This is a companion notebook for the book [Deep Learning with Python, Second Edition](). For readability, it only contains runnable code blocks and section titles, and omits everything else in the book: text paragraphs, figures, and pseudocode.

If you want to be able to follow what's going on, I recommend reading the notebook side by side with your copy of the book.

This notebook was generated for TensorFlow 2.6.

Deep learning for timeseries

Different kinds of timeseries tasks

A temperature-forecasting example

In [0]:

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

Inspecting the data of the Jena weather dataset

In [1]:

```
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))
```

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

Parsing the data

In [2]:

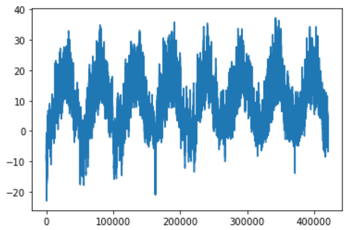
```
import numpy as np
temperature = np.zeros((len(lines),))
raw_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(",")[1:]]
    temperature[i] = values[1]
    raw_data[i, :] = values[:]
```

Plotting the temperature timeseries

In [3]:

```
from matplotlib import pyplot as plt
plt.plot(range(len(temperature)), temperature)
```

Out[3]: []

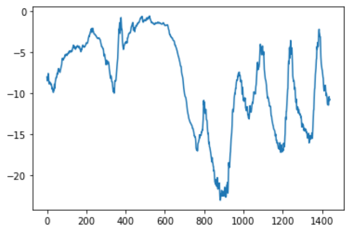


Plotting the first 10 days of the temperature timeseries

In [4]:

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

Out[4]: []



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

In [5]:

```
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
```

Preparing the data

Normalizing the data

In [6]:

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

In [7]:

```
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

In [8]:

```
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

In [9]:

```python
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,)
```

A common-sense, non-machine-learning baseline

Computing the common-sense baseline MAE

In [10]:

```python
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
```

Let's try a basic machine-learning model

Training and evaluating a densely connected model

In [11]:

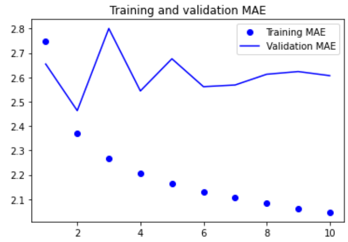
```python
from tensorflow import keras
from tensorflow.keras import layers

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=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 [==============================] - 6s 6ms/step - loss: 12.6572 - mae: 2.7477 - val_loss: 11.3838 - val_mae: 2.6549
Epoch 2/10
819/819 [==============================] - 5s 6ms/step - loss: 9.0960 - mae: 2.3711 - val_loss: 9.8224 - val_mae: 2.4633
Epoch 3/10
819/819 [==============================] - 5s 6ms/step - loss: 8.2734 - mae: 2.2656 - val_loss: 12.5271 - val_mae: 2.8009
Epoch 4/10
819/819 [==============================] - 5s 6ms/step - loss: 7.8064 - mae: 2.2059 - val_loss: 10.3681 - val_mae: 2.5442
Epoch 5/10
819/819 [==============================] - 5s 6ms/step - loss: 7.4812 - mae: 2.1629 - val_loss: 11.4768 - val_mae: 2.6764
Epoch 6/10
819/819 [==============================] - 5s 6ms/step - loss: 7.2546 - mae: 2.1293 - val_loss: 10.5296 - val_mae: 2.5616
Epoch 7/10
819/819 [==============================] - 5s 6ms/step - loss: 7.0873 - mae: 2.1059 - val_loss: 10.5754 - val_mae: 2.5686
Epoch 8/10
819/819 [==============================] - 5s 6ms/step - loss: 6.9135 - mae: 2.0817 - val_loss: 10.9478 - val_mae: 2.6126
Epoch 9/10
819/819 [==============================] - 5s 6ms/step - loss: 6.7732 - mae: 2.0615 - val_loss: 10.9876 - val_mae: 2.6238
Epoch 10/10
819/819 [==============================] - 5s 6ms/step - loss: 6.6668 - mae: 2.0456 - val_loss: 10.8723 - val_mae: 2.6069
405/405 [==============================] - 2s 4ms/step - loss: 10.7727 - mae: 2.5842
Test MAE: 2.58
```

Plotting results

In [12]:

```
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("Training and validation MAE")
plt.legend()
plt.show()
```



Let's try a 1D convolutional model

In [13]:

```
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)

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

```
Epoch 1/10
819/819 [==============================] - 10s 9ms/step - loss: 24.3670 - mae: 3.8371 - val_loss: 15.6560 - val_mae: 3.1207
Epoch 2/10
819/819 [==============================] - 7s 9ms/step - loss: 15.7505 - mae: 3.1497 - val_loss: 18.1392 - val_mae: 3.3539
Epoch 3/10
819/819 [==============================] - 9s 10ms/step - loss: 14.4414 - mae: 3.0116 - val_loss: 14.8862 - val_mae: 3.0594
Epoch 4/10
819/819 [==============================] - 9s 11ms/step - loss: 13.6781 - mae: 2.9235 - val_loss: 16.0429 - val_mae: 3.1482
Epoch 5/10
819/819 [==============================] - 9s 11ms/step - loss: 13.1143 - mae: 2.8607 - val_loss: 17.8093 - val_mae: 3.3372
Epoch 6/10
819/819 [==============================] - 9s 11ms/step - loss: 12.6263 - mae: 2.8090 - val_loss: 15.4455 - val_mae: 3.0965
Epoch 7/10
819/819 [==============================] - 9s 11ms/step - loss: 12.2076 - mae: 2.7621 - val_loss: 16.6135 - val_mae: 3.1888
Epoch 8/10
819/819 [==============================] - 9s 12ms/step - loss: 11.8068 - mae: 2.7177 - val_loss: 15.1065 - val_mae: 3.0761
Epoch 9/10
819/819 [==============================] - 9s 11ms/step - loss: 11.4506 - mae: 2.6756 - val_loss: 17.4583 - val_mae: 3.3382
Epoch 10/10
819/819 [==============================] - 9s 11ms/step - loss: 11.1447 - mae: 2.6420 - val_loss: 15.3919 - val_mae: 3.0959
405/405 [==============================] - 2s 4ms/step - loss: 17.0281 - mae: 3.2675
Test MAE: 3.27
```

A first recurrent baseline

A simple LSTM-based model

In [14]:

```
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)

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

```
Epoch 1/10
819/819 [==============================] - 11s 11ms/step - loss: 41.6994 - mae: 4.6888 - val_loss: 12.3566 - val_mae: 2.6717
Epoch 2/10
819/819 [==============================] - 9s 11ms/step - loss: 10.9468 - mae: 2.5680 - val_loss: 9.8223 - val_mae: 2.4259
Epoch 3/10
819/819 [==============================] - 9s 11ms/step - loss: 9.8583 - mae: 2.4479 - val_loss: 10.0782 - val_mae: 2.4494
Epoch 4/10
819/819 [==============================] - 9s 11ms/step - loss: 9.5077 - mae: 2.3999 - val_loss: 9.9815 - val_mae: 2.4336
Epoch 5/10
819/819 [==============================] - 9s 11ms/step - loss: 9.2706 - mae: 2.3689 - val_loss: 10.3790 - val_mae: 2.4633
Epoch 6/10
819/819 [==============================] - 9s 11ms/step - loss: 9.0969 - mae: 2.3448 - val_loss: 10.4309 - val_mae: 2.4690
Epoch 7/10
819/819 [==============================] - 9s 11ms/step - loss: 8.8533 - mae: 2.3102 - val_loss: 10.0912 - val_mae: 2.4289
Epoch 8/10
819/819 [==============================] - 9s 11ms/step - loss: 8.6181 - mae: 2.2783 - val_loss: 9.8560 - val_mae: 2.4033
Epoch 9/10
819/819 [==============================] - 9s 11ms/step - loss: 8.4430 - mae: 2.2563 - val_loss: 9.6263 - val_mae: 2.3911
Epoch 10/10
819/819 [==============================] - 9s 11ms/step - loss: 8.2831 - mae: 2.2365 - val_loss: 9.6786 - val_mae: 2.4035
405/405 [==============================] - 2s 5ms/step - loss: 10.5798 - mae: 2.5739
Test MAE: 2.57
```

In [15]:

```
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)
```

A recurrent layer in Keras

An RNN layer that can process sequences of any length

In [16]:

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

An RNN layer that returns only its last output step

In [17]:

```
num_features = 14
steps = 120
inputs = keras.Input(shape=(steps, num_features))
outputs = layers.SimpleRNN(16, return_sequences=False)(inputs)
print(outputs.shape)
```

```
(None, 16)
```

An RNN layer that returns its full output sequence

In [18]:

```
num_features = 14
steps = 120
inputs = keras.Input(shape=(steps, num_features))
outputs = layers.SimpleRNN(16, return_sequences=True)(inputs)
print(outputs.shape)
```

```
(None, 120, 16)
```

Stacking RNN layers

In [19]:

```
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)
```

Advanced use of recurrent neural networks

Using recurrent dropout to fight overfitting

Training and evaluating a dropout-regularized LSTM

In [20]:

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(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("jena_lstm_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=50,
                    validation_data=val_dataset,
                    callbacks=callbacks)
```

```
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when runni
ng on GPU.
Epoch 1/50
819/819 [==============================] - 502s 611ms/step - loss: 26.0826 - mae: 3.7911 - val_loss: 10.0547 - val_mae: 2.4713
Epoch 2/50
819/819 [==============================] - 508s 620ms/step - loss: 14.7710 - mae: 2.9846 - val_loss: 9.3839 - val_mae: 2.3778
Epoch 3/50
819/819 [==============================] - 479s 585ms/step - loss: 13.8350 - mae: 2.8839 - val_loss: 9.4168 - val_mae: 2.3872
Epoch 4/50
819/819 [==============================] - 482s 589ms/step - loss: 13.1707 - mae: 2.8115 - val_loss: 9.5503 - val_mae: 2.4078
Epoch 5/50
819/819 [==============================] - 530s 647ms/step - loss: 12.7362 - mae: 2.7667 - val_loss: 9.3157 - val_mae: 2.3709
Epoch 6/50
819/819 [==============================] - 549s 670ms/step - loss: 12.3723 - mae: 2.7239 - val_loss: 9.5854 - val_mae: 2.4084
Epoch 7/50
819/819 [==============================] - 552s 674ms/step - loss: 12.1062 - mae: 2.6944 - val_loss: 9.6112 - val_mae: 2.4133
Epoch 8/50
819/819 [==============================] - 552s 674ms/step - loss: 11.7730 - mae: 2.6580 - val_loss: 9.4661 - val_mae: 2.3912
Epoch 9/50
819/819 [==============================] - 552s 674ms/step - loss: 11.6212 - mae: 2.6433 - val_loss: 9.5470 - val_mae: 2.4002
Epoch 10/50
819/819 [==============================] - 549s 671ms/step - loss: 11.3629 - mae: 2.6142 - val_loss: 9.3815 - val_mae: 2.3874
Epoch 11/50
819/819 [==============================] - 551s 672ms/step - loss: 11.1733 - mae: 2.5892 - val_loss: 9.2729 - val_mae: 2.3727
Epoch 12/50
819/819 [==============================] - 550s 671ms/step - loss: 11.0498 - mae: 2.5778 - val_loss: 9.0728 - val_mae: 2.3498
Epoch 13/50
819/819 [==============================] - 551s 673ms/step - loss: 10.9135 - mae: 2.5632 - val_loss: 9.7261 - val_mae: 2.4293
```

```
Epoch 14/50
819/819 [==============================] - 495s 604ms/step - loss: 10.7813 - mae: 2.5452 - val_loss: 9.6143 - val_mae: 2.4207
Epoch 15/50
819/819 [==============================] - 479s 585ms/step - loss: 10.6615 - mae: 2.5338 - val_loss: 9.5337 - val_mae: 2.4074
Epoch 16/50
819/819 [==============================] - 482s 588ms/step - loss: 10.5750 - mae: 2.5201 - val_loss: 9.7370 - val_mae: 2.4239
Epoch 17/50
819/819 [==============================] - 482s 588ms/step - loss: 10.4448 - mae: 2.5028 - val_loss: 9.5460 - val_mae: 2.4016
Epoch 18/50
819/819 [==============================] - 481s 588ms/step - loss: 10.3932 - mae: 2.4982 - val_loss: 9.6767 - val_mae: 2.4194
Epoch 19/50
819/819 [==============================] - 481s 588ms/step - loss: 10.2259 - mae: 2.4783 - val_loss: 9.5299 - val_mae: 2.4056
Epoch 20/50
819/819 [==============================] - 479s 585ms/step - loss: 10.1608 - mae: 2.4707 - val_loss: 9.7668 - val_mae: 2.4257
Epoch 21/50
819/819 [==============================] - 483s 589ms/step - loss: 10.0752 - mae: 2.4594 - val_loss: 9.8139 - val_mae: 2.4370
Epoch 22/50
819/819 [==============================] - 506s 618ms/step - loss: 10.0123 - mae: 2.4503 - val_loss: 10.0859 - val_mae: 2.4763
Epoch 23/50
819/819 [==============================] - 550s 671ms/step - loss: 9.9241 - mae: 2.4408 - val_loss: 9.9001 - val_mae: 2.4541
Epoch 24/50
819/819 [==============================] - 550s 672ms/step - loss: 9.8951 - mae: 2.4353 - val_loss: 10.1830 - val_mae: 2.4942
Epoch 25/50
819/819 [==============================] - 552s 673ms/step - loss: 9.7739 - mae: 2.4231 - val_loss: 9.7975 - val_mae: 2.4368
Epoch 26/50
819/819 [==============================] - 548s 670ms/step - loss: 9.6988 - mae: 2.4156 - val_loss: 10.0036 - val_mae: 2.4678
Epoch 27/50
819/819 [==============================] - 551s 673ms/step - loss: 9.6365 - mae: 2.4066 - val_loss: 9.8540 - val_mae: 2.4537
Epoch 28/50
819/819 [==============================] - 552s 674ms/step - loss: 9.6589 - mae: 2.4034 - val_loss: 10.1126 - val_mae: 2.4809
Epoch 29/50
819/819 [==============================] - 552s 674ms/step - loss: 9.5380 - mae: 2.3942 - val_loss: 10.1637 - val_mae: 2.4869
Epoch 30/50
819/819 [==============================] - 551s 673ms/step - loss: 9.4882 - mae: 2.3855 - val_loss: 9.9820 - val_mae: 2.4600
Epoch 31/50
819/819 [==============================] - 520s 635ms/step - loss: 9.4816 - mae: 2.3809 - val_loss: 10.0841 - val_mae: 2.4738
Epoch 32/50
819/819 [==============================] - 554s 676ms/step - loss: 9.3876 - mae: 2.3742 - val_loss: 10.0639 - val_mae: 2.4763
Epoch 33/50
819/819 [==============================] - 554s 676ms/step - loss: 9.3043 - mae: 2.3635 - val_loss: 9.9912 - val_mae: 2.4707
Epoch 34/50
819/819 [==============================] - 560s 684ms/step - loss: 9.3159 - mae: 2.3617 - val_loss: 9.8645 - val_mae: 2.4549
Epoch 35/50
819/819 [==============================] - 553s 675ms/step - loss: 9.2982 - mae: 2.3625 - val_loss: 10.0490 - val_mae: 2.4717
Epoch 36/50
819/819 [==============================] - 498s 608ms/step - loss: 9.2160 - mae: 2.3523 - val_loss: 10.3507 - val_mae: 2.5121
Epoch 37/50
819/819 [==============================] - 481s 588ms/step - loss: 9.1973 - mae: 2.3462 - val_loss: 10.2429 - val_mae: 2.4996
Epoch 38/50
819/819 [==============================] - 483s 589ms/step - loss: 9.1645 - mae: 2.3458 - val_loss: 10.2882 - val_mae: 2.4938
Epoch 39/50
819/819 [==============================] - 480s 587ms/step - loss: 9.1218 - mae: 2.3373 - val_loss: 10.2649 - val_mae: 2.4956
```

```
Epoch 40/50
819/819 [==============================] - 509s 622ms/step - loss: 9.0897 - mae: 2.3331 - val_loss: 10.2998 - val_mae: 2.4999
Epoch 41/50
819/819 [==============================] - 484s 590ms/step - loss: 9.0293 - mae: 2.3298 - val_loss: 10.4864 - val_mae: 2.5366
Epoch 42/50
819/819 [==============================] - 483s 590ms/step - loss: 9.0028 - mae: 2.3237 - val_loss: 10.3656 - val_mae: 2.5125
Epoch 43/50
819/819 [==============================] - 481s 587ms/step - loss: 8.9756 - mae: 2.3209 - val_loss: 10.3081 - val_mae: 2.5020
Epoch 44/50
819/819 [==============================] - 533s 651ms/step - loss: 8.9321 - mae: 2.3154 - val_loss: 10.6029 - val_mae: 2.5384
Epoch 45/50
819/819 [==============================] - 552s 674ms/step - loss: 8.9227 - mae: 2.3089 - val_loss: 10.3620 - val_mae: 2.5142
Epoch 46/50
819/819 [==============================] - 551s 673ms/step - loss: 8.9272 - mae: 2.3135 - val_loss: 10.2842 - val_mae: 2.5016
Epoch 47/50
485/819 [================>.............] - ETA: 3:22 - loss: 8.8418 - mae: 2.3056
```

In [21]:

```
inputs = keras.Input(shape=(sequence_length, num_features))
x = layers.LSTM(32, recurrent_dropout=0.2, unroll=True)(inputs)
```

```
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
```

Stacking recurrent layers

Training and evaluating a dropout-regularized, stacked GRU model

In [22]:

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.GRU(32, recurrent_dropout=0.5, return_sequences=True)(inputs)
x = layers.GRU(32, recurrent_dropout=0.5)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

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=5,
                    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}")
```

```
WARNING:tensorflow:Layer gru will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/5
106/819 [==>...........................] - ETA: 12:14 - loss: 57.7047 - mae: 5.8984
```

Using bidirectional RNNs

Training and evaluating a bidirectional LSTM

In [23]:

```
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=5,
                    validation_data=val_dataset)
```

```
Epoch 1/5
819/819 [==============================] - 15s 16ms/step - loss: 22.8377 - mae: 3.4853 - val_loss: 10.5646 - val_mae: 2.5131
Epoch 2/5
819/819 [==============================] - 13s 16ms/step - loss: 9.4007 - mae: 2.3916 - val_loss: 10.1300 - val_mae: 2.4671
Epoch 3/5
819/819 [==============================] - 13s 16ms/step - loss: 8.4834 - mae: 2.2660 - val_loss: 11.2669 - val_mae: 2.5690
Epoch 4/5
819/819 [==============================] - 13s 16ms/step - loss: 8.0228 - mae: 2.2049 - val_loss: 10.4065 - val_mae: 2.4891
Epoch 5/5
819/819 [==============================] - 13s 16ms/step - loss: 7.6331 - mae: 2.1498 - val_loss: 11.1995 - val_mae: 2.5748
```

Going even further

Summary