assignment5ML

July 4, 2023

Smith Shauna

DSC650 Week 5

5.1 (3.4 IMDB Dataset)

[8]:
```
from keras.datasets import imdb

(train_data, train_labels),(test_data, test_labels) = imdb.
 ↪load_data(num_words=10000)
print(train_data[0])
print(train_labels[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36,
256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112,
167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16,
6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530,
38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8,
316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619,
5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14,
407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71,
43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98,
32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5,
144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88,
12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]
1
```

[9]:
```
max([max(sequence) for sequence in train_data])
```

[9]: 9999

[10]:
```
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
1641221/1641221 [==============================] - 0s 0us/step
```

[12]:
```
import numpy as np

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)

y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')

print(x_train[0])
```

```
[0. 1. 1. … 0. 0. 0.]
```

[ ]:
```
#output = relu(dot(w, input) + b)
```

[13]:
```
from keras import models
from keras import layers

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'))
```

[17]:
```
#from keras import optimizers

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

[18]:
```
from keras import losses
from keras import metrics

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

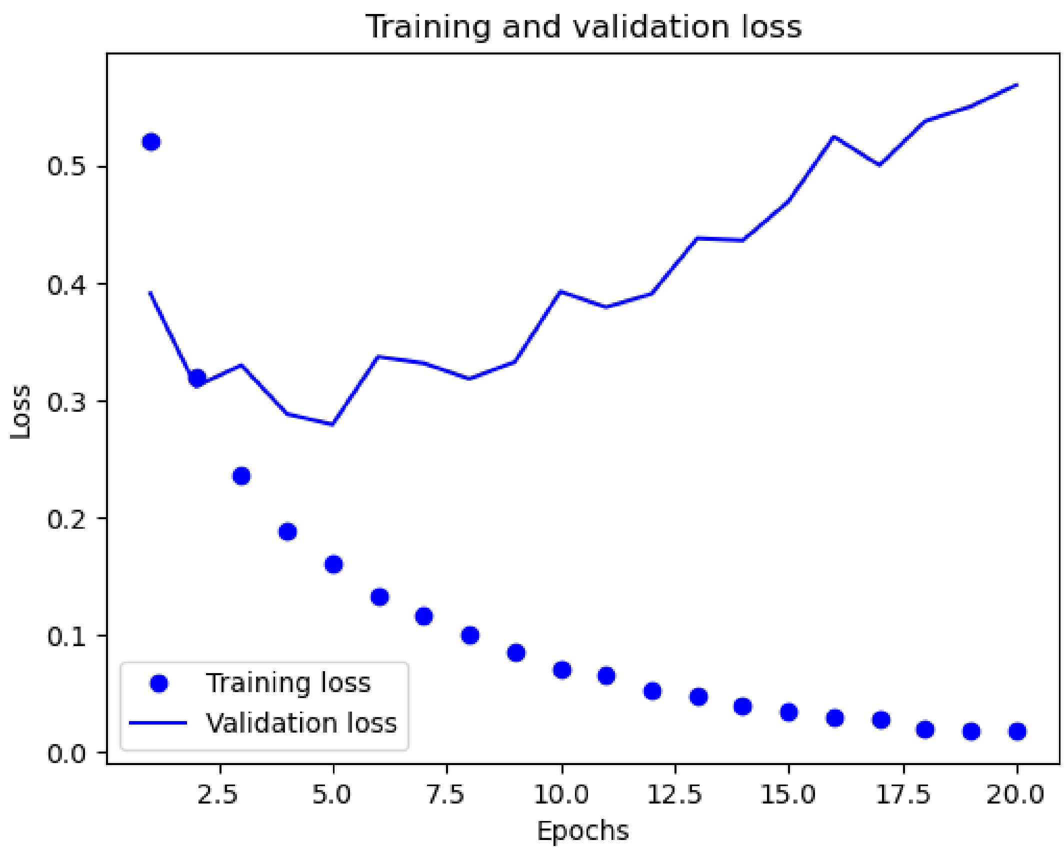
[19]:
```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

[20]:
```
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))
history_dict = history.history

print(history_dict.keys())
```

```
Epoch 1/20
30/30 [==============================] - 2s 54ms/step - loss: 0.5215 - acc:
0.7821 - val_loss: 0.3910 - val_acc: 0.8657
Epoch 2/20
30/30 [==============================] - 0s 15ms/step - loss: 0.3190 - acc:
0.8959 - val_loss: 0.3117 - val_acc: 0.8841
Epoch 3/20
30/30 [==============================] - 0s 14ms/step - loss: 0.2362 - acc:
0.9222 - val_loss: 0.3297 - val_acc: 0.8630
Epoch 4/20
30/30 [==============================] - 0s 15ms/step - loss: 0.1893 - acc:
0.9377 - val_loss: 0.2882 - val_acc: 0.8832
Epoch 5/20
30/30 [==============================] - 0s 15ms/step - loss: 0.1609 - acc:
0.9459 - val_loss: 0.2792 - val_acc: 0.8875
Epoch 6/20
30/30 [==============================] - 0s 15ms/step - loss: 0.1335 - acc:
0.9585 - val_loss: 0.3370 - val_acc: 0.8729
Epoch 7/20
30/30 [==============================] - 0s 14ms/step - loss: 0.1161 - acc:
0.9637 - val_loss: 0.3314 - val_acc: 0.8777
Epoch 8/20
30/30 [==============================] - 0s 17ms/step - loss: 0.0998 - acc:
0.9691 - val_loss: 0.3181 - val_acc: 0.8828
Epoch 9/20
30/30 [==============================] - 0s 14ms/step - loss: 0.0854 - acc:
0.9748 - val_loss: 0.3325 - val_acc: 0.8827
Epoch 10/20
30/30 [==============================] - 0s 14ms/step - loss: 0.0713 - acc:
0.9812 - val_loss: 0.3926 - val_acc: 0.8667
Epoch 11/20
30/30 [==============================] - 0s 14ms/step - loss: 0.0662 - acc:
0.9813 - val_loss: 0.3792 - val_acc: 0.8713
Epoch 12/20
30/30 [==============================] - 0s 14ms/step - loss: 0.0527 - acc:
0.9878 - val_loss: 0.3906 - val_acc: 0.8779
Epoch 13/20
```

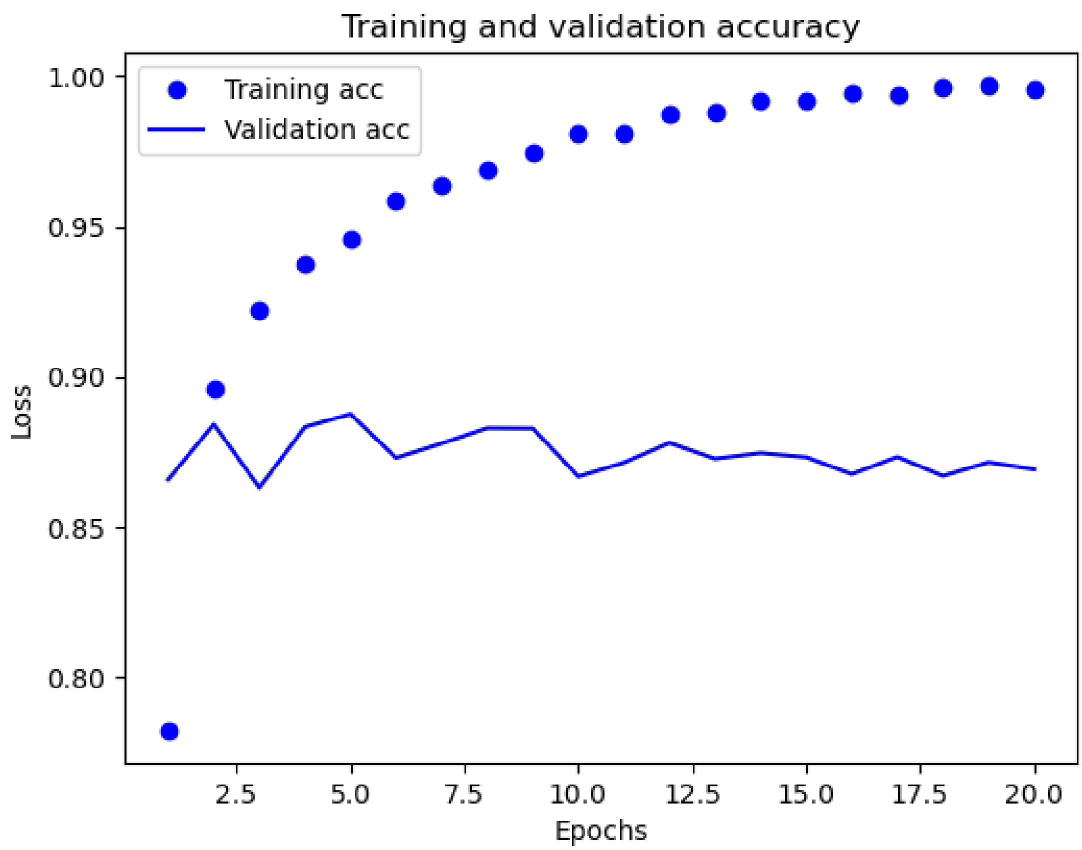
```
30/30 [==============================] - 0s 15ms/step - loss: 0.0478 - acc:
0.9881 - val_loss: 0.4379 - val_acc: 0.8726
Epoch 14/20
30/30 [==============================] - 1s 17ms/step - loss: 0.0401 - acc:
0.9917 - val_loss: 0.4361 - val_acc: 0.8745
Epoch 15/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0352 - acc:
0.9919 - val_loss: 0.4692 - val_acc: 0.8731
Epoch 16/20
30/30 [==============================] - 0s 15ms/step - loss: 0.0293 - acc:
0.9947 - val_loss: 0.5245 - val_acc: 0.8675
Epoch 17/20
30/30 [==============================] - 1s 17ms/step - loss: 0.0284 - acc:
0.9936 - val_loss: 0.5002 - val_acc: 0.8732
Epoch 18/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0207 - acc:
0.9962 - val_loss: 0.5378 - val_acc: 0.8669
Epoch 19/20
30/30 [==============================] - 0s 17ms/step - loss: 0.0190 - acc:
0.9971 - val_loss: 0.5503 - val_acc: 0.8714
Epoch 20/20
30/30 [==============================] - 0s 17ms/step - loss: 0.0184 - acc:
0.9959 - val_loss: 0.5685 - val_acc: 0.8691
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

[23]:
```python
import matplotlib.pyplot as plt

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

epochs = range(1, len(loss_values) + 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()
```



[25]:

```
plt.clf()
acc = history_dict['acc']
acc_values = history_dict['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()
```



[26]:

```
#Retrain
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)

print(results)
```

```
Epoch 1/4
49/49 [==============================] - 1s 10ms/step - loss: 0.4703 - accuracy:
0.8150
Epoch 2/4
49/49 [==============================] - 0s 9ms/step - loss: 0.2780 - accuracy:
```

```
0.9028
Epoch 3/4
49/49 [==============================] - 0s 9ms/step - loss: 0.2188 - accuracy:
0.9206
Epoch 4/4
49/49 [==============================] - 0s 10ms/step - loss: 0.1860 - accuracy:
0.9331
782/782 [==============================] - 2s 2ms/step - loss: 0.3078 -
accuracy: 0.8764
[0.3077734112739563, 0.8763999938964844]
```

[27]:
```
predictions = model.predict(x_test)

print(predictions)
```

```
782/782 [==============================] - 2s 2ms/step
[[0.2922592 ]
 [0.99982643]
 [0.9674441 ]
 …
 [0.19456838]
 [0.1267533 ]
 [0.7197722 ]]
```

Smith Shauna

5.2 (3.5 Classifying Newswires)

[28]:
```
from keras.datasets import reuters

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

print(len(train_data))
print(len(test_data))

print(train_data[10])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/reuters.npz
2110848/2110848 [==============================] - 0s 0us/step
8982
2246
[1, 245, 273, 207, 156, 53, 74, 160, 26, 14, 46, 296, 26, 39, 74, 2979, 3554,
14, 46, 4689, 4329, 86, 61, 3499, 4795, 14, 61, 451, 4329, 17, 12]
```

[30]:
```
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]])

train_labels[10]
```

[30]: 3

[31]:
```
import numpy as np

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)
```

[32]:
```
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)
```

[33]:
```
from keras.utils.np_utils import to_categorical

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

[34]:
```
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'))
```

[35]:
```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics='accuracy')
```

[36]:
```
x_val= x_train[:1000]
partial_x_train = x_train[1000:]
```

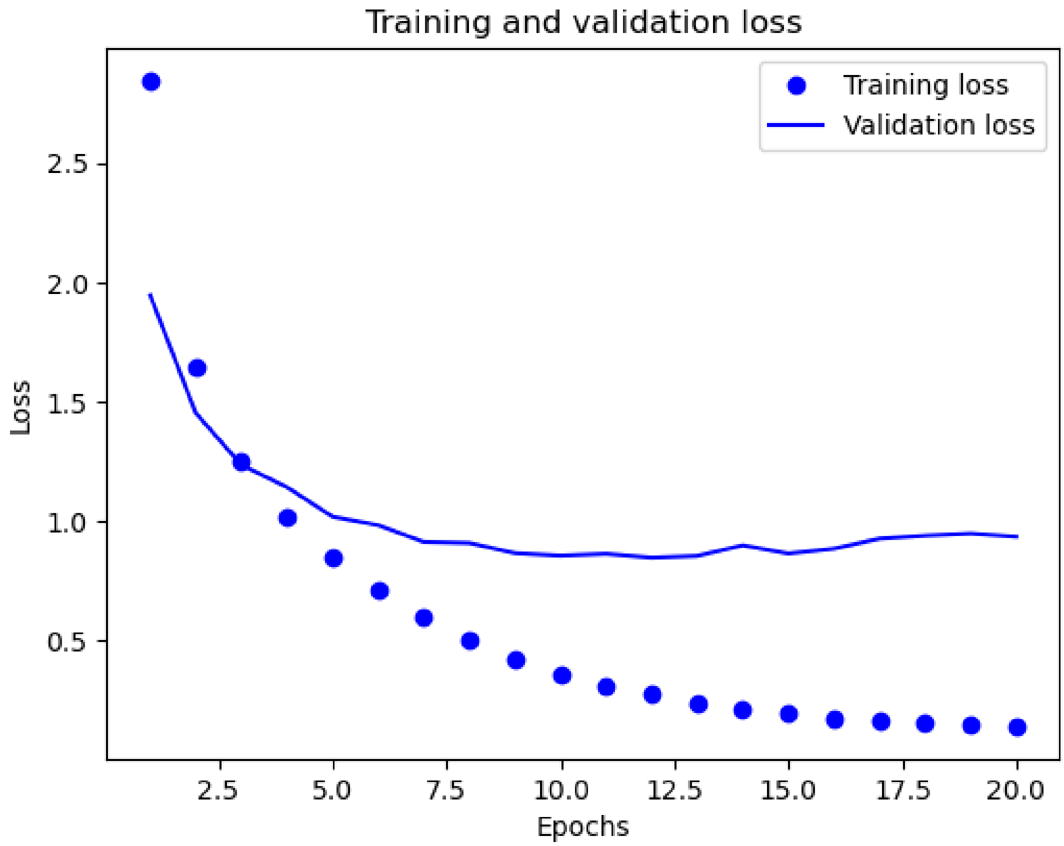
```
y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
```

[37]:
```
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 43ms/step - loss: 2.8430 - accuracy:
0.4888 - val_loss: 1.9448 - val_accuracy: 0.5890
Epoch 2/20
16/16 [==============================] - 0s 23ms/step - loss: 1.6417 - accuracy:
0.6518 - val_loss: 1.4528 - val_accuracy: 0.6830
Epoch 3/20
16/16 [==============================] - 0s 23ms/step - loss: 1.2496 - accuracy:
0.7326 - val_loss: 1.2360 - val_accuracy: 0.7220
Epoch 4/20
16/16 [==============================] - 0s 30ms/step - loss: 1.0196 - accuracy:
0.7798 - val_loss: 1.1429 - val_accuracy: 0.7320
Epoch 5/20
16/16 [==============================] - 0s 21ms/step - loss: 0.8519 - accuracy:
0.8167 - val_loss: 1.0195 - val_accuracy: 0.7850
Epoch 6/20
16/16 [==============================] - 0s 21ms/step - loss: 0.7148 - accuracy:
0.8492 - val_loss: 0.9839 - val_accuracy: 0.7880
Epoch 7/20
16/16 [==============================] - 0s 22ms/step - loss: 0.5991 - accuracy:
0.8735 - val_loss: 0.9137 - val_accuracy: 0.8090
Epoch 8/20
16/16 [==============================] - 0s 23ms/step - loss: 0.4999 - accuracy:
0.8961 - val_loss: 0.9093 - val_accuracy: 0.7980
Epoch 9/20
16/16 [==============================] - 0s 22ms/step - loss: 0.4245 - accuracy:
0.9132 - val_loss: 0.8670 - val_accuracy: 0.8150
Epoch 10/20
16/16 [==============================] - 0s 23ms/step - loss: 0.3566 - accuracy:
0.9265 - val_loss: 0.8566 - val_accuracy: 0.8120
Epoch 11/20
16/16 [==============================] - 0s 22ms/step - loss: 0.3092 - accuracy:
0.9357 - val_loss: 0.8642 - val_accuracy: 0.8130
Epoch 12/20
16/16 [==============================] - 0s 22ms/step - loss: 0.2770 - accuracy:
0.9391 - val_loss: 0.8484 - val_accuracy: 0.8240
Epoch 13/20
```

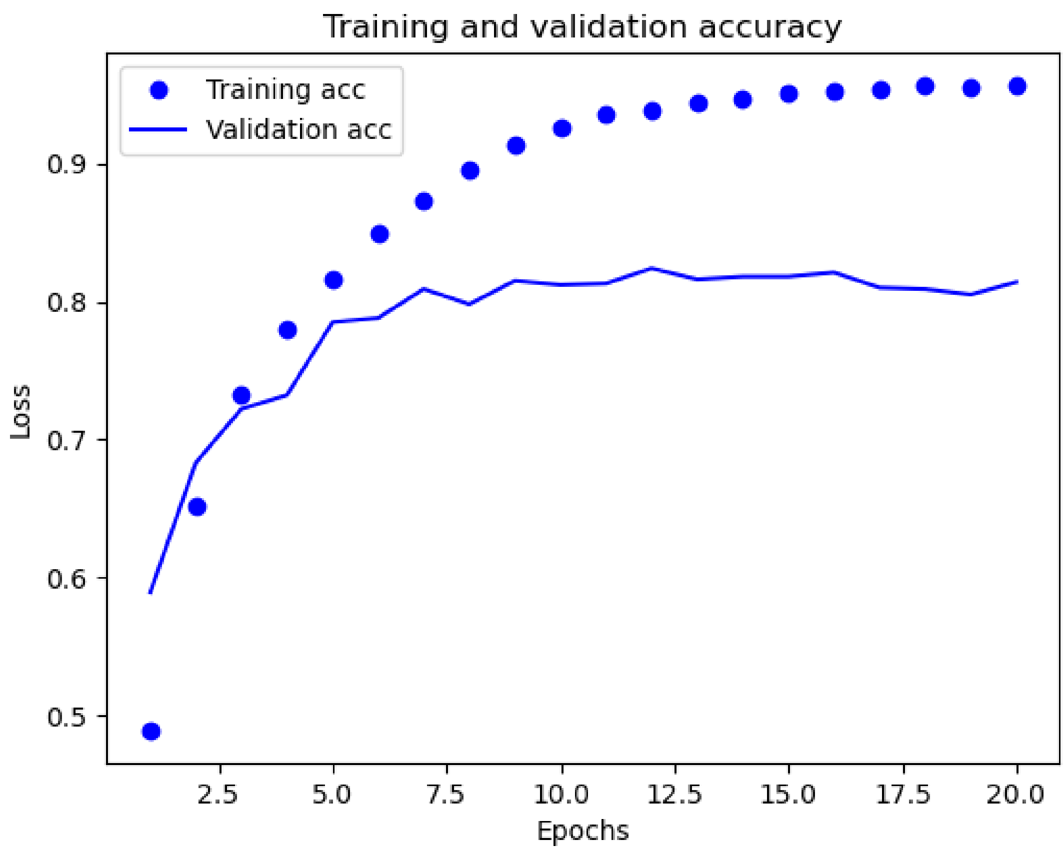
```
16/16 [==============================] - 0s 24ms/step - loss: 0.2396 - accuracy:
0.9449 - val_loss: 0.8555 - val_accuracy: 0.8160
Epoch 14/20
16/16 [==============================] - 0s 22ms/step - loss: 0.2150 - accuracy:
0.9476 - val_loss: 0.8995 - val_accuracy: 0.8180
Epoch 15/20
16/16 [==============================] - 0s 21ms/step - loss: 0.1952 - accuracy:
0.9518 - val_loss: 0.8655 - val_accuracy: 0.8180
Epoch 16/20
16/16 [==============================] - 0s 21ms/step - loss: 0.1741 - accuracy:
0.9530 - val_loss: 0.8856 - val_accuracy: 0.8210
Epoch 17/20
16/16 [==============================] - 0s 24ms/step - loss: 0.1666 - accuracy:
0.9539 - val_loss: 0.9274 - val_accuracy: 0.8100
Epoch 18/20
16/16 [==============================] - 0s 21ms/step - loss: 0.1568 - accuracy:
0.9569 - val_loss: 0.9402 - val_accuracy: 0.8090
Epoch 19/20
16/16 [==============================] - 0s 22ms/step - loss: 0.1468 - accuracy:
0.9550 - val_loss: 0.9486 - val_accuracy: 0.8050
Epoch 20/20
16/16 [==============================] - 0s 24ms/step - loss: 0.1426 - accuracy:
0.9569 - val_loss: 0.9360 - val_accuracy: 0.8140
```

[38]:
```
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()
```



[39]:

```
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()
```



[40]:
```
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=152,
          validation_data=(x_val,y_val))
results=model.evaluate(x_test, one_hot_test_labels)

results
```

```
Epoch 1/9
53/53 [==============================] - 2s 17ms/step - loss: 1.9627 - accuracy:
0.6144 - val_loss: 1.3237 - val_accuracy: 0.7010
```

```
Epoch 2/9
53/53 [==============================] - 1s 10ms/step - loss: 1.0660 - accuracy:
0.7653 - val_loss: 1.0750 - val_accuracy: 0.7610
Epoch 3/9
53/53 [==============================] - 1s 10ms/step - loss: 0.7514 - accuracy:
0.8370 - val_loss: 0.9544 - val_accuracy: 0.7980
Epoch 4/9
53/53 [==============================] - 1s 10ms/step - loss: 0.5339 - accuracy:
0.8866 - val_loss: 0.8946 - val_accuracy: 0.8170
Epoch 5/9
53/53 [==============================] - 1s 11ms/step - loss: 0.3887 - accuracy:
0.9177 - val_loss: 0.8792 - val_accuracy: 0.8190
Epoch 6/9
53/53 [==============================] - 1s 10ms/step - loss: 0.2959 - accuracy:
0.9347 - val_loss: 0.8708 - val_accuracy: 0.8200
Epoch 7/9
53/53 [==============================] - 1s 10ms/step - loss: 0.2416 - accuracy:
0.9437 - val_loss: 0.8795 - val_accuracy: 0.8240
Epoch 8/9
53/53 [==============================] - 1s 10ms/step - loss: 0.2051 - accuracy:
0.9481 - val_loss: 0.8873 - val_accuracy: 0.8170
Epoch 9/9
53/53 [==============================] - 1s 10ms/step - loss: 0.1832 - accuracy:
0.9526 - val_loss: 0.9775 - val_accuracy: 0.8120
71/71 [==============================] - 0s 3ms/step - loss: 1.0326 - accuracy:
0.7912
```

[40]: [1.0326011180877686, 0.7911843061447144]

[41]:
```
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]:
```
predictions = model.predict(x_test)
```

```
71/71 [==============================] - 0s 3ms/step
```

[44]:
```
predictions[0].shape, np.sum(predictions[0]),np.argmax(predictions[0])
```

[44]: ((46,), 1.0, 3)

[45]:
```
y_train = np.array(train_labels)
y_test = np.array(test_labels)
```

[46]:
```
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])
```

[47]:
```
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 12ms/step - loss: 3.6224 - accuracy:
0.0752 - val_loss: 3.4061 - val_accuracy: 0.0940
Epoch 2/20
63/63 [==============================] - 1s 9ms/step - loss: 3.1091 - accuracy:
0.1084 - val_loss: 2.8570 - val_accuracy: 0.1140
Epoch 3/20
63/63 [==============================] - 1s 9ms/step - loss: 2.3438 - accuracy:
0.3083 - val_loss: 1.9294 - val_accuracy: 0.5730
Epoch 4/20
63/63 [==============================] - 1s 9ms/step - loss: 1.6310 - accuracy:
0.5767 - val_loss: 1.6096 - val_accuracy: 0.5830
Epoch 5/20
63/63 [==============================] - 1s 9ms/step - loss: 1.4396 - accuracy:
0.5905 - val_loss: 1.5377 - val_accuracy: 0.5850
Epoch 6/20
63/63 [==============================] - 1s 9ms/step - loss: 1.3330 - accuracy:
0.6186 - val_loss: 1.5021 - val_accuracy: 0.6120
Epoch 7/20
63/63 [==============================] - 1s 8ms/step - loss: 1.2509 - accuracy:
0.6577 - val_loss: 1.4882 - val_accuracy: 0.6170
Epoch 8/20
63/63 [==============================] - 1s 10ms/step - loss: 1.1835 - accuracy:
0.6726 - val_loss: 1.4739 - val_accuracy: 0.6330
Epoch 9/20
63/63 [==============================] - 1s 10ms/step - loss: 1.1262 - accuracy:
0.6850 - val_loss: 1.4778 - val_accuracy: 0.6410
```

```
Epoch 10/20
63/63 [==============================] - 1s 9ms/step - loss: 1.0761 - accuracy:
0.6963 - val_loss: 1.4887 - val_accuracy: 0.6410
Epoch 11/20
63/63 [==============================] - 1s 9ms/step - loss: 1.0312 - accuracy:
0.7075 - val_loss: 1.4802 - val_accuracy: 0.6450
Epoch 12/20
63/63 [==============================] - 1s 9ms/step - loss: 0.9886 - accuracy:
0.7229 - val_loss: 1.4988 - val_accuracy: 0.6450
Epoch 13/20
63/63 [==============================] - 1s 9ms/step - loss: 0.9526 - accuracy:
0.7430 - val_loss: 1.5068 - val_accuracy: 0.6530
Epoch 14/20
63/63 [==============================] - 1s 9ms/step - loss: 0.9177 - accuracy:
0.7565 - val_loss: 1.5849 - val_accuracy: 0.6450
Epoch 15/20
63/63 [==============================] - 1s 9ms/step - loss: 0.8834 - accuracy:
0.7622 - val_loss: 1.5762 - val_accuracy: 0.6540
Epoch 16/20
63/63 [==============================] - 1s 9ms/step - loss: 0.8549 - accuracy:
0.7741 - val_loss: 1.5946 - val_accuracy: 0.6590
Epoch 17/20
63/63 [==============================] - 1s 9ms/step - loss: 0.8237 - accuracy:
0.7821 - val_loss: 1.5804 - val_accuracy: 0.6630
Epoch 18/20
63/63 [==============================] - 1s 9ms/step - loss: 0.7986 - accuracy:
0.7881 - val_loss: 1.6486 - val_accuracy: 0.6630
Epoch 19/20
63/63 [==============================] - 1s 9ms/step - loss: 0.7714 - accuracy:
0.7934 - val_loss: 1.6406 - val_accuracy: 0.6690
Epoch 20/20
63/63 [==============================] - 1s 9ms/step - loss: 0.7498 - accuracy:
0.7953 - val_loss: 1.6795 - val_accuracy: 0.6710
```

[47]:
```
<keras.callbacks.History at 0x7f02939b7400>
```

Smith Shauna

5.3 - (3.6 Predicting house prices)

[48]:
```
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
57026/57026 [==============================] - 0s 0us/step
```

[49]:
```
train_data.shape, test_data.shape
```

[49]:
```
((404, 13), (102, 13))
```

[50]:
```
train_targets
```

[50]:
```
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])
```

[51]:
```
mean = train_data.mean(axis = 0)
train_data -= mean
std = train_data.std(axis=0)
```

```
train_data /= std

test_data -= mean
test_data /= std
```

[52]:
```
 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
```

[56]:
```
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
```

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

[57]:
```
([2.0328590869903564,
  2.388063907623291,
  2.7506163120269775,
  2.4969539642333984,
  2.4606409072875977,
```

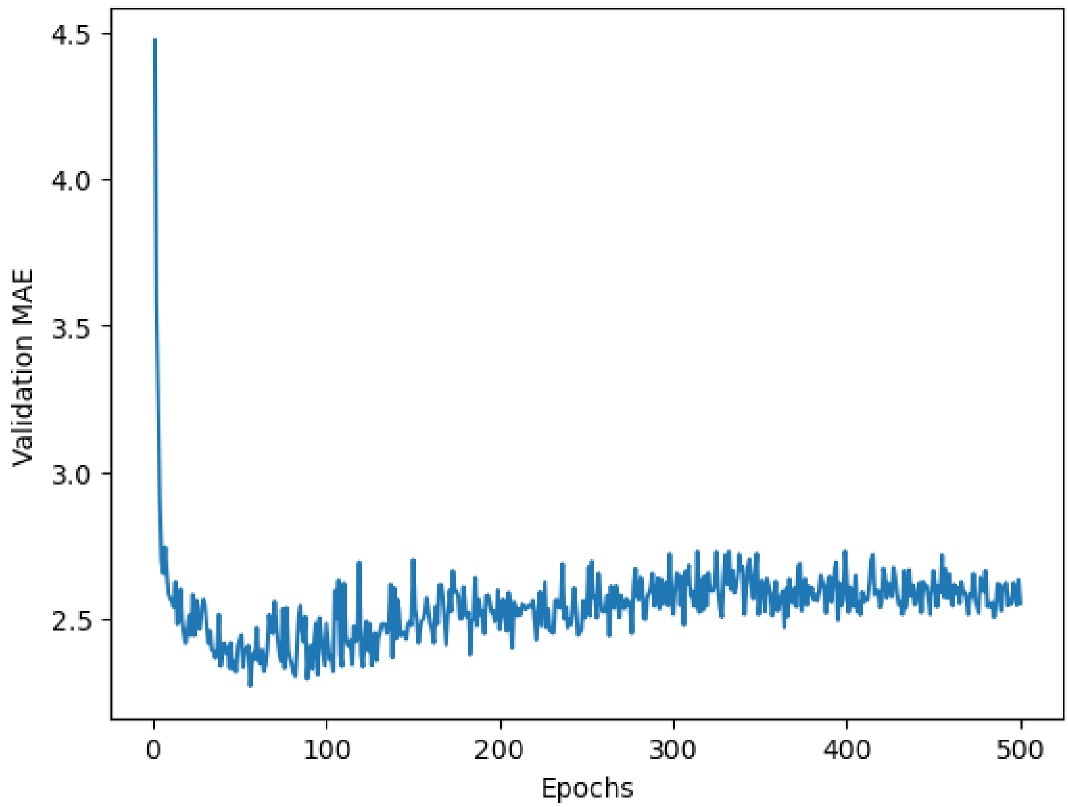
```
 2.934039354324341,
 2.809873104095459,
 2.3863813877105713,
 2.0728578567504883,
 2.402162551879883,
 2.7716777324676514,
 2.392277240753174],
2.491533617178599)
```

[59]:
```
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
```

[60]:
```
average_mae_history = [np.mean([x[i] for x in all_mae_histories]) for i in␣
 ↪range(num_epochs)]
```

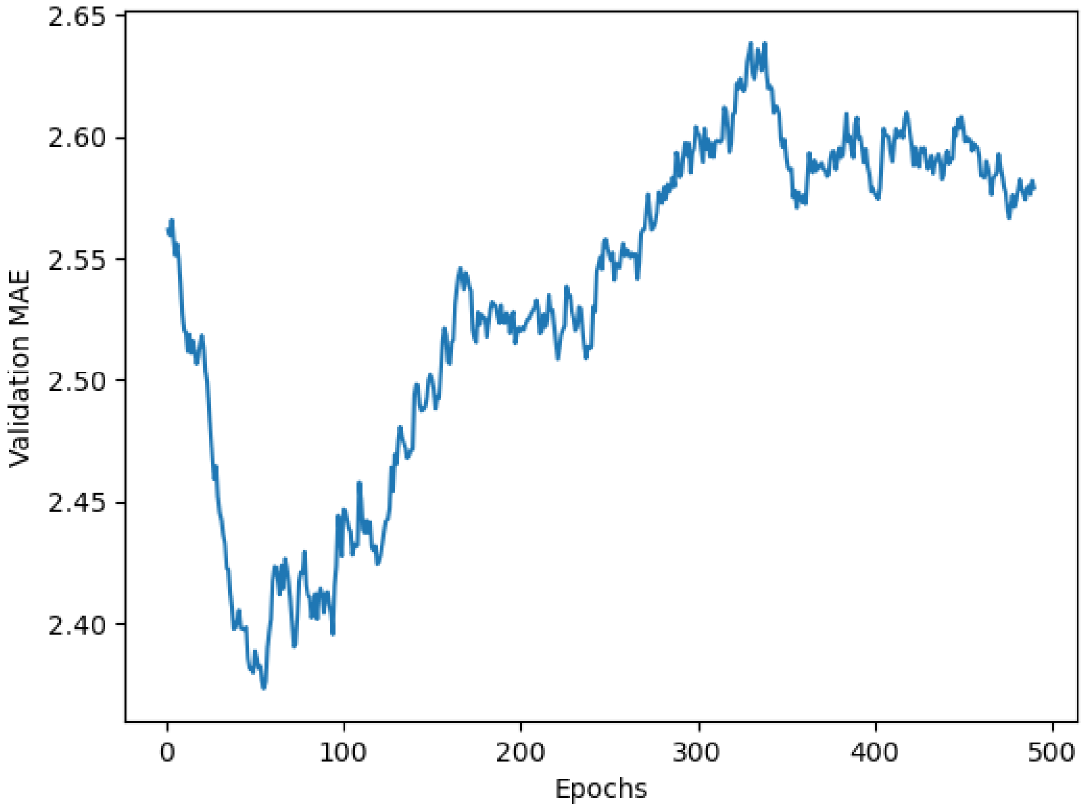
[67]:
```
plt.plot(range(1, len(average_mae_history)+ 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')

plt.show()
```



[68]:
```
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
```

[69]:
```
smooth_mae_history = smooth_curve(average_mae_history[10:])
```

[70]:
```
plt.plot(range(1, len(smooth_mae_history)+ 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')

plt.show()
```



[71]:
```
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 5ms/step - loss: 18.5580 - mae: 2.8200
```

[72]:
```
test_mae_score
```

[72]: 2.8200182914733887

[ ]:
```

```