Assignment 5.1

```
from keras import models
from keras import layers
```

Loading the IMDB dataset

```
from keras.datasets import imdb
```

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
17465344/17464789 [==============================] - 0s 0us/step
17473536/17464789 [==============================] - 0s 0us/step
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested se
/usr/local/lib/python3.7/dist-packages/keras/datasets/imdb.py:155: VisibleDeprec
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/keras/datasets/imdb.py:156: VisibleDeprec
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

```
train_data[0]
```

```
 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,
```

```
train_labels[0]
```

```
1
```

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

```
9999
```

For kicks, here's how you can quickly decode one of these reviews back to English words:

```
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-dataset
1646592/1641221 [==============================] - 0s 0us/step
1654784/1641221 [==============================] - 0s 0us/step
```

Encoding the integer sequences into a binary matrix

```
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)
```

```
x_train[0]
```

```
array([0., 1., 1., ..., 0., 0., 0.])
```

You should also vectorize your labels

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

Now the data is ready to be fed into a neural network.

The model definition

```
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'))
```

Compiling the model

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

Configuring the optimizer

```
from keras import optimizers

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

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/opti
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

Using custom 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])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/opti
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

Setting aside a validation set

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

Training your model

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

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

```
Epoch 1/20
30/30 [==============================] - 16s 60ms/step - loss: 0.6064 - acc: 0.7
Epoch 2/20
30/30 [==============================] - 1s 38ms/step - loss: 0.3463 - acc: 0.90
Epoch 3/20
30/30 [==============================] - 1s 38ms/step - loss: 0.2475 - acc: 0.92
Epoch 4/20
```

```
30/30 [==============================] - 1s 38ms/step - loss: 0.1843 - acc: 0.94
Epoch 5/20
30/30 [==============================] - 1s 38ms/step - loss: 0.1530 - acc: 0.95
Epoch 6/20
30/30 [==============================] - 1s 37ms/step - loss: 0.1215 - acc: 0.96
Epoch 7/20
30/30 [==============================] - 1s 37ms/step - loss: 0.1006 - acc: 0.96
Epoch 8/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0835 - acc: 0.97
Epoch 9/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0663 - acc: 0.98
Epoch 10/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0569 - acc: 0.98
Epoch 11/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0443 - acc: 0.99
Epoch 12/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0378 - acc: 0.99
Epoch 13/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0310 - acc: 0.99
Epoch 14/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0230 - acc: 0.99
Epoch 15/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0171 - acc: 0.99
Epoch 16/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0143 - acc: 0.99
Epoch 17/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0109 - acc: 0.99
Epoch 18/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0085 - acc: 0.99
Epoch 19/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0055 - acc: 0.99
Epoch 20/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0057 - acc: 0.99
```

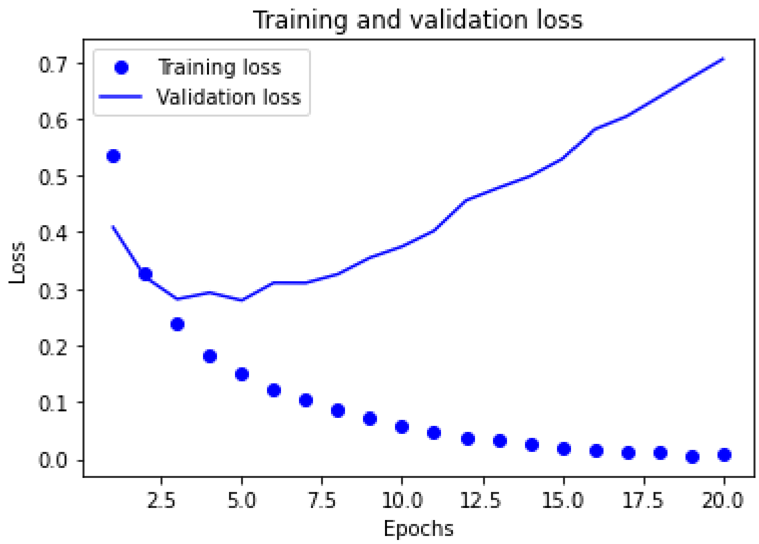
```
history_dict = history.history
history_dict.keys()
```

```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

Plotting the 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)

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()
```



Plotting the training and validation accuracy

```
plt.clf()
acc = 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('Accuracy')
plt.legend()

plt.show()
```

Training and validation accuracy

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 [==============================] - 2s 31ms/step - loss: 0.5601 - accuracy:
Epoch 2/4
49/49 [==============================] - 1s 30ms/step - loss: 0.2835 - accuracy:
Epoch 3/4
49/49 [==============================] - 1s 30ms/step - loss: 0.1989 - accuracy:
Epoch 4/4
49/49 [==============================] - 1s 30ms/step - loss: 0.1661 - accuracy:
782/782 [==============================] - 2s 2ms/step - loss: 0.3099 - accuracy
```

USING A TRAINED NETWORK TO GENERATE PREDICTIONS ON NEW DATA

```
model.predict(x_test)
```

```
array([[0.22002155],
       [0.9996809 ],
       [0.9457886 ],
       ...,
       [0.17937511],
       [0.13360816],
       [0.78052926]], dtype=float32)
```

Assignment 5.2

Loading the Reuters dataset

```
from keras.datasets import reuters

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets
2113536/2110848 [==============================] - 0s 0us/step
2121728/2110848 [==============================] - 0s 0us/step
/usr/local/lib/python3.7/dist-packages/keras/datasets/reuters.py:143: VisibleDep
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/keras/datasets/reuters.py:144: VisibleDep
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

```
len(train_data)
```

```
8982
```

```
len(test_data)
```

```
2246
```

```
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]
```

Decoding newswires 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]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
557056/550378 [==============================] - 0s 0us/step
565248/550378 [==============================] - 0s 0us/step
```

```
train_labels[10]
```

```
3
```

PREPARING THE DATA

```
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)
```

Vectorized training data and test data

```
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)
```

One-hot encoding / Categorical encoding

```
from keras.utils.np_utils import to_categorical

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

BUILDING YOUR NETWORK Model definition

```
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'))
```

Compiling the model

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

VALIDATING YOUR APPROACH Setting aside a validation set

```
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:]
```

Training the model

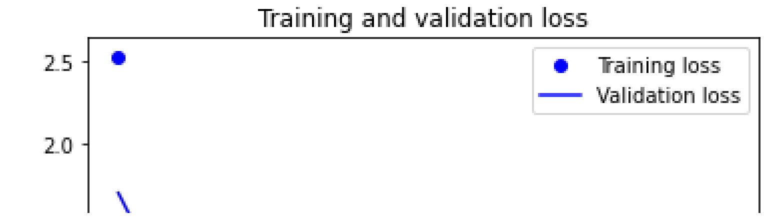
```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
16/16 [==============================] - 2s 54ms/step - loss: 3.0169 - accuracy:
Epoch 2/20
16/16 [==============================] - 1s 37ms/step - loss: 1.4741 - accuracy:
Epoch 3/20
16/16 [==============================] - 1s 37ms/step - loss: 1.0467 - accuracy:
Epoch 4/20
16/16 [==============================] - 1s 36ms/step - loss: 0.8252 - accuracy:
Epoch 5/20
16/16 [==============================] - 1s 37ms/step - loss: 0.6746 - accuracy:
Epoch 6/20
16/16 [==============================] - 1s 36ms/step - loss: 0.5336 - accuracy:
Epoch 7/20
16/16 [==============================] - 1s 37ms/step - loss: 0.4217 - accuracy:
Epoch 8/20
16/16 [==============================] - 1s 36ms/step - loss: 0.3478 - accuracy:
```

```
Epoch 9/20
16/16 [==============================] - 1s 36ms/step - loss: 0.2774 - accuracy:
Epoch 10/20
16/16 [==============================] - 1s 36ms/step - loss: 0.2313 - accuracy:
Epoch 11/20
16/16 [==============================] - 1s 36ms/step - loss: 0.2095 - accuracy:
Epoch 12/20
16/16 [==============================] - 1s 37ms/step - loss: 0.1675 - accuracy:
Epoch 13/20
16/16 [==============================] - 1s 37ms/step - loss: 0.1529 - accuracy:
Epoch 14/20
16/16 [==============================] - 1s 36ms/step - loss: 0.1386 - accuracy:
Epoch 15/20
16/16 [==============================] - 1s 38ms/step - loss: 0.1321 - accuracy:
Epoch 16/20
16/16 [==============================] - 1s 35ms/step - loss: 0.1265 - accuracy:
Epoch 17/20
16/16 [==============================] - 1s 38ms/step - loss: 0.1197 - accuracy:
Epoch 18/20
16/16 [==============================] - 1s 37ms/step - loss: 0.1090 - accuracy:
Epoch 19/20
16/16 [==============================] - 1s 37ms/step - loss: 0.1026 - accuracy:
Epoch 20/20
16/16 [==============================] - 1s 37ms/step - loss: 0.0977 - accuracy:
```

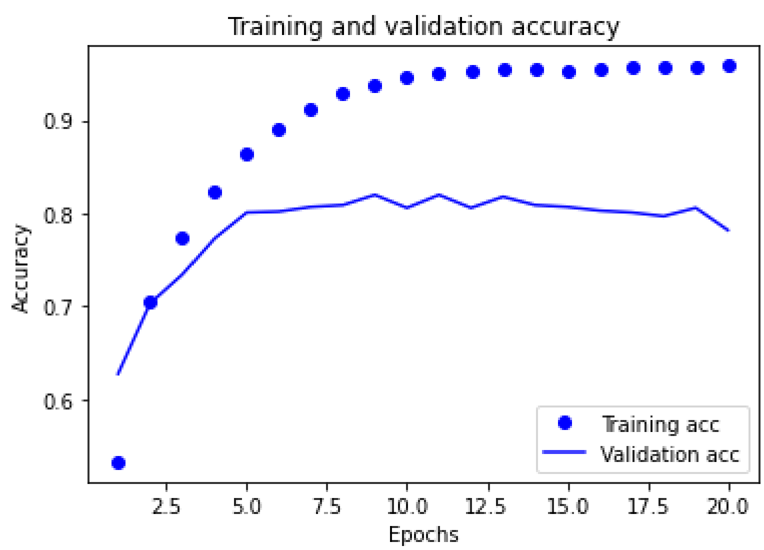
Plotting the training and validation loss

```
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()
```



Plotting the training and validation accuracy


```
plt.clf()
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# acc = history.history['acc']
# val_acc = history.history['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('Accuracy')
plt.legend()

plt.show()
```



Retraining a model from scratch

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

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
```

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

```
Epoch 1/9
16/16 [==============================] - 2s 59ms/step - loss: 3.2298 - accuracy:
Epoch 2/9
16/16 [==============================] - 1s 36ms/step - loss: 1.4850 - accuracy:
Epoch 3/9
16/16 [==============================] - 1s 36ms/step - loss: 1.0689 - accuracy:
Epoch 4/9
16/16 [==============================] - 1s 37ms/step - loss: 0.8392 - accuracy:
Epoch 5/9
16/16 [==============================] - 1s 37ms/step - loss: 0.6815 - accuracy:
Epoch 6/9
16/16 [==============================] - 1s 35ms/step - loss: 0.5318 - accuracy:
Epoch 7/9
16/16 [==============================] - 1s 38ms/step - loss: 0.4417 - accuracy:
Epoch 8/9
16/16 [==============================] - 1s 37ms/step - loss: 0.3565 - accuracy:
Epoch 9/9
16/16 [==============================] - 1s 36ms/step - loss: 0.2855 - accuracy:
71/71 [==============================] - 0s 2ms/step - loss: 0.9826 - accuracy: (
```

```
results
```

```
[0.9826157093048096, 0.7853962779045105]
```

GENERATING PREDICTIONS ON NEW DATA

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

```
predictions[0].shape
```

```
(46,)
```

```
np.sum(predictions[0])
```

```
1.0000001
```

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

```
3
```

Assignment 5.3

Loading the Boston housing dataset

```
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
57344/57026 [==============================] - 0s 0us/step
65536/57026 [==================================] - 0s 0us/step
```

```
train_data.shape
```

```
(404, 13)
```

```
test_data.shape
```

```
(102, 13)
```

```
train_targets
```

```
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])
```

PREPARING THE DATA Normalizing the data

```
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

```
from keras import models
from keras import layers

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

K-fold validation

```
import numpy as np

k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []
for i in range(k):
    print('processing fold #', i)
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
```

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

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

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

```
all_scores
```

```
[2.3133180141448975, 2.34952712059021, 3.068028450012207, 2.5455784797668457]
```

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

```
2.56911301612854
```

200 epochs. Saving the validation logs at each fold

```
num_epochs = 200
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']
```

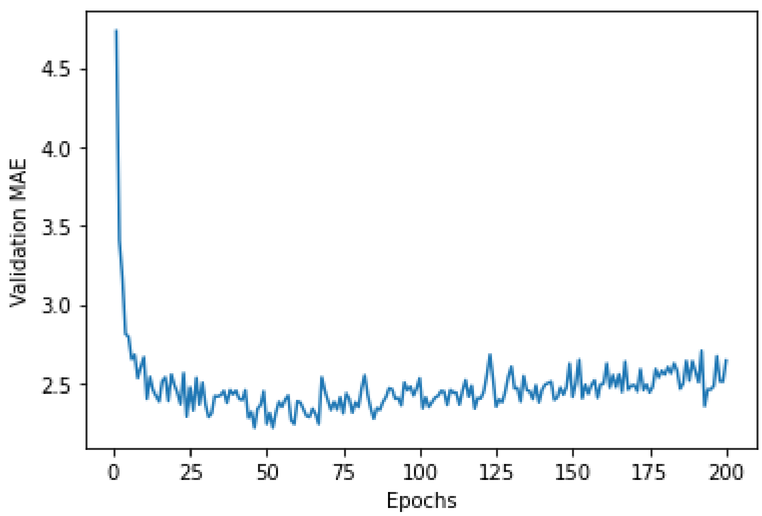
```
    mae_history = history.history['val_mae']
    all_mae_histories.append(mae_history)
```

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

Building the 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)]
```

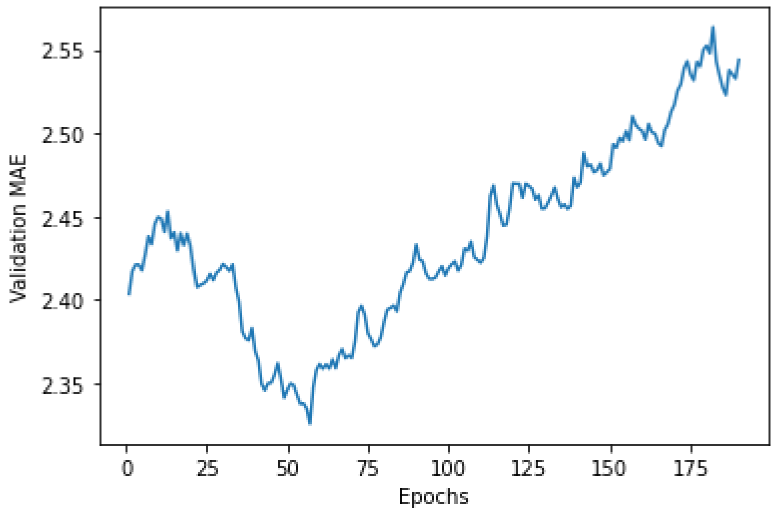
```
import matplotlib.pyplot as plt

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



Plotting validation scores, excluding the first 10 data points

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

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

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



Training the final 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 4ms/step - loss: 16.2873 - mae: 2.5957
```

```
test_mae_score
```

```
2.5956995487213135
```

✓ 0s completed at 3:05 PM