Taylor Imhof

Bellevue University | DSC 650

Assignment05

Date: 6/26/2022

Movie Review Classifier

In [1]:

```
# import imdb dataset from keras
from tensorflow.keras.datasets import imdb
```

In [2]:

```
# load imdb movie dataset from TF
(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 [==============================] - 0s 0us/step
```

In [5]:

```
train_data[:5]
```

Out[5]:

```
array([list([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, 3
16, 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]),
       list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634, 1
4, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93,
4, 114, 9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19,
4, 1002, 5, 89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11,
3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 123, 125, 68, 2, 6853, 15,
349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 437
3, 228, 8255, 5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78
, 22, 625, 64, 1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78,
285, 16, 145, 95]),
       list([1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5974, 54, 61, 369, 13, 71, 149, 14, 22, 112, 4
, 2401, 311, 12, 16, 3711, 33, 75, 43, 1829, 296, 4, 86, 320, 35, 534, 19, 263, 4821, 1301, 4, 1873, 33
, 89, 78, 12, 66, 16, 4, 360, 7, 4, 58, 316, 334, 11, 4, 1716, 43, 645, 662, 8, 257, 85, 1200, 42, 1228
, 2578, 83, 68, 3912, 15, 36, 165, 1539, 278, 36, 69, 2, 780, 8, 106, 14, 6905, 1338, 18, 6, 22, 12, 21
5, 28, 610, 40, 6, 87, 326, 23, 2300, 21, 23, 22, 12, 272, 40, 57, 31, 11, 4, 22, 47, 6, 2307, 51, 9, 1
70, 23, 595, 116, 595, 1352, 13, 191, 79, 638, 89, 2, 14, 9, 8, 106, 607, 624, 35, 534, 6, 227, 7, 129,
113]),
       list([1, 4, 2, 2, 33, 2804, 4, 2040, 432, 111, 153, 103, 4, 1494, 13, 70, 131, 67, 11, 61, 2, 74
4, 35, 3715, 761, 61, 5766, 452, 9214, 4, 985, 7, 2, 59, 166, 4, 105, 216, 1239, 41, 1797, 9, 15, 7, 35
, 744, 2413, 31, 8, 4, 687, 23, 4, 2, 7339, 6, 3693, 42, 38, 39, 121, 59, 456, 10, 10, 7, 265, 12, 575,
111, 153, 159, 59, 16, 1447, 21, 25, 586, 482, 39, 4, 96, 59, 716, 12, 4, 172, 65, 9, 579, 11, 6004, 4,
1615, 5, 2, 7, 5168, 17, 13, 7064, 12, 19, 6, 464, 31, 314, 11, 2, 6, 719, 605, 11, 8, 202, 27, 310, 4,
3772, 3501, 8, 2722, 58, 10, 10, 537, 2116, 180, 40, 14, 413, 173, 7, 263, 112, 37, 152, 377, 4, 537, 2
63, 846, 579, 178, 54, 75, 71, 476, 36, 413, 263, 2504, 182, 5, 17, 75, 2306, 922, 36, 279, 131, 2895,
17, 2867, 42, 17, 35, 921, 2, 192, 5, 1219, 3890, 19, 2, 217, 4122, 1710, 537, 2, 1236, 5, 736, 10, 10,
61, 403, 9, 2, 40, 61, 4494, 5, 27, 4494, 159, 90, 263, 2311, 4319, 309, 8, 178, 5, 82, 4319, 4, 65, 15
```

```
, 9225, 145, 143, 5122, 12, 7039, 537, 746, 537, 537, 15, 7979, 4, 2, 594, 7, 5168, 94, 9096, 3987, 2,
11, 2, 4, 538, 7, 1795, 246, 2, 9, 2, 11, 635, 14, 9, 51, 408, 12, 94, 318, 1382, 12, 47, 6, 2683, 936,
5, 6307, 2, 19, 49, 7, 4, 1885, 2, 1118, 25, 80, 126, 842, 10, 10, 2, 2, 4726, 27, 4494, 11, 1550, 3633
, 159, 27, 341, 29, 2733, 19, 4185, 173, 7, 90, 2, 8, 30, 11, 4, 1784, 86, 1117, 8, 3261, 46, 11, 2, 21
, 29, 9, 2841, 23, 4, 1010, 2, 793, 6, 2, 1386, 1830, 10, 10, 246, 50, 9, 6, 2750, 1944, 746, 90, 29, 2
, 8, 124, 4, 882, 4, 882, 496, 27, 2, 2213, 537, 121, 127, 1219, 130, 5, 29, 494, 8, 124, 4, 882, 496,
4, 341, 7, 27, 846, 10, 10, 29, 9, 1906, 8, 97, 6, 236, 2, 1311, 8, 4, 2, 7, 31, 7, 2, 91, 2, 3987, 70,
4, 882, 30, 579, 42, 9, 12, 32, 11, 537, 10, 10, 11, 14, 65, 44, 537, 75, 2, 1775, 3353, 2, 1846, 4, 2,
7, 154, 5, 4, 518, 53, 2, 2, 7, 3211, 882, 11, 399, 38, 75, 257, 3807, 19, 2, 17, 29, 456, 4, 65, 7, 27
, 205, 113, 10, 10, 2, 4, 2, 2, 9, 242, 4, 91, 1202, 2, 5, 2070, 307, 22, 7, 5168, 126, 93, 40, 2, 13,
188, 1076, 3222, 19, 4, 2, 7, 2348, 537, 23, 53, 537, 21, 82, 40, 2, 13, 2, 14, 280, 13, 219, 4, 2, 431
, 758, 859, 4, 953, 1052, 2, 7, 5991, 5, 94, 40, 25, 238, 60, 2, 4, 2, 804, 2, 7, 4, 9941, 132, 8, 67,
6, 22, 15, 9, 283, 8, 5168, 14, 31, 9, 242, 955, 48, 25, 279, 2, 23, 12, 1685, 195, 25, 238, 60, 796, 2
, 4, 671, 7, 2804, 5, 4, 559, 154, 888, 7, 726, 50, 26, 49, 7008, 15, 566, 30, 579, 21, 64, 2574]),
       list([1, 249, 1323, 7, 61, 113, 10, 10, 13, 1637, 14, 20, 56, 33, 2401, 18, 457, 88, 13, 2626, 1
400, 45, 3171, 13, 70, 79, 49, 706, 919, 13, 16, 355, 340, 355, 1696, 96, 143, 4, 22, 32, 289, 7, 61, 3
69, 71, 2359, 5, 13, 16, 131, 2073, 249, 114, 249, 229, 249, 20, 13, 28, 126, 110, 13, 473, 8, 569, 61,
419, 56, 429, 6, 1513, 18, 35, 534, 95, 474, 570, 5, 25, 124, 138, 88, 12, 421, 1543, 52, 725, 6397, 61
, 419, 11, 13, 1571, 15, 1543, 20, 11, 4, 2, 5, 296, 12, 3524, 5, 15, 421, 128, 74, 233, 334, 207, 126,
224, 12, 562, 298, 2167, 1272, 7, 2601, 5, 516, 988, 43, 8, 79, 120, 15, 595, 13, 784, 25, 3171, 18, 16
5, 170, 143, 19, 14, 5, 7224, 6, 226, 251, 7, 61, 113])],
      dtype=object)
```

In [7]:

```python
train_labels[:5]
```

Out[7]:

```
array([1, 0, 0, 1, 0], dtype=int64)
```

In [9]:

```python
# word index is restricted to < 10,000
# set by passing num_words param to imdb.load_data()
max([max(sequence) for sequence in train_data])
```

Out[9]:

```
9999
```

In [10]:

```python
# decoding reviews back to text
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
```

In [12]:

```python
type(decoded_review)
decoded_review
```

Out[12]:

```
"? this film was just brilliant casting location scenery story direction everyone's really suited the p
art they played and you could just imagine being there robert ? is an amazing actor and now the same be
ing director ? father came from the same scottish island as myself so i loved the fact there was a real
connection with this film the witty remarks throughout the film were great it was just brilliant so muc
h that i bought the film as soon as it was released for ? and would recommend it to everyone to watch a
nd the fly fishing was amazing really cried at the end it was so sad and you know what they say if you
cry at a film it must have been good and this definitely was also ? to the two little boy's that played
the ? of norman and paul they were just brilliant children are often left out of the ? list i think bec
ause the stars that play them all grown up are such a big profile for the whole film but these children
```

are amazing and should be praised for what they have done don't you think the whole story was so lovely because it was true and was someone's life after all that was shared with us all"

Preparing the data

- pad lists to have same length
- change to integer tensor of shape(samples, max_length)
- embedding to start capable model
- multi-hot encode to turn into vectors of 0s and 1s

In [14]:

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

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

In [15]:

```
# vectorize labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

Building the Model

In [16]:

```
# model definition
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(16, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

In [20]:

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

In [21]:

```
# set aside validation data to test how model performs on "new" data
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

In [22]:

```
# training the model
history = model.fit(
    partial_x_train,
    partial_y_train,
```

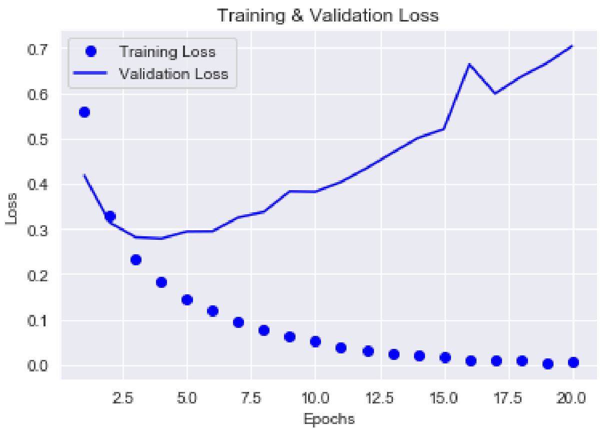
```
    partial_y_train,
    epochs=20,
    batch_size=512,
    validation_data=(x_val, y_val)
)
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 2s 115us/sample - loss: 0.5592 - accuracy: 0.7380 - val_
loss: 0.4177 - val_accuracy: 0.8493
Epoch 2/20
15000/15000 [==============================] - 1s 79us/sample - loss: 0.3298 - accuracy: 0.8955 - val_l
oss: 0.3134 - val_accuracy: 0.8876
Epoch 3/20
15000/15000 [==============================] - 1s 78us/sample - loss: 0.2329 - accuracy: 0.9255 - val_l
oss: 0.2812 - val_accuracy: 0.8924
Epoch 4/20
15000/15000 [==============================] - 1s 76us/sample - loss: 0.1820 - accuracy: 0.9424 - val_l
oss: 0.2782 - val_accuracy: 0.8899
Epoch 5/20
15000/15000 [==============================] - 1s 85us/sample - loss: 0.1453 - accuracy: 0.9537 - val_l
oss: 0.2939 - val_accuracy: 0.8815
Epoch 6/20
15000/15000 [==============================] - 1s 81us/sample - loss: 0.1191 - accuracy: 0.9636 - val_l
oss: 0.2944 - val_accuracy: 0.8857
Epoch 7/20
15000/15000 [==============================] - 1s 81us/sample - loss: 0.0936 - accuracy: 0.9732 - val_l
oss: 0.3251 - val_accuracy: 0.8767
Epoch 8/20
15000/15000 [==============================] - 1s 91us/sample - loss: 0.0783 - accuracy: 0.9785 - val_l
oss: 0.3375 - val_accuracy: 0.8850
Epoch 9/20
15000/15000 [==============================] - 1s 88us/sample - loss: 0.0639 - accuracy: 0.9832 - val_l
oss: 0.3829 - val_accuracy: 0.8693
Epoch 10/20
15000/15000 [==============================] - 1s 76us/sample - loss: 0.0524 - accuracy: 0.9863 - val_l
oss: 0.3819 - val_accuracy: 0.8817
Epoch 11/20
15000/15000 [==============================] - 1s 78us/sample - loss: 0.0397 - accuracy: 0.9917 - val_l
oss: 0.4035 - val_accuracy: 0.8772
Epoch 12/20
15000/15000 [==============================] - 1s 81us/sample - loss: 0.0304 - accuracy: 0.9943 - val_l
oss: 0.4338 - val_accuracy: 0.8741
Epoch 13/20
15000/15000 [==============================] - 1s 87us/sample - loss: 0.0245 - accuracy: 0.9958 - val_l
oss: 0.4679 - val_accuracy: 0.8766
Epoch 14/20
15000/15000 [==============================] - 1s 89us/sample - loss: 0.0192 - accuracy: 0.9967 - val_l
oss: 0.5007 - val_accuracy: 0.8742
Epoch 15/20
15000/15000 [==============================] - 1s 80us/sample - loss: 0.0155 - accuracy: 0.9973 - val_l
oss: 0.5204 - val_accuracy: 0.8724
Epoch 16/20
15000/15000 [==============================] - 1s 77us/sample - loss: 0.0084 - accuracy: 0.9995 - val_l
oss: 0.6634 - val_accuracy: 0.8625
Epoch 17/20
15000/15000 [==============================] - 1s 85us/sample - loss: 0.0091 - accuracy: 0.9991 - val_l
oss: 0.5987 - val_accuracy: 0.8683
Epoch 18/20
15000/15000 [==============================] - 1s 86us/sample - loss: 0.0083 - accuracy: 0.9983 - val_l
oss: 0.6357 - val_accuracy: 0.8699
Epoch 19/20
15000/15000 [==============================] - 1s 84us/sample - loss: 0.0030 - accuracy: 0.9999 - val_l
oss: 0.6657 - val_accuracy: 0.8690
Epoch 20/20
15000/15000 [==============================] - 1s 82us/sample - loss: 0.0047 - accuracy: 0.9989 - val_l
oss: 0.7039 - val_accuracy: 0.8657
```

In [24]:

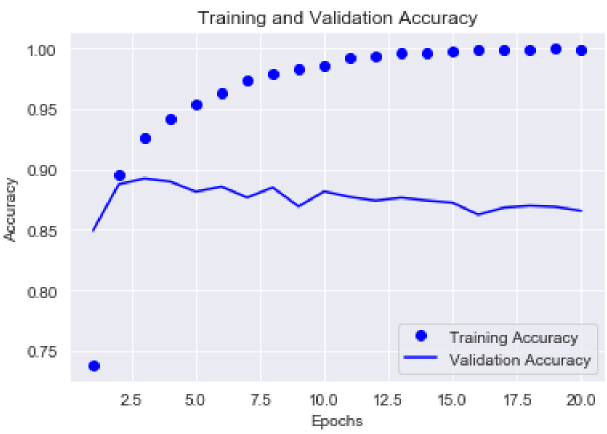
```
# plotting training and validation loss
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
```

```
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 & Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [25]:

```
# plotting training and validation accuracy
plt.clf() # clears figure
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [27]:

```
# retrain the model without four epochs as to avoid overfitting/overoptimizing
model = keras.Sequential([
    layers.Dense(16, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
model.compile(
    optimizer='rmsprop',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

model.fit(x_train, y_train, epochs=4, batch_size=128)
```

```
Train on 25000 samples
```

```
Train on 25000 samples
Epoch 1/4
25000/25000 [==============================] - 2s 74us/sample - loss: 0.3604 - accuracy: 0.8560
Epoch 2/4
25000/25000 [==============================] - 1s 56us/sample - loss: 0.2122 - accuracy: 0.9195
Epoch 3/4
25000/25000 [==============================] - 2s 61us/sample - loss: 0.1684 - accuracy: 0.9376
Epoch 4/4
25000/25000 [==============================] - 1s 59us/sample - loss: 0.1419 - accuracy: 0.9477
25000/25000 [==============================] - 2s 63us/sample - loss: 0.3640 - accuracy: 0.8704
```

In [31]:

```
results = model.evaluate(x_test, y_test)
```

```
25000/25000 [==============================] - 2s 60us/sample - loss: 0.3640 - accuracy: 0.8704
```

In [34]:

```
print(f'Test Loss:\t\t{results[0]:.4f}\nTest Accuracy:\t{results[1]:.4f}')
```

```
Test Loss:  0.3640
Test Accuracy: 0.8704
```

In [35]:

```
# predict on "new" data
model.predict(x_test)
```

Out[35]:

```
array([[0.04830925],
       [0.99933356],
       [0.76204664],
       ...,
       [0.09218684],
       [0.02938751],
       [0.51056576]], dtype=float32)
```

News Classifier

In [74]:

```
# load reuters news dataset from keras
from tensorflow.keras.datasets import reuters
(train_data, train_labels), (test_data, test_labels) = reuters.load_data(num_words=10000)
```

In [75]:

```
# check length of training and testing data
print(len(train_data))
print(len(test_data))
```

```
8982
2246
```

In [76]:

```
# decode newswire back to text
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]]
)
decoded_newswire
```

```
decoded_newswire
```

Out[76]:

```
'? ? ? said as a result of its december acquisition of space co it expects earnings per share in 1987 o
f 1 15 to 1 30 dlrs per share up from 70 cts in 1986 the company said pretax net should rise to nine to
10 mln dlrs from six mln dlrs in 1986 and rental operation revenues to 19 to 22 mln dlrs from 12 5 mln
dlrs it said cash flow per share this year should be 2 50 to three dlrs reuter 3'
```

Preparing the Data

In [77]:

```
# vectorize input data
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

In [65]:

```
# vectors labels using one-hot encoding
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
y_train = to_one_hot(train_labels)
y_test = to_one_hot(test_labels)
```

In [78]:

```
# one-hot encoding implementation built into keras
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(train_labels)
y_test = to_categorical(test_labels)
```

In [79]:

```
# have to use larger units than movie review as there are much
# more dimensions for the different cats of news articles
model = keras.Sequential([
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(46, activation='softmax')
])
```

In [80]:

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

In [83]:

```
# set aside validation set
x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = y_train[:1000]
partial_y_train = y_train[1000:]
```

In [84]:

```
# training the model
history = model.fit(
    partial_x_train,
    partial_y_train,
```

```
    epochs=20,
    batch_size=512,
    validation_data=(x_val, y_val)
)
```

```
Train on 7982 samples, validate on 1000 samples
Epoch 1/20
7982/7982 [==============================] - 1s 185us/sample - loss: 2.6698 - accuracy: 0.5232 - val_lo
ss: 1.7391 - val_accuracy: 0.6380
Epoch 2/20
7982/7982 [==============================] - 1s 94us/sample - loss: 1.4030 - accuracy: 0.7062 - val_los
s: 1.2808 - val_accuracy: 0.7120
Epoch 3/20
7982/7982 [==============================] - 1s 108us/sample - loss: 1.0261 - accuracy: 0.7769 - val_lo
ss: 1.1107 - val_accuracy: 0.7550
Epoch 4/20
7982/7982 [==============================] - 1s 104us/sample - loss: 0.8030 - accuracy: 0.8305 - val_lo
ss: 1.0081 - val_accuracy: 0.7880
Epoch 5/20
7982/7982 [==============================] - 1s 95us/sample - loss: 0.6407 - accuracy: 0.8636 - val_los
s: 0.9504 - val_accuracy: 0.8000
Epoch 6/20
7982/7982 [==============================] - 1s 91us/sample - loss: 0.5115 - accuracy: 0.8934 - val_los
s: 0.9092 - val_accuracy: 0.8070
Epoch 7/20
7982/7982 [==============================] - 1s 89us/sample - loss: 0.4144 - accuracy: 0.9126 - val_los
s: 0.8771 - val_accuracy: 0.8170
Epoch 8/20
7982/7982 [==============================] - 1s 85us/sample - loss: 0.3352 - accuracy: 0.9266 - val_los
s: 0.9557 - val_accuracy: 0.8060
Epoch 9/20
7982/7982 [==============================] - 1s 90us/sample - loss: 0.2835 - accuracy: 0.9385 - val_los
s: 0.8873 - val_accuracy: 0.8160
Epoch 10/20
7982/7982 [==============================] - 1s 90us/sample - loss: 0.2349 - accuracy: 0.9451 - val_los
s: 0.9228 - val_accuracy: 0.8030
Epoch 11/20
7982/7982 [==============================] - 1s 95us/sample - loss: 0.2086 - accuracy: 0.9488 - val_los
s: 0.9468 - val_accuracy: 0.8060
Epoch 12/20
7982/7982 [==============================] - 1s 101us/sample - loss: 0.1836 - accuracy: 0.9500 - val_lo
ss: 0.9144 - val_accuracy: 0.8180
Epoch 13/20
7982/7982 [==============================] - 1s 102us/sample - loss: 0.1639 - accuracy: 0.9513 - val_lo
ss: 0.9390 - val_accuracy: 0.8210
Epoch 14/20
7982/7982 [==============================] - 1s 94us/sample - loss: 0.1497 - accuracy: 0.9555 - val_los
s: 1.0399 - val_accuracy: 0.7980
Epoch 15/20
7982/7982 [==============================] - 1s 92us/sample - loss: 0.1397 - accuracy: 0.9565 - val_los
s: 1.0184 - val_accuracy: 0.8060
Epoch 16/20
7982/7982 [==============================] - 1s 85us/sample - loss: 0.1314 - accuracy: 0.9562 - val_los
s: 1.0573 - val_accuracy: 0.7980
Epoch 17/20
7982/7982 [==============================] - 1s 88us/sample - loss: 0.1255 - accuracy: 0.9557 - val_los
s: 1.0651 - val_accuracy: 0.8030
Epoch 18/20
7982/7982 [==============================] - 1s 89us/sample - loss: 0.1168 - accuracy: 0.9564 - val_los
s: 1.0372 - val_accuracy: 0.8140
Epoch 19/20
7982/7982 [==============================] - 1s 103us/sample - loss: 0.1198 - accuracy: 0.9557 - val_lo
ss: 1.0235 - val_accuracy: 0.8180
Epoch 20/20
7982/7982 [==============================] - 1s 78us/sample - loss: 0.1109 - accuracy: 0.9573 - val_los
s: 1.0548 - val_accuracy: 0.8100
```

Plotting Losses To Determine Best # of Epochs

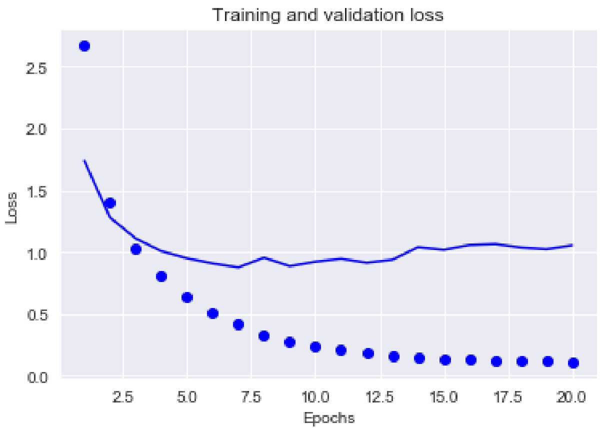
In [85]:

```
# plot train and validation loss
loss = history.history['loss']
val_loss = history.history['val_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.lenend()
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-85-78c05e3b474f> in <module>
      8 plt.xlabel('Epochs')
      9 plt.ylabel('Loss')
---> 10 plt.lenend()
     11 plt.show()

AttributeError: module 'matplotlib.pyplot' has no attribute 'lenend'
```

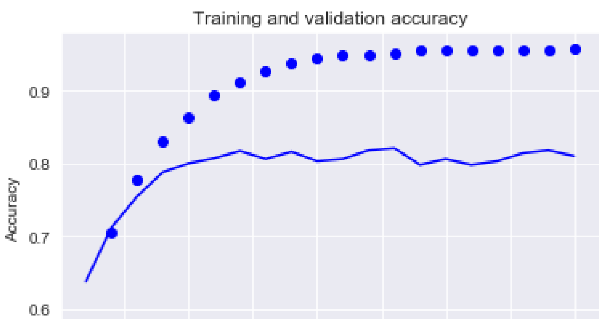


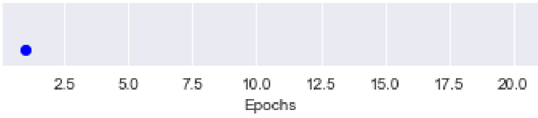
In [86]:

```
# plotting train/validation accuracy
plt.clf()
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.lenged()
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-86-ba6be0a0ff77> in <module>
      8 plt.xlabel('Epochs')
      9 plt.ylabel('Accuracy')
---> 10 plt.lenged()
     11 plt.show()

AttributeError: module 'matplotlib.pyplot' has no attribute 'lenged'
```





It would appear that the values seem to level/taper around 9 epochs

In [87]:

```
# retrain model with "better" # of epochs
model = keras.Sequential([
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(46, activation='softmax')
])
model.compile(
    optimizer='rmsprop',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
model.fit(
    x_train,
    y_train,
    epochs=9,
    batch_size=512
)
results = model.evaluate(x_test, y_test)
```

```
Train on 8982 samples
Epoch 1/9
8982/8982 [==============================] - 1s 126us/sample - loss: 2.3783 - accuracy: 0.5490
Epoch 2/9
8982/8982 [==============================] - 1s 78us/sample - loss: 1.3029 - accuracy: 0.7182
Epoch 3/9
8982/8982 [==============================] - 1s 80us/sample - loss: 0.9606 - accuracy: 0.7958
Epoch 4/9
8982/8982 [==============================] - 1s 92us/sample - loss: 0.7470 - accuracy: 0.8435
Epoch 5/9
8982/8982 [==============================] - 1s 87us/sample - loss: 0.5814 - accuracy: 0.8815
Epoch 6/9
8982/8982 [==============================] - 1s 85us/sample - loss: 0.4609 - accuracy: 0.9058
Epoch 7/9
8982/8982 [==============================] - 1s 90us/sample - loss: 0.3687 - accuracy: 0.9243
Epoch 8/9
8982/8982 [==============================] - 1s 86us/sample - loss: 0.3021 - accuracy: 0.9363
Epoch 9/9
8982/8982 [==============================] - 1s 76us/sample - loss: 0.2496 - accuracy: 0.9443
2246/2246 [==============================] - 0s 157us/sample - loss: 0.9694 - accuracy: 0.7890
```

In [90]:

```
print(f'Test loss:\t{results[0]:.4f}\nTest Acc:\t{results[1]:.4f}')
```

```
Test loss: 0.9694
Test Acc: 0.7890
```

In [91]:

```
# check accuracy of random baseline
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)
hits_array.mean()
```

Out[91]:

```
0.18432769367764915
```

Generate predictions on new data

In [92]:

```
preds = model.predict(x_test)
```

In [93]:

```
preds[0].shape
```

Out[93]:

```
(46,)
```

In [94]:

```
# coefficients of vector values should be 1 as it forms a probability distribution
np.sum(preds[0])
```

Out[94]:

```
0.99999994
```

In [95]:

```
# class with highest probability
np.argmax(preds[0])
```

Out[95]:

```
3
```

Housing Price Regression Model

In [96]:

```
# load boston housing price data from keras
from tensorflow.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 1us/step
```

In [97]:

```
print(train_data.shape)
print(test_data.shape)
```

```
(404, 13)
(102, 13)
```

Preparing the data

In [98]:

```
# feature-wise normalization to account for different ranges of measured observations
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std
test_data -= mean
test_data /= std
```

Model Definition

In [106]:

```
#
def build_model():
    model = keras.Sequential([
        layers.Dense(64, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(1) # no activation as it is a linear layer
    ])
    model.compile(
        optimizer='rmsprop',
        loss='mse',
        metrics=['mae']
    )
    return model
```

Validation via K-fold Validation

In [102]:

```
k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []
for i in range(k):
    print(f'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=16,
        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
```

In [107]:

```
all_scores
```

Out[107]:

```
[1.9519291, 2.5321727, 2.5141358, 2.418055]
```

In [108]:

```
np.mean(all_scores)
```

Out[108]:

```
2.354073
```

In [110]:

```
# same implementation but saving validation logs
num_epochs = 500
all_mae_histories = []
for i in range(k):
    print(f'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=16,
        verbose=0
    )
    mae_history = history.history['val_mae']
    all_mae_histories.append(mae_history)
```

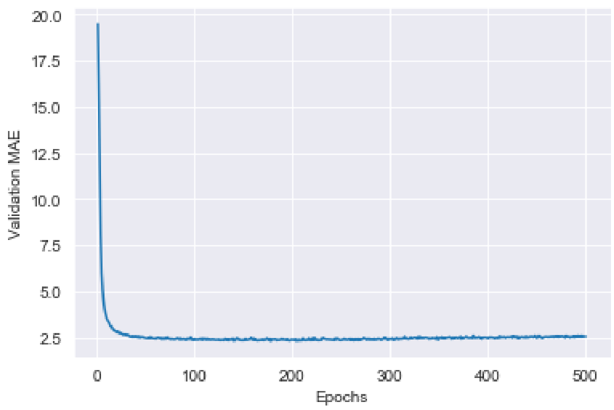
```
Processing fold #0
Processing fold #1
Processing fold #2
Processing fold #3
```

In [111]:

```
# build history of successive mean k-fold validation scores
average_mae_history = [
    np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)
]
```

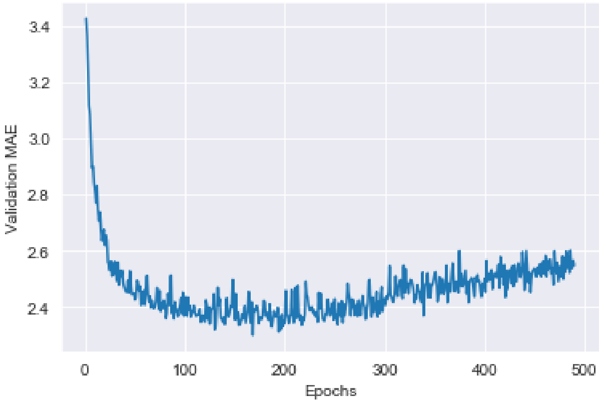
In [112]:

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



In [113]:

```python
# omit first few data points and re-draw the plot
truncated_mae_history = average_mae_history[10:]
plt.plot(range(1, len(truncated_mae_history) + 1), truncated_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



In [114]:

```python
## Training Final Model
```

In [118]:

```python
# retrain model with "better" # of epochs
model = build_model()
model.fit(
    train_data,
    train_targets,
    epochs=130,
    batch_size=16,
    verbose=0
)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

```
102/102 [==============================] - 0s 598us/sample - loss: 17.8547 - mae: 2.8682
```

In [119]:

```python
test_mae_score
```

Out[119]:

```
2.8682077
```

Generate Predictions on New Data

In [120]:

```python
preds = model.predict(test_data)
preds[0]
```

Out[120]:

```
array([9.15679], dtype=float32)
```