In [2]:
```python
#Import libraries
import warnings
warnings.filterwarnings("ignore")
import keras
import numpy as np
from keras import models
from keras import layers
from keras import optimizers
from keras import losses
from keras import metrics
import copy
import matplotlib.pyplot as plt
```

Assignment 5.1 - Movie Reviews

In [3]:
```python
#Import data
from keras.datasets import imdb
```

In [4]:
```python
#Split data
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

In [5]:
```python
#Vectorize sequence with dimension 10000
def vectorize_sequence(sequences, dimension=10000):
    #create all-zero matrix of shape (len(seq), dim)
    results = np.zeros((len(sequences),dimension))
    for i, seq in enumerate(sequences):
        results[i,seq]= 1.
    return results

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

In [6]:
```python
#vectorized labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

In [7]:
```python
#Building model
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'))
```

```
2023-04-16 23:24:27.487735: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is opti
mized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-cri
tical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

In [8]:
```python
#Compile model
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [9]:
```python
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [10]:
```python
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

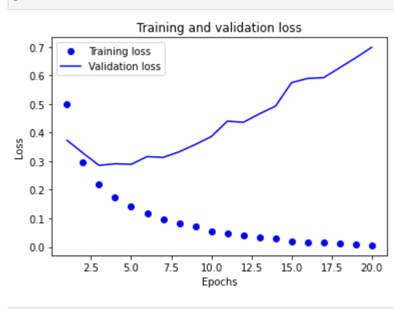
In [11]:
```python
#Validation of the model
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]

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 [==============================] - 1s 29ms/step - loss: 0.4986 - binary_accuracy: 0.7897 - val_loss: 0.37
30 - val_binary_accuracy: 0.8770
Epoch 2/20
30/30 [==============================] - 0s 14ms/step - loss: 0.2959 - binary_accuracy: 0.9036 - val_loss: 0.32
82 - val_binary_accuracy: 0.8655
Epoch 3/20
30/30 [==============================] - 0s 14ms/step - loss: 0.2173 - binary_accuracy: 0.9300 - val_loss: 0.28
53 - val_binary_accuracy: 0.8884
Epoch 4/20
30/30 [==============================] - 0s 15ms/step - loss: 0.1729 - binary_accuracy: 0.9424 - val_loss: 0.29
08 - val_binary_accuracy: 0.8835
Epoch 5/20
30/30 [==============================] - 0s 14ms/step - loss: 0.1407 - binary_accuracy: 0.9563 - val_loss: 0.28
90 - val_binary_accuracy: 0.8852
Epoch 6/20
30/30 [==============================] - 0s 13ms/step - loss: 0.1170 - binary_accuracy: 0.9633 - val_loss: 0.31
59 - val_binary_accuracy: 0.8782
Epoch 7/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0967 - binary_accuracy: 0.9720 - val_loss: 0.31
33 - val_binary_accuracy: 0.8843
Epoch 8/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0812 - binary_accuracy: 0.9769 - val_loss: 0.33
29 - val_binary_accuracy: 0.8809
Epoch 9/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0710 - binary_accuracy: 0.9801 - val_loss: 0.35
83 - val_binary_accuracy: 0.8799
Epoch 10/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0559 - binary_accuracy: 0.9853 - val_loss: 0.38
61 - val_binary_accuracy: 0.8790
Epoch 11/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0481 - binary_accuracy: 0.9879 - val_loss: 0.43
99 - val_binary_accuracy: 0.8724
Epoch 12/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0417 - binary_accuracy: 0.9899 - val_loss: 0.43
59 - val_binary_accuracy: 0.8732
Epoch 13/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0333 - binary_accuracy: 0.9923 - val_loss: 0.46
56 - val_binary_accuracy: 0.8716
Epoch 14/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0304 - binary_accuracy: 0.9932 - val_loss: 0.49
23 - val_binary_accuracy: 0.8716
Epoch 15/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0200 - binary_accuracy: 0.9968 - val_loss: 0.57
47 - val_binary_accuracy: 0.8658
Epoch 16/20
30/30 [==============================] - 0s 14ms/step - loss: 0.0160 - binary_accuracy: 0.9981 - val_loss: 0.58
92 - val_binary_accuracy: 0.8590
Epoch 17/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0152 - binary_accuracy: 0.9978 - val_loss: 0.59
20 - val_binary_accuracy: 0.8664
Epoch 18/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0123 - binary_accuracy: 0.9982 - val_loss: 0.62
66 - val_binary_accuracy: 0.8657
Epoch 19/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0104 - binary_accuracy: 0.9981 - val_loss: 0.66
14 - val_binary_accuracy: 0.8643
Epoch 20/20
30/30 [==============================] - 0s 13ms/step - loss: 0.0056 - binary_accuracy: 0.9996 - val_loss: 0.69
81 - val_binary_accuracy: 0.8640
```

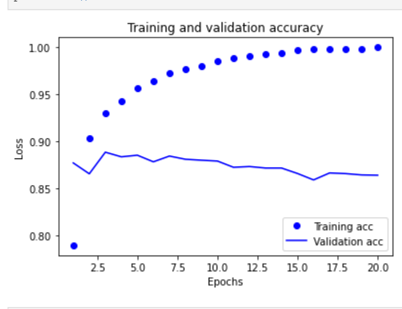
In [12]:
```python
history_dict = history.history
history_dict.keys()
```

Out[12]:
```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

In [13]:
```python
# Data Visualization

acc = history.history['binary_accuracy']
val_acc = history.history['val_binary_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

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

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
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 [14]:
```python
plt.clf()   # clear 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, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



In [15]:
```python
#Build new model using 4 epochs
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 9ms/step - loss: 0.4404 - accuracy: 0.8170
Epoch 2/4
49/49 [==============================] - 0s 9ms/step - loss: 0.2512 - accuracy: 0.9105
Epoch 3/4
49/49 [==============================] - 0s 9ms/step - loss: 0.1968 - accuracy: 0.9295
Epoch 4/4
49/49 [==============================] - 0s 9ms/step - loss: 0.1656 - accuracy: 0.9404
782/782 [==============================] - 1s 1ms/step - loss: 0.3289 - accuracy: 0.8718
```

In [16]:
```python
#Display results of naive approach
results
```

Out[16]:
```
[0.3288571834564209, 0.8718000054359436]
```

In [17]:
```python
#Predict the results
model.predict(x_test)
```

```
782/782 [==============================] - 1s 990us/step
```

Out[17]:
```
array([[0.3457677 ],
       [0.99996424],
       [0.9860464 ],
       ...,
       [0.22828163],
       [0.13901812],
       [0.76328766]], dtype=float32)
```

Assignment 5.2 - News Classifier

In [18]:
```python
#importing data set
from keras.datasets import reuters
```

In [19]:
```python
#Splitting data
(train_data, train_labels), (test_data, test_labels) = reuters.load_data(num_words=10000)
```

In [20]:
```python
#Data preparation using same vectorizer from 5.1
x_train = vectorize_sequence(train_data)
x_test = vectorize_sequence(test_data)
```

In [21]:
```python
#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
```

In [22]:
```python
# Vectorized training and test labels
one_hot_train_labels = to_one_hot(train_labels)
one_hot_test_labels = to_one_hot(test_labels)
```

In [23]:
```python
from keras.utils.np_utils import to_categorical
one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

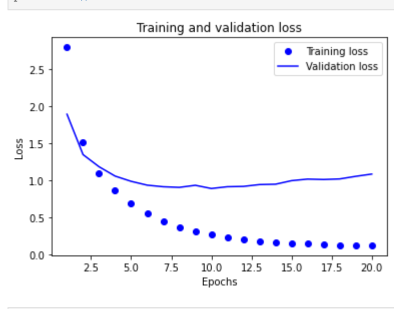
In [24]:
```python
#Building model
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
```

In [25]:
```python
#Model compilation
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

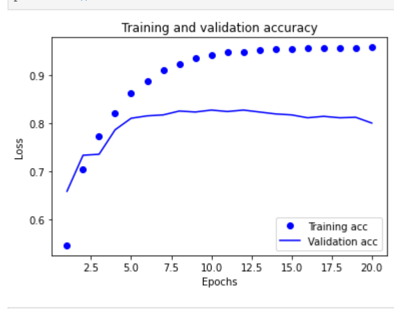
In [26]:
```python
#Model validation
x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]

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 26ms/step - loss: 2.7911 - accuracy: 0.5455 - val_loss: 1.8878 - va
l_accuracy: 0.6580
Epoch 2/20
16/16 [==============================] - 0s 17ms/step - loss: 1.5126 - accuracy: 0.7037 - val_loss: 1.3432 - va
l_accuracy: 0.7330
Epoch 3/20
16/16 [==============================] - 0s 16ms/step - loss: 1.0858 - accuracy: 0.7726 - val_loss: 1.1792 - va
l_accuracy: 0.7350
Epoch 4/20
16/16 [==============================] - 0s 16ms/step - loss: 0.8578 - accuracy: 0.8211 - val_loss: 1.0532 - va
l_accuracy: 0.7860
Epoch 5/20
16/16 [==============================] - 0s 16ms/step - loss: 0.6819 - accuracy: 0.8616 - val_loss: 0.9824 - va
l_accuracy: 0.8100
Epoch 6/20
16/16 [==============================] - 0s 16ms/step - loss: 0.5530 - accuracy: 0.8866 - val_loss: 0.9308 - va
l_accuracy: 0.8150
Epoch 7/20
16/16 [==============================] - 0s 17ms/step - loss: 0.4455 - accuracy: 0.9095 - val_loss: 0.9095 - va
l_accuracy: 0.8170
Epoch 8/20
16/16 [==============================] - 0s 16ms/step - loss: 0.3650 - accuracy: 0.9236 - val_loss: 0.9009 - va
l_accuracy: 0.8250
Epoch 9/20
16/16 [==============================] - 0s 17ms/step - loss: 0.3028 - accuracy: 0.9356 - val_loss: 0.9291 - va
l_accuracy: 0.8230
Epoch 10/20
16/16 [==============================] - 0s 16ms/step - loss: 0.2606 - accuracy: 0.9416 - val_loss: 0.8856 - va
l_accuracy: 0.8270
Epoch 11/20
16/16 [==============================] - 0s 15ms/step - loss: 0.2234 - accuracy: 0.9478 - val_loss: 0.9098 - va
l_accuracy: 0.8240
Epoch 12/20
16/16 [==============================] - 0s 16ms/step - loss: 0.1931 - accuracy: 0.9485 - val_loss: 0.9140 - va
l_accuracy: 0.8270
Epoch 13/20
16/16 [==============================] - 0s 16ms/step - loss: 0.1749 - accuracy: 0.9528 - val_loss: 0.9387 - va
l_accuracy: 0.8230
Epoch 14/20
16/16 [==============================] - 0s 16ms/step - loss: 0.1579 - accuracy: 0.9538 - val_loss: 0.9435 - va
l_accuracy: 0.8190
Epoch 15/20
16/16 [==============================] - 0s 16ms/step - loss: 0.1448 - accuracy: 0.9543 - val_loss: 0.9915 - va
l_accuracy: 0.8170
Epoch 16/20
16/16 [==============================] - 0s 17ms/step - loss: 0.1393 - accuracy: 0.9555 - val_loss: 1.0126 - va
l_accuracy: 0.8110
Epoch 17/20
16/16 [==============================] - 0s 22ms/step - loss: 0.1314 - accuracy: 0.9563 - val_loss: 1.0071 - va
l_accuracy: 0.8140
Epoch 18/20
16/16 [==============================] - 0s 23ms/step - loss: 0.1205 - accuracy: 0.9568 - val_loss: 1.0147 - va
l_accuracy: 0.8110
Epoch 19/20
16/16 [==============================] - 0s 21ms/step - loss: 0.1190 - accuracy: 0.9568 - val_loss: 1.0499 - va
l_accuracy: 0.8120
Epoch 20/20
16/16 [==============================] - 0s 16ms/step - loss: 0.1199 - accuracy: 0.9577 - val_loss: 1.0805 - va
l_accuracy: 0.8000
```

In [27]:
```python
#Data Visualization
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 [28]:
```python
plt.clf()   # clear figure

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()
```



In [29]:
```python
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=8,
          batch_size=512,
          validation_data=(x_val, y_val))
results = model.evaluate(x_test, one_hot_test_labels)
```

```
Epoch 1/8
16/16 [==============================] - 1s 24ms/step - loss: 2.6462 - accuracy: 0.5449 - val_loss: 1.7509 - va
l_accuracy: 0.6340
Epoch 2/8
16/16 [==============================] - 0s 16ms/step - loss: 1.4434 - accuracy: 0.6998 - val_loss: 1.3178 - va
l_accuracy: 0.7120
Epoch 3/8
16/16 [==============================] - 0s 16ms/step - loss: 1.0614 - accuracy: 0.7680 - val_loss: 1.1424 - va
l_accuracy: 0.7420
Epoch 4/8
16/16 [==============================] - 0s 16ms/step - loss: 0.8332 - accuracy: 0.8216 - val_loss: 1.0297 - va
l_accuracy: 0.7730
Epoch 5/8
16/16 [==============================] - 0s 17ms/step - loss: 0.6614 - accuracy: 0.8644 - val_loss: 0.9583 - va
l_accuracy: 0.7890
Epoch 6/8
16/16 [==============================] - 0s 16ms/step - loss: 0.5279 - accuracy: 0.8899 - val_loss: 0.9222 - va
l_accuracy: 0.8000
Epoch 7/8
16/16 [==============================] - 0s 16ms/step - loss: 0.4235 - accuracy: 0.9156 - val_loss: 0.8896 - va
l_accuracy: 0.8100
Epoch 8/8
16/16 [==============================] - 0s 15ms/step - loss: 0.3462 - accuracy: 0.9278 - val_loss: 0.8902 - va
l_accuracy: 0.8110
71/71 [==============================] - 0s 2ms/step - loss: 0.9941 - accuracy: 0.7814
```

In [30]:
```python
#Results of naive model
print(results)
```

```
[0.9941007494926453, 0.7813891172409058]
```

In [31]:
```python
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[31]:
```
0.182546749777382
```

Assignment 5.3 - Housing Prices

In [32]:
```python
#Import data
from keras.datasets import boston_housing
```

In [33]:
```python
#Splitting data
(train_data, train_targets), (test_data, test_targets) =  boston_housing.load_data()
```

In [34]:
```python
#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 [35]:
```python
#Building model
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 [36]:
```python
#Model Validation
k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []
for i in range(k):
    print('processing fold #', i)
    # Prepare the validation data: data from partition # k
    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]

    # Prepare the training data: data from all other partitions
    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)

    # Build the Keras model (already compiled)
    model = build_model()
    # Train the model (in silent mode, verbose=0)
    model.fit(partial_train_data, partial_train_targets,
              epochs=num_epochs, batch_size=1, verbose=0)
    # Evaluate the model on the validation data
    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 [37]:
```python
#Printing scores
print(all_scores)
print(np.mean(all_scores))
```

```
[2.0773348808288574, 2.6227023601531982, 2.5399224758148193, 2.4558420181274414]
2.423950433731079
```

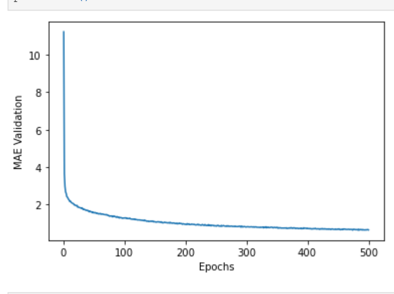
In [38]:
```python
from keras import backend as K
# Clear session
K.clear_session()
```

In [39]:
```python
num_epochs = 500
all_mae_histories = []
for i in range(k):
    print('processing fold #', i)
    # Prepare the validation data: data from partition # k
    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]

    # Prepare the training data: data from all other partitions
    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)

    # Build the Keras model (already compiled)
    model = build_model()
    # Train the model (in silent mode, verbose=0)
    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['mae']
    all_mae_histories.append(mae_history)
```

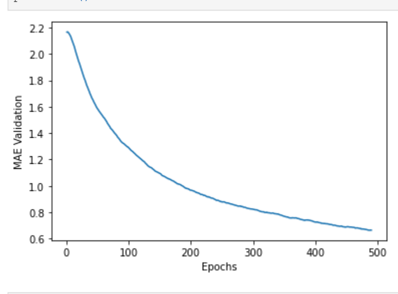
```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
```

In [40]:
```python
average_mae_history = [
    np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

In [41]:
```python
plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('MAE Validation')
plt.show()
```



In [43]:
```python
def smooth_curve(points, factor=0.9):
    smoothed_points = []
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point * (1 - factor))
        else:
            smoothed_points.append(point)
    return smoothed_points

smooth_mae_history = smooth_curve(average_mae_history[10:])

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



In [44]:
```python
# Get a fresh, compiled model
model = build_model()
# Train it on the entire dataset
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: 17.6108 - mae: 2.7527
```

In [45]:
```python
test_mae_score
```

Out[45]:
```
2.7526745796203613
```

In [ ]: