In [5]:
```
#Shaquiel Pashtunyar
#DSC650 Assignment 5.1
```

# Assignment 5.1 Deep learning with Movie classifier

In [6]:
```
#Importing data from tensor flow
from tensorflow.keras.datasets import imdb
```

In [7]:
```
# Getting the data into my tables
(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/imd
b.npz
17464789/17464789 [==============================] - 2s 0us/step
```

In [34]:
```
train_data
```

Out[34]:
```
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 17
3, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 11
2, 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, 1
5, 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, 1
8, 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, 53
45, 19, 178, 32]),
       list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 71
5, 8, 118, 1634, 14, 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, 816
3, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 952, 46, 3
7, 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, 685
3, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174, 11,
220, 175, 136, 50, 9, 4373, 228, 8255, 5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 49
1, 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, 1
49, 14, 22, 112, 4, 2401, 311, 12, 16, 3711, 33, 75, 43, 1829, 296, 4, 86, 320, 35, 5
34, 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, 16
5, 1539, 278, 36, 69, 2, 780, 8, 106, 14, 6905, 1338, 18, 6, 22, 12, 215, 28, 610, 4
0, 6, 87, 326, 23, 2300, 21, 23, 22, 12, 272, 40, 57, 31, 11, 4, 22, 47, 6, 2307, 51,
9, 170, 23, 595, 116, 595, 1352, 13, 191, 79, 638, 89, 2, 14, 9, 8, 106, 607, 624, 3
5, 534, 6, 227, 7, 129, 113]),
       ...,
       list([1, 11, 6, 230, 245, 6401, 9, 6, 1225, 446, 2, 45, 2174, 84, 8322, 4007,
21, 4, 912, 84, 2, 325, 725, 134, 2, 1715, 84, 5, 36, 28, 57, 1099, 21, 8, 140, 8, 70
3, 5, 2, 84, 56, 18, 1644, 14, 9, 31, 7, 4, 9406, 1209, 2295, 2, 1008, 18, 6, 20, 20
7, 110, 563, 12, 8, 2901, 2, 8, 97, 6, 20, 53, 4767, 74, 4, 460, 364, 1273, 29, 270,
11, 960, 108, 45, 40, 29, 2961, 395, 11, 6, 4065, 500, 7, 2, 89, 364, 70, 29, 140, 4,
64, 4780, 11, 4, 2678, 26, 178, 4, 529, 443, 2, 5, 27, 710, 117, 2, 8123, 165, 47, 8
4, 37, 131, 818, 14, 595, 10, 10, 61, 1242, 1209, 10, 10, 288, 2260, 1702, 34, 2901,
2, 4, 65, 496, 4, 231, 7, 790, 5, 6, 320, 234, 2766, 234, 1119, 1574, 7, 496, 4, 139,
929, 2901, 2, 7750, 5, 4241, 18, 4, 8497, 2, 250, 11, 1818, 7561, 4, 4217, 5408, 747,
1115, 372, 1890, 1006, 541, 9303, 7, 4, 59, 2, 4, 3586, 2]),
       list([1, 1446, 7079, 69, 72, 3305, 13, 610, 930, 8, 12, 582, 23, 5, 16, 484, 6
85, 54, 349, 11, 4120, 2959, 45, 58, 1466, 13, 197, 12, 16, 43, 23, 2, 5, 62, 30, 14
5, 402, 11, 4131, 51, 575, 32, 61, 369, 71, 66, 770, 12, 1054, 75, 100, 2198, 8, 4, 1
05, 37, 69, 147, 712, 75, 3543, 44, 257, 390, 5, 69, 263, 514, 105, 50, 286, 1814, 2
3, 4, 123, 13, 161, 40, 5, 421, 4, 116, 16, 897, 13, 2, 40, 319, 5872, 112, 6700, 11,
4803, 121, 25, 70, 3468, 4, 719, 3798, 13, 18, 31, 62, 40, 8, 7200, 4, 2, 7, 14, 123,
5, 942, 25, 8, 721, 12, 145, 5, 202, 12, 160, 580, 202, 12, 6, 52, 58, 2, 92, 401, 72
8, 12, 39, 14, 251, 8, 15, 251, 5, 2, 12, 38, 84, 80, 124, 12, 9, 23]),
       list([1, 17, 6, 194, 337, 7, 4, 204, 22, 45, 254, 8, 106, 14, 123, 4, 2, 270,
2, 5, 2, 2, 732, 2098, 101, 405, 39, 14, 1034, 4, 1310, 9, 115, 50, 305, 12, 47, 4, 1
68, 5, 235, 7, 38, 111, 699, 102, 7, 4, 4039, 9245, 9, 24, 6, 78, 1099, 17, 2345, 2,
21, 27, 9685, 6139, 5, 2, 1603, 92, 1183, 4, 1310, 7, 4, 204, 42, 97, 90, 35, 221, 10
9, 29, 127, 27, 118, 8, 97, 12, 157, 21, 6789, 2, 9, 6, 66, 78, 1099, 4, 631, 1191,
5, 2642, 272, 191, 1070, 6, 7585, 8, 2197, 2, 2, 544, 5, 383, 1271, 848, 1468, 2, 49
7, 2, 8, 1597, 8778, 2, 21, 60, 27, 239, 9, 43, 8368, 209, 405, 10, 10, 12, 764, 40,
4, 248, 20, 12, 16, 5, 174, 1791, 72, 7, 51, 6, 1739, 22, 4, 204, 131, 9])],
      dtype=object)
```

In [35]:
```
train_labels
```
Out[35]:
```
array([1, 0, 0, ..., 0, 1, 0], dtype=int64)
```

In [36]:
```
#Removing max sequence
word_index = imdb.get_word_index()

# reverse it by mapping integer indices to words
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])

# Decode the review - indices are offset by 3 because 0, 1, 2 are reserved, indices fo
decoded_review = " ".join(
    [reverse_word_index.get(i - 3, "?") for i in train_data[0]])

decoded_review
```
Out[36]:
```
"? this film was just brilliant casting location scenery story direction everyone's r
eally suited the part they played and you could just imagine being there robert ? is
an amazing actor and now the same being director ? father came from the same scottish
island as myself so i loved the fact there was a real connection with this film the w
itty remarks throughout the film were great it was just brilliant so much that i boug
ht the film as soon as it was released for ? and would recommend it to everyone to wa
tch and the fly fishing was amazing really cried at the end it was so sad and you kno
w what they say if you cry at a film it must have been good and this definitely was a
lso ? to the two little boy's that played the ? of norman and paul they were just bri
lliant children are often left out of the ? list i think because the stars that play
them all grown up are such a big profile for the whole film but these children are am
azing and should be praised for what they have done don't you think the whole story w
as so lovely because it was true and was someone's life after all that was shared wit
h us all"
```

In [37]:
```
# create my training and testing dataset
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)
```

In [17]:
```
x_train[0]
```
Out[17]:
```
array([0., 1., 1., ..., 0., 0., 0.])
```

In [18]:
```
# my target to go along with the x
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

In [19]:
```
# Keras implementation
from keras import models
from keras import layers
```

In [20]:
```
#Sequential modeling
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
```

```python
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [21]:
```python
model.compile(optimizer="rmsprop",
            loss="binary_crossentropy",
            metrics=["accuracy"])
```

In [22]:
```python
from keras import optimizers
```

In [23]:
```python
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
            loss='binary_crossentropy',
            metrics=['accuracy'])
```

```
C:\Users\spashtunyar\Anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\rmspro
p.py:140: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

In [39]:
```python
#Adding losses and metrics
from keras import losses
from keras import metrics

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

In [25]:
```python
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

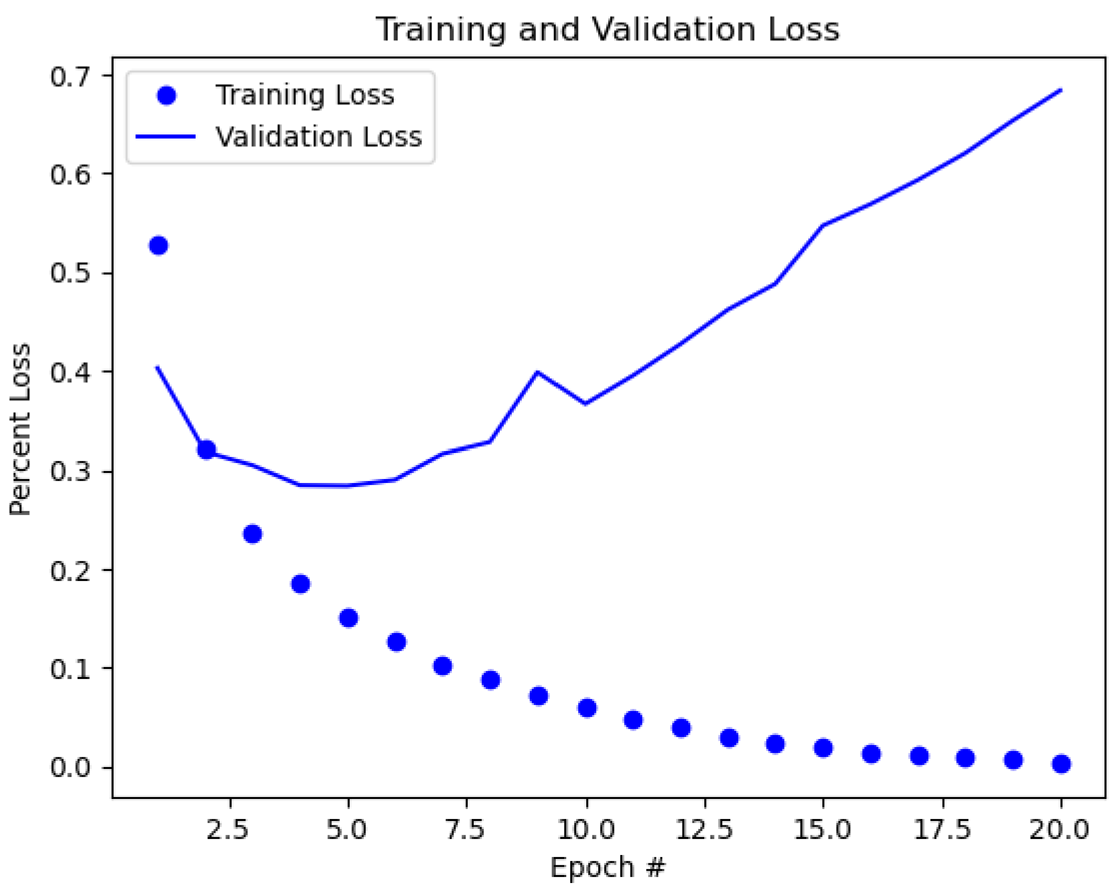
In [26]:
```python
# finally time to train my model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 2s 35ms/step - loss: 0.5268 - binary_accurac
y: 0.7945 - val_loss: 0.4029 - val_binary_accuracy: 0.8682
Epoch 2/20
30/30 [==============================] - 0s 13ms/step - loss: 0.3208 - binary_accurac
y: 0.9008 - val_loss: 0.3185 - val_binary_accuracy: 0.8796
Epoch 3/20
30/30 [==============================] - 0s 14ms/step - loss: 0.2354 - binary_accurac
y: 0.9243 - val_loss: 0.3049 - val_binary_accuracy: 0.8742
Epoch 4/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1855 - binary_accurac
y: 0.9393 - val_loss: 0.2846 - val_binary_accuracy: 0.8862
Epoch 5/20
30/30 [==============================] - 0s 13ms/step - loss: 0.1504 - binary_accurac
y: 0.9515 - val_loss: 0.2840 - val_binary_accuracy: 0.8870
Epoch 6/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1279 - binary_accurac
y: 0.9591 - val_loss: 0.2901 - val_binary_accuracy: 0.8865
Epoch 7/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1036 - binary_accurac
y: 0.9690 - val_loss: 0.3163 - val_binary_accuracy: 0.8817
Epoch 8/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0879 - binary_accurac
y: 0.9750 - val_loss: 0.3280 - val_binary_accuracy: 0.8796
Epoch 9/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0720 - binary_accurac
y: 0.9806 - val_loss: 0.3986 - val_binary_accuracy: 0.8660
Epoch 10/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0599 - binary_accurac
y: 0.9854 - val_loss: 0.3665 - val_binary_accuracy: 0.8783
Epoch 11/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0474 - binary_accurac
y: 0.9893 - val_loss: 0.3951 - val_binary_accuracy: 0.8760
Epoch 12/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0392 - binary_accurac
y: 0.9919 - val_loss: 0.4270 - val_binary_accuracy: 0.8723
Epoch 13/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0294 - binary_accurac
y: 0.9947 - val_loss: 0.4619 - val_binary_accuracy: 0.8726
Epoch 14/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0242 - binary_accurac
y: 0.9953 - val_loss: 0.4880 - val_binary_accuracy: 0.8713
Epoch 15/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0192 - binary_accurac
y: 0.9968 - val_loss: 0.5469 - val_binary_accuracy: 0.8640
Epoch 16/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0146 - binary_accurac
y: 0.9987 - val_loss: 0.5686 - val_binary_accuracy: 0.8640
Epoch 17/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0118 - binary_accurac
y: 0.9987 - val_loss: 0.5930 - val_binary_accuracy: 0.8714
Epoch 18/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0086 - binary_accurac
y: 0.9990 - val_loss: 0.6203 - val_binary_accuracy: 0.8665
Epoch 19/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0084 - binary_accurac
y: 0.9987 - val_loss: 0.6533 - val_binary_accuracy: 0.8665
Epoch 20/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0037 - binary_accurac
y: 0.9999 - val_loss: 0.6838 - val_binary_accuracy: 0.8680
```

In [27]:
```
#creating dictionary keys
history_dict = history.history
history_dict.keys()
```

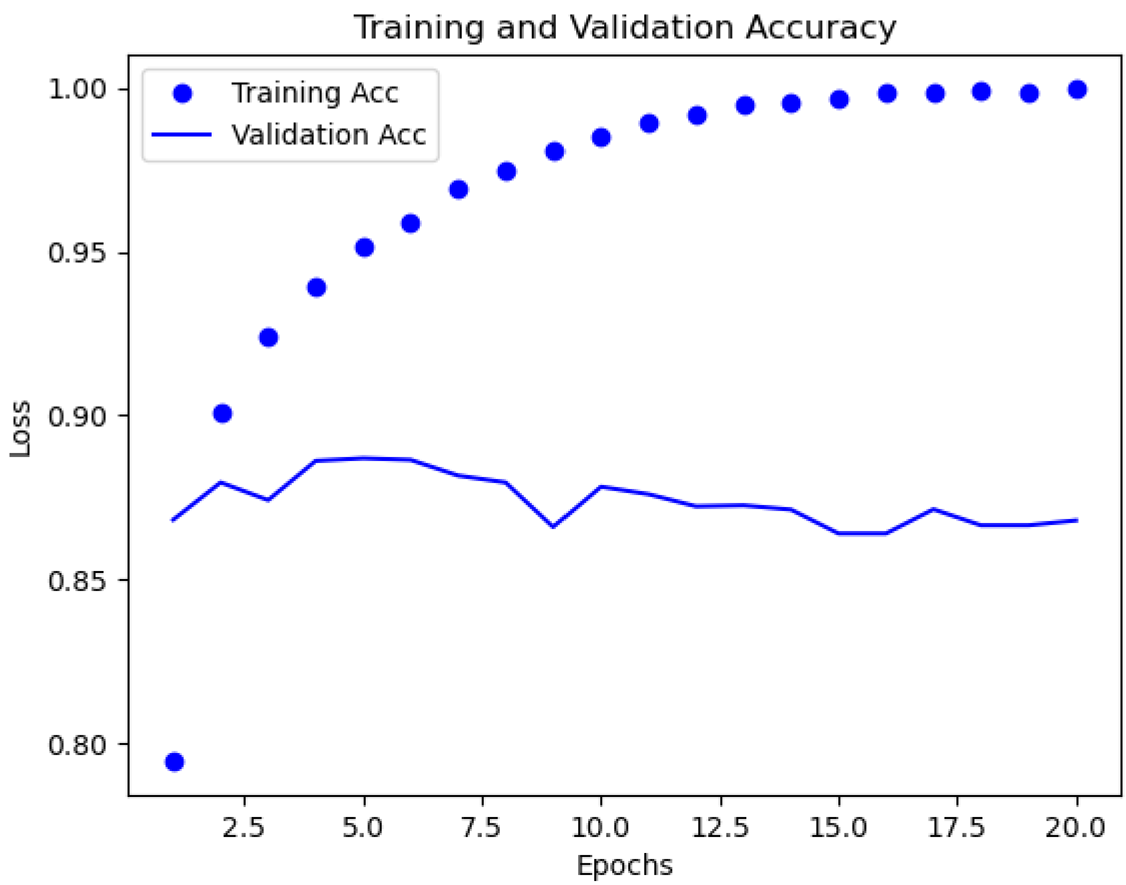
Out[27]:
```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

In [29]:
```
#Plotting the training data with pyplot
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
acc = history_dict["binary_accuracy"]
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('Epoch #')
plt.ylabel('Percent Loss')
plt.legend()
plt.show()
```




In [30]:
```
# plot training & validation accuracy
plt.clf() # clears figure
acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['val_binary_accuracy']

plt.plot(epochs, acc, 'bo', label='Training Acc')
plt.plot(epochs, val_acc_values, 'b', label='Validation Acc')
plt.title('Training and Validation Accuracy')
```

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




In [31]:

```
# retraining a model from scratch
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 8ms/step - loss: 0.4739 - accuracy: 0.814
1
Epoch 2/4
49/49 [==============================] - 0s 8ms/step - loss: 0.2692 - accuracy: 0.906
7
Epoch 3/4
49/49 [==============================] - 0s 8ms/step - loss: 0.2037 - accuracy: 0.927
6
Epoch 4/4
49/49 [==============================] - 0s 7ms/step - loss: 0.1709 - accuracy: 0.940
2
782/782 [==============================] - 2s 2ms/step - loss: 0.2922 - accuracy: 0.8
839
```

In [32]:

```
results
```

Out[32]:

```
[0.29221242666244507, 0.8839200139045715]
```

In [33]:

```
#Test Predictions
model.predict(x_test)
```

```
782/782 [==============================] - 1s 1ms/step
```

Out[33]:

```
array([[0.18759479],
       [0.99955153],
       [0.8243988 ],
       ...,
       [0.09366523],
       [0.06851666],
       [0.4542775 ]], dtype=float32)
```

# Part 2 Implementing 3.5 classifier

In [40]:

```
#Getting reuters dataset
import keras
from keras.datasets import reuters
```

In [41]:

```
#Same set, getting reuters dataset
(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/reu
ters.npz
2110848/2110848 [==============================] - 0s 0us/step
```

In [43]:

```
print(len(train_data))
```

```
8982
```

In [44]:

```
print(len(test_data))
```

```
2246
```

In [45]:

```
print(train_data[10])
```

```
[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]
```

In [46]:
```
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/reu
ters_word_index.json
550378/550378 [==============================] - 0s 0us/step
```

In [48]:
```
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
```

In [50]:
```
#Creating training datasets
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

In [51]:
```
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
```

In [52]:
```
one_hot_train_labels = to_one_hot(train_labels)
one_hot_test_labels = to_one_hot(test_labels)
```

In [53]:
```
from keras.utils.np_utils import to_categorical
one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

In [54]:
```
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'))
```

In [57]:
```
model.compile(optimizer='rmsprop',
        loss='categorical_crossentropy',
        metrics='accuracy')
```

In [58]:
```
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:]
```

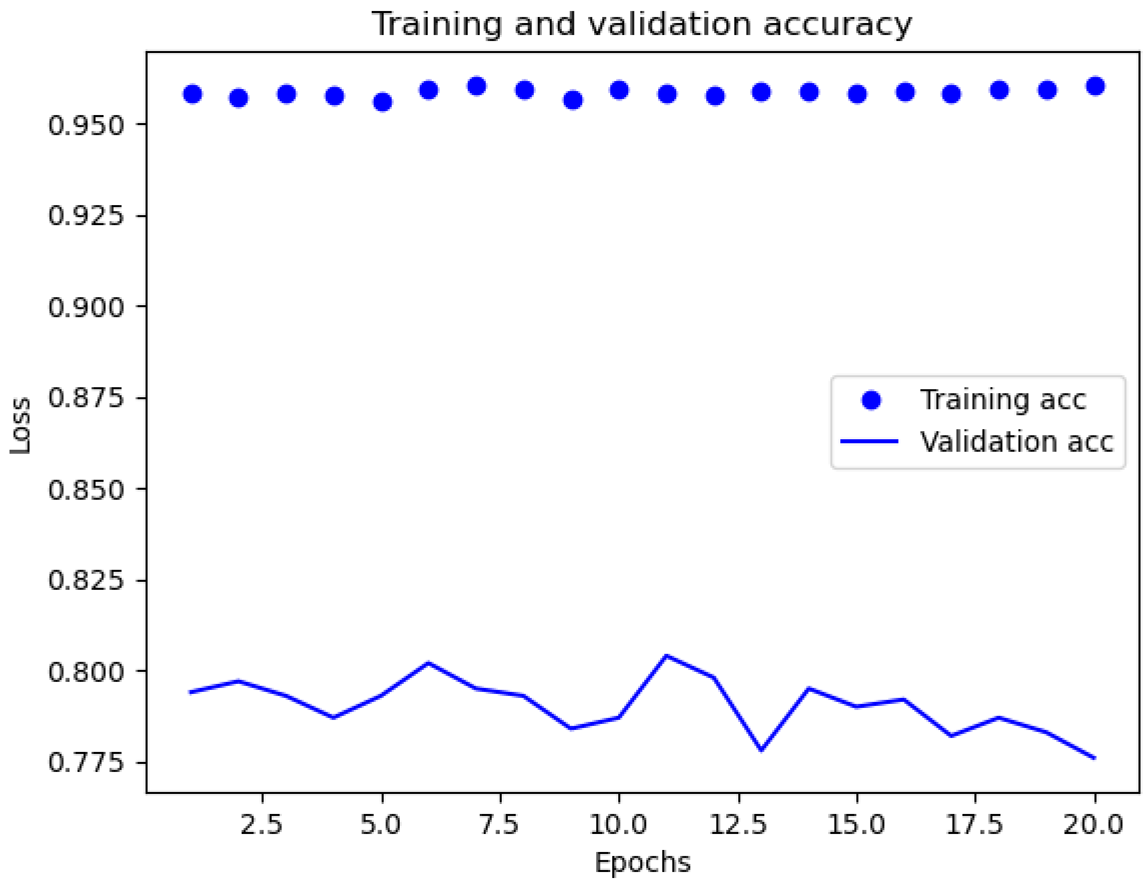
In [60]:
```
#20 EPOCHS model training
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
16/16 [==============================] - 0s 20ms/step - loss: 0.1096 - accuracy: 0.95
85 - val_loss: 1.1502 - val_accuracy: 0.7940
Epoch 2/20
16/16 [==============================] - 0s 16ms/step - loss: 0.1078 - accuracy: 0.95
74 - val_loss: 1.1601 - val_accuracy: 0.7970
Epoch 3/20
16/16 [==============================] - 0s 16ms/step - loss: 0.1049 - accuracy: 0.95
80 - val_loss: 1.1908 - val_accuracy: 0.7930
Epoch 4/20
16/16 [==============================] - 0s 18ms/step - loss: 0.1008 - accuracy: 0.95
77 - val_loss: 1.2393 - val_accuracy: 0.7870
Epoch 5/20
16/16 [==============================] - 0s 17ms/step - loss: 0.1050 - accuracy: 0.95
59 - val_loss: 1.1985 - val_accuracy: 0.7930
Epoch 6/20
16/16 [==============================] - 0s 17ms/step - loss: 0.0969 - accuracy: 0.95
94 - val_loss: 1.1530 - val_accuracy: 0.8020
Epoch 7/20
16/16 [==============================] - 0s 17ms/step - loss: 0.0999 - accuracy: 0.96
05 - val_loss: 1.2202 - val_accuracy: 0.7950
Epoch 8/20
16/16 [==============================] - 0s 17ms/step - loss: 0.0957 - accuracy: 0.95
95 - val_loss: 1.2355 - val_accuracy: 0.7930
Epoch 9/20
16/16 [==============================] - 0s 17ms/step - loss: 0.0996 - accuracy: 0.95
65 - val_loss: 1.2885 - val_accuracy: 0.7840
Epoch 10/20
16/16 [==============================] - 0s 17ms/step - loss: 0.0933 - accuracy: 0.95
95 - val_loss: 1.3203 - val_accuracy: 0.7870
Epoch 11/20
16/16 [==============================] - 0s 19ms/step - loss: 0.0948 - accuracy: 0.95
84 - val_loss: 1.2486 - val_accuracy: 0.8040
Epoch 12/20
16/16 [==============================] - 0s 19ms/step - loss: 0.0956 - accuracy: 0.95
75 - val_loss: 1.2524 - val_accuracy: 0.7980
Epoch 13/20
16/16 [==============================] - 0s 17ms/step - loss: 0.0918 - accuracy: 0.95
89 - val_loss: 1.3519 - val_accuracy: 0.7780
Epoch 14/20
16/16 [==============================] - 0s 17ms/step - loss: 0.0931 - accuracy: 0.95
89 - val_loss: 1.2459 - val_accuracy: 0.7950
Epoch 15/20
16/16 [==============================] - 0s 17ms/step - loss: 0.0885 - accuracy: 0.95
83 - val_loss: 1.3602 - val_accuracy: 0.7900
Epoch 16/20
16/16 [==============================] - 0s 17ms/step - loss: 0.0920 - accuracy: 0.95
87 - val_loss: 1.2805 - val_accuracy: 0.7920
Epoch 17/20
16/16 [==============================] - 0s 17ms/step - loss: 0.0910 - accuracy: 0.95
82 - val_loss: 1.3820 - val_accuracy: 0.7820
Epoch 18/20
16/16 [==============================] - 0s 17ms/step - loss: 0.0902 - accuracy: 0.95
94 - val_loss: 1.3538 - val_accuracy: 0.7870
Epoch 19/20
16/16 [==============================] - 0s 17ms/step - loss: 0.0879 - accuracy: 0.95
97 - val_loss: 1.3522 - val_accuracy: 0.7830
Epoch 20/20
16/16 [==============================] - 0s 18ms/step - loss: 0.0876 - accuracy: 0.96
04 - val_loss: 1.4541 - val_accuracy: 0.7760
```

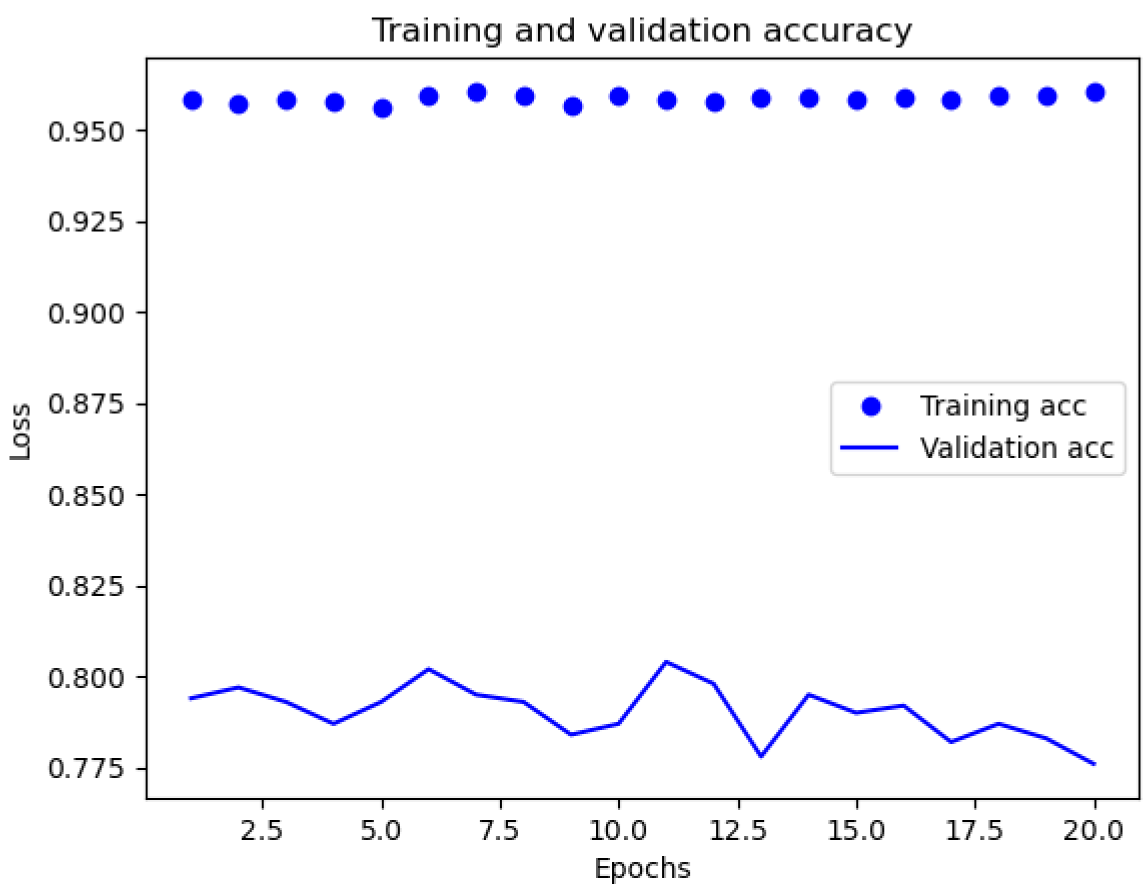
In [61]:
```
#plotting the results, same code as above from section 1
import matplotlib.pyplot as plt
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)

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()
```


In [64]:
```
for key in history.history.keys():
    print(key)
```

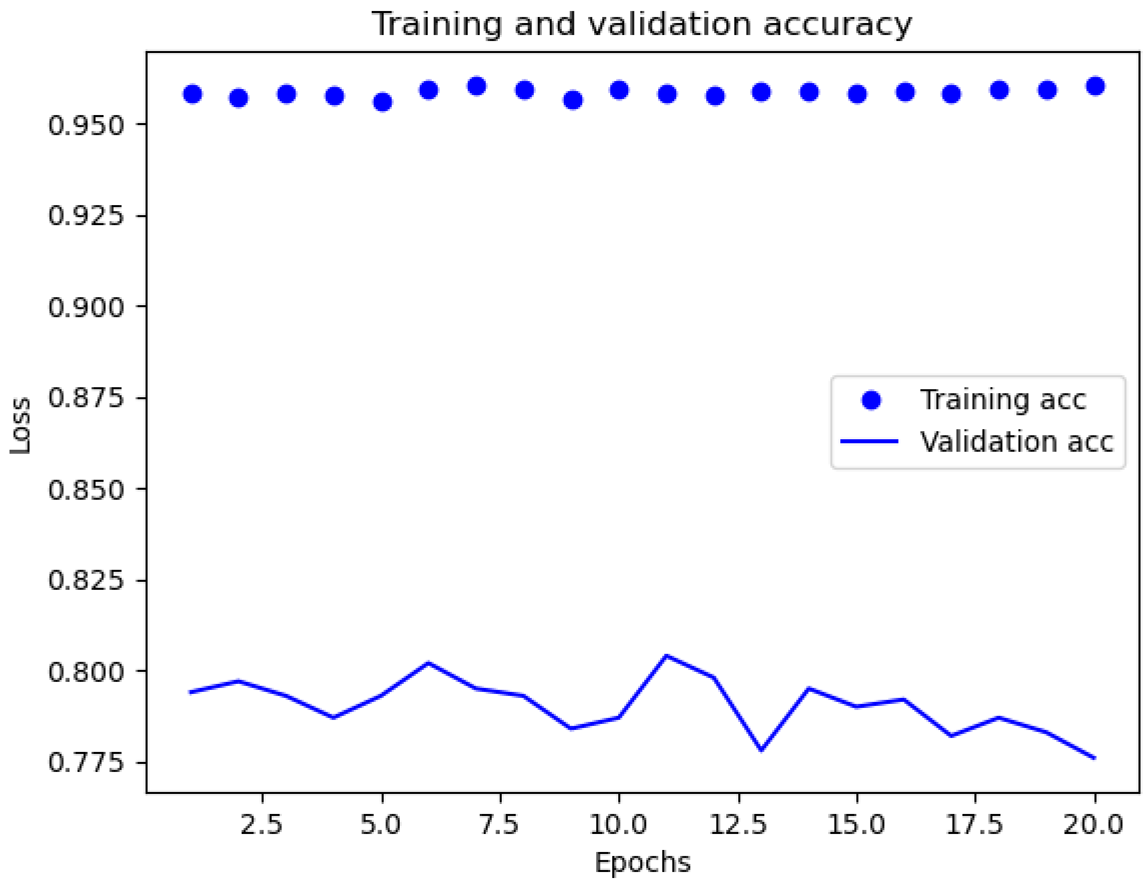
```
loss
```



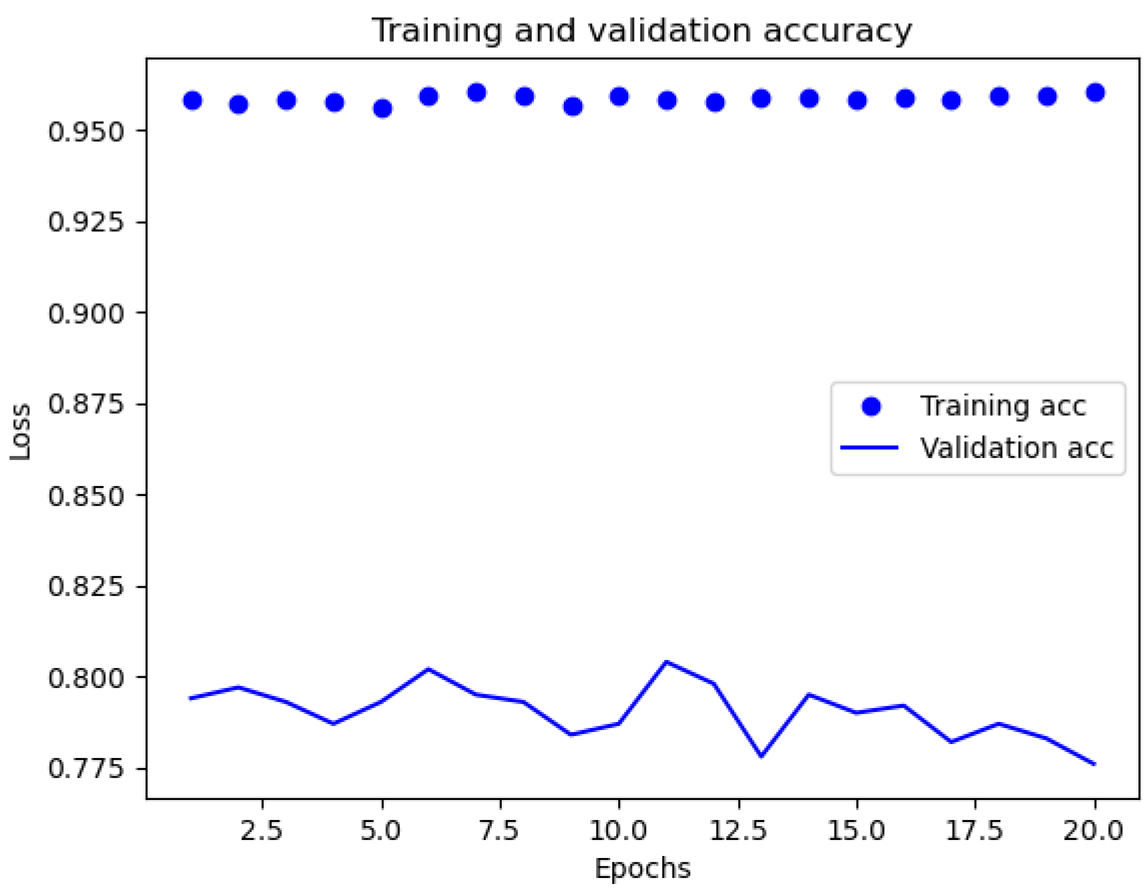

accuracy




val_loss

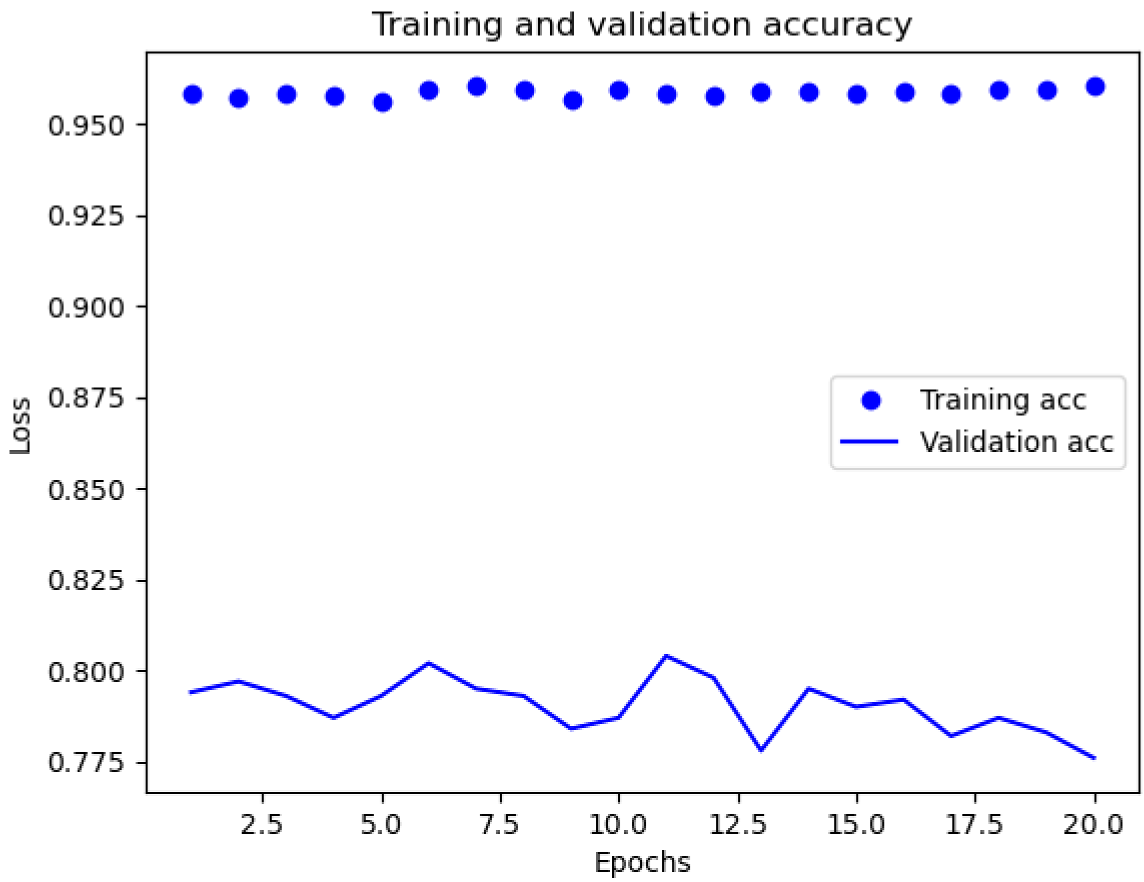



val_accuracy




In [68]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
```

```python
    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()
```




In [69]:

```python
# retrain model from scratch
model = models.Sequential([
  layers.Dense(64, activation="relu", input_shape = (10000,)),
  layers.Dense(64, activation="relu"),
  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 34ms/step - loss: 2.6953 - accuracy: 0.52
37 - val_loss: 1.7832 - val_accuracy: 0.6310
Epoch 2/9
16/16 [==============================] - 0s 17ms/step - loss: 1.4653 - accuracy: 0.69
11 - val_loss: 1.3148 - val_accuracy: 0.7190
Epoch 3/9
16/16 [==============================] - 0s 21ms/step - loss: 1.0693 - accuracy: 0.77
57 - val_loss: 1.1222 - val_accuracy: 0.7690
Epoch 4/9
16/16 [==============================] - 0s 16ms/step - loss: 0.8325 - accuracy: 0.82
86 - val_loss: 1.0087 - val_accuracy: 0.7900
Epoch 5/9
16/16 [==============================] - 0s 20ms/step - loss: 0.6594 - accuracy: 0.86
52 - val_loss: 0.9784 - val_accuracy: 0.7990
Epoch 6/9
16/16 [==============================] - 0s 16ms/step - loss: 0.5300 - accuracy: 0.89
19 - val_loss: 0.9290 - val_accuracy: 0.8080
Epoch 7/9
16/16 [==============================] - 0s 16ms/step - loss: 0.4256 - accuracy: 0.91
19 - val_loss: 0.8883 - val_accuracy: 0.8150
Epoch 8/9
16/16 [==============================] - 0s 16ms/step - loss: 0.3475 - accuracy: 0.92
67 - val_loss: 0.9179 - val_accuracy: 0.8140
Epoch 9/9
16/16 [==============================] - 0s 20ms/step - loss: 0.2873 - accuracy: 0.93
92 - val_loss: 0.8924 - val_accuracy: 0.8160
71/71 [==============================] - 0s 2ms/step - loss: 0.9965 - accuracy: 0.787
6
```

In [70]:
```
results
```

Out[70]:
```
[0.9964867830276489, 0.7876224517822266]
```

In [71]:
```
import copy
test_labels_copy = copy.copy(test_labels)
np.random.shuffle(test_labels_copy)
float(np.sum(np.array(test_labels) == np.array(test_labels_copy))) /len(test_labels)
```

Out[71]:
```
0.1918967052537845
```

In [72]:
```
predictions = model.predict(x_test)
```

```
71/71 [==============================] - 0s 2ms/step
```

In [73]:
```
predictions[0].shape
```

Out[73]:
```
(46,)
```

In [74]:
```
np.sum(predictions[0])
```

Out[74]:
```
1.0000004
```

In [75]:
```
y_train = np.array(train_labels)
y_test = np.array(test_labels)
```

In [76]:
```
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
```

```
                  metrics=["accuracy"])
```

In [77]:
```
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 13ms/step - loss: 2.8058 - accuracy: 0.25
65 - val_loss: 1.9797 - val_accuracy: 0.6020
Epoch 2/20
63/63 [==============================] - 1s 10ms/step - loss: 1.5739 - accuracy: 0.60
31 - val_loss: 1.4724 - val_accuracy: 0.6010
Epoch 3/20
63/63 [==============================] - 1s 8ms/step - loss: 1.2589 - accuracy: 0.650
6 - val_loss: 1.3555 - val_accuracy: 0.6480
Epoch 4/20
63/63 [==============================] - 1s 8ms/step - loss: 1.0997 - accuracy: 0.703
5 - val_loss: 1.2851 - val_accuracy: 0.6910
Epoch 5/20
63/63 [==============================] - 1s 9ms/step - loss: 0.9798 - accuracy: 0.750
7 - val_loss: 1.2536 - val_accuracy: 0.7120
Epoch 6/20
63/63 [==============================] - 1s 9ms/step - loss: 0.8806 - accuracy: 0.770
1 - val_loss: 1.2448 - val_accuracy: 0.7220
Epoch 7/20
63/63 [==============================] - 1s 8ms/step - loss: 0.7972 - accuracy: 0.790
8 - val_loss: 1.2692 - val_accuracy: 0.7200
Epoch 8/20
63/63 [==============================] - 1s 8ms/step - loss: 0.7295 - accuracy: 0.805
3 - val_loss: 1.2718 - val_accuracy: 0.7230
Epoch 9/20
63/63 [==============================] - 1s 9ms/step - loss: 0.6735 - accuracy: 0.817
0 - val_loss: 1.2979 - val_accuracy: 0.7330
Epoch 10/20
63/63 [==============================] - 1s 8ms/step - loss: 0.6260 - accuracy: 0.831
0 - val_loss: 1.3119 - val_accuracy: 0.7350
Epoch 11/20
63/63 [==============================] - 1s 8ms/step - loss: 0.5816 - accuracy: 0.838
1 - val_loss: 1.3554 - val_accuracy: 0.7340
Epoch 12/20
63/63 [==============================] - 1s 9ms/step - loss: 0.5466 - accuracy: 0.848
8 - val_loss: 1.3826 - val_accuracy: 0.7410
Epoch 13/20
63/63 [==============================] - 1s 8ms/step - loss: 0.5162 - accuracy: 0.853
0 - val_loss: 1.4549 - val_accuracy: 0.7380
Epoch 14/20
63/63 [==============================] - 1s 8ms/step - loss: 0.4885 - accuracy: 0.862
3 - val_loss: 1.5206 - val_accuracy: 0.7440
Epoch 15/20
63/63 [==============================] - 1s 8ms/step - loss: 0.4624 - accuracy: 0.869
6 - val_loss: 1.5241 - val_accuracy: 0.7400
Epoch 16/20
63/63 [==============================] - 1s 8ms/step - loss: 0.4440 - accuracy: 0.873
1 - val_loss: 1.6299 - val_accuracy: 0.7340
Epoch 17/20
63/63 [==============================] - 1s 8ms/step - loss: 0.4231 - accuracy: 0.880
9 - val_loss: 1.6232 - val_accuracy: 0.7420
Epoch 18/20
63/63 [==============================] - 1s 9ms/step - loss: 0.4053 - accuracy: 0.883
1 - val_loss: 1.7539 - val_accuracy: 0.7330
Epoch 19/20
63/63 [==============================] - 1s 8ms/step - loss: 0.3904 - accuracy: 0.885
5 - val_loss: 1.7902 - val_accuracy: 0.7280
Epoch 20/20
63/63 [==============================] - 1s 8ms/step - loss: 0.3768 - accuracy: 0.891
5 - val_loss: 1.8838 - val_accuracy: 0.7200
```

```
Out[77]: <keras.callbacks.History at 0x27336778490>
```

```
In [79]: # we got 72% accuracy which isn't to bad
```

## 5.3 Housing price regression with deep learning

```
In [80]: #Housing data from keras
         from keras.datasets import boston_housing
```

```
In [81]: #Same data load method
         (train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/bos
ton_housing.npz
57026/57026 [==============================] - 0s 1us/step
```

```
In [83]: train_targets
```

```
Out[83]:
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])
```

In [84]:
```
# data preparation
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std
test_data -= mean
test_data /= std
```

In [85]:
```
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
```

In [86]:
```
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=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 [87]:
```
#scores from previous processing
all_scores
```

Out[87]:
```
[1.9889638423919678,
 2.4483425617218018,
 2.4551186561584473,
 2.4192421436309814]
```

In [88]:
```
#average of the 4
np.mean(all_scores)
```

Out[88]:
```
2.3279168009757996
```

In [89]:

```
# save validation logs for each fold
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=16, verbose=0)
    mae_history = history.history["val_mae"]
    all_mae_histories.append(mae_history)
```

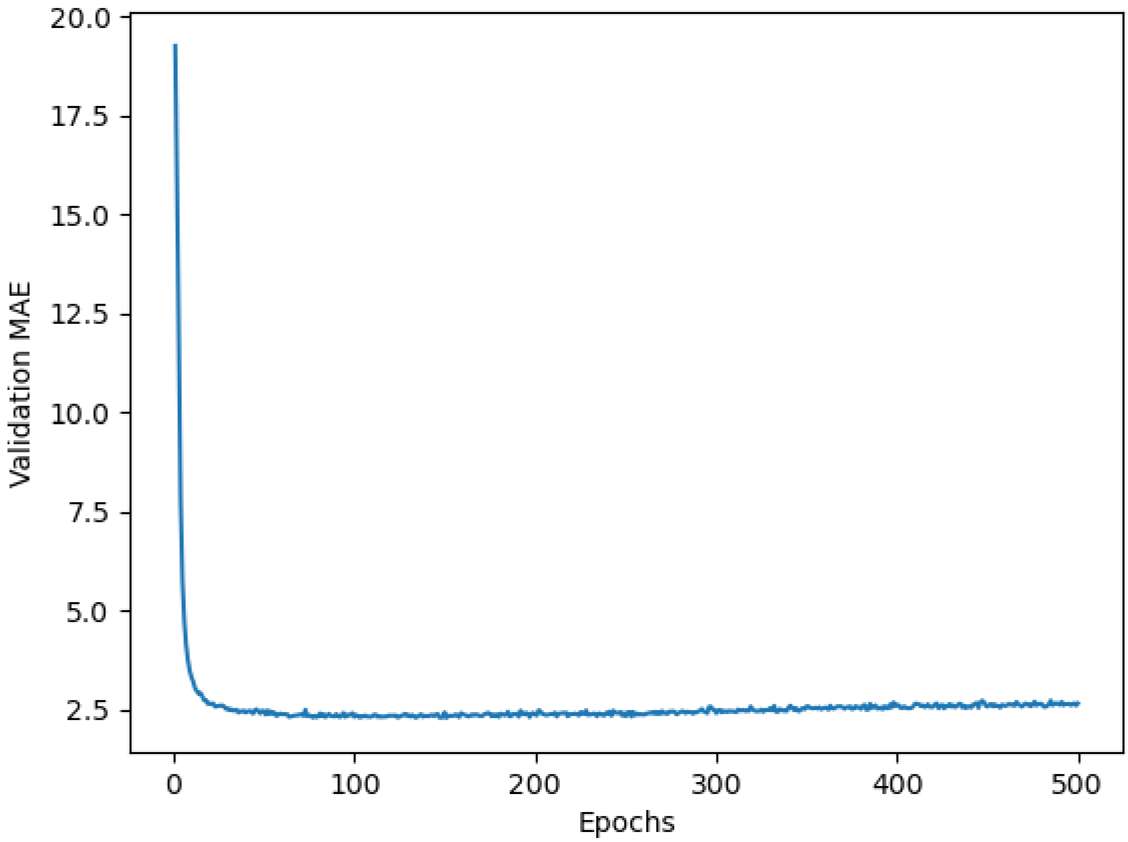
```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
```

In [90]:

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

In [91]:

```
# ploting the results and MAE
plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel("Epochs")
plt.ylabel("Validation MAE")
plt.show()
```

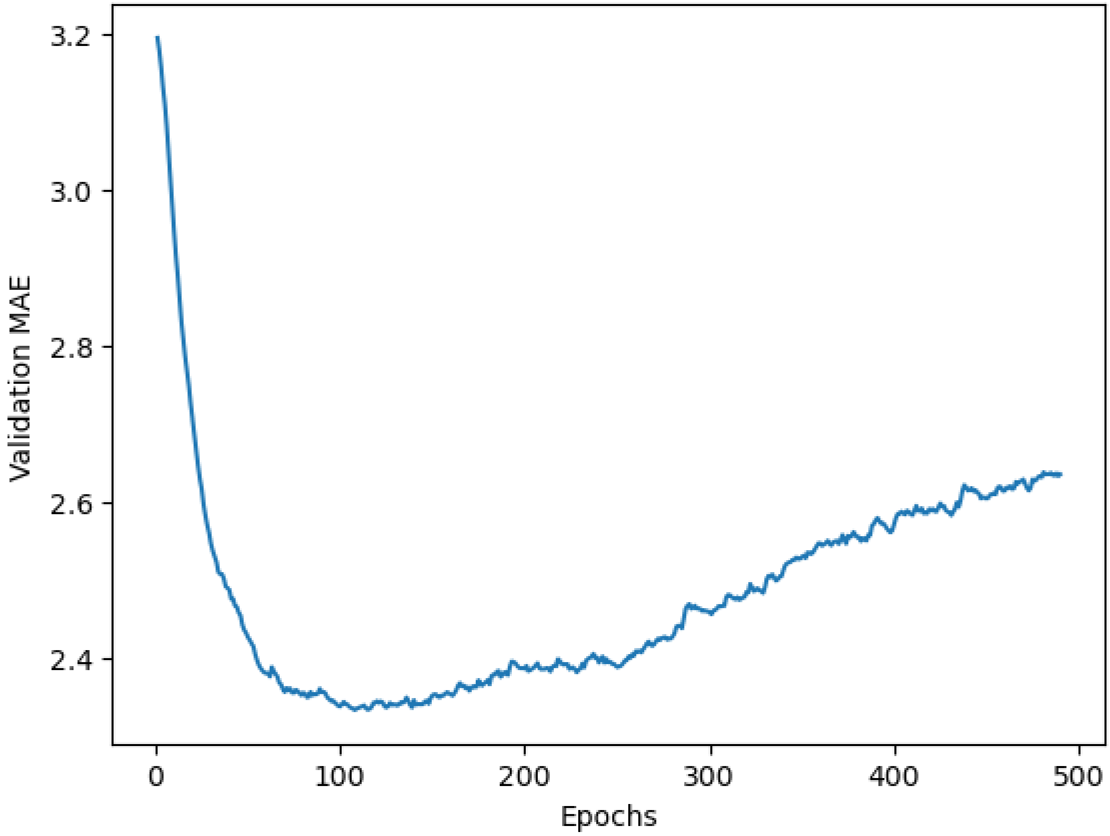



In [92]:

```
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:])
```

In [93]:

```
# plot validation scores but excluding the first points to remove the deep curve we sc
plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel("Epochs")
plt.ylabel("Validation MAE")
plt.show()
```




In [94]:
```
# train the model
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 2ms/step - loss: 17.5741 - mae: 2.7417
```

In [95]:
```
#Testing mae score
test_mae_score
```

Out[95]:
```
2.741657257080078
```

In [96]:
```
# off by not much at all
```

In [ ]: