Assignment 3 RNN

April 9, 2024

1 Assignment 3: Time-Series Data

1.1 Group 3 - Members - Gaurav Kudeshia & Anurodh Singh

1.2 Date: 04-07-2024

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

```
--2024-04-08 22:54:05--  https://s3.amazonaws.com/keras-
datasets/jena_climate_2009_2016.csv.zip
Resolving s3.amazonaws.com (s3.amazonaws.com)… 52.217.11.174, 16.182.74.128,
52.217.227.48, …
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.11.174|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 13565642 (13M) [application/zip]
Saving to: 'jena_climate_2009_2016.csv.zip.10'

100%[======================================>] 13,565,642  42.6MB/s   in 0.3s

2024-04-08 22:54:05 (42.6 MB/s) - 'jena_climate_2009_2016.csv.zip.10' saved
[13565642/13565642]
```

The study leverages the Jena Climate dataset, which comprises over 420,000 data points, each capturing 15 distinct weather-related features

[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:]
print(header)
print(len(lines))
```

```
num_variables = len(header)
print("Number of variables:", num_variables)
num_rows = len(lines)
print("Number of rows:", num_rows)
```

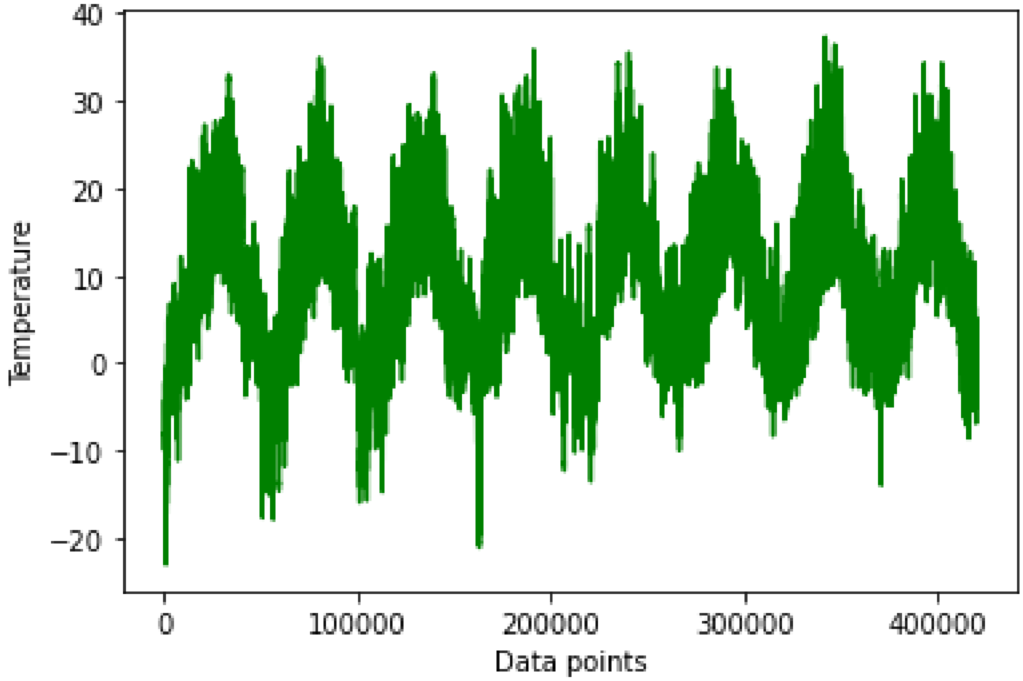
```
['"Date Time"', '"p (mbar)"', '"T (degC)"', '"Tpot (K)"', '"Tdew (degC)"', '"rh
(%)"', '"VPmax (mbar)"', '"VPact (mbar)"', '"VPdef (mbar)"', '"sh (g/kg)"',
'"H2OC (mmol/mol)"', '"rho (g/m**3)"', '"wv (m/s)"', '"max. wv (m/s)"', '"wd
(deg)"']
420451
Number of variables: 15
Number of rows: 420451
```

Processing the data

[3]:
```
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

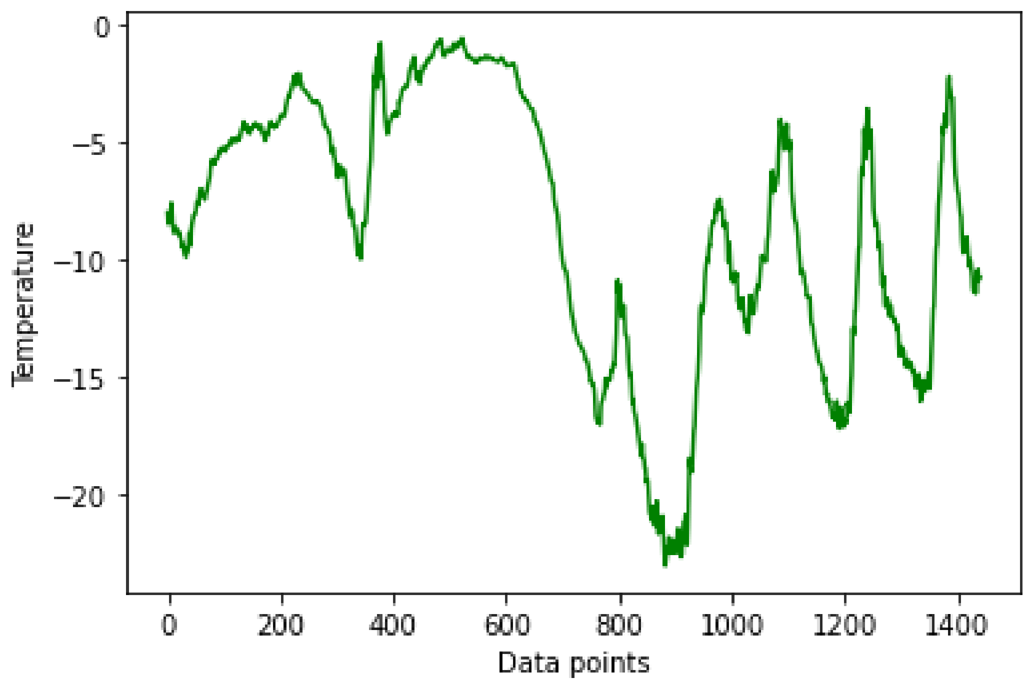
[4]:
```
from matplotlib import pyplot as plt
plt.plot(range(len(temperature)), temperature, color='green')
plt.xlabel('Data points')
plt.ylabel('Temperature')
plt.show()
```



When plotting a temperature timeseries for the first 10 days, each day contributes 144 data points, resulting in a total of 1440 data points. This extensive dataset allows for detailed analysis and visualization of temperature trends over this period.

[5]:
```
plt.plot(range(1440), temperature[:1440], color='green')
plt.xlabel('Data points')
plt.ylabel('Temperature')
```

[5]: Text(0, 0.5, 'Temperature')



Determining the allocation of our dataset samples: 50% will be dedicated to training, ensuring a robust learning foundation. An additional 25% is reserved for validation, enabling rigorous testing and optimization of model performance, aiming for balance and efficiency.

[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
```

Data normalization is essential for numerical datasets, eliminating the need for vectorization. This process ensures uniformity, improves algorithm efficiency, enhances model accuracy, and facilitates faster convergence in machine learning tasks. It streamlines data analysis by bringing different scales to a common range

[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
```

[8]:
```python
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
```

Created training, validation, and testing datasets is crucial due to the high redundancy in the samples. Allocating memory for each sample is inefficient. Therefore, we dynamically generate samples, enhancing memory utilization and computational efficiency. This approach also allows for more flexible dataset management and scalability.

[9]:
```python
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)
```

Reviewing our dataset's output revealed key insights, indicated trends, showcased data consistency, and identified areas for improvement, ensuring reliability and guiding future data collection and analysis strategies.

[10]:
```
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,)
```

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

The "evaluate__naive__method" function sets a baseline for simple forecast models by predicting the next value based on the last input sequence value, aiding in performance assessment and comparison.

[11]:
```
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
```

Used a simple baseline method where future temperature is presumed equal to the current temperature, the validation and test Mean Absolute Error (MAE) are 2.44 and 2.62 degrees Celsius, respectively. This method, though straightforward, demonstrates a noteworthy predictive accuracy, indicating an average error margin of about 2.5 degrees, suggesting its utility for initial forecasting models or when complex data is unavailable.

1.2.2 A basic machine-learning model - Dense Layer

Training and evaluating a densely connected model

[12]:
```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(16, 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=30,
                    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 1/30
819/819 [==============================] - 11s 13ms/step - loss: 12.3729 - mae:
2.7216 - val_loss: 10.4065 - val_mae: 2.5519
Epoch 2/30
819/819 [==============================] - 10s 12ms/step - loss: 9.0209 - mae:
2.3609 - val_loss: 13.4101 - val_mae: 2.8971
Epoch 3/30
819/819 [==============================] - 10s 12ms/step - loss: 8.2655 - mae:
2.2619 - val_loss: 10.5275 - val_mae: 2.5691
Epoch 4/30
819/819 [==============================] - 10s 12ms/step - loss: 7.7936 - mae:
2.1969 - val_loss: 10.9505 - val_mae: 2.6133
Epoch 5/30
819/819 [==============================] - 10s 12ms/step - loss: 7.4816 - mae:
2.1544 - val_loss: 10.4495 - val_mae: 2.5515
Epoch 6/30
819/819 [==============================] - 10s 12ms/step - loss: 7.2376 - mae:
2.1185 - val_loss: 10.9362 - val_mae: 2.6168
Epoch 7/30
```

```
819/819 [==============================] - 9s 11ms/step - loss: 7.0303 - mae:
2.0885 - val_loss: 10.8968 - val_mae: 2.6033
Epoch 8/30
819/819 [==============================] - 10s 13ms/step - loss: 6.8960 - mae:
2.0701 - val_loss: 13.5724 - val_mae: 2.9132
Epoch 9/30
819/819 [==============================] - 10s 12ms/step - loss: 6.7522 - mae:
2.0482 - val_loss: 10.8489 - val_mae: 2.6010
Epoch 10/30
819/819 [==============================] - 10s 13ms/step - loss: 6.6443 - mae:
2.0309 - val_loss: 11.3066 - val_mae: 2.6547
Epoch 11/30
819/819 [==============================] - 9s 11ms/step - loss: 6.5437 - mae:
2.0156 - val_loss: 11.4976 - val_mae: 2.6884
Epoch 12/30
819/819 [==============================] - 10s 12ms/step - loss: 6.4623 - mae:
2.0042 - val_loss: 12.2063 - val_mae: 2.7603
Epoch 13/30
819/819 [==============================] - 10s 13ms/step - loss: 6.3861 - mae:
1.9912 - val_loss: 11.0990 - val_mae: 2.6323
Epoch 14/30
819/819 [==============================] - 9s 11ms/step - loss: 6.3129 - mae:
1.9811 - val_loss: 11.1253 - val_mae: 2.6375
Epoch 15/30
819/819 [==============================] - 10s 12ms/step - loss: 6.2378 - mae:
1.9706 - val_loss: 11.1658 - val_mae: 2.6379
Epoch 16/30
819/819 [==============================] - 9s 11ms/step - loss: 6.1769 - mae:
1.9603 - val_loss: 11.7137 - val_mae: 2.7016
Epoch 17/30
819/819 [==============================] - 10s 12ms/step - loss: 6.1187 - mae:
1.9500 - val_loss: 11.4597 - val_mae: 2.6761
Epoch 18/30
819/819 [==============================] - 10s 12ms/step - loss: 6.0824 - mae:
1.9446 - val_loss: 11.9534 - val_mae: 2.7393
Epoch 19/30
819/819 [==============================] - 9s 11ms/step - loss: 6.0227 - mae:
1.9369 - val_loss: 11.2479 - val_mae: 2.6525
Epoch 20/30
819/819 [==============================] - 9s 11ms/step - loss: 5.9698 - mae:
1.9279 - val_loss: 11.3687 - val_mae: 2.6634
Epoch 21/30
819/819 [==============================] - 10s 12ms/step - loss: 5.9248 - mae:
1.9204 - val_loss: 11.5728 - val_mae: 2.6916
Epoch 22/30
819/819 [==============================] - 10s 12ms/step - loss: 5.8994 - mae:
1.9162 - val_loss: 12.3158 - val_mae: 2.7773
Epoch 23/30
```

```
819/819 [==============================] - 10s 12ms/step - loss: 5.8554 - mae:
1.9111 - val_loss: 11.5504 - val_mae: 2.6841
Epoch 24/30
819/819 [==============================] - 9s 11ms/step - loss: 5.8154 - mae:
1.9036 - val_loss: 12.4807 - val_mae: 2.7958
Epoch 25/30
819/819 [==============================] - 10s 12ms/step - loss: 5.8054 - mae:
1.9016 - val_loss: 11.5668 - val_mae: 2.6864
Epoch 26/30
819/819 [==============================] - 10s 12ms/step - loss: 5.7430 - mae:
1.8923 - val_loss: 11.7283 - val_mae: 2.7107
Epoch 27/30
819/819 [==============================] - 10s 12ms/step - loss: 5.7213 - mae:
1.8875 - val_loss: 11.6114 - val_mae: 2.6949
Epoch 28/30
819/819 [==============================] - 9s 11ms/step - loss: 5.7042 - mae:
1.8860 - val_loss: 12.9244 - val_mae: 2.8538
Epoch 29/30
819/819 [==============================] - 10s 12ms/step - loss: 5.6744 - mae:
1.8790 - val_loss: 12.3332 - val_mae: 2.7786
Epoch 30/30
819/819 [==============================] - 10s 12ms/step - loss: 5.6522 - mae:
1.8759 - val_loss: 12.5454 - val_mae: 2.8089
405/405 [==============================] - 3s 7ms/step - loss: 11.3753 - mae:
2.6472
Test MAE: 2.65
```

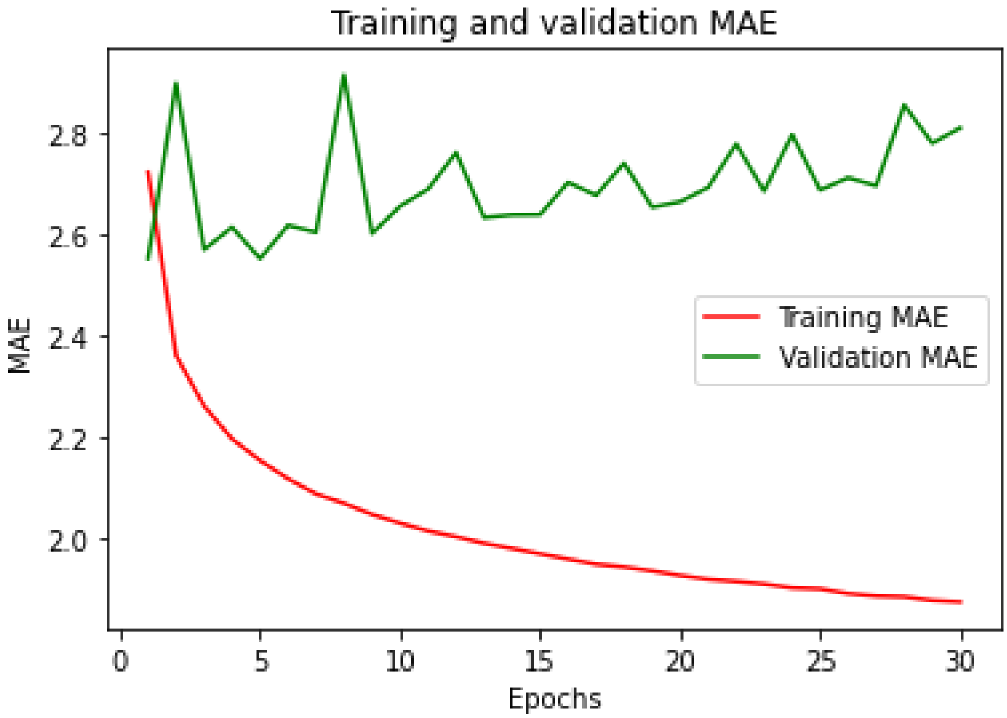
[13]:
```python
model = keras.models.load_model("jena_dense.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
405/405 [==============================] - 4s 8ms/step - loss: 11.3753 - mae:
2.6472
Test MAE: 2.65
```

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, color="red", linestyle="solid", label="Training MAE")
plt.plot(epochs, val_loss, color="green", linestyle="solid", label="Validation␣
 ↪MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
```

```
plt.show()
```



1.2.3 Let's experiment with 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=20,
                    validation_data=val_dataset,
                    callbacks=callbacks)
```

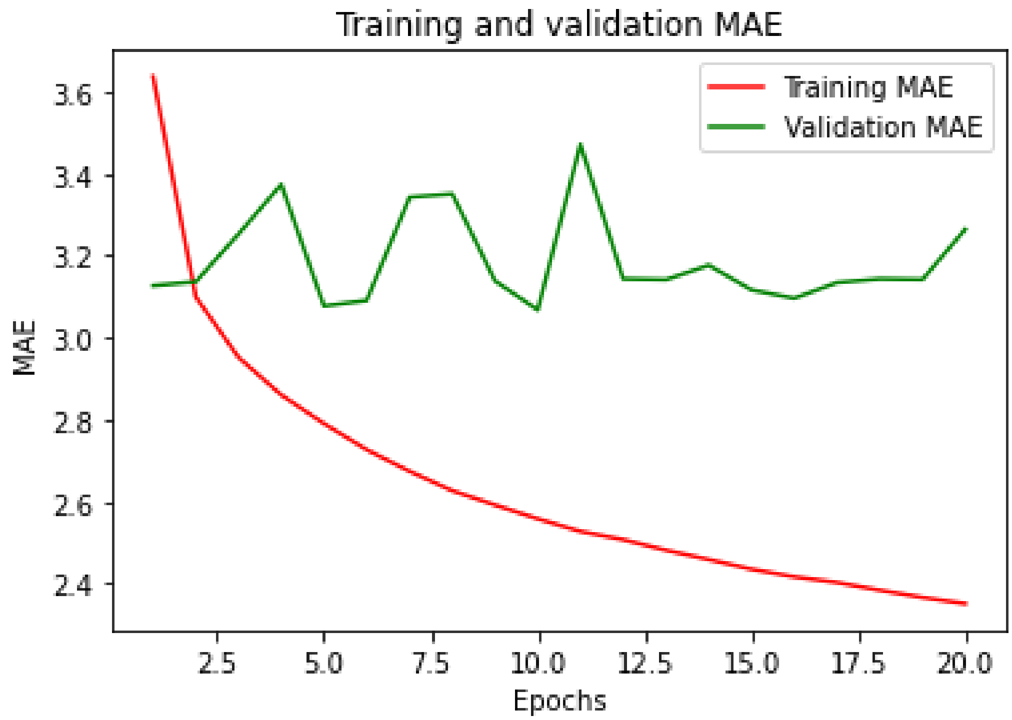
```python
model = keras.models.load_model("jena_conv.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/20
819/819 [==============================] - 26s 31ms/step - loss: 21.2633 - mae:
3.6372 - val_loss: 15.3470 - val_mae: 3.1274
Epoch 2/20
819/819 [==============================] - 25s 30ms/step - loss: 15.1568 - mae:
3.0997 - val_loss: 15.6724 - val_mae: 3.1370
Epoch 3/20
819/819 [==============================] - 24s 29ms/step - loss: 13.8019 - mae:
2.9523 - val_loss: 16.9867 - val_mae: 3.2529
Epoch 4/20
819/819 [==============================] - 24s 30ms/step - loss: 13.0037 - mae:
2.8607 - val_loss: 18.0498 - val_mae: 3.3741
Epoch 5/20
819/819 [==============================] - 23s 28ms/step - loss: 12.4018 - mae:
2.7906 - val_loss: 15.1613 - val_mae: 3.0777
Epoch 6/20
819/819 [==============================] - 25s 30ms/step - loss: 11.8691 - mae:
2.7270 - val_loss: 15.2491 - val_mae: 3.0912
Epoch 7/20
819/819 [==============================] - 24s 29ms/step - loss: 11.4292 - mae:
2.6746 - val_loss: 17.8043 - val_mae: 3.3428
Epoch 8/20
819/819 [==============================] - 24s 29ms/step - loss: 11.0452 - mae:
2.6272 - val_loss: 17.9853 - val_mae: 3.3509
Epoch 9/20
819/819 [==============================] - 24s 29ms/step - loss: 10.7494 - mae:
2.5923 - val_loss: 15.8565 - val_mae: 3.1390
Epoch 10/20
819/819 [==============================] - 23s 27ms/step - loss: 10.4797 - mae:
2.5587 - val_loss: 15.1883 - val_mae: 3.0681
Epoch 11/20
819/819 [==============================] - 24s 29ms/step - loss: 10.2268 - mae:
2.5278 - val_loss: 19.2901 - val_mae: 3.4719
Epoch 12/20
819/819 [==============================] - 23s 28ms/step - loss: 10.0517 - mae:
2.5077 - val_loss: 15.9613 - val_mae: 3.1439
Epoch 13/20
819/819 [==============================] - 24s 29ms/step - loss: 9.8513 - mae:
2.4819 - val_loss: 15.9297 - val_mae: 3.1420
Epoch 14/20
819/819 [==============================] - 23s 29ms/step - loss: 9.6506 - mae:
2.4591 - val_loss: 16.3516 - val_mae: 3.1771
Epoch 15/20
```

```
819/819 [==============================] - 24s 29ms/step - loss: 9.4759 - mae:
2.4356 - val_loss: 15.7939 - val_mae: 3.1170
Epoch 16/20
819/819 [==============================] - 23s 28ms/step - loss: 9.3215 - mae:
2.4170 - val_loss: 15.6826 - val_mae: 3.0965
Epoch 17/20
819/819 [==============================] - 24s 29ms/step - loss: 9.2122 - mae:
2.4032 - val_loss: 15.9202 - val_mae: 3.1348
Epoch 18/20
819/819 [==============================] - 24s 29ms/step - loss: 9.0721 - mae:
2.3842 - val_loss: 16.0635 - val_mae: 3.1439
Epoch 19/20
819/819 [==============================] - 24s 29ms/step - loss: 8.9354 - mae:
2.3666 - val_loss: 15.9977 - val_mae: 3.1423
Epoch 20/20
819/819 [==============================] - 24s 29ms/step - loss: 8.8290 - mae:
2.3515 - val_loss: 17.3839 - val_mae: 3.2645
405/405 [==============================] - 5s 11ms/step - loss: 16.5197 - mae:
3.2062
Test MAE: 3.21
```

[16]:

```
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, color="red", linestyle="solid", label="Training MAE")
plt.plot(epochs, val_loss, color="green", linestyle="solid", label="Validation␣
↪MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



Convolutional models underperform for weather predictions as they struggle with non-translation-invariant data like weather patterns. Additionally, their inability to prioritize recent data significantly hinders accurate temperature forecasting. Unlike dense models, 1D CNNs fail to grasp essential temporal sequences, crucial for predicting short-term weather changes effectively, leading to inferior results.

1.3 A Simple RNN

1.3.1 An RNN layer that can process sequences of any length

[38]:
```
num_features = 14
inputs = keras.Input(shape=(None, num_features))
outputs = layers.SimpleRNN(16)(inputs)

model = keras.Model(inputs, outputs)

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

```python
                     callbacks=callbacks)

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

```
Epoch 1/10
819/819 [==============================] - 18s 21ms/step - loss: 138.4891 - mae:
9.6677 - val_loss: 143.9164 - val_mae: 9.8833
Epoch 2/10
819/819 [==============================] - 18s 22ms/step - loss: 136.3632 - mae:
9.5542 - val_loss: 143.6891 - val_mae: 9.8626
Epoch 3/10
819/819 [==============================] - 17s 21ms/step - loss: 136.2844 - mae:
9.5472 - val_loss: 143.6355 - val_mae: 9.8585
Epoch 4/10
819/819 [==============================] - 19s 23ms/step - loss: 136.2434 - mae:
9.5443 - val_loss: 143.6126 - val_mae: 9.8582
Epoch 5/10
819/819 [==============================] - 18s 22ms/step - loss: 136.2070 - mae:
9.5419 - val_loss: 143.5994 - val_mae: 9.8552
Epoch 6/10
819/819 [==============================] - 18s 22ms/step - loss: 136.1809 - mae:
9.5384 - val_loss: 143.5842 - val_mae: 9.8526
Epoch 7/10
819/819 [==============================] - 18s 22ms/step - loss: 136.1483 - mae:
9.5355 - val_loss: 143.5832 - val_mae: 9.8572
Epoch 8/10
819/819 [==============================] - 17s 21ms/step - loss: 136.1323 - mae:
9.5338 - val_loss: 143.5513 - val_mae: 9.8513
Epoch 9/10
819/819 [==============================] - 16s 19ms/step - loss: 136.1149 - mae:
9.5322 - val_loss: 143.5497 - val_mae: 9.8498
Epoch 10/10
819/819 [==============================] - 16s 19ms/step - loss: 136.1085 - mae:
9.5308 - val_loss: 143.5481 - val_mae: 9.8528
405/405 [==============================] - 4s 10ms/step - loss: 151.2877 - mae:
9.9194
Test MAE: 9.92
```

1.3.2 Stacking RNN layers enhances model depth and learning capacity, allowing for more complex data patterns to be captured. This method involves placing multiple RNN layers on top of each other, with each layer's output serving as the input for the next, thereby improving the network's ability to learn from data with long-term dependencies.

[17]:
```python
num_features = 14
steps = 120
inputs = keras.Input(shape=(steps, num_features))
x = layers.SimpleRNN(16, return_sequences=True)(inputs)
x = layers.SimpleRNN(16, return_sequences=True)(x)
outputs = layers.SimpleRNN(16)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_SRNN2.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=40,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_SRNN2.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/40
819/819 [==============================] - 56s 66ms/step - loss: 136.9822 - mae:
9.5763 - val_loss: 143.4180 - val_mae: 9.8338
Epoch 2/40
819/819 [==============================] - 54s 66ms/step - loss: 135.9801 - mae:
9.5181 - val_loss: 143.4305 - val_mae: 9.8371
Epoch 3/40
819/819 [==============================] - 51s 63ms/step - loss: 135.9546 - mae:
9.5133 - val_loss: 143.3928 - val_mae: 9.8310
Epoch 4/40
819/819 [==============================] - 51s 63ms/step - loss: 135.9235 - mae:
9.5088 - val_loss: 143.4037 - val_mae: 9.8352
Epoch 5/40
819/819 [==============================] - 51s 62ms/step - loss: 135.9237 - mae:
9.5077 - val_loss: 143.4266 - val_mae: 9.8368
Epoch 6/40
819/819 [==============================] - 51s 63ms/step - loss: 135.8961 - mae:
9.5041 - val_loss: 143.4307 - val_mae: 9.8374
Epoch 7/40
819/819 [==============================] - 53s 64ms/step - loss: 135.8733 - mae:
9.5004 - val_loss: 143.4155 - val_mae: 9.8351
```

```
Epoch 8/40
819/819 [==============================] - 52s 64ms/step - loss: 135.8650 - mae:
9.4990 - val_loss: 143.4593 - val_mae: 9.8409
Epoch 9/40
819/819 [==============================] - 51s 63ms/step - loss: 135.8499 - mae:
9.4967 - val_loss: 143.4382 - val_mae: 9.8384
Epoch 10/40
819/819 [==============================] - 52s 63ms/step - loss: 135.8482 - mae:
9.4962 - val_loss: 143.4309 - val_mae: 9.8375
Epoch 11/40
819/819 [==============================] - 49s 60ms/step - loss: 135.8481 - mae:
9.4964 - val_loss: 143.4476 - val_mae: 9.8397
Epoch 12/40
819/819 [==============================] - 52s 64ms/step - loss: 135.8408 - mae:
9.4950 - val_loss: 143.5978 - val_mae: 9.8592
Epoch 13/40
819/819 [==============================] - 55s 67ms/step - loss: 135.8320 - mae:
9.4941 - val_loss: 143.4532 - val_mae: 9.8387
Epoch 14/40
819/819 [==============================] - 54s 66ms/step - loss: 135.8175 - mae:
9.4920 - val_loss: 143.4548 - val_mae: 9.8419
Epoch 15/40
819/819 [==============================] - 54s 66ms/step - loss: 135.8243 - mae:
9.4928 - val_loss: 143.4483 - val_mae: 9.8395
Epoch 16/40
819/819 [==============================] - 55s 66ms/step - loss: 135.8089 - mae:
9.4908 - val_loss: 143.4418 - val_mae: 9.8387
Epoch 17/40
819/819 [==============================] - 55s 67ms/step - loss: 135.7935 - mae:
9.4882 - val_loss: 143.4572 - val_mae: 9.8376
Epoch 18/40
819/819 [==============================] - 51s 63ms/step - loss: 135.8004 - mae:
9.4892 - val_loss: 143.4356 - val_mae: 9.8384
Epoch 19/40
819/819 [==============================] - 51s 62ms/step - loss: 135.7951 - mae:
9.4883 - val_loss: 143.4390 - val_mae: 9.8377
Epoch 20/40
819/819 [==============================] - 53s 65ms/step - loss: 135.7812 - mae:
9.4859 - val_loss: 143.4339 - val_mae: 9.8383
Epoch 21/40
819/819 [==============================] - 52s 64ms/step - loss: 135.7886 - mae:
9.4873 - val_loss: 143.4608 - val_mae: 9.8448
Epoch 22/40
819/819 [==============================] - 49s 60ms/step - loss: 135.7743 - mae:
9.4855 - val_loss: 143.4287 - val_mae: 9.8363
Epoch 23/40
819/819 [==============================] - 50s 62ms/step - loss: 135.7749 - mae:
9.4856 - val_loss: 143.4482 - val_mae: 9.8371
```

```
Epoch 24/40
819/819 [==============================] - 51s 63ms/step - loss: 135.7755 - mae:
9.4857 - val_loss: 143.4340 - val_mae: 9.8360
Epoch 25/40
819/819 [==============================] - 50s 61ms/step - loss: 135.7661 - mae:
9.4837 - val_loss: 143.4787 - val_mae: 9.8437
Epoch 26/40
819/819 [==============================] - 50s 61ms/step - loss: 135.7626 - mae:
9.4831 - val_loss: 143.4123 - val_mae: 9.8348
Epoch 27/40
819/819 [==============================] - 51s 62ms/step - loss: 135.7595 - mae:
9.4825 - val_loss: 143.4423 - val_mae: 9.8402
Epoch 28/40
819/819 [==============================] - 52s 63ms/step - loss: 135.7639 - mae:
9.4824 - val_loss: 143.4260 - val_mae: 9.8348
Epoch 29/40
819/819 [==============================] - 52s 63ms/step - loss: 135.7579 - mae:
9.4816 - val_loss: 143.4276 - val_mae: 9.8368
Epoch 30/40
819/819 [==============================] - 49s 60ms/step - loss: 135.7439 - mae:
9.4798 - val_loss: 143.4371 - val_mae: 9.8370
Epoch 31/40
819/819 [==============================] - 50s 61ms/step - loss: 135.7532 - mae:
9.4808 - val_loss: 143.4422 - val_mae: 9.8397
Epoch 32/40
819/819 [==============================] - 44s 54ms/step - loss: 135.7509 - mae:
9.4805 - val_loss: 143.4735 - val_mae: 9.8461
Epoch 33/40
819/819 [==============================] - 49s 59ms/step - loss: 135.7408 - mae:
9.4793 - val_loss: 143.4489 - val_mae: 9.8391
Epoch 34/40
819/819 [==============================] - 47s 58ms/step - loss: 135.7446 - mae:
9.4792 - val_loss: 143.4908 - val_mae: 9.8461
Epoch 35/40
819/819 [==============================] - 51s 62ms/step - loss: 135.7469 - mae:
9.4793 - val_loss: 143.4392 - val_mae: 9.8393
Epoch 36/40
819/819 [==============================] - 47s 57ms/step - loss: 135.7373 - mae:
9.4777 - val_loss: 143.4908 - val_mae: 9.8465
Epoch 37/40
819/819 [==============================] - 49s 60ms/step - loss: 135.7389 - mae:
9.4779 - val_loss: 143.4745 - val_mae: 9.8419
Epoch 38/40
819/819 [==============================] - 49s 60ms/step - loss: 135.7384 - mae:
9.4777 - val_loss: 143.4525 - val_mae: 9.8411
Epoch 39/40
819/819 [==============================] - 47s 57ms/step - loss: 135.7329 - mae:
9.4768 - val_loss: 143.4516 - val_mae: 9.8382
```

```
Epoch 40/40
819/819 [==============================] - 46s 56ms/step - loss: 135.7367 - mae:
9.4770 - val_loss: 143.4478 - val_mae: 9.8404
405/405 [==============================] - 6s 14ms/step - loss: 151.1607 - mae:
9.9069
Test MAE: 9.91
```

1.4 The GRU, a streamlined version of LSTM, efficiently handles sequences for tasks like speech recognition. It simplifies training by blending forget and input gates into a single update gate, enhancing model performance with fewer parameters.

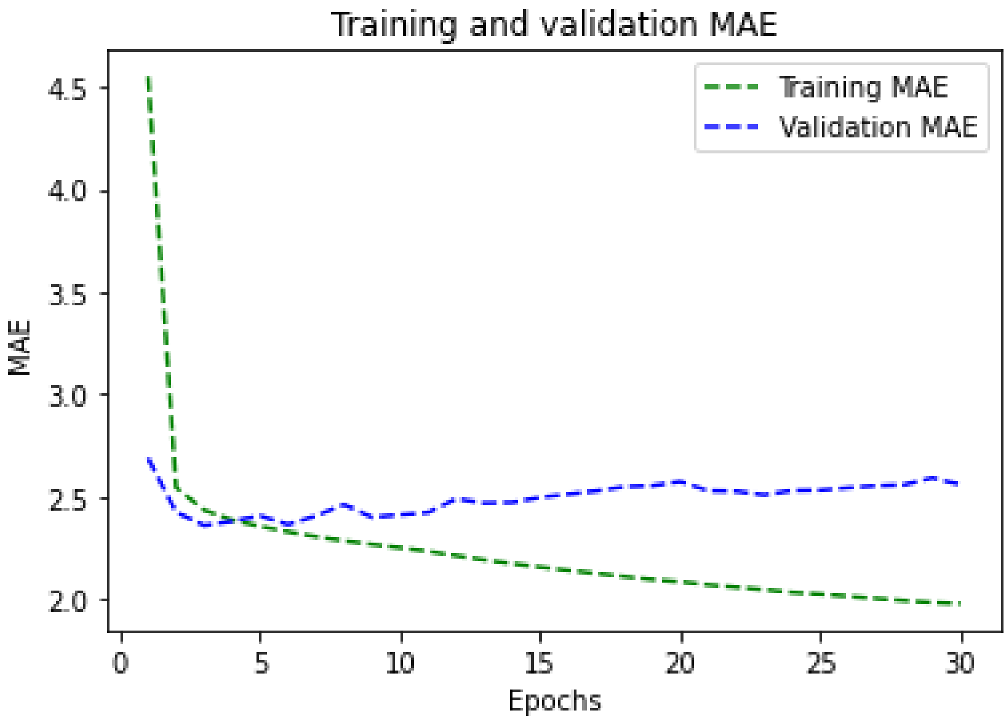
[19]:
```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.GRU(16)(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_gru.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=30,
                    validation_data=val_dataset,
                    callbacks=callbacks)

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

```
Epoch 1/30
819/819 [==============================] - 39s 46ms/step - loss: 39.1661 - mae:
4.5509 - val_loss: 12.6680 - val_mae: 2.6873
Epoch 2/30
819/819 [==============================] - 37s 46ms/step - loss: 10.6713 - mae:
2.5445 - val_loss: 9.9131 - val_mae: 2.4240
Epoch 3/30
819/819 [==============================] - 34s 42ms/step - loss: 9.6828 - mae:
2.4345 - val_loss: 9.3459 - val_mae: 2.3593
Epoch 4/30
819/819 [==============================] - 37s 45ms/step - loss: 9.3208 - mae:
2.3857 - val_loss: 9.6408 - val_mae: 2.3802
Epoch 5/30
819/819 [==============================] - 34s 41ms/step - loss: 9.0877 - mae:
2.3553 - val_loss: 9.8975 - val_mae: 2.4063
Epoch 6/30
819/819 [==============================] - 38s 47ms/step - loss: 8.8860 - mae:
2.3283 - val_loss: 9.4372 - val_mae: 2.3637
```

```
Epoch 7/30
819/819 [==============================] - 35s 43ms/step - loss: 8.7000 - mae:
2.3047 - val_loss: 9.8557 - val_mae: 2.4062
Epoch 8/30
819/819 [==============================] - 35s 42ms/step - loss: 8.5249 - mae:
2.2843 - val_loss: 10.4197 - val_mae: 2.4618
Epoch 9/30
819/819 [==============================] - 36s 44ms/step - loss: 8.3723 - mae:
2.2669 - val_loss: 9.7597 - val_mae: 2.3992
Epoch 10/30
819/819 [==============================] - 36s 44ms/step - loss: 8.2310 - mae:
2.2503 - val_loss: 9.9118 - val_mae: 2.4113
Epoch 11/30
819/819 [==============================] - 41s 50ms/step - loss: 8.0820 - mae:
2.2331 - val_loss: 9.9359 - val_mae: 2.4226
Epoch 12/30
819/819 [==============================] - 40s 49ms/step - loss: 7.9184 - mae:
2.2115 - val_loss: 10.6653 - val_mae: 2.4896
Epoch 13/30
819/819 [==============================] - 39s 47ms/step - loss: 7.7728 - mae:
2.1912 - val_loss: 10.4094 - val_mae: 2.4688
Epoch 14/30
819/819 [==============================] - 36s 44ms/step - loss: 7.6478 - mae:
2.1729 - val_loss: 10.2348 - val_mae: 2.4713
Epoch 15/30
819/819 [==============================] - 34s 41ms/step - loss: 7.5306 - mae:
2.1562 - val_loss: 10.7848 - val_mae: 2.4966
Epoch 16/30
819/819 [==============================] - 33s 41ms/step - loss: 7.4148 - mae:
2.1400 - val_loss: 10.9982 - val_mae: 2.5119
Epoch 17/30
819/819 [==============================] - 33s 40ms/step - loss: 7.3086 - mae:
2.1249 - val_loss: 11.3207 - val_mae: 2.5278
Epoch 18/30
819/819 [==============================] - 37s 45ms/step - loss: 7.2070 - mae:
2.1101 - val_loss: 11.4810 - val_mae: 2.5484
Epoch 19/30
819/819 [==============================] - 35s 43ms/step - loss: 7.1101 - mae:
2.0950 - val_loss: 11.5724 - val_mae: 2.5534
Epoch 20/30
819/819 [==============================] - 34s 42ms/step - loss: 7.0291 - mae:
2.0840 - val_loss: 11.8174 - val_mae: 2.5738
Epoch 21/30
819/819 [==============================] - 33s 40ms/step - loss: 6.9421 - mae:
2.0696 - val_loss: 11.1486 - val_mae: 2.5289
Epoch 22/30
819/819 [==============================] - 35s 43ms/step - loss: 6.8576 - mae:
2.0576 - val_loss: 11.2917 - val_mae: 2.5262
```

```
Epoch 23/30
819/819 [==============================] - 31s 38ms/step - loss: 6.7783 - mae:
2.0456 - val_loss: 11.0894 - val_mae: 2.5085
Epoch 24/30
819/819 [==============================] - 34s 42ms/step - loss: 6.6897 - mae:
2.0323 - val_loss: 11.3618 - val_mae: 2.5298
Epoch 25/30
819/819 [==============================] - 32s 39ms/step - loss: 6.6299 - mae:
2.0238 - val_loss: 11.2273 - val_mae: 2.5318
Epoch 26/30
819/819 [==============================] - 34s 42ms/step - loss: 6.5583 - mae:
2.0133 - val_loss: 11.3817 - val_mae: 2.5435
Epoch 27/30
819/819 [==============================] - 35s 42ms/step - loss: 6.4882 - mae:
2.0025 - val_loss: 11.5728 - val_mae: 2.5529
Epoch 28/30
819/819 [==============================] - 36s 44ms/step - loss: 6.4262 - mae:
1.9925 - val_loss: 11.6671 - val_mae: 2.5581
Epoch 29/30
819/819 [==============================] - 34s 42ms/step - loss: 6.3675 - mae:
1.9832 - val_loss: 12.0826 - val_mae: 2.5915
Epoch 30/30
819/819 [==============================] - 37s 45ms/step - loss: 6.3223 - mae:
1.9774 - val_loss: 11.5193 - val_mae: 2.5596
405/405 [==============================] - 5s 12ms/step - loss: 10.0938 - mae:
2.4951
Test MAE: 2.50
```

[21]:
```
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, color="green", linestyle="dashed", label="Training MAE")
plt.plot(epochs,  val_loss, color="blue",linestyle="dashed", label="Validation␣
↪MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



1.5 LSTM(Long Short-Term Memory)

1.5.1 LSTM-Simple

[22]:
```
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=30,
                    validation_data=val_dataset,
                    callbacks=callbacks)

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

```
Epoch 1/30
819/819 [==============================] - 40s 47ms/step - loss: 41.9183 - mae:
```

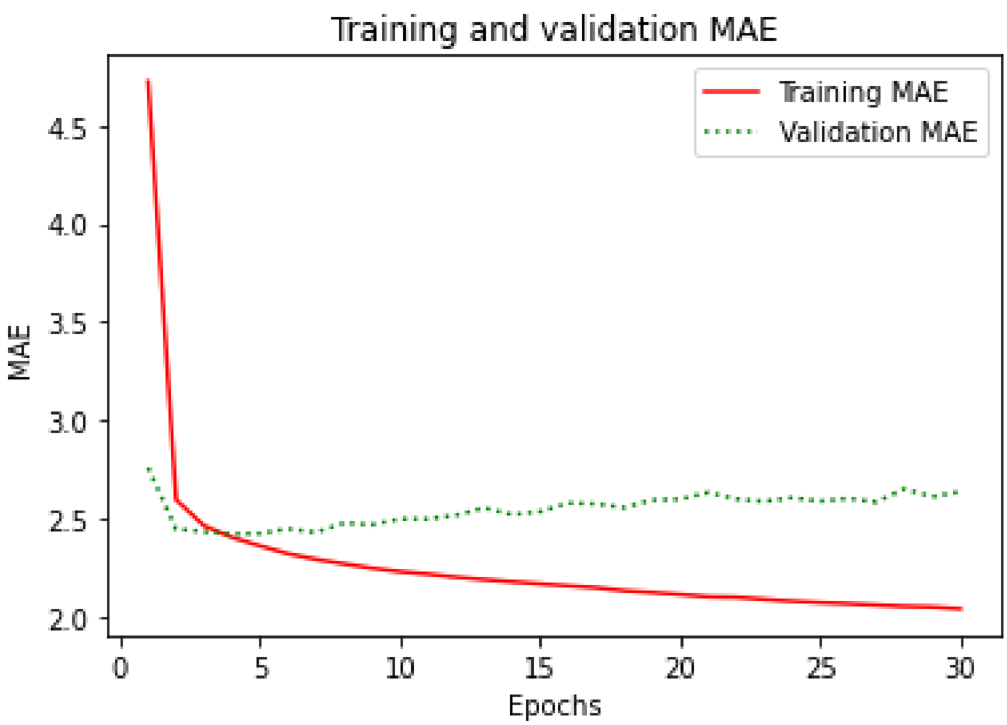
```
4.7232 - val_loss: 12.9314 - val_mae: 2.7574
Epoch 2/30
819/819 [==============================] - 38s 46ms/step - loss: 11.1444 - mae:
2.5952 - val_loss: 9.8891 - val_mae: 2.4461
Epoch 3/30
819/819 [==============================] - 31s 38ms/step - loss: 9.9398 - mae:
2.4626 - val_loss: 9.8411 - val_mae: 2.4330
Epoch 4/30
819/819 [==============================] - 37s 45ms/step - loss: 9.5035 - mae:
2.4035 - val_loss: 9.6953 - val_mae: 2.4203
Epoch 5/30
819/819 [==============================] - 39s 47ms/step - loss: 9.2092 - mae:
2.3604 - val_loss: 9.6946 - val_mae: 2.4234
Epoch 6/30
819/819 [==============================] - 38s 47ms/step - loss: 8.8934 - mae:
2.3203 - val_loss: 9.8256 - val_mae: 2.4471
Epoch 7/30
819/819 [==============================] - 38s 46ms/step - loss: 8.6851 - mae:
2.2920 - val_loss: 9.7042 - val_mae: 2.4289
Epoch 8/30
819/819 [==============================] - 40s 49ms/step - loss: 8.5223 - mae:
2.2689 - val_loss: 10.0398 - val_mae: 2.4770
Epoch 9/30
819/819 [==============================] - 39s 47ms/step - loss: 8.3621 - mae:
2.2468 - val_loss: 10.0288 - val_mae: 2.4688
Epoch 10/30
819/819 [==============================] - 38s 46ms/step - loss: 8.2145 - mae:
2.2292 - val_loss: 10.2581 - val_mae: 2.4973
Epoch 11/30
819/819 [==============================] - 38s 46ms/step - loss: 8.0964 - mae:
2.2164 - val_loss: 10.3227 - val_mae: 2.4975
Epoch 12/30
819/819 [==============================] - 36s 43ms/step - loss: 7.9722 - mae:
2.2011 - val_loss: 10.4095 - val_mae: 2.5162
Epoch 13/30
819/819 [==============================] - 31s 38ms/step - loss: 7.8857 - mae:
2.1889 - val_loss: 10.7614 - val_mae: 2.5541
Epoch 14/30
819/819 [==============================] - 38s 47ms/step - loss: 7.8018 - mae:
2.1774 - val_loss: 10.4732 - val_mae: 2.5190
Epoch 15/30
819/819 [==============================] - 37s 45ms/step - loss: 7.7180 - mae:
2.1673 - val_loss: 10.5897 - val_mae: 2.5348
Epoch 16/30
819/819 [==============================] - 35s 42ms/step - loss: 7.6437 - mae:
2.1570 - val_loss: 11.0445 - val_mae: 2.5781
Epoch 17/30
819/819 [==============================] - 34s 41ms/step - loss: 7.5777 - mae:
```

```
2.1475 - val_loss: 10.9230 - val_mae: 2.5743
Epoch 18/30
819/819 [==============================] - 32s 39ms/step - loss: 7.4733 - mae:
2.1321 - val_loss: 10.7923 - val_mae: 2.5550
Epoch 19/30
819/819 [==============================] - 36s 44ms/step - loss: 7.4143 - mae:
2.1231 - val_loss: 11.1205 - val_mae: 2.5938
Epoch 20/30
819/819 [==============================] - 33s 41ms/step - loss: 7.3421 - mae:
2.1131 - val_loss: 11.1418 - val_mae: 2.5979
Epoch 21/30
819/819 [==============================] - 32s 39ms/step - loss: 7.2667 - mae:
2.1015 - val_loss: 11.4120 - val_mae: 2.6316
Epoch 22/30
819/819 [==============================] - 34s 42ms/step - loss: 7.2473 - mae:
2.0989 - val_loss: 11.0679 - val_mae: 2.5991
Epoch 23/30
819/819 [==============================] - 38s 46ms/step - loss: 7.1831 - mae:
2.0883 - val_loss: 10.9676 - val_mae: 2.5866
Epoch 24/30
819/819 [==============================] - 33s 41ms/step - loss: 7.1127 - mae:
2.0790 - val_loss: 11.0673 - val_mae: 2.6054
Epoch 25/30
819/819 [==============================] - 32s 38ms/step - loss: 7.0611 - mae:
2.0716 - val_loss: 11.0533 - val_mae: 2.5881
Epoch 26/30
819/819 [==============================] - 36s 44ms/step - loss: 7.0205 - mae:
2.0662 - val_loss: 11.1042 - val_mae: 2.6020
Epoch 27/30
819/819 [==============================] - 34s 41ms/step - loss: 6.9765 - mae:
2.0592 - val_loss: 11.0187 - val_mae: 2.5834
Epoch 28/30
819/819 [==============================] - 35s 42ms/step - loss: 6.9331 - mae:
2.0524 - val_loss: 11.5445 - val_mae: 2.6506
Epoch 29/30
819/819 [==============================] - 37s 46ms/step - loss: 6.9051 - mae:
2.0492 - val_loss: 11.2120 - val_mae: 2.6106
Epoch 30/30
819/819 [==============================] - 38s 46ms/step - loss: 6.8531 - mae:
2.0401 - val_loss: 11.4896 - val_mae: 2.6359
405/405 [==============================] - 6s 14ms/step - loss: 11.0631 - mae:
2.5796
Test MAE: 2.58
```

[23]:

```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, color="red", linestyle="-", label="Training MAE")
plt.plot(epochs, val_loss, color="green", linestyle=":", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



1.5.2 LSTM - dropout Regularization

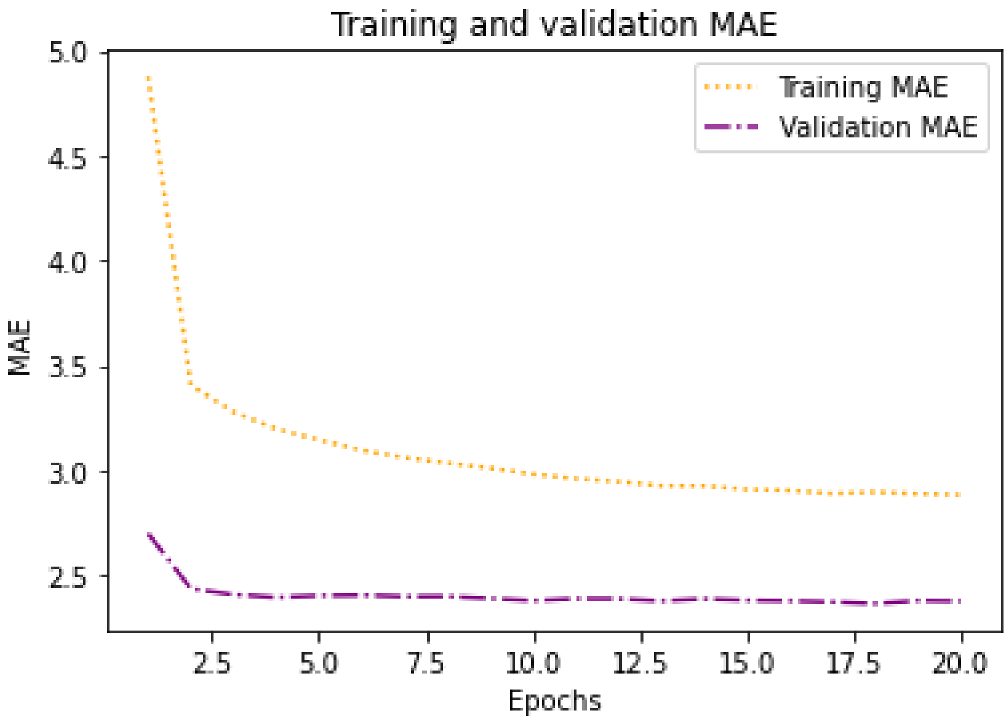
[24]:
```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(16, recurrent_dropout=0.25)(inputs)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
```

```python
history = model.fit(train_dataset,
                    epochs=20,
                    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}")
```

```
Epoch 1/20
819/819 [==============================] - 74s 88ms/step - loss: 42.8306 - mae:
4.8766 - val_loss: 12.6001 - val_mae: 2.6992
Epoch 2/20
819/819 [==============================] - 71s 87ms/step - loss: 19.6392 - mae:
3.4055 - val_loss: 9.8227 - val_mae: 2.4364
Epoch 3/20
819/819 [==============================] - 69s 84ms/step - loss: 18.1403 - mae:
3.2784 - val_loss: 9.5779 - val_mae: 2.4080
Epoch 4/20
819/819 [==============================] - 69s 84ms/step - loss: 17.2146 - mae:
3.1977 - val_loss: 9.4252 - val_mae: 2.3947
Epoch 5/20
819/819 [==============================] - 68s 83ms/step - loss: 16.6788 - mae:
3.1455 - val_loss: 9.4004 - val_mae: 2.4024
Epoch 6/20
819/819 [==============================] - 68s 84ms/step - loss: 16.1785 - mae:
3.0958 - val_loss: 9.4544 - val_mae: 2.4045
Epoch 7/20
819/819 [==============================] - 68s 83ms/step - loss: 15.7867 - mae:
3.0611 - val_loss: 9.4373 - val_mae: 2.3986
Epoch 8/20
819/819 [==============================] - 70s 85ms/step - loss: 15.5167 - mae:
3.0355 - val_loss: 9.4196 - val_mae: 2.3994
Epoch 9/20
819/819 [==============================] - 69s 84ms/step - loss: 15.2544 - mae:
3.0113 - val_loss: 9.2791 - val_mae: 2.3897
Epoch 10/20
819/819 [==============================] - 70s 85ms/step - loss: 14.9283 - mae:
2.9821 - val_loss: 9.3039 - val_mae: 2.3789
Epoch 11/20
819/819 [==============================] - 68s 83ms/step - loss: 14.7177 - mae:
2.9608 - val_loss: 9.3669 - val_mae: 2.3884
Epoch 12/20
819/819 [==============================] - 66s 81ms/step - loss: 14.5666 - mae:
2.9468 - val_loss: 9.3733 - val_mae: 2.3882
Epoch 13/20
819/819 [==============================] - 68s 83ms/step - loss: 14.4209 - mae:
2.9266 - val_loss: 9.2837 - val_mae: 2.3776
```

```
Epoch 14/20
819/819 [==============================] - 70s 85ms/step - loss: 14.3778 - mae:
2.9257 - val_loss: 9.3869 - val_mae: 2.3877
Epoch 15/20
819/819 [==============================] - 70s 85ms/step - loss: 14.2251 - mae:
2.9101 - val_loss: 9.3310 - val_mae: 2.3807
Epoch 16/20
819/819 [==============================] - 69s 84ms/step - loss: 14.1683 - mae:
2.9042 - val_loss: 9.3061 - val_mae: 2.3783
Epoch 17/20
819/819 [==============================] - 67s 82ms/step - loss: 14.0164 - mae:
2.8895 - val_loss: 9.3080 - val_mae: 2.3735
Epoch 18/20
819/819 [==============================] - 66s 81ms/step - loss: 14.0843 - mae:
2.8969 - val_loss: 9.2455 - val_mae: 2.3646
Epoch 19/20
819/819 [==============================] - 70s 86ms/step - loss: 13.9841 - mae:
2.8886 - val_loss: 9.3927 - val_mae: 2.3795
Epoch 20/20
819/819 [==============================] - 70s 86ms/step - loss: 13.9408 - mae:
2.8832 - val_loss: 9.3388 - val_mae: 2.3771
405/405 [==============================] - 5s 12ms/step - loss: 10.8784 - mae:
2.5956
Test MAE: 2.60
```

[25]:
```
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, color="orange", linestyle="dotted", label="Training MAE")
plt.plot(epochs, val_loss, color="purple", linestyle="dashdot",␣
 ↪label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



1.5.3 Stacked LSTM configuration, multiple layers with 16 units each are sequentially arranged to enhance the model's learning capacity and complexity, allowing it to capture deeper sequential patterns in data.

[26]:
```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(16, return_sequences=True)(inputs)
x = layers.LSTM(16)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_LSTM_stacked1.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=20,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_LSTM_stacked1.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

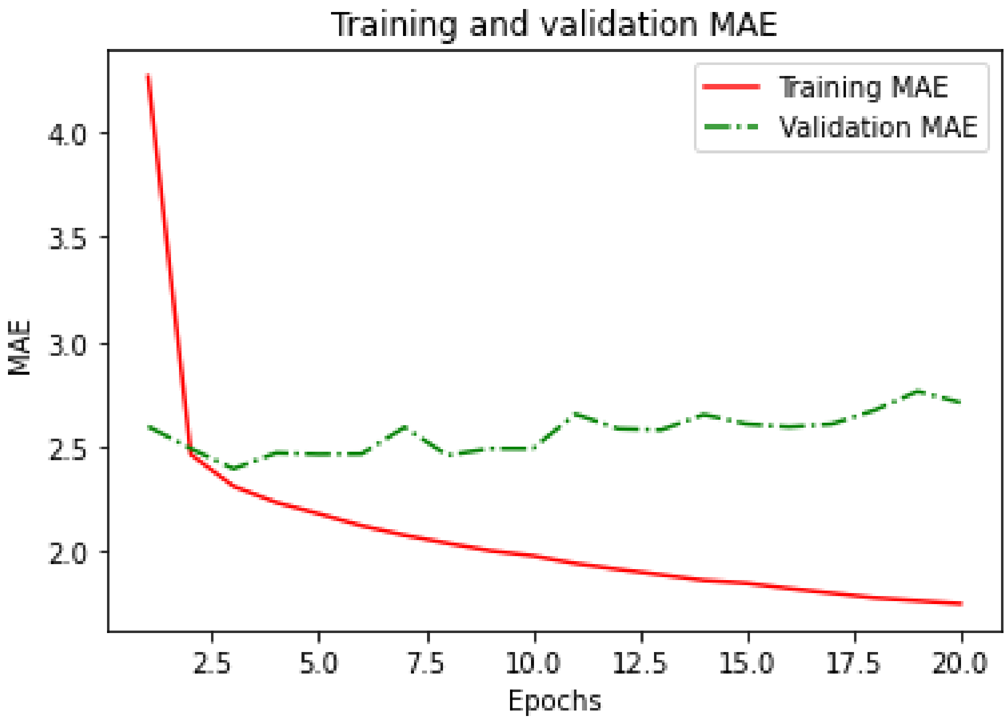
```
Epoch 1/20
```

```
819/819 [==============================] - 71s 83ms/step - loss: 34.7504 - mae:
4.2678 - val_loss: 11.5198 - val_mae: 2.5950
Epoch 2/20
819/819 [==============================] - 74s 90ms/step - loss: 10.1138 - mae:
2.4619 - val_loss: 10.1606 - val_mae: 2.4894
Epoch 3/20
819/819 [==============================] - 78s 95ms/step - loss: 8.8856 - mae:
2.3099 - val_loss: 9.5042 - val_mae: 2.3937
Epoch 4/20
819/819 [==============================] - 79s 97ms/step - loss: 8.2719 - mae:
2.2323 - val_loss: 10.0632 - val_mae: 2.4689
Epoch 5/20
819/819 [==============================] - 71s 87ms/step - loss: 7.8765 - mae:
2.1779 - val_loss: 9.9985 - val_mae: 2.4624
Epoch 6/20
819/819 [==============================] - 69s 85ms/step - loss: 7.4622 - mae:
2.1210 - val_loss: 10.1141 - val_mae: 2.4644
Epoch 7/20
819/819 [==============================] - 76s 93ms/step - loss: 7.1394 - mae:
2.0755 - val_loss: 11.1059 - val_mae: 2.5912
Epoch 8/20
819/819 [==============================] - 73s 90ms/step - loss: 6.8853 - mae:
2.0373 - val_loss: 9.9321 - val_mae: 2.4572
Epoch 9/20
819/819 [==============================] - 72s 87ms/step - loss: 6.6558 - mae:
2.0014 - val_loss: 10.2633 - val_mae: 2.4898
Epoch 10/20
819/819 [==============================] - 74s 91ms/step - loss: 6.4975 - mae:
1.9770 - val_loss: 10.2299 - val_mae: 2.4891
Epoch 11/20
819/819 [==============================] - 70s 85ms/step - loss: 6.2568 - mae:
1.9403 - val_loss: 11.5789 - val_mae: 2.6542
Epoch 12/20
819/819 [==============================] - 67s 81ms/step - loss: 6.0780 - mae:
1.9122 - val_loss: 11.1013 - val_mae: 2.5844
Epoch 13/20
819/819 [==============================] - 69s 84ms/step - loss: 5.9252 - mae:
1.8861 - val_loss: 10.9796 - val_mae: 2.5785
Epoch 14/20
819/819 [==============================] - 70s 85ms/step - loss: 5.7558 - mae:
1.8598 - val_loss: 11.5564 - val_mae: 2.6520
Epoch 15/20
819/819 [==============================] - 67s 82ms/step - loss: 5.6706 - mae:
1.8461 - val_loss: 11.4090 - val_mae: 2.6069
Epoch 16/20
819/819 [==============================] - 74s 90ms/step - loss: 5.5141 - mae:
1.8205 - val_loss: 11.0221 - val_mae: 2.5931
Epoch 17/20
```

```
819/819 [==============================] - 71s 86ms/step - loss: 5.3875 - mae:
1.7983 - val_loss: 11.3963 - val_mae: 2.6068
Epoch 18/20
819/819 [==============================] - 68s 83ms/step - loss: 5.2565 - mae:
1.7759 - val_loss: 11.9647 - val_mae: 2.6729
Epoch 19/20
819/819 [==============================] - 69s 85ms/step - loss: 5.1767 - mae:
1.7622 - val_loss: 12.6780 - val_mae: 2.7633
Epoch 20/20
819/819 [==============================] - 64s 78ms/step - loss: 5.0866 - mae:
1.7481 - val_loss: 12.3647 - val_mae: 2.7098
405/405 [==============================] - 10s 23ms/step - loss: 11.2406 - mae:
2.6483
Test MAE: 2.65
```

[27]:

```
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, color="red", linestyle="-", label="Training MAE")
plt.plot(epochs, val_loss, color="green", linestyle="-.", label="Validation␣
↪MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



1.5.4 LSTM Architecture: 32-Unit Stacked Configuration

- This design employs multiple LSTM layers, each with 32 units, enhancing the network's ability to learn complex patterns and sequences for improved predictive accuracy and depth of learning.**

[28]:
```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(32, return_sequences=True)(inputs)
x = layers.LSTM(32)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_LSTM_stacked2.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=3,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_LSTM_stacked2.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/3
819/819 [==============================] - 97s 115ms/step - loss: 21.1750 - mae:
3.2993 - val_loss: 10.4814 - val_mae: 2.5439
Epoch 2/3
819/819 [==============================] - 106s 129ms/step - loss: 7.9393 - mae:
2.1989 - val_loss: 10.7691 - val_mae: 2.5724
Epoch 3/3
819/819 [==============================] - 102s 125ms/step - loss: 6.4760 - mae:
1.9792 - val_loss: 11.6964 - val_mae: 2.6838
405/405 [==============================] - 16s 38ms/step - loss: 11.1635 - mae:
2.6281
Test MAE: 2.63
```

1.5.5 LSTM - Enhanced Configuration with 8 Units

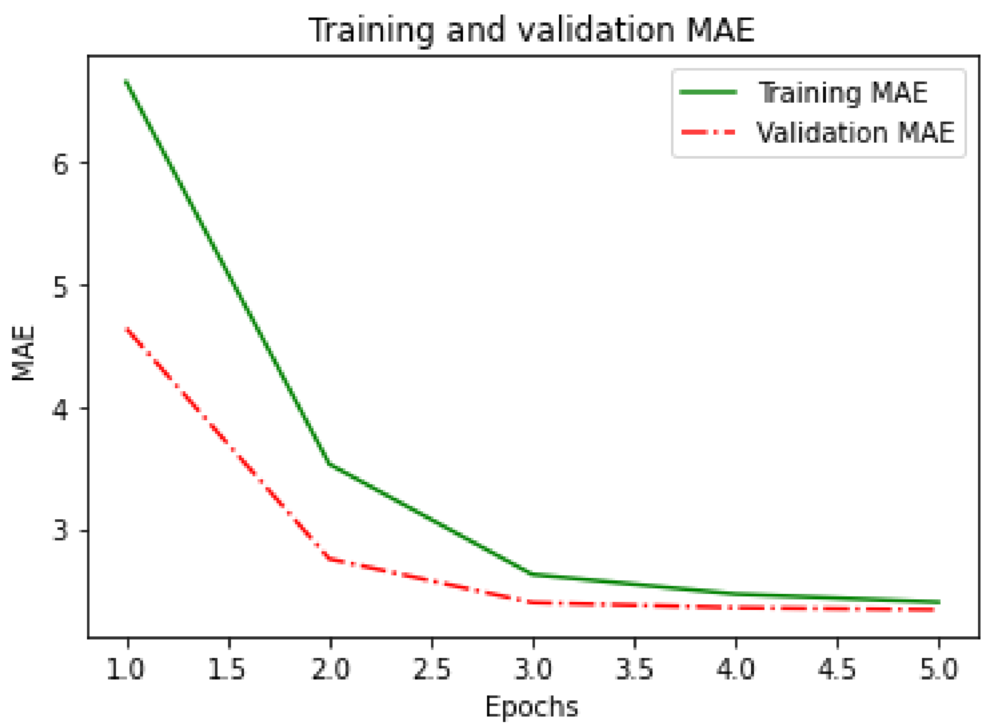
- A multi-layer, 8-unit LSTM architecture enhances model depth for improved feature capture, promoting complex pattern learning and boosting prediction accuracy across various sequential data applications.

[29]:
```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(8, return_sequences=True)(inputs)
x = layers.LSTM(8)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_LSTM_stacked3.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=5,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_LSTM_stacked3.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/5
819/819 [==============================] - 59s 69ms/step - loss: 75.4028 - mae:
6.6551 - val_loss: 39.5502 - val_mae: 4.6391
Epoch 2/5
819/819 [==============================] - 56s 68ms/step - loss: 23.0456 - mae:
3.5298 - val_loss: 13.8421 - val_mae: 2.7544
Epoch 3/5
819/819 [==============================] - 60s 73ms/step - loss: 11.5716 - mae:
2.6270 - val_loss: 9.5712 - val_mae: 2.3969
Epoch 4/5
819/819 [==============================] - 55s 67ms/step - loss: 9.9887 - mae:
```

```
2.4667 - val_loss: 9.1485 - val_mae: 2.3548
Epoch 5/5
819/819 [==============================] - 52s 63ms/step - loss: 9.4877 - mae:
2.4006 - val_loss: 9.0698 - val_mae: 2.3386
405/405 [==============================] - 8s 17ms/step - loss: 10.6134 - mae:
2.5451
Test MAE: 2.55
```

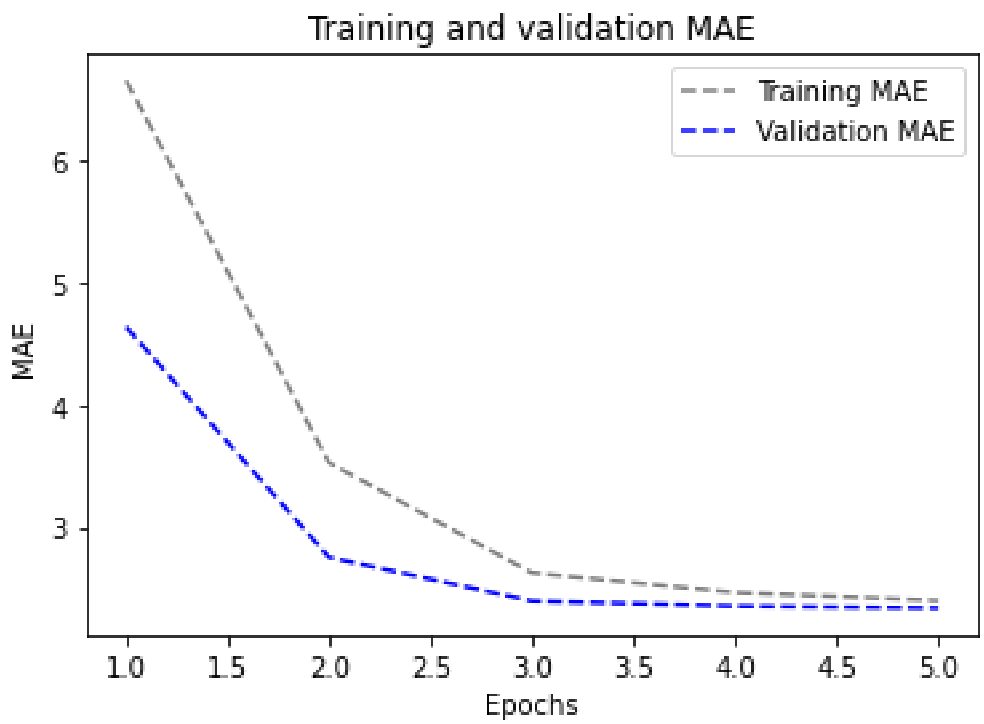
[30]:
```
import matplotlib.pyplot as plt
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="green", linestyle="-", label="Training MAE")
plt.plot(epochs, val_loss, color="red", linestyle="-.", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



1.5.6 Enhanced LSTM Model

- Features dropout regularization and employs a multi-layered architecture for improved performance and robustness against overfitting, ensuring better generalization on diverse datasets.

[32]:
```
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, color="grey", linestyle="dashed", label="Training MAE")
plt.plot(epochs,  val_loss, color="blue",linestyle="dashed", label="Validation␣
↪MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



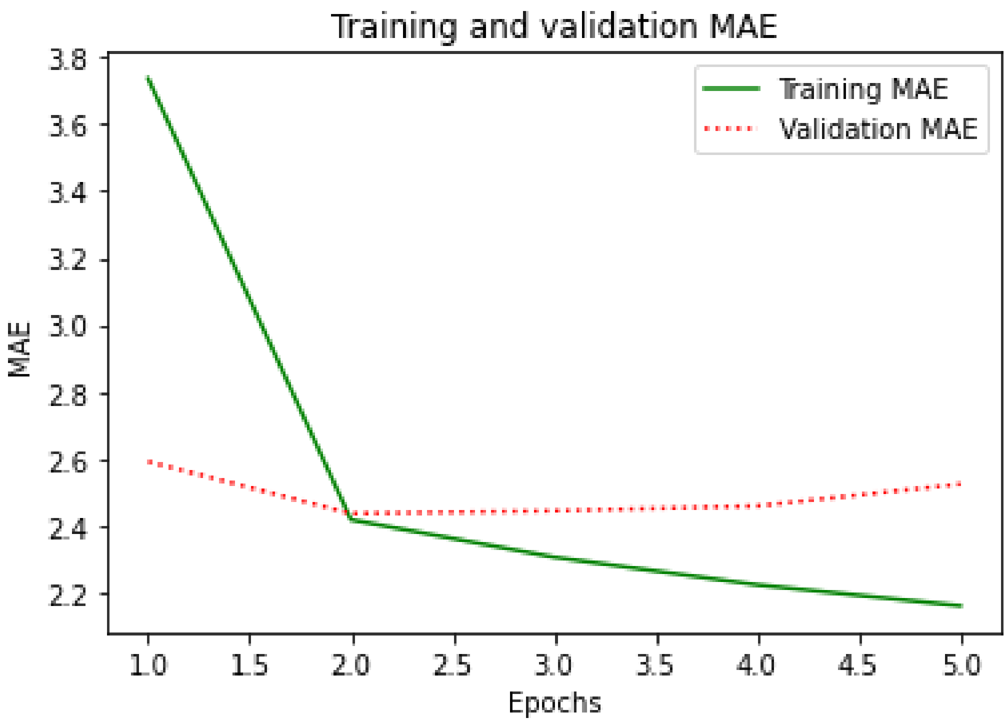
1.6 Bidirectional LSTMs enhance understanding by processing data in both forward and reverse directions, improving prediction and context awareness.

[33]:
```
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)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_bidirec_LSTM.keras",
                                    save_best_only=True)
]

model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=5,
                    validation_data=val_dataset,
                     callbacks=callbacks)

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

```
Epoch 1/5
819/819 [==============================] - 47s 54ms/step - loss: 27.1035 - mae:
3.7360 - val_loss: 10.9731 - val_mae: 2.5919
Epoch 2/5
819/819 [==============================] - 41s 50ms/step - loss: 9.5349 - mae:
2.4180 - val_loss: 9.8281 - val_mae: 2.4369
Epoch 3/5
819/819 [==============================] - 44s 54ms/step - loss: 8.7013 - mae:
2.3062 - val_loss: 9.9125 - val_mae: 2.4447
Epoch 4/5
819/819 [==============================] - 43s 53ms/step - loss: 8.0864 - mae:
2.2236 - val_loss: 9.9827 - val_mae: 2.4593
Epoch 5/5
819/819 [==============================] - 43s 52ms/step - loss: 7.6541 - mae:
2.1610 - val_loss: 10.6079 - val_mae: 2.5247
405/405 [==============================] - 8s 18ms/step - loss: 11.0557 - mae:
2.6226
Test MAE: 2.62
```

[34]:
```
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, color="green", linestyle="-", label="Training MAE")
```

```
plt.plot(epochs, val_loss, color="red", linestyle=":", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



1.6.1 Combining 1D ConvNets with LSTMs enhances feature extraction and sequential data processing, improving time series prediction and text classification efficiency.

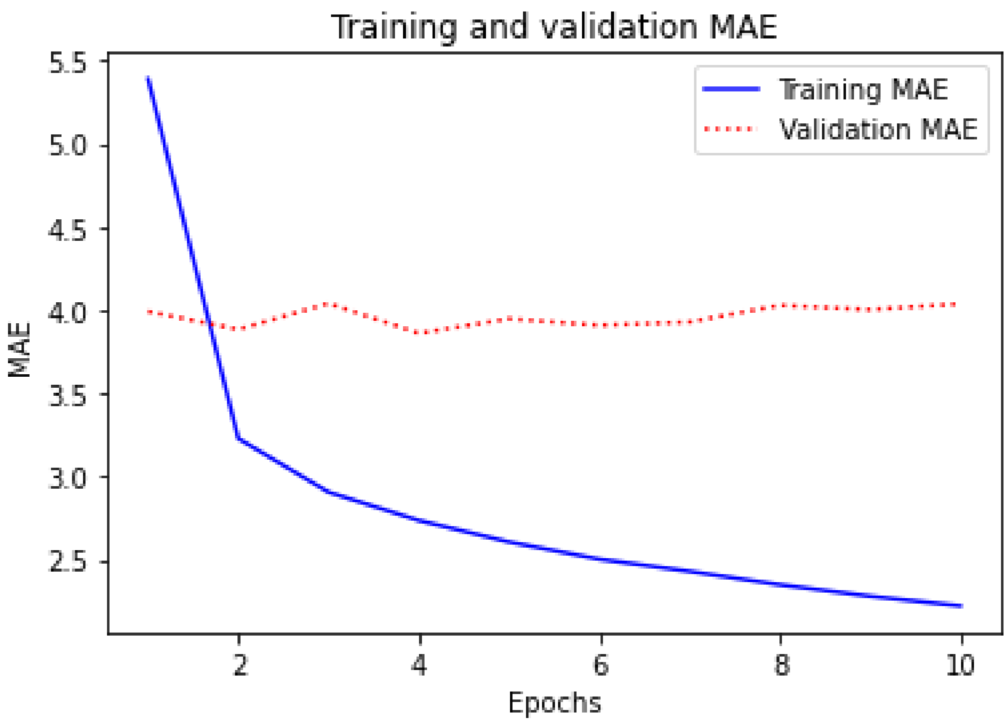
[35]:
```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Conv1D(64, 3, activation='relu')(inputs)
x = layers.MaxPooling1D(3)(x)
x = layers.Conv1D(128, 3, activation='relu')(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Reshape((-1, 128))(x)
x = layers.LSTM(16)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

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

```python
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_Conv_LSTM.keras", save_best_only=True)
]

history = model.fit(train_dataset, epochs=10, validation_data=val_dataset,␣
 ↪callbacks=callbacks)

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

```
Epoch 1/10
819/819 [==============================] - 42s 49ms/step - loss: 51.8175 - mae:
5.3903 - val_loss: 26.9152 - val_mae: 3.9919
Epoch 2/10
819/819 [==============================] - 39s 47ms/step - loss: 17.7467 - mae:
3.2294 - val_loss: 23.5092 - val_mae: 3.8843
Epoch 3/10
819/819 [==============================] - 39s 47ms/step - loss: 14.1392 - mae:
2.9071 - val_loss: 25.1284 - val_mae: 4.0405
Epoch 4/10
819/819 [==============================] - 38s 47ms/step - loss: 12.5786 - mae:
2.7366 - val_loss: 23.2584 - val_mae: 3.8610
Epoch 5/10
819/819 [==============================] - 38s 47ms/step - loss: 11.4935 - mae:
2.6074 - val_loss: 24.4073 - val_mae: 3.9481
Epoch 6/10
819/819 [==============================] - 39s 48ms/step - loss: 10.6868 - mae:
2.5035 - val_loss: 23.8940 - val_mae: 3.9089
Epoch 7/10
819/819 [==============================] - 39s 47ms/step - loss: 10.0983 - mae:
2.4306 - val_loss: 24.2641 - val_mae: 3.9288
Epoch 8/10
819/819 [==============================] - 38s 46ms/step - loss: 9.4901 - mae:
2.3501 - val_loss: 25.4257 - val_mae: 4.0290
Epoch 9/10
819/819 [==============================] - 38s 47ms/step - loss: 8.9507 - mae:
2.2811 - val_loss: 25.6006 - val_mae: 4.0031
Epoch 10/10
819/819 [==============================] - 37s 46ms/step - loss: 8.5471 - mae:
2.2231 - val_loss: 25.8764 - val_mae: 4.0380
405/405 [==============================] - 6s 15ms/step - loss: 25.4850 - mae:
4.0514
Test MAE: 4.05
```

[36]:
```python
import matplotlib.pyplot as plt
loss = history.history["mae"]
val_loss = history.history["val_mae"]
```

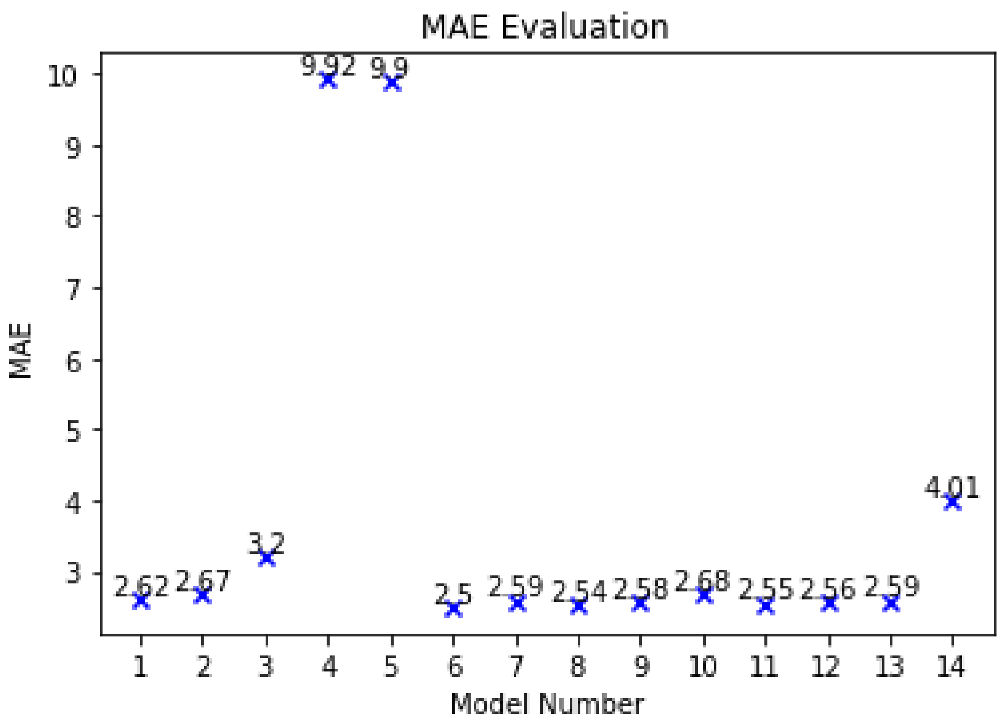
```
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="blue", linestyle="-", label="Training MAE")
plt.plot(epochs,  val_loss, color="red",linestyle=":", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



Different models were developed and compared as part of our project. These progressed through different neural network topologies, starting from a common sense baseline that wasn't machine learning and going all the way up to basic machine learning. We looked at 1D convolutional models as well as various RNN topologies, such as basic, stacked layers, and GRUs. We worked with LSTM models for a long time, ranging in complexity from straightforward to stacked configurations with 8, 16, and 32 unit sizes, and adding dropout regularization. In order to assess their performance on various tasks, we also experimented with a bidirectional LSTM and a hybrid model that combined 1D convnets with LSTM. We were able to cover a wide range of machine learning approaches thanks to this thorough methodology, from basic concepts to state-of-the-art neural network structures.

[41]:
```
import matplotlib.pyplot as plt

Models = ("1","2","3","4","5","6","7","8","9","10","11","12","13","14")
```

```
Mae = (2.62,2.67,3.2,9.92,9.9,2.5,2.59,2.54,2.58,2.68,2.55,2.56,2.59,4.01)

# MAE Evaluation
plt.scatter(Models, Mae, color="blue", marker='x')
plt.title("MAE Evaluation")
plt.xlabel("Model Number")
plt.ylabel("MAE")

for (xi, yi) in zip(Models,Mae):
    plt.text(xi, yi, str(yi), va='bottom', ha='center')

plt.show()
```



We started with a non-machine learning, common sense baseline method and achieved a Mean Absolute Error (MAE) of 2.62 in our investigation of several models for time series analysis. A somewhat higher MAE of 2.65 was obtained in later tests using a simple machine learning model that used dense layers. This was mostly because the data preprocessing step lost some temporal sequence. Our attempt to employ convolutional models did not work well, mostly due to the insensitivity of the convolutional method to the data's sequential order, which resulted in an inadequate performance. This revealed the need for architectures made especially for time series data, such as Recurrent Neural Networks (RNNs), whose capacity to retain past inputs makes them excellent at capturing temporal dependencies. Nevertheless, because to the vanishing gradient issue that impedes deep network learning, the SimpleRNN model performed poorly. We investigated GRU and LSTM designs in order to get around this, with the straightforward GRU model outperforming the others by effectively capturing long-range dependencies. Our efforts to improve LSTM models by adding bidirectional layers, recurrent dropout, and unit adjustments demonstrated encouraging outcomes that were closely matched by the common sense baseline. The poor MAE of 4.05 obtained by testing a hybrid model that combined 1D convolution with LSTM highlights the limits of convolutional models in maintaining the order of sequential data. In summary, our study highlights the significance of selecting an appropriate architecture for time series data, with RNN variations such as GRU and LSTM presenting considerable promise for capturing intricate temporal patterns.