C4W4_Assignment

March 18, 2023

1 Week 4: Using real world data

Welcome! So far you have worked exclusively with generated data. This time you will be using the Daily Minimum Temperatures in Melbourne dataset which contains data of the daily minimum temperatures recorded in Melbourne from 1981 to 1990. In addition to be using Tensorflow's layers for processing sequence data such as Recurrent layers or LSTMs you will also use Convolutional layers to improve the model's performance.

Let's get started!

NOTE: *To prevent errors from the autograder, you are not allowed to edit or delete some of the cells in this notebook . Please only put your solutions in between the `### START CODE HERE` and `### END CODE HERE` code comments, and also refrain from adding any new cells.* ***Once you have passed this assignment*** *and want to experiment with any of the locked cells, you may follow the instructions at the bottom of this notebook.*

[1]:
```
import csv
import pickle
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from dataclasses import dataclass
from absl import logging
logging.set_verbosity(logging.ERROR)
```

Begin by looking at the structure of the csv that contains the data:

[2]:
```
TEMPERATURES_CSV = './data/daily-min-temperatures.csv'

with open(TEMPERATURES_CSV, 'r') as csvfile:
    print(f"Header looks like this:\n\n{csvfile.readline()}")
    print(f"First data point looks like this:\n\n{csvfile.readline()}")
    print(f"Second data point looks like this:\n\n{csvfile.readline()}")
```

```
Header looks like this:

"Date","Temp"

First data point looks like this:
```

```
"1981-01-01",20.7

Second data point looks like this:

"1981-01-02",17.9
```

As you can see, each data point is composed of the date and the recorded minimum temperature for that date.

In the first exercise you will code a function to read the data from the csv but for now run the next cell to load a helper function to plot the time series.

[3]:
```python
def plot_series(time, series, format="-", start=0, end=None):
    plt.plot(time[start:end], series[start:end], format)
    plt.xlabel("Time")
    plt.ylabel("Value")
    plt.grid(True)
```

1.1 Parsing the raw data

Now you need to read the data from the csv file. To do so, complete the `parse_data_from_file` function.

A couple of things to note:

- You should omit the first line as the file contains headers.
- There is no need to save the data points as numpy arrays, regular lists is fine.
- To read from csv files use `csv.reader` by passing the appropriate arguments.
- `csv.reader` returns an iterable that returns each row in every iteration. So the temperature can be accessed via row[1] and the date can be discarded.
- The `times` list should contain every timestep (starting at zero), which is just a sequence of ordered numbers with the same length as the `temperatures` list.
- The values of the `temperatures` should be of `float` type. You can use Python's built-in `float` function to ensure this.

[4]:
```python
def parse_data_from_file(filename):

    times = []
    temperatures = []

    with open(filename) as csvfile:

        ### START CODE HERE

        reader = csv.reader(csvfile, delimiter=',')
        next(reader)
        step=0
        for row in reader:
            temperatures.append(float(row[1]))
```

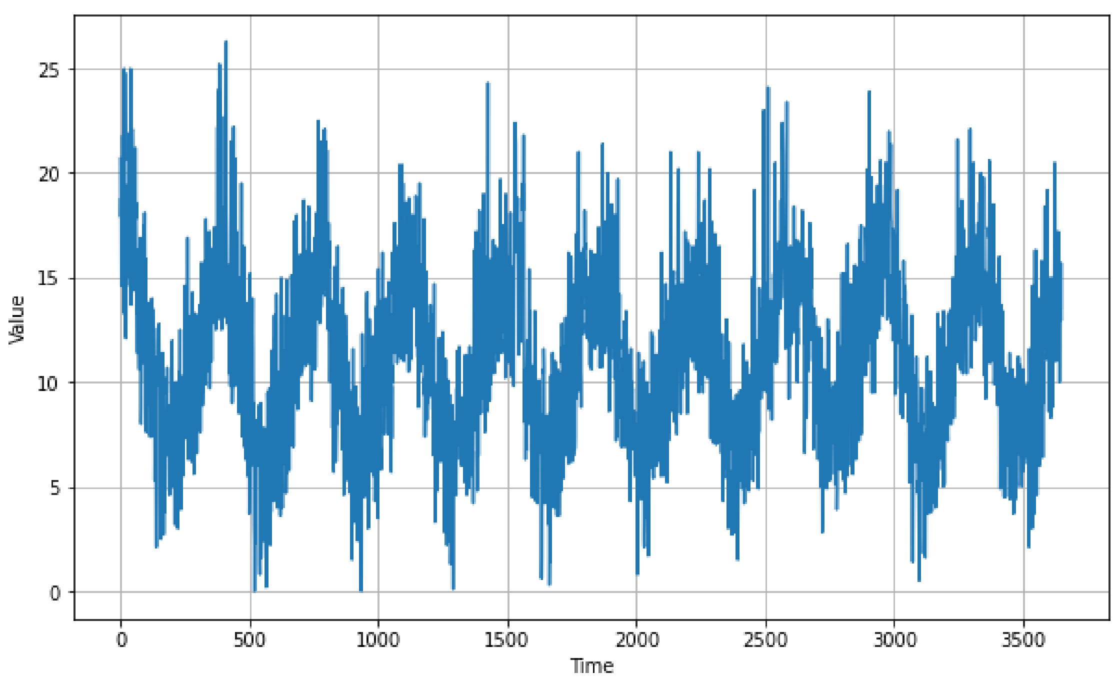
```
            times.append(step)
            step = step + 1
        ### END CODE HERE

    return times, temperatures
```

The next cell will use your function to compute the `times` and `temperatures` and will save these as numpy arrays within the `G` dataclass. This cell will also plot the time series:

[5]:
```
# Test your function and save all "global" variables within the G class (G␣
 ↪stands for global)
@dataclass
class G:
    TEMPERATURES_CSV = './data/daily-min-temperatures.csv'
    times, temperatures = parse_data_from_file(TEMPERATURES_CSV)
    TIME = np.array(times)
    SERIES = np.array(temperatures)
    SPLIT_TIME = 2500
    WINDOW_SIZE = 64
    BATCH_SIZE = 32
    SHUFFLE_BUFFER_SIZE = 1000


plt.figure(figsize=(10, 6))
plot_series(G.TIME, G.SERIES)
plt.show()
```



Expected Output:

1.2 Processing the data

Since you already coded the `train_val_split` and `windowed_dataset` functions during past week's assignments, this time they are provided for you:

[6]:
```
def train_val_split(time, series, time_step=G.SPLIT_TIME):

    time_train = time[:time_step]
    series_train = series[:time_step]
    time_valid = time[time_step:]
    series_valid = series[time_step:]

    return time_train, series_train, time_valid, series_valid


# Split the dataset
time_train, series_train, time_valid, series_valid = train_val_split(G.TIME, G.
↪SERIES)
```

[7]:
```
def windowed_dataset(series, window_size=G.WINDOW_SIZE, batch_size=G.
↪BATCH_SIZE, shuffle_buffer=G.SHUFFLE_BUFFER_SIZE):
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size + 1, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size + 1))
    ds = ds.shuffle(shuffle_buffer)
    ds = ds.map(lambda w: (w[:-1], w[-1]))
    ds = ds.batch(batch_size).prefetch(1)
    return ds


# Apply the transformation to the training set
train_set = windowed_dataset(series_train, window_size=G.WINDOW_SIZE,␣
↪batch_size=G.BATCH_SIZE, shuffle_buffer=G.SHUFFLE_BUFFER_SIZE)
```

1.3 Defining the model architecture

Now that you have a function that will process the data before it is fed into your neural network for training, it is time to define your layer architecture. Just as in last week's assignment you will do the layer definition and compilation in two separate steps. Begin by completing the `create_uncompiled_model` function below.

This is done so you can reuse your model's layers for the learning rate adjusting and the actual training.

Hint:

- `Lambda` layers are not required.
- Use a combination of `Conv1D` and `LSTM` layers followed by `Dense` layers

[8]:
```
def create_uncompiled_model():

    ### START CODE HERE

    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv1D(filters=32, kernel_size=3,
                               strides=1, activation="relu",
                               padding='causal', input_shape=[None, 1]),
        tf.keras.layers.LSTM(64, return_sequences=True),
        tf.keras.layers.LSTM(64),
        tf.keras.layers.Dense(32, activation="relu"),
        tf.keras.layers.Dense(16, activation="relu"),
        tf.keras.layers.Dense(1)
    ])

    ### END CODE HERE

    return model
```

[9]:
```
# Test your uncompiled model
uncompiled_model = create_uncompiled_model()

try:
    uncompiled_model.predict(train_set)
except:
    print("Your current architecture is incompatible with the windowed dataset,␣
 ↪try adjusting it.")
else:
    print("Your current architecture is compatible with the windowed dataset! :
 ↪)")
```

```
Your current architecture is compatible with the windowed dataset! :)
```

1.4 Adjusting the learning rate - (Optional Exercise)

As you saw in the lecture you can leverage Tensorflow's callbacks to dinamically vary the learning rate during training. This can be helpful to get a better sense of which learning rate better acommodates to the problem at hand.

Notice that this is only changing the learning rate during the training process to give you an idea of what a reasonable learning rate is and should not be confused with selecting the best learning rate, this is known as hyperparameter optimization and it is outside the scope of this course.

For the optimizers you can try out:

- tf.keras.optimizers.Adam

- tf.keras.optimizers.SGD with a momentum of 0.9

[10]:
```
def adjust_learning_rate(dataset):

    model = create_uncompiled_model()

    lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 *␣
↪10**(epoch / 20))

    ### START CODE HERE

    # Select your optimizer
    optimizer = tf.keras.optimizers.SGD(lr=1e-8, momentum=0.9)

    # Compile the model passing in the appropriate loss
    model.compile(loss=tf.keras.losses.Huber(),
                  optimizer=optimizer,
                  metrics=["mae"])

    ### END CODE HERE

    history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])

    return history
```

[11]:
```
# Run the training with dynamic LR
lr_history = adjust_learning_rate(train_set)
```

```
Epoch 1/100
77/77 [==============================] - 10s 93ms/step - loss: 10.1973 - mae:
10.6962 - lr: 1.0000e-08
Epoch 2/100
77/77 [==============================] - 7s 89ms/step - loss: 10.1970 - mae:
10.6959 - lr: 1.1220e-08
Epoch 3/100
77/77 [==============================] - 7s 89ms/step - loss: 10.1966 - mae:
10.6956 - lr: 1.2589e-08
Epoch 4/100
77/77 [==============================] - 7s 85ms/step - loss: 10.1962 - mae:
10.6952 - lr: 1.4125e-08
Epoch 5/100
77/77 [==============================] - 7s 96ms/step - loss: 10.1958 - mae:
10.6947 - lr: 1.5849e-08
Epoch 6/100
77/77 [==============================] - 7s 87ms/step - loss: 10.1953 - mae:
10.6942 - lr: 1.7783e-08
Epoch 7/100
77/77 [==============================] - 7s 90ms/step - loss: 10.1947 - mae:
```

```
10.6937 - lr: 1.9953e-08
Epoch 8/100
77/77 [==============================] - 7s 86ms/step - loss: 10.1941 - mae:
10.6930 - lr: 2.2387e-08
Epoch 9/100
77/77 [==============================] - 7s 90ms/step - loss: 10.1934 - mae:
10.6923 - lr: 2.5119e-08
Epoch 10/100
77/77 [==============================] - 6s 84ms/step - loss: 10.1926 - mae:
10.6915 - lr: 2.8184e-08
Epoch 11/100
77/77 [==============================] - 7s 87ms/step - loss: 10.1917 - mae:
10.6906 - lr: 3.1623e-08
Epoch 12/100
77/77 [==============================] - 6s 82ms/step - loss: 10.1906 - mae:
10.6896 - lr: 3.5481e-08
Epoch 13/100
77/77 [==============================] - 7s 84ms/step - loss: 10.1895 - mae:
10.6884 - lr: 3.9811e-08
Epoch 14/100
77/77 [==============================] - 7s 97ms/step - loss: 10.1882 - mae:
10.6871 - lr: 4.4668e-08
Epoch 15/100
77/77 [==============================] - 7s 92ms/step - loss: 10.1868 - mae:
10.6857 - lr: 5.0119e-08
Epoch 16/100
77/77 [==============================] - 7s 90ms/step - loss: 10.1851 - mae:
10.6840 - lr: 5.6234e-08
Epoch 17/100
77/77 [==============================] - 7s 88ms/step - loss: 10.1833 - mae:
10.6822 - lr: 6.3096e-08
Epoch 18/100
77/77 [==============================] - 7s 87ms/step - loss: 10.1812 - mae:
10.6801 - lr: 7.0795e-08
Epoch 19/100
77/77 [==============================] - 7s 89ms/step - loss: 10.1788 - mae:
10.6778 - lr: 7.9433e-08
Epoch 20/100
77/77 [==============================] - 7s 90ms/step - loss: 10.1762 - mae:
10.6751 - lr: 8.9125e-08
Epoch 21/100
77/77 [==============================] - 8s 97ms/step - loss: 10.1732 - mae:
10.6721 - lr: 1.0000e-07
Epoch 22/100
77/77 [==============================] - 7s 92ms/step - loss: 10.1698 - mae:
10.6687 - lr: 1.1220e-07
Epoch 23/100
77/77 [==============================] - 7s 89ms/step - loss: 10.1659 - mae:
```

```
10.6648 - lr: 1.2589e-07
Epoch 24/100
77/77 [==============================] - 7s 93ms/step - loss: 10.1616 - mae:
10.6605 - lr: 1.4125e-07
Epoch 25/100
77/77 [==============================] - 7s 90ms/step - loss: 10.1568 - mae:
10.6557 - lr: 1.5849e-07
Epoch 26/100
77/77 [==============================] - 7s 91ms/step - loss: 10.1514 - mae:
10.6503 - lr: 1.7783e-07
Epoch 27/100
77/77 [==============================] - 7s 92ms/step - loss: 10.1456 - mae:
10.6445 - lr: 1.9953e-07
Epoch 28/100
77/77 [==============================] - 8s 97ms/step - loss: 10.1392 - mae:
10.6381 - lr: 2.2387e-07
Epoch 29/100
77/77 [==============================] - 7s 89ms/step - loss: 10.1325 - mae:
10.6314 - lr: 2.5119e-07
Epoch 30/100
77/77 [==============================] - 7s 88ms/step - loss: 10.1252 - mae:
10.6241 - lr: 2.8184e-07
Epoch 31/100
77/77 [==============================] - 7s 91ms/step - loss: 10.1172 - mae:
10.6161 - lr: 3.1623e-07
Epoch 32/100
77/77 [==============================] - 7s 86ms/step - loss: 10.1087 - mae:
10.6075 - lr: 3.5481e-07
Epoch 33/100
77/77 [==============================] - 7s 88ms/step - loss: 10.0998 - mae:
10.5986 - lr: 3.9811e-07
Epoch 34/100
77/77 [==============================] - 8s 98ms/step - loss: 10.0908 - mae:
10.5896 - lr: 4.4668e-07
Epoch 35/100
77/77 [==============================] - 7s 87ms/step - loss: 10.0817 - mae:
10.5806 - lr: 5.0119e-07
Epoch 36/100
77/77 [==============================] - 7s 89ms/step - loss: 10.0723 - mae:
10.5711 - lr: 5.6234e-07
Epoch 37/100
77/77 [==============================] - 7s 91ms/step - loss: 10.0625 - mae:
10.5613 - lr: 6.3096e-07
Epoch 38/100
77/77 [==============================] - 7s 93ms/step - loss: 10.0526 - mae:
10.5514 - lr: 7.0795e-07
Epoch 39/100
77/77 [==============================] - 7s 94ms/step - loss: 10.0427 - mae:
```

```
10.5415 - lr: 7.9433e-07
Epoch 40/100
77/77 [==============================] - 7s 93ms/step - loss: 10.0329 - mae:
10.5317 - lr: 8.9125e-07
Epoch 41/100
77/77 [==============================] - 7s 87ms/step - loss: 10.0233 - mae:
10.5220 - lr: 1.0000e-06
Epoch 42/100
77/77 [==============================] - 7s 88ms/step - loss: 10.0141 - mae:
10.5128 - lr: 1.1220e-06
Epoch 43/100
77/77 [==============================] - 7s 91ms/step - loss: 10.0050 - mae:
10.5038 - lr: 1.2589e-06
Epoch 44/100
77/77 [==============================] - 7s 95ms/step - loss: 9.9957 - mae:
10.4945 - lr: 1.4125e-06
Epoch 45/100
77/77 [==============================] - 8s 103ms/step - loss: 9.9858 - mae:
10.4846 - lr: 1.5849e-06
Epoch 46/100
77/77 [==============================] - 7s 90ms/step - loss: 9.9750 - mae:
10.4737 - lr: 1.7783e-06
Epoch 47/100
77/77 [==============================] - 7s 90ms/step - loss: 9.9630 - mae:
10.4617 - lr: 1.9953e-06
Epoch 48/100
77/77 [==============================] - 7s 94ms/step - loss: 9.9494 - mae:
10.4480 - lr: 2.2387e-06
Epoch 49/100
77/77 [==============================] - 7s 92ms/step - loss: 9.9334 - mae:
10.4321 - lr: 2.5119e-06
Epoch 50/100
77/77 [==============================] - 7s 90ms/step - loss: 9.9153 - mae:
10.4139 - lr: 2.8184e-06
Epoch 51/100
77/77 [==============================] - 7s 88ms/step - loss: 9.8952 - mae:
10.3938 - lr: 3.1623e-06
Epoch 52/100
77/77 [==============================] - 7s 89ms/step - loss: 9.8728 - mae:
10.3714 - lr: 3.5481e-06
Epoch 53/100
77/77 [==============================] - 7s 89ms/step - loss: 9.8476 - mae:
10.3462 - lr: 3.9811e-06
Epoch 54/100
77/77 [==============================] - 7s 92ms/step - loss: 9.8189 - mae:
10.3176 - lr: 4.4668e-06
Epoch 55/100
77/77 [==============================] - 7s 94ms/step - loss: 9.7862 - mae:
```

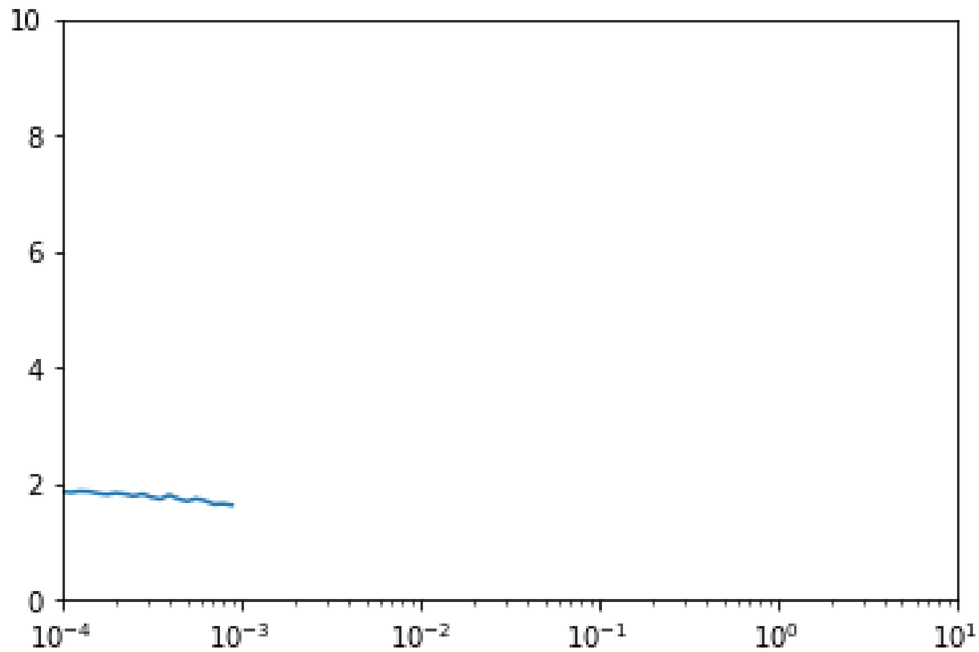
```
10.2849 - lr: 5.0119e-06
Epoch 56/100
77/77 [==============================] - 7s 95ms/step - loss: 9.7487 - mae:
10.2473 - lr: 5.6234e-06
Epoch 57/100
77/77 [==============================] - 7s 92ms/step - loss: 9.7053 - mae:
10.2038 - lr: 6.3096e-06
Epoch 58/100
77/77 [==============================] - 7s 94ms/step - loss: 9.6549 - mae:
10.1534 - lr: 7.0795e-06
Epoch 59/100
77/77 [==============================] - 7s 88ms/step - loss: 9.5960 - mae:
10.0944 - lr: 7.9433e-06
Epoch 60/100
77/77 [==============================] - 7s 93ms/step - loss: 9.5261 - mae:
10.0244 - lr: 8.9125e-06
Epoch 61/100
77/77 [==============================] - 7s 95ms/step - loss: 9.4423 - mae:
9.9406 - lr: 1.0000e-05
Epoch 62/100
77/77 [==============================] - 7s 94ms/step - loss: 9.3413 - mae:
9.8396 - lr: 1.1220e-05
Epoch 63/100
77/77 [==============================] - 6s 84ms/step - loss: 9.2179 - mae:
9.7163 - lr: 1.2589e-05
Epoch 64/100
77/77 [==============================] - 7s 89ms/step - loss: 9.0646 - mae:
9.5629 - lr: 1.4125e-05
Epoch 65/100
77/77 [==============================] - 7s 92ms/step - loss: 8.8694 - mae:
9.3676 - lr: 1.5849e-05
Epoch 66/100
77/77 [==============================] - 8s 99ms/step - loss: 8.6145 - mae:
9.1126 - lr: 1.7783e-05
Epoch 67/100
77/77 [==============================] - 7s 97ms/step - loss: 8.2784 - mae:
8.7764 - lr: 1.9953e-05
Epoch 68/100
77/77 [==============================] - 8s 106ms/step - loss: 7.8323 - mae:
8.3284 - lr: 2.2387e-05
Epoch 69/100
77/77 [==============================] - 8s 101ms/step - loss: 7.2203 - mae:
7.7159 - lr: 2.5119e-05
Epoch 70/100
77/77 [==============================] - 8s 100ms/step - loss: 6.3570 - mae:
6.8487 - lr: 2.8184e-05
Epoch 71/100
77/77 [==============================] - 8s 98ms/step - loss: 5.1580 - mae:
```

```
5.6436 - lr: 3.1623e-05
Epoch 72/100
77/77 [==============================] - 8s 98ms/step - loss: 3.7843 - mae:
4.2598 - lr: 3.5481e-05
Epoch 73/100
77/77 [==============================] - 8s 99ms/step - loss: 2.8467 - mae:
3.3131 - lr: 3.9811e-05
Epoch 74/100
77/77 [==============================] - 8s 101ms/step - loss: 2.5976 - mae:
3.0619 - lr: 4.4668e-05
Epoch 75/100
77/77 [==============================] - 8s 100ms/step - loss: 2.5334 - mae:
2.9968 - lr: 5.0119e-05
Epoch 76/100
77/77 [==============================] - 8s 98ms/step - loss: 2.4229 - mae:
2.8843 - lr: 5.6234e-05
Epoch 77/100
77/77 [==============================] - 8s 101ms/step - loss: 2.2456 - mae:
2.7046 - lr: 6.3096e-05
Epoch 78/100
77/77 [==============================] - 8s 104ms/step - loss: 2.0371 - mae:
2.4929 - lr: 7.0795e-05
Epoch 79/100
77/77 [==============================] - 8s 108ms/step - loss: 1.9567 - mae:
2.4092 - lr: 7.9433e-05
Epoch 80/100
77/77 [==============================] - 8s 97ms/step - loss: 1.9042 - mae:
2.3550 - lr: 8.9125e-05
Epoch 81/100
77/77 [==============================] - 8s 98ms/step - loss: 1.8573 - mae:
2.3085 - lr: 1.0000e-04
Epoch 82/100
77/77 [==============================] - 8s 97ms/step - loss: 1.8414 - mae:
2.2932 - lr: 1.1220e-04
Epoch 83/100
77/77 [==============================] - 8s 98ms/step - loss: 1.8640 - mae:
2.3161 - lr: 1.2589e-04
Epoch 84/100
77/77 [==============================] - 8s 102ms/step - loss: 1.8569 - mae:
2.3075 - lr: 1.4125e-04
Epoch 85/100
77/77 [==============================] - 8s 104ms/step - loss: 1.8270 - mae:
2.2774 - lr: 1.5849e-04
Epoch 86/100
77/77 [==============================] - 8s 99ms/step - loss: 1.8052 - mae:
2.2560 - lr: 1.7783e-04
Epoch 87/100
77/77 [==============================] - 8s 97ms/step - loss: 1.8327 - mae:
```

```
2.2853 - lr: 1.9953e-04
Epoch 88/100
77/77 [==============================] - 8s 100ms/step - loss: 1.8143 - mae:
2.2657 - lr: 2.2387e-04
Epoch 89/100
77/77 [==============================] - 8s 99ms/step - loss: 1.7792 - mae:
2.2286 - lr: 2.5119e-04
Epoch 90/100
77/77 [==============================] - 8s 98ms/step - loss: 1.8151 - mae:
2.2666 - lr: 2.8184e-04
Epoch 91/100
77/77 [==============================] - 8s 98ms/step - loss: 1.7591 - mae:
2.2082 - lr: 3.1623e-04
Epoch 92/100
77/77 [==============================] - 7s 95ms/step - loss: 1.7267 - mae:
2.1796 - lr: 3.5481e-04
Epoch 93/100
77/77 [==============================] - 7s 96ms/step - loss: 1.7971 - mae:
2.2468 - lr: 3.9811e-04
Epoch 94/100
77/77 [==============================] - 8s 99ms/step - loss: 1.7301 - mae:
2.1796 - lr: 4.4668e-04
Epoch 95/100
77/77 [==============================] - 8s 101ms/step - loss: 1.6888 - mae:
2.1329 - lr: 5.0119e-04
Epoch 96/100
77/77 [==============================] - 8s 97ms/step - loss: 1.7359 - mae:
2.1857 - lr: 5.6234e-04
Epoch 97/100
77/77 [==============================] - 8s 102ms/step - loss: 1.6973 - mae:
2.1475 - lr: 6.3096e-04
Epoch 98/100
77/77 [==============================] - 8s 107ms/step - loss: 1.6361 - mae:
2.0861 - lr: 7.0795e-04
Epoch 99/100
77/77 [==============================] - 9s 111ms/step - loss: 1.6461 - mae:
2.0941 - lr: 7.9433e-04
Epoch 100/100
77/77 [==============================] - 8s 106ms/step - loss: 1.6232 - mae:
2.0704 - lr: 8.9125e-04
```

[12]:
```
plt.semilogx(lr_history.history["lr"], lr_history.history["loss"])
plt.axis([1e-4, 10, 0, 10])
```

[12]:
```
(0.0001, 10.0, 0.0, 10.0)
```



1.5 Compiling the model

Now that you have trained the model while varying the learning rate, it is time to do the actual training that will be used to forecast the time series. For this complete the `create_model` function below.

Notice that you are reusing the architecture you defined in the `create_uncompiled_model` earlier. Now you only need to compile this model using the appropriate loss, optimizer (and learning rate).

Hints:

- The training should be really quick so if you notice that each epoch is taking more than a few seconds, consider trying a different architecture.
- If after the first epoch you get an output like this: loss: nan - mae: nan it is very likely that your network is suffering from exploding gradients. This is a common problem if you used SGD as optimizer and set a learning rate that is too high. If you encounter this problem consider lowering the learning rate or using Adam with the default learning rate.

[13]:
```
def create_model():


    model = create_uncompiled_model()

    ### START CODE HERE

    model.compile(loss=tf.keras.losses.Huber(),
```

```
                    optimizer=tf.keras.optimizers.SGD(learning_rate=1e-3,␣
↪momentum=0.9),
                    metrics=["mae"])


    ### END CODE HERE

    return model
```

[14]:
```
# Save an instance of the model
model = create_model()

# Train it
history = model.fit(train_set, epochs=50)
```

```
Epoch 1/50
77/77 [==============================] - 11s 98ms/step - loss: 7.2255 - mae:
7.7143
Epoch 2/50
77/77 [==============================] - 7s 96ms/step - loss: 2.2916 - mae:
2.7523
Epoch 3/50
77/77 [==============================] - 7s 95ms/step - loss: 1.9018 - mae:
2.3542
Epoch 4/50
77/77 [==============================] - 8s 100ms/step - loss: 1.8559 - mae:
2.3064
Epoch 5/50
77/77 [==============================] - 8s 98ms/step - loss: 1.7805 - mae:
2.2305
Epoch 6/50
77/77 [==============================] - 7s 95ms/step - loss: 1.7804 - mae:
2.2294
Epoch 7/50
77/77 [==============================] - 7s 95ms/step - loss: 1.7615 - mae:
2.2143
Epoch 8/50
77/77 [==============================] - 8s 99ms/step - loss: 1.6718 - mae:
2.1230
Epoch 9/50
77/77 [==============================] - 7s 96ms/step - loss: 1.6649 - mae:
2.1148
Epoch 10/50
77/77 [==============================] - 8s 99ms/step - loss: 1.6227 - mae:
2.0636
Epoch 11/50
77/77 [==============================] - 8s 97ms/step - loss: 1.6284 - mae:
```

```
2.0751
Epoch 12/50
77/77 [==============================] - 8s 99ms/step - loss: 1.6286 - mae:
2.0800
Epoch 13/50
77/77 [==============================] - 7s 96ms/step - loss: 1.5416 - mae:
1.9824
Epoch 14/50
77/77 [==============================] - 8s 98ms/step - loss: 1.5560 - mae:
2.0035
Epoch 15/50
77/77 [==============================] - 7s 97ms/step - loss: 1.5619 - mae:
2.0057
Epoch 16/50
77/77 [==============================] - 8s 99ms/step - loss: 1.5215 - mae:
1.9671
Epoch 17/50
77/77 [==============================] - 8s 99ms/step - loss: 1.5501 - mae:
1.9944
Epoch 18/50
77/77 [==============================] - 8s 99ms/step - loss: 1.5458 - mae:
1.9884
Epoch 19/50
77/77 [==============================] - 8s 102ms/step - loss: 1.5385 - mae:
1.9822
Epoch 20/50
77/77 [==============================] - 8s 101ms/step - loss: 1.5797 - mae:
2.0252
Epoch 21/50
77/77 [==============================] - 8s 102ms/step - loss: 1.5461 - mae:
1.9883
Epoch 22/50
77/77 [==============================] - 8s 99ms/step - loss: 1.5962 - mae:
2.0417
Epoch 23/50
77/77 [==============================] - 7s 97ms/step - loss: 1.5450 - mae:
1.9864
Epoch 24/50
77/77 [==============================] - 7s 92ms/step - loss: 1.5262 - mae:
1.9674
Epoch 25/50
77/77 [==============================] - 7s 95ms/step - loss: 1.4977 - mae:
1.9373
Epoch 26/50
77/77 [==============================] - 7s 88ms/step - loss: 1.5283 - mae:
1.9723
Epoch 27/50
77/77 [==============================] - 7s 92ms/step - loss: 1.5332 - mae:
```

```
1.9772
Epoch 28/50
77/77 [==============================] - 7s 92ms/step - loss: 1.5017 - mae:
1.9426
Epoch 29/50
77/77 [==============================] - 7s 91ms/step - loss: 1.5226 - mae:
1.9621
Epoch 30/50
77/77 [==============================] - 7s 90ms/step - loss: 1.5066 - mae:
1.9460
Epoch 31/50
77/77 [==============================] - 7s 94ms/step - loss: 1.5265 - mae:
1.9670
Epoch 32/50
77/77 [==============================] - 7s 92ms/step - loss: 1.5150 - mae:
1.9567
Epoch 33/50
77/77 [==============================] - 7s 89ms/step - loss: 1.5136 - mae:
1.9566
Epoch 34/50
77/77 [==============================] - 7s 92ms/step - loss: 1.5146 - mae:
1.9542
Epoch 35/50
77/77 [==============================] - 7s 89ms/step - loss: 1.5073 - mae:
1.9509
Epoch 36/50
77/77 [==============================] - 7s 90ms/step - loss: 1.5312 - mae:
1.9709
Epoch 37/50
77/77 [==============================] - 7s 87ms/step - loss: 1.4871 - mae:
1.9285
Epoch 38/50
77/77 [==============================] - 7s 90ms/step - loss: 1.5331 - mae:
1.9739
Epoch 39/50
77/77 [==============================] - 8s 98ms/step - loss: 1.4891 - mae:
1.9302
Epoch 40/50
77/77 [==============================] - 8s 99ms/step - loss: 1.4897 - mae:
1.9269
Epoch 41/50
77/77 [==============================] - 7s 90ms/step - loss: 1.4891 - mae:
1.9313
Epoch 42/50
77/77 [==============================] - 8s 101ms/step - loss: 1.5239 - mae:
1.9645
Epoch 43/50
77/77 [==============================] - 8s 99ms/step - loss: 1.5119 - mae:
```

```
1.9508
Epoch 44/50
77/77 [==============================] - 8s 98ms/step - loss: 1.4866 - mae:
1.9301
Epoch 45/50
77/77 [==============================] - 7s 93ms/step - loss: 1.5046 - mae:
1.9463
Epoch 46/50
77/77 [==============================] - 7s 94ms/step - loss: 1.4793 - mae:
1.9172
Epoch 47/50
77/77 [==============================] - 7s 95ms/step - loss: 1.5098 - mae:
1.9493
Epoch 48/50
77/77 [==============================] - 7s 96ms/step - loss: 1.5039 - mae:
1.9461
Epoch 49/50
77/77 [==============================] - 8s 97ms/step - loss: 1.5191 - mae:
1.9597
Epoch 50/50
77/77 [==============================] - 7s 94ms/step - loss: 1.4855 - mae:
1.9254
```

1.6 Evaluating the forecast

Now it is time to evaluate the performance of the forecast. For this you can use the `compute_metrics` function that you coded in a previous assignment:

[15]:
```
def compute_metrics(true_series, forecast):

    mse = tf.keras.metrics.mean_squared_error(true_series, forecast).numpy()
    mae = tf.keras.metrics.mean_absolute_error(true_series, forecast).numpy()

    return mse, mae
```

At this point only the model that will perform the forecast is ready but you still need to compute the actual forecast.

1.7 Faster model forecasts

In the previous week you saw a faster approach compared to using a for loop to compute the forecasts for every point in the sequence. Remember that this faster approach uses batches of data.

The code to implement this is provided in the `model_forecast` below. Notice that the code is very similar to the one in the `windowed_dataset` function with the differences that: - The dataset is windowed using `window_size` rather than `window_size + 1` - No shuffle should be used - No need to split the data into features and labels - A model is used to predict batches of the dataset

[16]:
```python
def model_forecast(model, series, window_size):
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size))
    ds = ds.batch(32).prefetch(1)
    forecast = model.predict(ds)
    return forecast
```

Now compute the actual forecast:

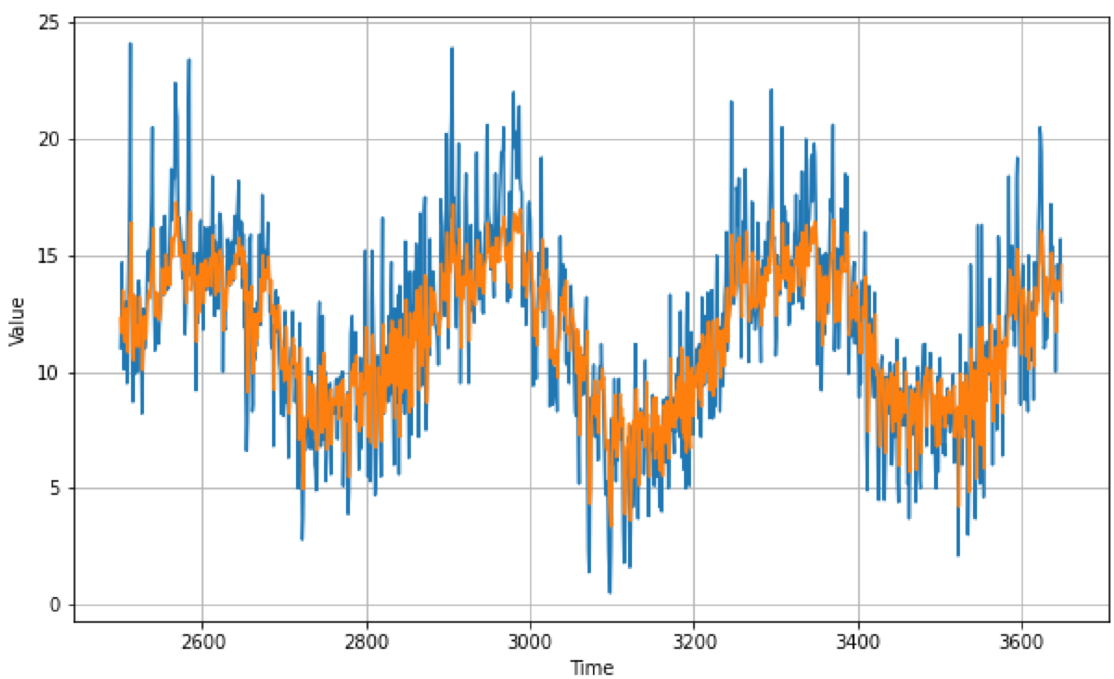
Note: Don't modify the cell below.

The grader uses the same slicing to get the forecast so if you change the cell below you risk having issues when submitting your model for grading.

[17]:
```python
# Compute the forecast for all the series
rnn_forecast = model_forecast(model, G.SERIES, G.WINDOW_SIZE).squeeze()

# Slice the forecast to get only the predictions for the validation set
rnn_forecast = rnn_forecast[G.SPLIT_TIME - G.WINDOW_SIZE:-1]

# Plot the forecast
plt.figure(figsize=(10, 6))
plot_series(time_valid, series_valid)
plot_series(time_valid, rnn_forecast)
```



[18]:
```
mse, mae = compute_metrics(series_valid, rnn_forecast)

print(f"mse: {mse:.2f}, mae: {mae:.2f} for forecast")
```

```
mse: 5.42, mae: 1.82 for forecast
```

To pass this assignment your forecast should achieve a MSE of 6 or less and a MAE of 2 or less.

- If your forecast didn't achieve this threshold try re-training your model with a different architecture (you will need to re-run both `create_uncompiled_model` and `create_model` functions) or tweaking the optimizer's parameters.
- If your forecast did achieve this threshold run the following cell to save the model in the SavedModel format which will be used for grading and after doing so, submit your assigment for grading.
- This environment includes a dummy SavedModel directory which contains a dummy model trained for one epoch. **To replace this file with your actual model you need to run the next cell before submitting for grading.**

[19]:
```
# Save your model in the SavedModel format
model.save('saved_model/my_model')

# Compress the directory using tar
! tar -czvf saved_model.tar.gz saved_model/
```

```
INFO:tensorflow:Assets written to: saved_model/my_model/assets
```

```
INFO:tensorflow:Assets written to: saved_model/my_model/assets
```

```
saved_model/
saved_model/my_model/
saved_model/my_model/keras_metadata.pb
saved_model/my_model/variables/
saved_model/my_model/variables/variables.data-00000-of-00001
saved_model/my_model/variables/variables.index
saved_model/my_model/saved_model.pb
saved_model/my_model/assets/
```

Congratulations on finishing this week's assignment!

You have successfully implemented a neural network capable of forecasting time series leveraging a combination of Tensorflow's layers such as Convolutional and LSTMs! This resulted in a forecast that surpasses all the ones you did previously.

By finishing this assignment you have finished the specialization! Give yourself a pat on the back!!!

Please click here if you want to experiment with any of the non-graded code.

Important Note: Please only do this when you've already passed the assignment to avoid problems with the autograder.

On the notebook's menu, click "View" > "Cell Toolbar" > "Edit Metadata"

Hit the "Edit Metadata" button next to the code cell which you want to lock/unlock

Set the attribute value for "editable" to:

"true" if you want to unlock it

"false" if you want to lock it

```
    </li>
    <li> On the notebook's menu, click "View" > "Cell Toolbar" > "None" </li>
</ol>
<p> Here's a short demo of how to do the steps above:
    <br>
    <img src="https://drive.google.com/uc?export=view&id=14Xy_Mb17CZVgzVAgq7NCjMVBvSae3xO1" al
```