Assignment 5

November 10, 2021

0.1 Assignment 5

Nick Reardon

0.1.1 Assignment 5.1

Implement the movie review classifier found in section 3.4 of Deep Learning with Python

[1]:
```
# Import libraries
import numpy as np

# Load the imdb dataset
from keras.datasets import imdb

# turn off warnings
np.warnings.filterwarnings('ignore', category = np.VisibleDeprecationWarning)
```

[2]:
```
(train_data, train_labels), (test_data, test_labels) = imdb.
 ↪load_data(num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/imdb.npz
17465344/17464789 [==============================] - 1s 0us/step
```

[4]:
```
word_index = imdb.get_word_index()
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])
decoded_review = ' '.join(
    [reverse_word_index.get(i - 3, '?') for i in train_data[0]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/imdb_word_index.json
1646592/1641221 [==============================] - 0s 0us/step
```

[5]:
```
# 3.2

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
```

```
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

x_train[0]
```

[5]: array([0., 1., 1., …, 0., 0., 0.])

[6]:
```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

[7]:
```
# 3.3

from keras import models
from keras import layers
```

[8]:
```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

[9]:
```
# 3.4

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[10]:
```
# 3.5

from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[11]:
```
# 3.6

from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

[12]:
```
# 3.7

x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

[13]:
```
# 3.8

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

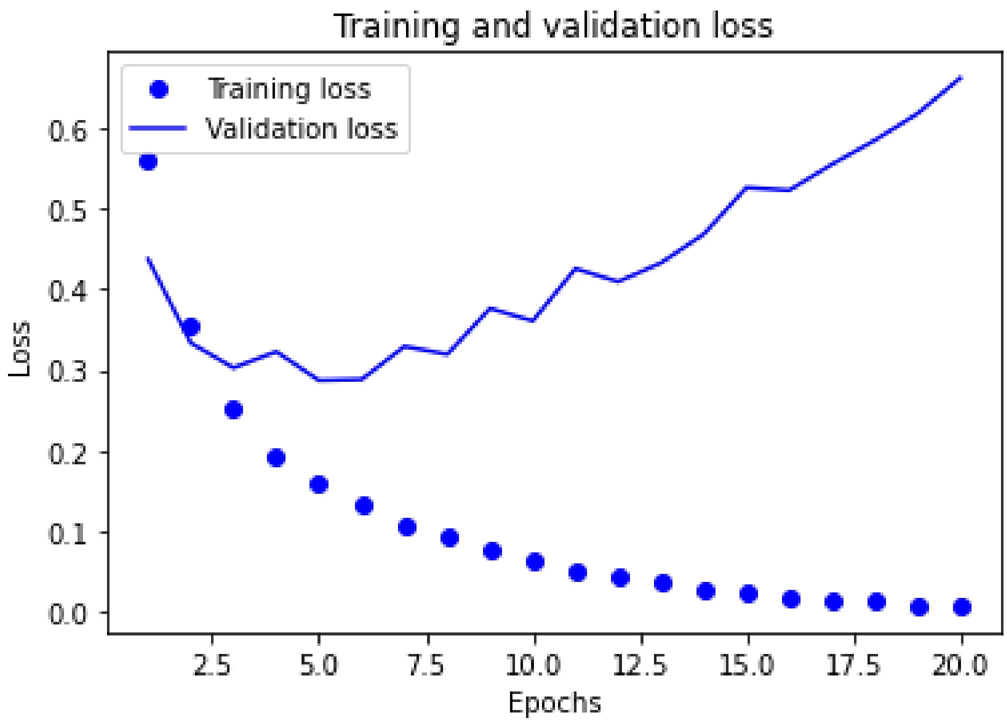
```
Epoch 1/20
30/30 [==============================] - 2s 51ms/step - loss: 0.6200 - acc:
0.6885 - val_loss: 0.4377 - val_acc: 0.8667
Epoch 2/20
30/30 [==============================] - 0s 13ms/step - loss: 0.3802 - acc:
0.8958 - val_loss: 0.3334 - val_acc: 0.8846
Epoch 3/20
30/30 [==============================] - 0s 13ms/step - loss: 0.2619 - acc:
0.9261 - val_loss: 0.3026 - val_acc: 0.8845
Epoch 4/20
30/30 [==============================] - 0s 13ms/step - loss: 0.1968 - acc:
0.9407 - val_loss: 0.3227 - val_acc: 0.8686
Epoch 5/20
30/30 [==============================] - 0s 12ms/step - loss: 0.1628 - acc:
0.9506 - val_loss: 0.2871 - val_acc: 0.8835
Epoch 6/20
30/30 [==============================] - 0s 12ms/step - loss: 0.1289 - acc:
0.9630 - val_loss: 0.2883 - val_acc: 0.8882
Epoch 7/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1034 - acc:
0.9721 - val_loss: 0.3292 - val_acc: 0.8808
Epoch 8/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0910 - acc:
0.9764 - val_loss: 0.3193 - val_acc: 0.8826
Epoch 9/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0731 - acc:
0.9799 - val_loss: 0.3763 - val_acc: 0.8759
Epoch 10/20
```

```
30/30 [==============================] - 0s 12ms/step - loss: 0.0615 - acc:
0.9857 - val_loss: 0.3608 - val_acc: 0.8805
Epoch 11/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0470 - acc:
0.9908 - val_loss: 0.4258 - val_acc: 0.8727
Epoch 12/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0412 - acc:
0.9932 - val_loss: 0.4096 - val_acc: 0.8731
Epoch 13/20
30/30 [==============================] - 1s 17ms/step - loss: 0.0339 - acc:
0.9943 - val_loss: 0.4329 - val_acc: 0.8752
Epoch 14/20
30/30 [==============================] - 1s 21ms/step - loss: 0.0256 - acc:
0.9970 - val_loss: 0.4686 - val_acc: 0.8763
Epoch 15/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0202 - acc:
0.9979 - val_loss: 0.5265 - val_acc: 0.8615
Epoch 16/20
30/30 [==============================] - 0s 14ms/step - loss: 0.0177 - acc:
0.9980 - val_loss: 0.5231 - val_acc: 0.8688
Epoch 17/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0145 - acc:
0.9984 - val_loss: 0.5554 - val_acc: 0.8715
Epoch 18/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0106 - acc:
0.9986 - val_loss: 0.5851 - val_acc: 0.8684
Epoch 19/20
30/30 [==============================] - 0s 15ms/step - loss: 0.0080 - acc:
0.9993 - val_loss: 0.6184 - val_acc: 0.8657
Epoch 20/20
30/30 [==============================] - 0s 14ms/step - loss: 0.0075 - acc:
0.9992 - val_loss: 0.6626 - val_acc: 0.8617
```

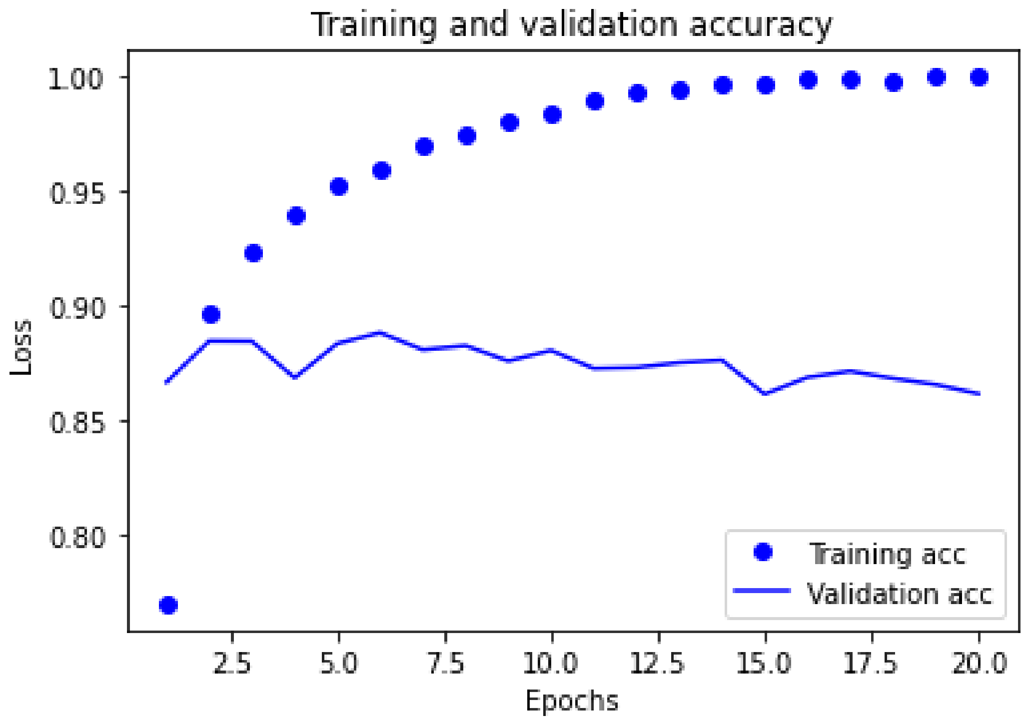
[14]:

```python
# 3.9

import matplotlib.pyplot as plt

acc = history.history['acc']
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
```

```
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



[18]:
```
# 3.10

plt.clf()
acc = history.history['acc']
val_acc = history_dict['val_acc']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



[16]:
```
# 3.11

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 1s 10ms/step - loss: 0.5722 - accuracy:
0.7440
Epoch 2/4
49/49 [==============================] - 0s 8ms/step - loss: 0.3002 - accuracy:
0.9116
Epoch 3/4
49/49 [==============================] - 0s 8ms/step - loss: 0.2129 - accuracy:
0.9283
Epoch 4/4
```

```
49/49 [==============================] - 0s 7ms/step - loss: 0.1708 - accuracy:
0.9437
782/782 [==============================] - 2s 2ms/step - loss: 0.2867 -
accuracy: 0.8862
```

[17]:
```
results
```

[17]:
```
[0.28666865825653076, 0.8861600160598755]
```

[24]:
```
predictions = model.predict(x_test)
```

[26]:
```
model.predict(x_test)
```

[26]:
```
array([[0.182661  ],
       [0.99820447],
       [0.78975165],
       …,
       [0.14726576],
       [0.0808821 ],
       [0.7332363 ]], dtype=float32)
```

0.1.2 5.2

[27]:
```
# 3.12
# The Reuters dataset
from keras.datasets import reuters

(train_data, train_labels), (test_data, test_labels) = reuters.load_data(
    num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/reuters.npz
2113536/2110848 [==============================] - 0s 0us/step
```

[28]:
```
# 3.13

word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_newswire = ' '.join([reverse_word_index.get(i - 3, '?') for i in
    train_data[0]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/reuters_word_index.json
557056/550378 [==============================] - 0s 0us/step
```

[29]:
```
# 3.14

import numpy as np
```

```python
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

[30]:
```python
def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results

one_hot_train_labels = to_one_hot(train_labels)
one_hot_test_labels = to_one_hot(test_labels)
```

[31]:
```python
from keras.utils.np_utils import to_categorical

one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

[32]:
```python
# 3.15

from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
```

[33]:
```python
# 3.16

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

[34]:
```python
# 3.17

x_val = x_train[:1000]
partial_x_train = x_train[1000:]

y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
```

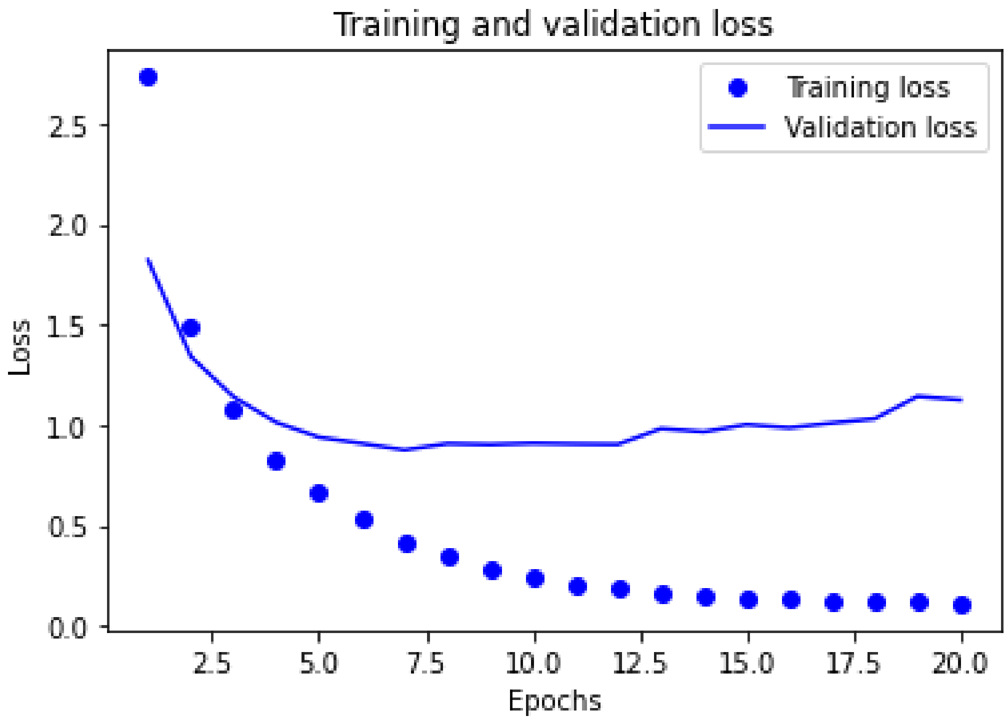
[35]:
```python
# 3.18

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
16/16 [==============================] - 1s 31ms/step - loss: 3.2064 - accuracy:
0.3901 - val_loss: 1.8208 - val_accuracy: 0.6220
Epoch 2/20
16/16 [==============================] - 0s 17ms/step - loss: 1.6155 - accuracy:
0.6755 - val_loss: 1.3416 - val_accuracy: 0.6980
Epoch 3/20
16/16 [==============================] - 0s 17ms/step - loss: 1.1392 - accuracy:
0.7593 - val_loss: 1.1399 - val_accuracy: 0.7530
Epoch 4/20
16/16 [==============================] - 0s 16ms/step - loss: 0.8646 - accuracy:
0.8182 - val_loss: 1.0127 - val_accuracy: 0.7950
Epoch 5/20
16/16 [==============================] - 0s 18ms/step - loss: 0.6665 - accuracy:
0.8610 - val_loss: 0.9389 - val_accuracy: 0.8100
Epoch 6/20
16/16 [==============================] - 0s 20ms/step - loss: 0.5285 - accuracy:
0.8887 - val_loss: 0.9078 - val_accuracy: 0.8110
Epoch 7/20
16/16 [==============================] - 0s 22ms/step - loss: 0.4193 - accuracy:
0.9172 - val_loss: 0.8754 - val_accuracy: 0.8190
Epoch 8/20
16/16 [==============================] - 0s 15ms/step - loss: 0.3430 - accuracy:
0.9278 - val_loss: 0.9071 - val_accuracy: 0.8070
Epoch 9/20
16/16 [==============================] - 0s 15ms/step - loss: 0.2747 - accuracy:
0.9391 - val_loss: 0.9029 - val_accuracy: 0.8070
Epoch 10/20
16/16 [==============================] - 0s 16ms/step - loss: 0.2373 - accuracy:
0.9471 - val_loss: 0.9087 - val_accuracy: 0.8120
Epoch 11/20
16/16 [==============================] - 0s 16ms/step - loss: 0.2104 - accuracy:
0.9496 - val_loss: 0.9055 - val_accuracy: 0.8170
Epoch 12/20
16/16 [==============================] - 0s 18ms/step - loss: 0.1719 - accuracy:
0.9552 - val_loss: 0.9038 - val_accuracy: 0.8070
Epoch 13/20
16/16 [==============================] - 0s 17ms/step - loss: 0.1515 - accuracy:
0.9558 - val_loss: 0.9821 - val_accuracy: 0.7990
```

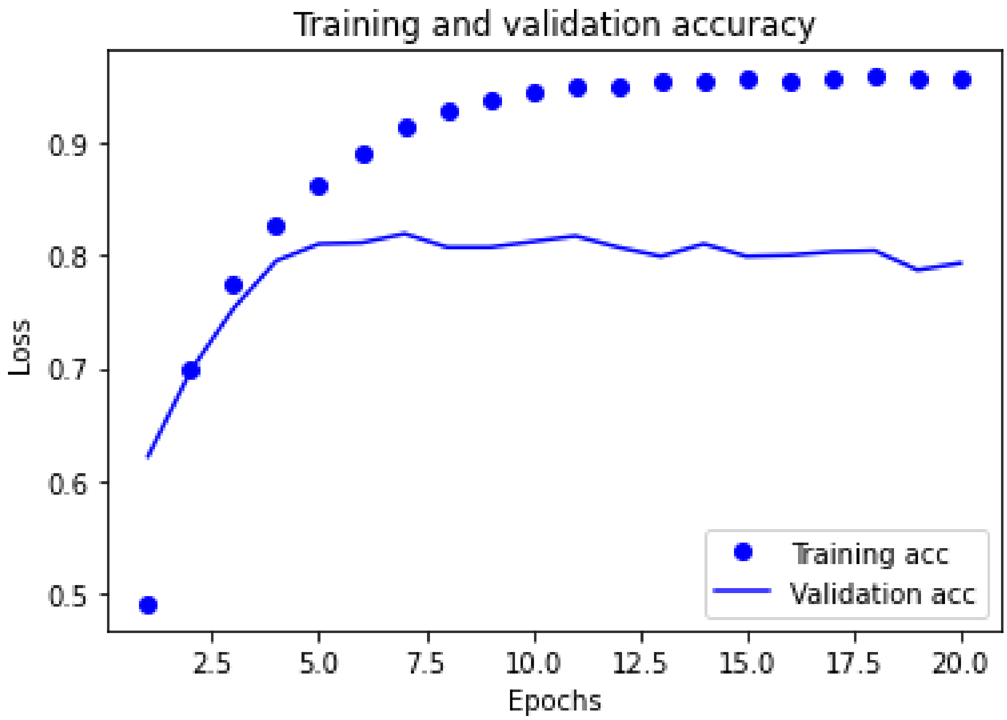
```
Epoch 14/20
16/16 [==============================] - 0s 17ms/step - loss: 0.1489 - accuracy:
0.9566 - val_loss: 0.9660 - val_accuracy: 0.8100
Epoch 15/20
16/16 [==============================] - 0s 16ms/step - loss: 0.1259 - accuracy:
0.9597 - val_loss: 1.0022 - val_accuracy: 0.7990
Epoch 16/20
16/16 [==============================] - 0s 18ms/step - loss: 0.1266 - accuracy:
0.9585 - val_loss: 0.9860 - val_accuracy: 0.8000
Epoch 17/20
16/16 [==============================] - 0s 16ms/step - loss: 0.1143 - accuracy:
0.9606 - val_loss: 1.0097 - val_accuracy: 0.8030
Epoch 18/20
16/16 [==============================] - 0s 16ms/step - loss: 0.1118 - accuracy:
0.9616 - val_loss: 1.0309 - val_accuracy: 0.8040
Epoch 19/20
16/16 [==============================] - 0s 17ms/step - loss: 0.1045 - accuracy:
0.9599 - val_loss: 1.1427 - val_accuracy: 0.7870
Epoch 20/20
16/16 [==============================] - 0s 18ms/step - loss: 0.1016 - accuracy:
0.9608 - val_loss: 1.1248 - val_accuracy: 0.7930
```

[36]:

```
# 3.19

import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



[38]:
```
# 3.20
plt.clf()

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



[39]:
```
# 3.21

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(partial_x_train,
          partial_y_train,
          epochs=9,
          batch_size=512,
          validation_data=(x_val, y_val))
results = model.evaluate(x_test, one_hot_test_labels)
```

```
Epoch 1/9
16/16 [==============================] - 1s 28ms/step - loss: 3.1996 - accuracy:
0.4276 - val_loss: 1.7330 - val_accuracy: 0.6440
Epoch 2/9
16/16 [==============================] - 0s 16ms/step - loss: 1.5320 - accuracy:
0.6843 - val_loss: 1.3029 - val_accuracy: 0.7080
Epoch 3/9
```

```
16/16 [==============================] - 0s 16ms/step - loss: 1.0870 - accuracy:
0.7708 - val_loss: 1.1422 - val_accuracy: 0.7580
Epoch 4/9
16/16 [==============================] - 0s 17ms/step - loss: 0.8440 - accuracy:
0.8264 - val_loss: 1.0292 - val_accuracy: 0.7920
Epoch 5/9
16/16 [==============================] - 0s 16ms/step - loss: 0.6766 - accuracy:
0.8645 - val_loss: 0.9897 - val_accuracy: 0.7910
Epoch 6/9
16/16 [==============================] - 0s 16ms/step - loss: 0.5482 - accuracy:
0.8887 - val_loss: 0.9277 - val_accuracy: 0.8140
Epoch 7/9
16/16 [==============================] - 0s 16ms/step - loss: 0.4238 - accuracy:
0.9180 - val_loss: 0.8841 - val_accuracy: 0.8150
Epoch 8/9
16/16 [==============================] - 0s 17ms/step - loss: 0.3438 - accuracy:
0.9296 - val_loss: 0.8904 - val_accuracy: 0.8210
Epoch 9/9
16/16 [==============================] - 0s 22ms/step - loss: 0.2860 - accuracy:
0.9397 - val_loss: 0.8823 - val_accuracy: 0.8170
71/71 [==============================] - 0s 2ms/step - loss: 0.9861 - accuracy:
0.7898
```

[40]:
```python
results
```

[40]: [0.9860702753067017, 0.7898486256599426]

[41]:
```python
import copy
test_labels_copy = copy.copy(test_labels)
np.random.shuffle(test_labels_copy)
hits_array = np.array(test_labels) == np.array(test_labels_copy)
float(np.sum(hits_array)) / len(test_labels)
```

[41]: 0.19278717720391808

[42]:
```python
# 3.22

predictions = model.predict(x_test)
```

[43]:
```python
predictions[0].shape
```

[43]: (46,)

[44]:
```python
np.sum(predictions[0])
```

[44]: 1.0

[45]:
```
np.argmax(predictions[0])
```

[45]: 3

[46]:
```
y_train = np.array(train_labels)
y_test = np.array(test_labels)
```

[47]:
```
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])
```

[48]:
```
# 3.23

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(partial_x_train,
          partial_y_train,
          epochs=20,
          batch_size=128,
          validation_data=(x_val, y_val))
```

```
Epoch 1/20
63/63 [==============================] - 1s 10ms/step - loss: 3.3822 - accuracy:
0.1589 - val_loss: 2.3247 - val_accuracy: 0.5400
Epoch 2/20
63/63 [==============================] - 0s 7ms/step - loss: 2.0866 - accuracy:
0.5668 - val_loss: 1.7974 - val_accuracy: 0.5810
Epoch 3/20
63/63 [==============================] - 0s 8ms/step - loss: 1.6455 - accuracy:
0.6008 - val_loss: 1.6569 - val_accuracy: 0.5880
Epoch 4/20
63/63 [==============================] - 0s 7ms/step - loss: 1.4947 - accuracy:
0.6036 - val_loss: 1.5632 - val_accuracy: 0.5900
Epoch 5/20
63/63 [==============================] - 0s 7ms/step - loss: 1.3265 - accuracy:
0.6202 - val_loss: 1.4852 - val_accuracy: 0.5940
Epoch 6/20
63/63 [==============================] - 0s 7ms/step - loss: 1.2048 - accuracy:
0.6343 - val_loss: 1.4753 - val_accuracy: 0.6160
Epoch 7/20
63/63 [==============================] - 0s 7ms/step - loss: 1.1261 - accuracy:
```

```
0.6673 - val_loss: 1.4440 - val_accuracy: 0.6310
Epoch 8/20
63/63 [==============================] - 0s 7ms/step - loss: 1.0653 - accuracy:
0.6924 - val_loss: 1.4598 - val_accuracy: 0.6500
Epoch 9/20
63/63 [==============================] - 0s 6ms/step - loss: 1.0029 - accuracy:
0.7140 - val_loss: 1.4933 - val_accuracy: 0.6440
Epoch 10/20
63/63 [==============================] - 0s 7ms/step - loss: 0.9520 - accuracy:
0.7214 - val_loss: 1.5150 - val_accuracy: 0.6400
Epoch 11/20
63/63 [==============================] - 0s 6ms/step - loss: 0.8933 - accuracy:
0.7386 - val_loss: 1.5606 - val_accuracy: 0.6500
Epoch 12/20
63/63 [==============================] - 0s 6ms/step - loss: 0.8750 - accuracy:
0.7483 - val_loss: 1.5502 - val_accuracy: 0.6610
Epoch 13/20
63/63 [==============================] - 0s 6ms/step - loss: 0.8492 - accuracy:
0.7612 - val_loss: 1.6314 - val_accuracy: 0.6790
Epoch 14/20
63/63 [==============================] - 0s 6ms/step - loss: 0.8106 - accuracy:
0.7867 - val_loss: 1.6443 - val_accuracy: 0.6780
Epoch 15/20
63/63 [==============================] - 0s 6ms/step - loss: 0.7521 - accuracy:
0.7947 - val_loss: 1.6647 - val_accuracy: 0.6730
Epoch 16/20
63/63 [==============================] - 0s 7ms/step - loss: 0.7350 - accuracy:
0.7989 - val_loss: 1.7521 - val_accuracy: 0.6780
Epoch 17/20
63/63 [==============================] - 0s 6ms/step - loss: 0.7356 - accuracy:
0.7959 - val_loss: 1.7887 - val_accuracy: 0.6740
Epoch 18/20
63/63 [==============================] - 0s 6ms/step - loss: 0.6877 - accuracy:
0.8100 - val_loss: 1.8185 - val_accuracy: 0.6800
Epoch 19/20
63/63 [==============================] - 0s 7ms/step - loss: 0.6621 - accuracy:
0.8098 - val_loss: 1.8892 - val_accuracy: 0.6720
Epoch 20/20
63/63 [==============================] - 0s 6ms/step - loss: 0.6600 - accuracy:
0.8081 - val_loss: 1.9709 - val_accuracy: 0.6730
```

```
[48]: <tensorflow.python.keras.callbacks.History at 0x7fb9f4c870d0>
```

0.1.3 5.3

[49]:
```
# 3.24

from keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) = boston_housing.
↪load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/boston_housing.npz
57344/57026 [==============================] - 0s 0us/step
```

[50]:
```
train_data.shape
```

[50]:
```
(404, 13)
```

[51]:
```
test_data.shape
```

[51]:
```
(102, 13)
```

[52]:
```
train_targets
```

[52]:
```
array([15.2, 42.3, 50. , 21.1, 17.7, 18.5, 11.3, 15.6, 15.6, 14.4, 12.1,
       17.9, 23.1, 19.9, 15.7,  8.8, 50. , 22.5, 24.1, 27.5, 10.9, 30.8,
       32.9, 24. , 18.5, 13.3, 22.9, 34.7, 16.6, 17.5, 22.3, 16.1, 14.9,
       23.1, 34.9, 25. , 13.9, 13.1, 20.4, 20. , 15.2, 24.7, 22.2, 16.7,
       12.7, 15.6, 18.4, 21. , 30.1, 15.1, 18.7,  9.6, 31.5, 24.8, 19.1,
       22. , 14.5, 11. , 32. , 29.4, 20.3, 24.4, 14.6, 19.5, 14.1, 14.3,
       15.6, 10.5,  6.3, 19.3, 19.3, 13.4, 36.4, 17.8, 13.5, 16.5,  8.3,
       14.3, 16. , 13.4, 28.6, 43.5, 20.2, 22. , 23. , 20.7, 12.5, 48.5,
       14.6, 13.4, 23.7, 50. , 21.7, 39.8, 38.7, 22.2, 34.9, 22.5, 31.1,
       28.7, 46. , 41.7, 21. , 26.6, 15. , 24.4, 13.3, 21.2, 11.7, 21.7,
       19.4, 50. , 22.8, 19.7, 24.7, 36.2, 14.2, 18.9, 18.3, 20.6, 24.6,
       18.2,  8.7, 44. , 10.4, 13.2, 21.2, 37. , 30.7, 22.9, 20. , 19.3,
       31.7, 32. , 23.1, 18.8, 10.9, 50. , 19.6,  5. , 14.4, 19.8, 13.8,
       19.6, 23.9, 24.5, 25. , 19.9, 17.2, 24.6, 13.5, 26.6, 21.4, 11.9,
       22.6, 19.6,  8.5, 23.7, 23.1, 22.4, 20.5, 23.6, 18.4, 35.2, 23.1,
       27.9, 20.6, 23.7, 28. , 13.6, 27.1, 23.6, 20.6, 18.2, 21.7, 17.1,
        8.4, 25.3, 13.8, 22.2, 18.4, 20.7, 31.6, 30.5, 20.3,  8.8, 19.2,
       19.4, 23.1, 23. , 14.8, 48.8, 22.6, 33.4, 21.1, 13.6, 32.2, 13.1,
       23.4, 18.9, 23.9, 11.8, 23.3, 22.8, 19.6, 16.7, 13.4, 22.2, 20.4,
       21.8, 26.4, 14.9, 24.1, 23.8, 12.3, 29.1, 21. , 19.5, 23.3, 23.8,
       17.8, 11.5, 21.7, 19.9, 25. , 33.4, 28.5, 21.4, 24.3, 27.5, 33.1,
       16.2, 23.3, 48.3, 22.9, 22.8, 13.1, 12.7, 22.6, 15. , 15.3, 10.5,
       24. , 18.5, 21.7, 19.5, 33.2, 23.2,  5. , 19.1, 12.7, 22.3, 10.2,
       13.9, 16.3, 17. , 20.1, 29.9, 17.2, 37.3, 45.4, 17.8, 23.2, 29. ,
       22. , 18. , 17.4, 34.6, 20.1, 25. , 15.6, 24.8, 28.2, 21.2, 21.4,
       23.8, 31. , 26.2, 17.4, 37.9, 17.5, 20. ,  8.3, 23.9,  8.4, 13.8,
```

```
        7.2, 11.7, 17.1, 21.6, 50. , 16.1, 20.4, 20.6, 21.4, 20.6, 36.5,
        8.5, 24.8, 10.8, 21.9, 17.3, 18.9, 36.2, 14.9, 18.2, 33.3, 21.8,
       19.7, 31.6, 24.8, 19.4, 22.8,  7.5, 44.8, 16.8, 18.7, 50. , 50. ,
       19.5, 20.1, 50. , 17.2, 20.8, 19.3, 41.3, 20.4, 20.5, 13.8, 16.5,
       23.9, 20.6, 31.5, 23.3, 16.8, 14. , 33.8, 36.1, 12.8, 18.3, 18.7,
       19.1, 29. , 30.1, 50. , 50. , 22. , 11.9, 37.6, 50. , 22.7, 20.8,
       23.5, 27.9, 50. , 19.3, 23.9, 22.6, 15.2, 21.7, 19.2, 43.8, 20.3,
       33.2, 19.9, 22.5, 32.7, 22. , 17.1, 19. , 15. , 16.1, 25.1, 23.7,
       28.7, 37.2, 22.6, 16.4, 25. , 29.8, 22.1, 17.4, 18.1, 30.3, 17.5,
       24.7, 12.6, 26.5, 28.7, 13.3, 10.4, 24.4, 23. , 20. , 17.8,  7. ,
       11.8, 24.4, 13.8, 19.4, 25.2, 19.4, 19.4, 29.1])
```

[53]:
```python
# 3.25

mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std
```

[54]:
```python
# 3.26

from keras import models
from keras import layers

def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu',
                           input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model
```

[55]:
```python
# 3.27
# K-fold validation

import numpy as np

k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []
for i in range(k):
    print('processing fold #', i)
```

```
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]

    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)

    model = build_model()
    model.fit(partial_train_data, partial_train_targets,
              epochs=num_epochs, batch_size=1, verbose=0)
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)
```

```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
```

[56]:
```
all_scores
```

[56]: [1.9402332305908203, 2.48775053024292, 2.757383108139038, 2.741459608078003]

[57]:
```
np.mean(all_scores)
```

[57]: 2.4817066192626953

[58]:
```
# 3.28

num_epochs = 500
all_mae_histories = []
for i in range(k):
    print('processing fold #', i)
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)
```

```
    model = build_model()
    history = model.fit(partial_train_data, partial_train_targets,
                        validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=1, verbose=0)
    mae_history = history.history['val_mae']
    all_mae_histories.append(mae_history)
```

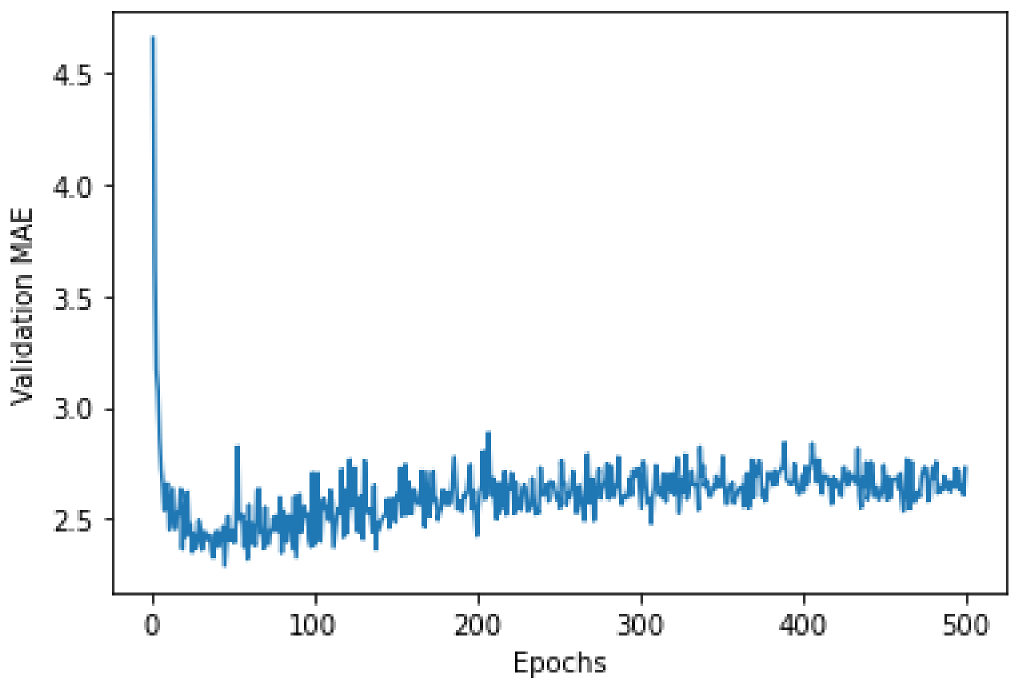
```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
```

[59]:

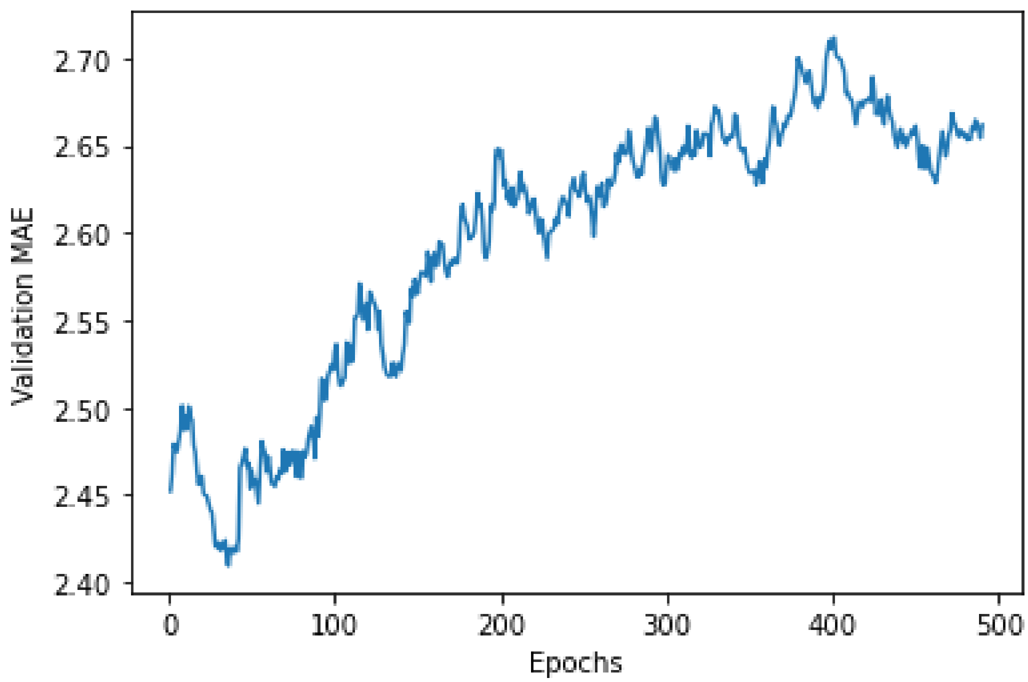
```
# 3.29

average_mae_history = [
    np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

[60]:

```
# 3.30

import matplotlib.pyplot as plt

plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



[61]:
```
# 3.31

def smooth_curve(points, factor=0.9):
  smoothed_points = []
  for point in points:
    if smoothed_points:
      previous = smoothed_points[-1]
      smoothed_points.append(previous * factor + point * (1 - factor))
    else:
      smoothed_points.append(point)
  return smoothed_points

smooth_mae_history = smooth_curve(average_mae_history[10:])

plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



[62]:
```
# 3.32

model = build_model()
model.fit(train_data, train_targets,
```

```
          epochs=80, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

```
4/4 [==============================] - 0s 1ms/step - loss: 17.7156 - mae: 2.5769
```

[63]:
```
test_mae_score
```

[63]: 2.576890230178833

[ ]:
```
```