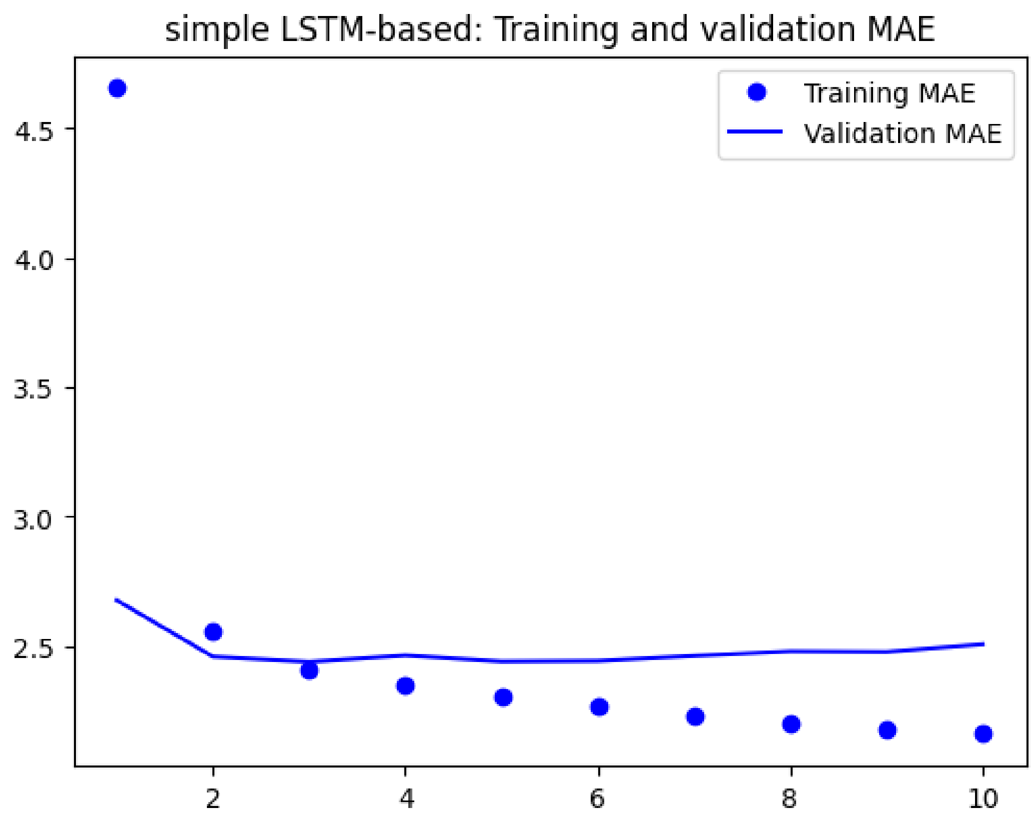
1.1.5 A first recurrent baseline

A simple LSTM-based model

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

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)
```

```python
model = keras.models.load_model("jena_lstm.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")

loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("simple LSTM-based: Training and validation MAE")
plt.legend()
plt.show()
```

```
Epoch 1/10
819/819 [==============================] - 16s 17ms/step - loss: 40.9196 - mae:
4.6549 - val_loss: 12.3398 - val_mae: 2.6779
Epoch 2/10
819/819 [==============================] - 13s 16ms/step - loss: 10.8266 - mae:
2.5570 - val_loss: 10.0843 - val_mae: 2.4600
Epoch 3/10
819/819 [==============================] - 13s 16ms/step - loss: 9.5208 - mae:
2.4072 - val_loss: 9.9154 - val_mae: 2.4402
Epoch 4/10
819/819 [==============================] - 13s 16ms/step - loss: 9.0954 - mae:
2.3496 - val_loss: 10.1160 - val_mae: 2.4648
Epoch 5/10
819/819 [==============================] - 13s 15ms/step - loss: 8.7942 - mae:
2.3056 - val_loss: 9.9226 - val_mae: 2.4418
Epoch 6/10
819/819 [==============================] - 13s 16ms/step - loss: 8.5086 - mae:
2.2684 - val_loss: 9.9726 - val_mae: 2.4438
Epoch 7/10
819/819 [==============================] - 13s 15ms/step - loss: 8.2426 - mae:
2.2342 - val_loss: 10.0058 - val_mae: 2.4631
Epoch 8/10
819/819 [==============================] - 13s 16ms/step - loss: 7.9939 - mae:
2.2030 - val_loss: 10.2501 - val_mae: 2.4803
Epoch 9/10
819/819 [==============================] - 13s 16ms/step - loss: 7.8109 - mae:
2.1806 - val_loss: 10.2247 - val_mae: 2.4783
Epoch 10/10
819/819 [==============================] - 13s 15ms/step - loss: 7.6667 - mae:
2.1615 - val_loss: 10.4526 - val_mae: 2.5076
405/405 [==============================] - 5s 10ms/step - loss: 11.0154 - mae:
2.6200
Test MAE: 2.62
```



A basic baseline RNN was built using LSTM and the test MAE has improved to 2.62

1.2 Understanding recurrent neural networks

NumPy implementation of a simple RNN

[17]:
```
import numpy as np
timesteps = 100
input_features = 32
output_features = 64
inputs = np.random.random((timesteps, input_features))
state_t = np.zeros((output_features,))
W = np.random.random((output_features, input_features))
U = np.random.random((output_features, output_features))
b = np.random.random((output_features,))
successive_outputs = []
for input_t in inputs:
    output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)
    successive_outputs.append(output_t)
    state_t = output_t
final_output_sequence = np.stack(successive_outputs, axis=0)
```

2 1. Adjusting the number of units in each recurrent layer in the stacked setup

2.0.1 Using SimpleRNN in Keras

Stacking RNN layers

- Stacked SimpleRNN layers with increasing units (32, 32) process sequential data.
- RMSprop optimizer is used with Mean Squared Error (MSE) loss and Mean Absolute Error (MAE) metric.

[18]:
```
steps = 120
num_features = 32
inputs = keras.Input(shape=(steps, num_features))
x = layers.SimpleRNN(32, return_sequences=True)(inputs)
x = layers.SimpleRNN(32, return_sequences=True)(x)
outputs = layers.SimpleRNN(16)(x)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_simple_rnn.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)
```

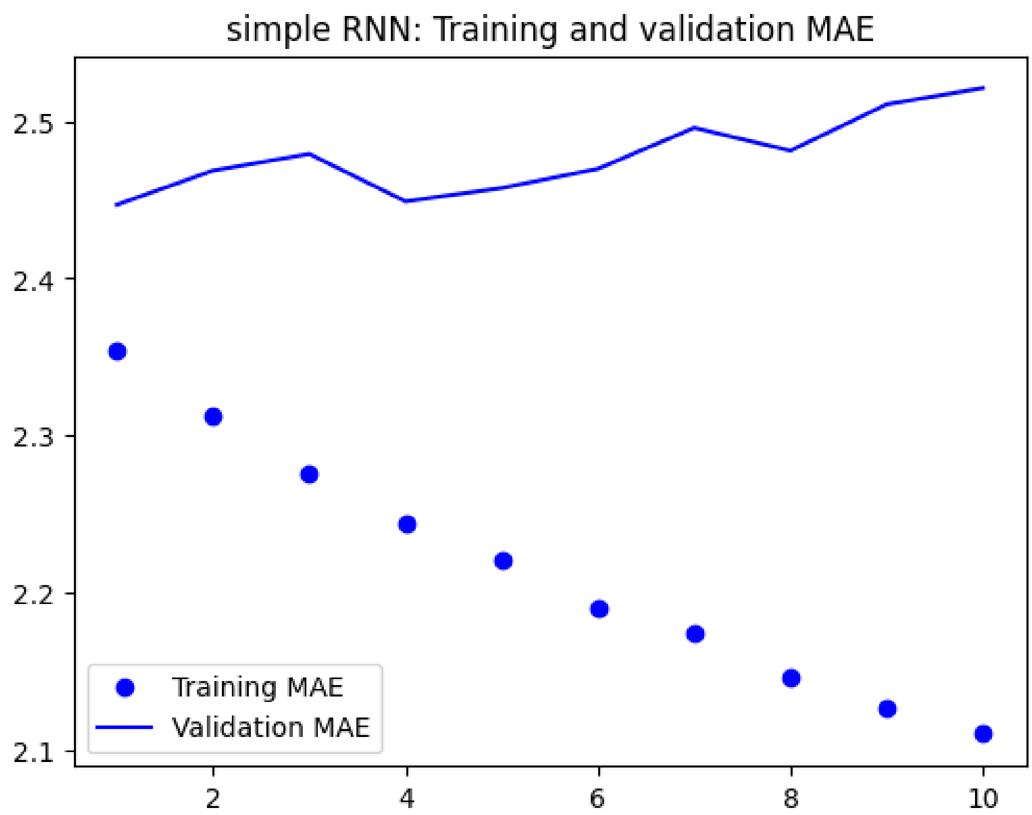
```
Epoch 1/10
819/819 [==============================] - 15s 16ms/step - loss: 9.1226 - mae:
2.3538 - val_loss: 9.8444 - val_mae: 2.4467
Epoch 2/10
819/819 [==============================] - 13s 16ms/step - loss: 8.8265 - mae:
2.3124 - val_loss: 10.1080 - val_mae: 2.4684
Epoch 3/10
819/819 [==============================] - 13s 15ms/step - loss: 8.5488 - mae:
2.2757 - val_loss: 10.1479 - val_mae: 2.4790
Epoch 4/10
819/819 [==============================] - 13s 16ms/step - loss: 8.2948 - mae:
2.2441 - val_loss: 9.9140 - val_mae: 2.4488
Epoch 5/10
819/819 [==============================] - 13s 16ms/step - loss: 8.1048 - mae:
2.2207 - val_loss: 10.0210 - val_mae: 2.4573
Epoch 6/10
819/819 [==============================] - 13s 15ms/step - loss: 7.8778 - mae:
2.1899 - val_loss: 10.0274 - val_mae: 2.4696
Epoch 7/10
819/819 [==============================] - 13s 16ms/step - loss: 7.7565 - mae:
2.1744 - val_loss: 10.2612 - val_mae: 2.4956
```

```
Epoch 8/10
819/819 [==============================] - 13s 16ms/step - loss: 7.5463 - mae:
2.1459 - val_loss: 10.2559 - val_mae: 2.4811
Epoch 9/10
819/819 [==============================] - 13s 16ms/step - loss: 7.4195 - mae:
2.1265 - val_loss: 10.3864 - val_mae: 2.5105
Epoch 10/10
819/819 [==============================] - 13s 16ms/step - loss: 7.3134 - mae:
2.1104 - val_loss: 10.6272 - val_mae: 2.5208
```

[19]:

```
model = keras.models.load_model("jena_simple_rnn.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")

loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("simple RNN: Training and validation MAE")
plt.legend()
plt.show()
```

```
405/405 [==============================] - 5s 10ms/step - loss: 11.0609 - mae:
2.5824
Test MAE: 2.58
```



- A simpleRNN with two layer has a MAE of 2.58

3 2. Using layer_lstm() instead of layer_gru()

3.0.1 Stacking RNNs with GRU and LSTM

Training and evaluating a dropout-regularized, stacked GRU model

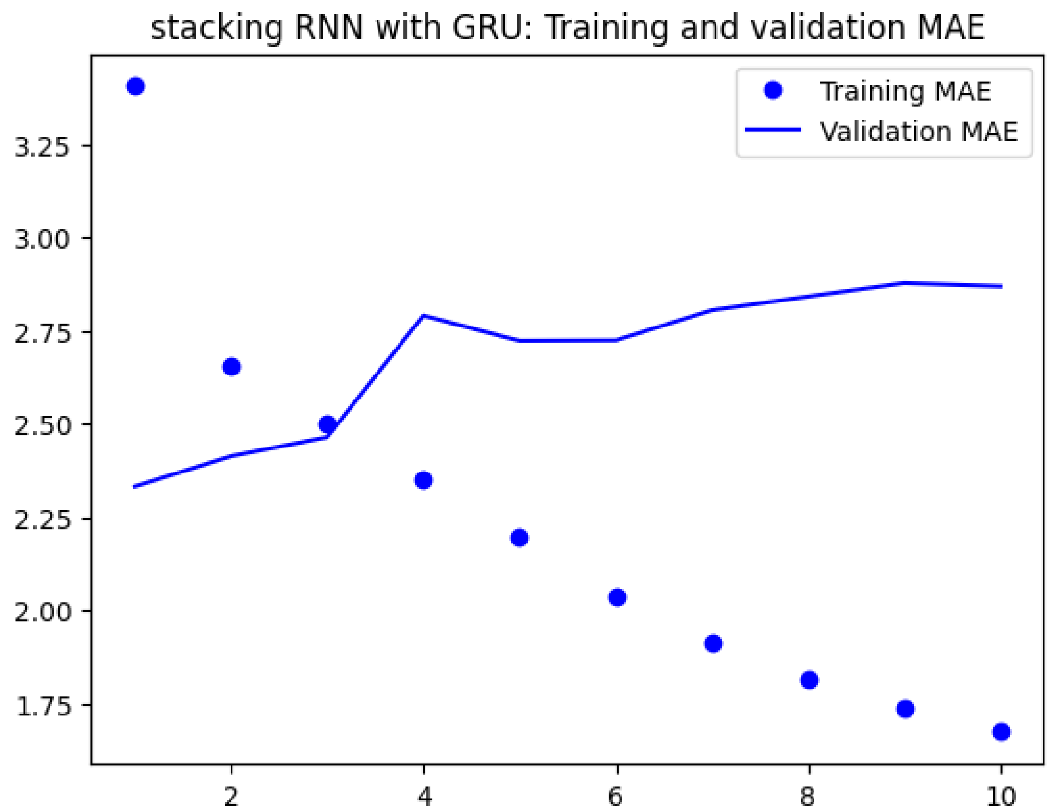
- Two stacked GRU layers are employed, with 64 units in the first layer and 32 units in the second layer.
- The second GRU layer is followed by a dropout layer with a dropout rate of 0.4 to prevent overfitting.
- The model is compiled using the RMSprop optimizer, Mean Squared Error (MSE) loss function, and Mean Absolute Error (MAE) metric.

[20]:
```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.GRU(64, return_sequences=True)(inputs)
x = layers.GRU(32)(x)
x = layers.Dropout(0.4)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
```

```python
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_stacked_gru_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_stacked_gru_dropout.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")

loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("stacking RNN with GRU: Training and validation MAE")
plt.legend()
plt.show()
```

```
Epoch 1/10
819/819 [==============================] - 21s 21ms/step - loss: 21.3686 - mae:
3.4066 - val_loss: 9.0560 - val_mae: 2.3318
Epoch 2/10
819/819 [==============================] - 16s 19ms/step - loss: 11.6831 - mae:
2.6539 - val_loss: 9.5710 - val_mae: 2.4131
Epoch 3/10
819/819 [==============================] - 15s 19ms/step - loss: 10.3468 - mae:
2.5007 - val_loss: 9.8497 - val_mae: 2.4640
Epoch 4/10
819/819 [==============================] - 15s 19ms/step - loss: 9.1452 - mae:
2.3523 - val_loss: 12.7537 - val_mae: 2.7907
Epoch 5/10
819/819 [==============================] - 16s 19ms/step - loss: 8.0173 - mae:
2.1960 - val_loss: 12.1706 - val_mae: 2.7229
Epoch 6/10
819/819 [==============================] - 15s 19ms/step - loss: 6.9577 - mae:
2.0366 - val_loss: 12.1050 - val_mae: 2.7241
Epoch 7/10
819/819 [==============================] - 16s 19ms/step - loss: 6.1719 - mae:
1.9115 - val_loss: 12.8373 - val_mae: 2.8050
Epoch 8/10
819/819 [==============================] - 15s 19ms/step - loss: 5.6127 - mae:
1.8143 - val_loss: 13.1800 - val_mae: 2.8416
Epoch 9/10
```

```
819/819 [==============================] - 16s 19ms/step - loss: 5.1673 - mae:
1.7378 - val_loss: 13.4377 - val_mae: 2.8777
Epoch 10/10
819/819 [==============================] - 16s 19ms/step - loss: 4.8124 - mae:
1.6728 - val_loss: 13.4197 - val_mae: 2.8691
405/405 [==============================] - 5s 10ms/step - loss: 10.5346 - mae:
2.5181
Test MAE: 2.52
```



- Using GRU stacked RNN the test MAE reduced to even more to **2.52**.
- It can be seen that a stacked two layer GRU RNN has better results than simpleRNN

Training and evaluating a dropout-regularized LSTM

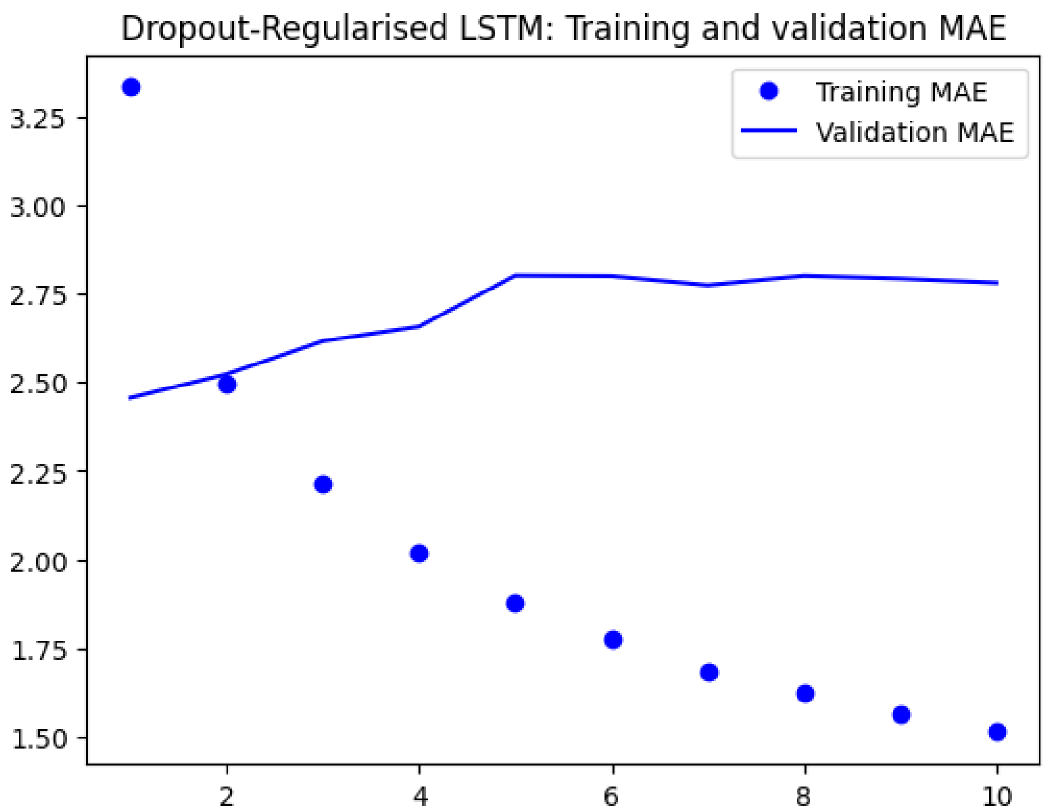
- This model comprises two LSTM (Long Short-Term Memory) layers. The first layer has 64 units, followed by a second layer with 32 units.
- A dropout layer with a dropout rate of 0.4 is inserted between the two LSTM layers. Dropout is effective for regularizing the model and reducing overfitting by randomly dropping 40% of the units during training.
- The model is compiled using the RMSprop optimizer, a robust optimizer for training recurrent neural networks.
- Mean Squared Error (MSE) is chosen as the loss function to measure the difference between predicted and actual values.

- Mean Absolute Error (MAE) is selected as a metric to monitor during training, providing insight into the model's performance on the validation set.

[21]:
```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(64, return_sequences=True)(inputs)
x = layers.LSTM(32)(x)
x = layers.Dropout(0.4)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_lstm_dropout.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("Dropout-Regularised LSTM: Training and validation MAE")
plt.legend()
plt.show()
```

```
Epoch 1/10
819/819 [==============================] - 19s 20ms/step - loss: 20.4452 - mae:
3.3329 - val_loss: 9.8503 - val_mae: 2.4560
Epoch 2/10
819/819 [==============================] - 16s 19ms/step - loss: 10.4097 - mae:
2.4956 - val_loss: 10.2442 - val_mae: 2.5225
Epoch 3/10
819/819 [==============================] - 16s 19ms/step - loss: 8.3368 - mae:
2.2126 - val_loss: 11.0479 - val_mae: 2.6160
Epoch 4/10
819/819 [==============================] - 16s 19ms/step - loss: 7.0495 - mae:
2.0177 - val_loss: 11.5107 - val_mae: 2.6571
Epoch 5/10
819/819 [==============================] - 15s 19ms/step - loss: 6.1942 - mae:
1.8789 - val_loss: 12.7000 - val_mae: 2.8008
Epoch 6/10
819/819 [==============================] - 16s 19ms/step - loss: 5.5482 - mae:
```

```
1.7738 - val_loss: 12.7147 - val_mae: 2.7994
Epoch 7/10
819/819 [==============================] - 16s 19ms/step - loss: 5.0375 - mae:
1.6846 - val_loss: 12.5165 - val_mae: 2.7737
Epoch 8/10
819/819 [==============================] - 16s 19ms/step - loss: 4.6890 - mae:
1.6222 - val_loss: 12.6873 - val_mae: 2.8002
Epoch 9/10
819/819 [==============================] - 15s 19ms/step - loss: 4.3845 - mae:
1.5650 - val_loss: 12.6682 - val_mae: 2.7922
Epoch 10/10
819/819 [==============================] - 16s 19ms/step - loss: 4.1085 - mae:
1.5130 - val_loss: 12.5147 - val_mae: 2.7805
405/405 [==============================] - 5s 10ms/step - loss: 11.1090 - mae:
2.6062
Test MAE: 2.61
```



- With LSTM, the test MAE is 2.61 which is little similar to GRU.
- Both LSTM and GRU performed similarly with slight changes.

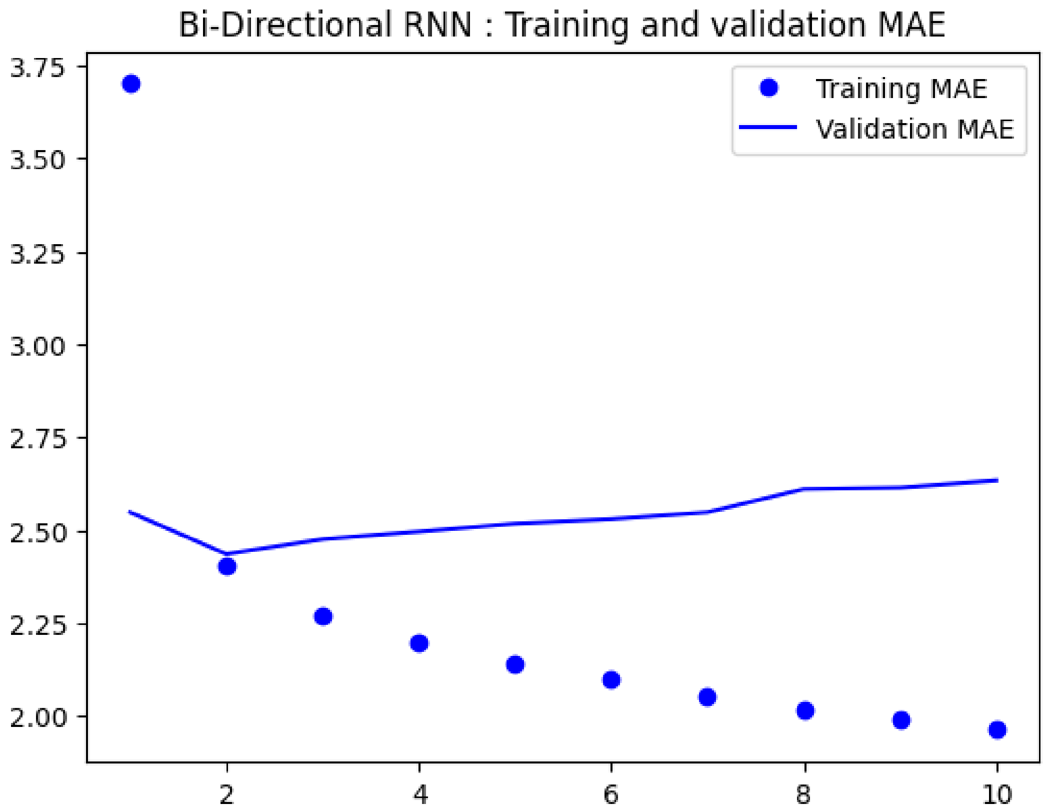
3.0.2 Using bidirectional RNNs

Training and evaluating a bidirectional LSTM

[25]:
```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Bidirectional(layers.LSTM(16))(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset)
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("Bi-Directional RNN : Training and validation MAE")
plt.legend()
plt.show()
```

```
Epoch 1/10
819/819 [==============================] - 19s 19ms/step - loss: 26.3529 - mae:
3.7003 - val_loss: 10.9238 - val_mae: 2.5477
Epoch 2/10
819/819 [==============================] - 15s 18ms/step - loss: 9.4402 - mae:
2.4032 - val_loss: 10.0707 - val_mae: 2.4354
Epoch 3/10
819/819 [==============================] - 15s 18ms/step - loss: 8.4654 - mae:
2.2721 - val_loss: 10.2811 - val_mae: 2.4752
Epoch 4/10
819/819 [==============================] - 16s 19ms/step - loss: 7.9361 - mae:
2.1987 - val_loss: 10.4670 - val_mae: 2.4955
Epoch 5/10
819/819 [==============================] - 15s 19ms/step - loss: 7.5218 - mae:
2.1399 - val_loss: 10.5808 - val_mae: 2.5170
Epoch 6/10
819/819 [==============================] - 15s 18ms/step - loss: 7.2509 - mae:
2.0977 - val_loss: 10.7009 - val_mae: 2.5293
Epoch 7/10
819/819 [==============================] - 15s 19ms/step - loss: 6.9774 - mae:
2.0532 - val_loss: 10.7251 - val_mae: 2.5476
Epoch 8/10
819/819 [==============================] - 16s 19ms/step - loss: 6.7403 - mae:
2.0184 - val_loss: 11.3408 - val_mae: 2.6101
```

```
Epoch 9/10
819/819 [==============================] - 15s 19ms/step - loss: 6.5583 - mae:
1.9922 - val_loss: 11.2898 - val_mae: 2.6139
Epoch 10/10
819/819 [==============================] - 15s 18ms/step - loss: 6.3602 - mae:
1.9624 - val_loss: 11.4001 - val_mae: 2.6332
405/405 [==============================] - 4s 11ms/step - loss: 12.4068 - mae:
2.7788
Test MAE: 2.78
```



4 3. Using a combination of 1d_convnets and RNN.

- A conv 1D stacked with RNN LSTM

[23]:
```
inputs = keras.Input(shape=(sequence_length, raw_data.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.LSTM(32)(x)
x = layers.Dropout(0.6)(x)
```

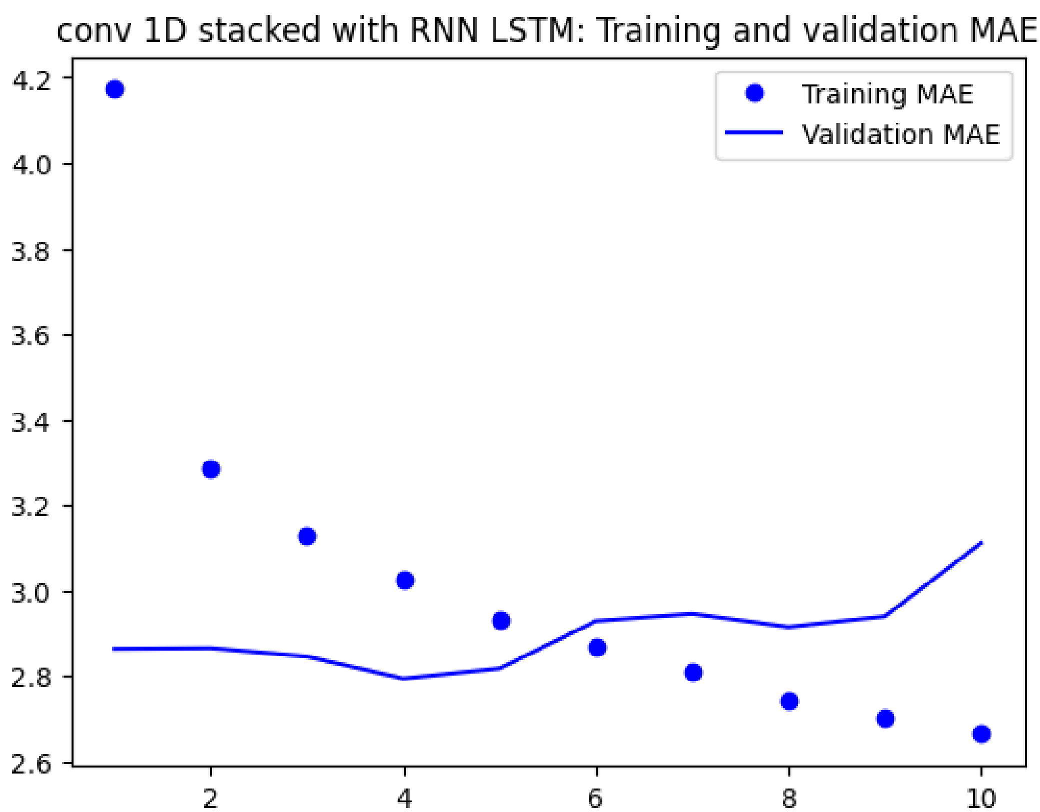
```
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm_conv_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)
```

```
Epoch 1/10
819/819 [==============================] - 15s 16ms/step - loss: 31.0849 - mae:
4.1731 - val_loss: 13.5187 - val_mae: 2.8632
Epoch 2/10
819/819 [==============================] - 13s 15ms/step - loss: 18.0567 - mae:
3.2865 - val_loss: 13.2226 - val_mae: 2.8647
Epoch 3/10
819/819 [==============================] - 13s 16ms/step - loss: 16.4051 - mae:
3.1280 - val_loss: 13.0865 - val_mae: 2.8453
Epoch 4/10
819/819 [==============================] - 13s 15ms/step - loss: 15.3971 - mae:
3.0244 - val_loss: 12.5535 - val_mae: 2.7937
Epoch 5/10
819/819 [==============================] - 13s 16ms/step - loss: 14.5094 - mae:
2.9334 - val_loss: 12.7918 - val_mae: 2.8172
Epoch 6/10
819/819 [==============================] - 13s 16ms/step - loss: 13.9335 - mae:
2.8688 - val_loss: 14.0951 - val_mae: 2.9281
Epoch 7/10
819/819 [==============================] - 13s 16ms/step - loss: 13.4136 - mae:
2.8093 - val_loss: 13.9580 - val_mae: 2.9451
Epoch 8/10
819/819 [==============================] - 13s 16ms/step - loss: 12.8030 - mae:
2.7429 - val_loss: 13.7579 - val_mae: 2.9143
Epoch 9/10
819/819 [==============================] - 13s 15ms/step - loss: 12.4418 - mae:
2.7003 - val_loss: 14.0394 - val_mae: 2.9389
Epoch 10/10
819/819 [==============================] - 13s 16ms/step - loss: 12.0993 - mae:
2.6641 - val_loss: 15.7882 - val_mae: 3.1108
```

[24]:
```
model = keras.models.load_model("jena_lstm_conv_dropout.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
loss = history.history["mae"]
```

```
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("conv 1D stacked with RNN LSTM: Training and validation MAE")
plt.legend()
plt.show()
```

```
405/405 [==============================] - 5s 9ms/step - loss: 13.5350 - mae:
2.9269
Test MAE: 2.93
```



With combination of conv1d and RNN lstm, the model got worsened with test MAE 2.93.