In [ ]:
```
#Classifying movie reviews: A binary classification example
#The IMDB dataset
#Loading the IMDB dataset

from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)

train_data[0]
train_labels[0]
max([max(sequence) for sequence in train_data])
```

Out[ ]:
```
9999
```

New Section

In [ ]:
```
#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]])
```

In [ ]:
```
#Preparing the data
#Encoding the integer sequences via multi-hot encoding

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)
x_train[0]
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

In [ ]:
```
#Building your model
#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 [ ]:
```
#Compiling the model

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

In [ ]:
```
#Validating your approach
#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:]
```

In [ ]:
```
#Training your model

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

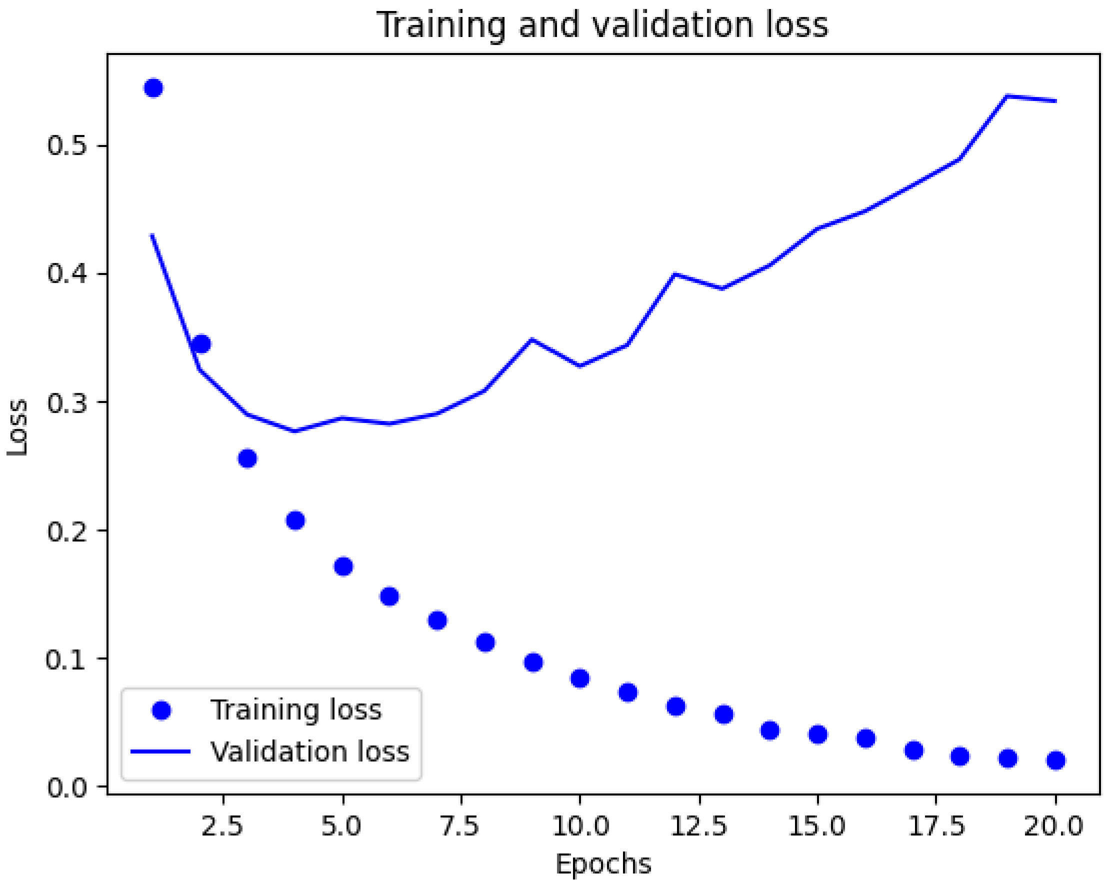
```
Epoch 1/20
30/30 [==============================] - 6s 189ms/step - loss: 0.5449 - accuracy:
0.7689 - val_loss: 0.4285 - val_accuracy: 0.8573
Epoch 2/20
30/30 [==============================] - 3s 101ms/step - loss: 0.3452 - accuracy:
0.8911 - val_loss: 0.3242 - val_accuracy: 0.8833
Epoch 3/20
30/30 [==============================] - 5s 160ms/step - loss: 0.2553 - accuracy:
0.9157 - val_loss: 0.2892 - val_accuracy: 0.8879
Epoch 4/20
30/30 [==============================] - 1s 39ms/step - loss: 0.2071 - accuracy:
0.9319 - val_loss: 0.2760 - val_accuracy: 0.8889
Epoch 5/20
30/30 [==============================] - 1s 49ms/step - loss: 0.1721 - accuracy:
0.9443 - val_loss: 0.2864 - val_accuracy: 0.8840
Epoch 6/20
30/30 [==============================] - 1s 48ms/step - loss: 0.1490 - accuracy:
0.9534 - val_loss: 0.2819 - val_accuracy: 0.8848
Epoch 7/20
30/30 [==============================] - 1s 40ms/step - loss: 0.1304 - accuracy:
0.9595 - val_loss: 0.2898 - val_accuracy: 0.8857
Epoch 8/20
30/30 [==============================] - 1s 41ms/step - loss: 0.1123 - accuracy:
0.9662 - val_loss: 0.3076 - val_accuracy: 0.8794
Epoch 9/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0972 - accuracy:
0.9722 - val_loss: 0.3477 - val_accuracy: 0.8752
Epoch 10/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0842 - accuracy:
0.9775 - val_loss: 0.3269 - val_accuracy: 0.8811
Epoch 11/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0727 - accuracy:
0.9803 - val_loss: 0.3434 - val_accuracy: 0.8806
Epoch 12/20
30/30 [==============================] - 2s 61ms/step - loss: 0.0626 - accuracy:
0.9835 - val_loss: 0.3986 - val_accuracy: 0.8726
Epoch 13/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0555 - accuracy:
0.9871 - val_loss: 0.3874 - val_accuracy: 0.8759
Epoch 14/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0444 - accuracy:
0.9915 - val_loss: 0.4056 - val_accuracy: 0.8766
Epoch 15/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0410 - accuracy:
0.9913 - val_loss: 0.4340 - val_accuracy: 0.8720
Epoch 16/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0370 - accuracy:
0.9923 - val_loss: 0.4477 - val_accuracy: 0.8740
Epoch 17/20
30/30 [==============================] - 1s 37ms/step - loss: 0.0280 - accuracy:
0.9951 - val_loss: 0.4677 - val_accuracy: 0.8721
Epoch 18/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0234 - accuracy:
0.9974 - val_loss: 0.4883 - val_accuracy: 0.8723
Epoch 19/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0219 - accuracy:
0.9968 - val_loss: 0.5377 - val_accuracy: 0.8633
Epoch 20/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0205 - accuracy:
0.9969 - val_loss: 0.5339 - val_accuracy: 0.8719
```

Out[ ]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

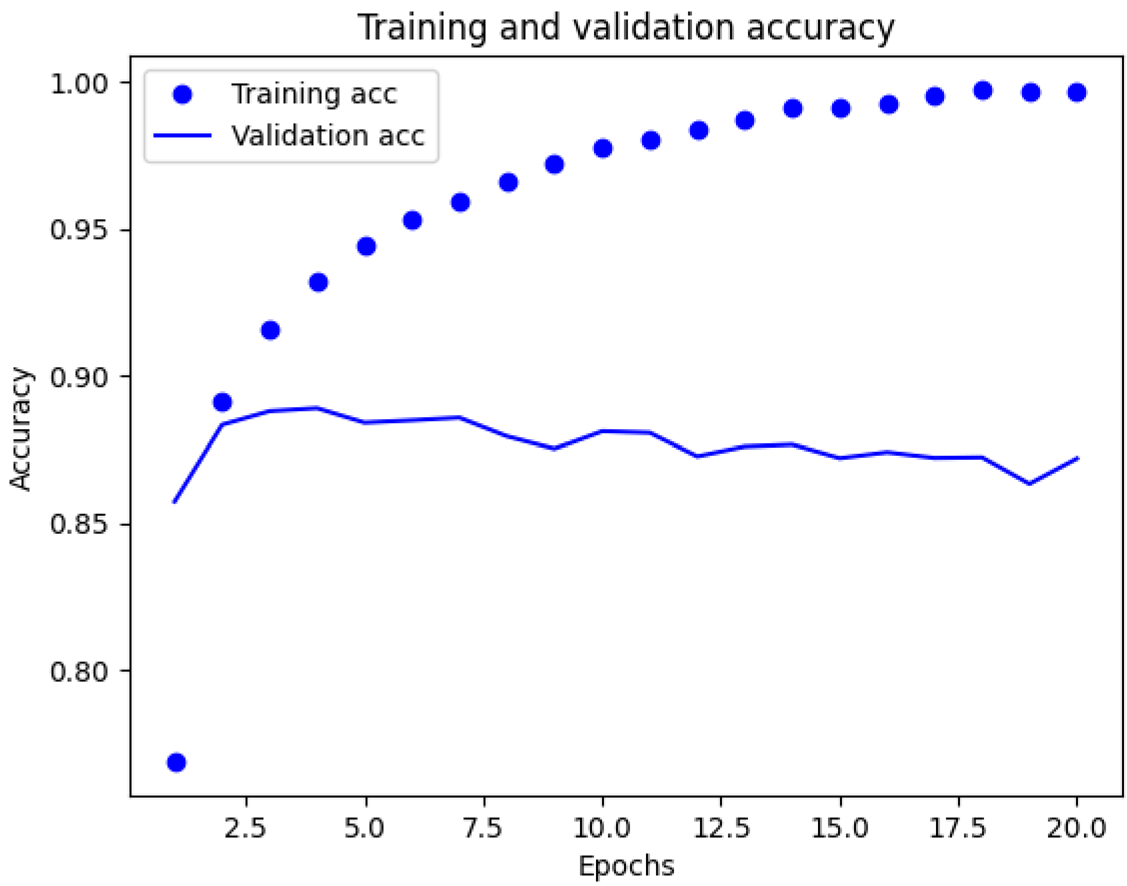
In [ ]:
```
#Plotting the training and validation loss

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()
```



In [ ]:
```
#Plotting the training and validation accuracy

plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["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("Accuracy")
plt.legend()
plt.show()
```



In [ ]:
```
#Retraining a model from scratch

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=512)
results = model.evaluate(x_test, y_test)
results
```

```
Epoch 1/4
49/49 [==============================] - 3s 28ms/step - loss: 0.4970 - accuracy:
0.8015
Epoch 2/4
49/49 [==============================] - 2s 31ms/step - loss: 0.2880 - accuracy:
0.8977
Epoch 3/4
49/49 [==============================] - 2s 39ms/step - loss: 0.2231 - accuracy:
0.9191
Epoch 4/4
49/49 [==============================] - 1s 29ms/step - loss: 0.1879 - accuracy:
0.9328
782/782 [==============================] - 2s 3ms/step - loss: 0.2890 - accuracy:
0.8852
```

Out[ ]:
```
[0.2890424132347107, 0.8851600289344788]
```

In [ ]:
```
#Using a trained model to generate predictions on new data

model.predict(x_test)
```

```
782/782 [==============================] - 3s 2ms/step
```

Out[ ]:
```
array([[0.22125062],
       [0.9995276 ],
       [0.8962404 ],
       ...,
       [0.09870885],
       [0.11608151],
       [0.667172  ]], dtype=float32)
```

In [ ]:
```
#Further experiments
#Wrapping up
#Classifying newswires: A multiclass classification example
#The Reuters dataset
#Loading the Reuters dataset

from tensorflow.keras.datasets import reuters
(train_data, train_labels), (test_data, test_labels) = reuters.load_data(
    num_words=10000)
len(train_data)
len(test_data)
train_data[10]
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/
reuters.npz
2110848/2110848 [==============================] - 0s 0us/step
```

Out[ ]:
```
[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 [ ]:
```
#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]])
train_labels[10]
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/
reuters_word_index.json
550378/550378 [==============================] - 0s 0us/step
```

Out[ ]:

```
3
```

In [ ]:

```python
#Preparing the data
#Encoding the input data

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

In [ ]:

```python
#Encoding the labels

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)
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(train_labels)
y_test = to_categorical(test_labels)
```

In [ ]:

```python
#Building your model
#Model definition

model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(46, activation="softmax")
])

#Compiling the model

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

In [ ]:

```python
#Validating your approach
#Setting aside a 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 [ ]:

```python
#Training the model

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

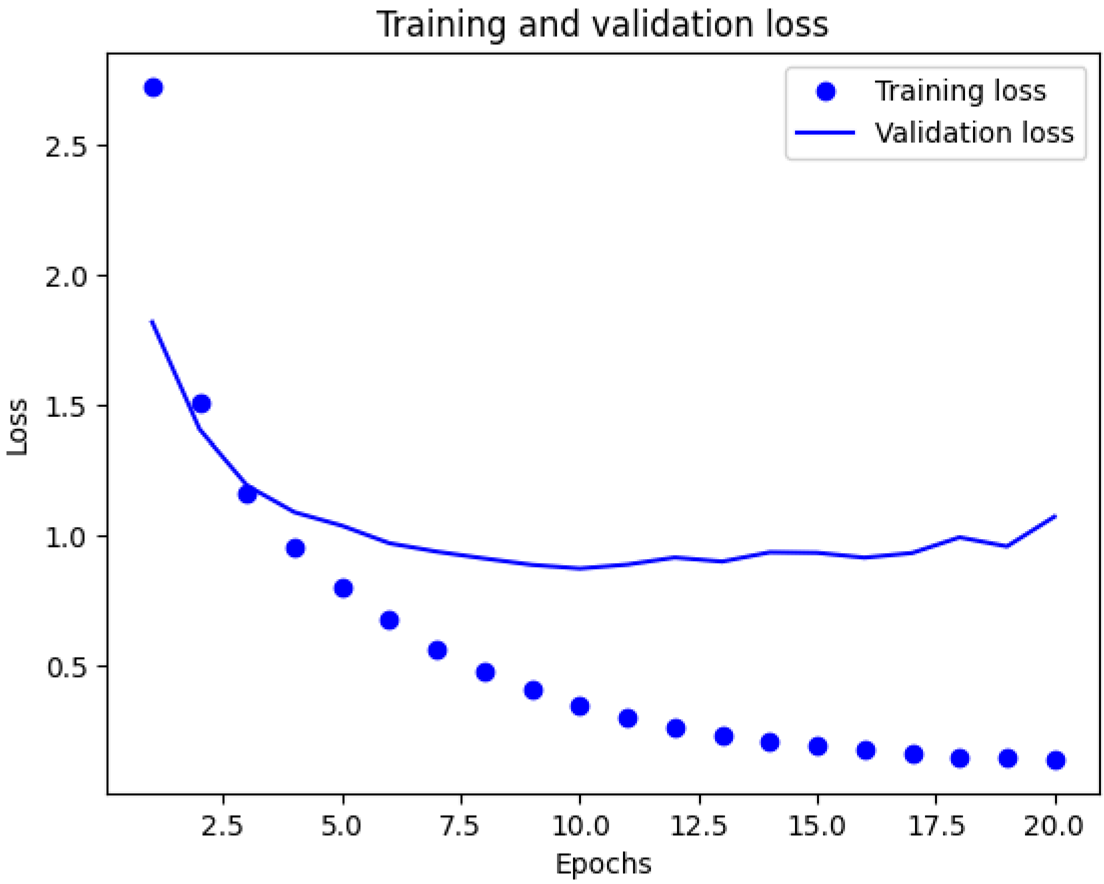
```
Epoch 1/20
16/16 [==============================] - 2s 71ms/step - loss: 2.7257 - accuracy:
0.5140 - val_loss: 1.8193 - val_accuracy: 0.6220
Epoch 2/20
16/16 [==============================] - 1s 54ms/step - loss: 1.5123 - accuracy:
0.6855 - val_loss: 1.4070 - val_accuracy: 0.6740
Epoch 3/20
16/16 [==============================] - 1s 53ms/step - loss: 1.1652 - accuracy:
0.7451 - val_loss: 1.1919 - val_accuracy: 0.7350
Epoch 4/20
16/16 [==============================] - 1s 52ms/step - loss: 0.9570 - accuracy:
0.7907 - val_loss: 1.0894 - val_accuracy: 0.7610
Epoch 5/20
16/16 [==============================] - 1s 57ms/step - loss: 0.8010 - accuracy:
0.8249 - val_loss: 1.0376 - val_accuracy: 0.7650
Epoch 6/20
16/16 [==============================] - 1s 83ms/step - loss: 0.6772 - accuracy:
0.8510 - val_loss: 0.9701 - val_accuracy: 0.7940
Epoch 7/20
16/16 [==============================] - 1s 86ms/step - loss: 0.5626 - accuracy:
0.8746 - val_loss: 0.9376 - val_accuracy: 0.7950
Epoch 8/20
16/16 [==============================] - 1s 64ms/step - loss: 0.4765 - accuracy:
0.8991 - val_loss: 0.9115 - val_accuracy: 0.8110
Epoch 9/20
16/16 [==============================] - 1s 53ms/step - loss: 0.4073 - accuracy:
0.9136 - val_loss: 0.8871 - val_accuracy: 0.8120
Epoch 10/20
16/16 [==============================] - 1s 50ms/step - loss: 0.3449 - accuracy:
0.9270 - val_loss: 0.8728 - val_accuracy: 0.8190
Epoch 11/20
16/16 [==============================] - 1s 52ms/step - loss: 0.2991 - accuracy:
0.9356 - val_loss: 0.8881 - val_accuracy: 0.8160
Epoch 12/20
16/16 [==============================] - 1s 50ms/step - loss: 0.2632 - accuracy:
0.9416 - val_loss: 0.9152 - val_accuracy: 0.8020
Epoch 13/20
16/16 [==============================] - 1s 50ms/step - loss: 0.2321 - accuracy:
0.9463 - val_loss: 0.8996 - val_accuracy: 0.8130
Epoch 14/20
16/16 [==============================] - 1s 51ms/step - loss: 0.2109 - accuracy:
0.9470 - val_loss: 0.9353 - val_accuracy: 0.8090
Epoch 15/20
16/16 [==============================] - 1s 51ms/step - loss: 0.1900 - accuracy:
0.9513 - val_loss: 0.9334 - val_accuracy: 0.8090
Epoch 16/20
16/16 [==============================] - 1s 49ms/step - loss: 0.1765 - accuracy:
0.9529 - val_loss: 0.9147 - val_accuracy: 0.8110
Epoch 17/20
16/16 [==============================] - 1s 51ms/step - loss: 0.1622 - accuracy:
0.9540 - val_loss: 0.9326 - val_accuracy: 0.8220
Epoch 18/20
16/16 [==============================] - 1s 51ms/step - loss: 0.1491 - accuracy:
0.9577 - val_loss: 0.9928 - val_accuracy: 0.7980
Epoch 19/20
16/16 [==============================] - 1s 52ms/step - loss: 0.1458 - accuracy:
0.9563 - val_loss: 0.9580 - val_accuracy: 0.8100
Epoch 20/20
16/16 [==============================] - 1s 61ms/step - loss: 0.1391 - accuracy:
0.9583 - val_loss: 1.0725 - val_accuracy: 0.7860
```

In [ ]:

```python
#Plotting the training and validation loss

loss = history.history["loss"]
```

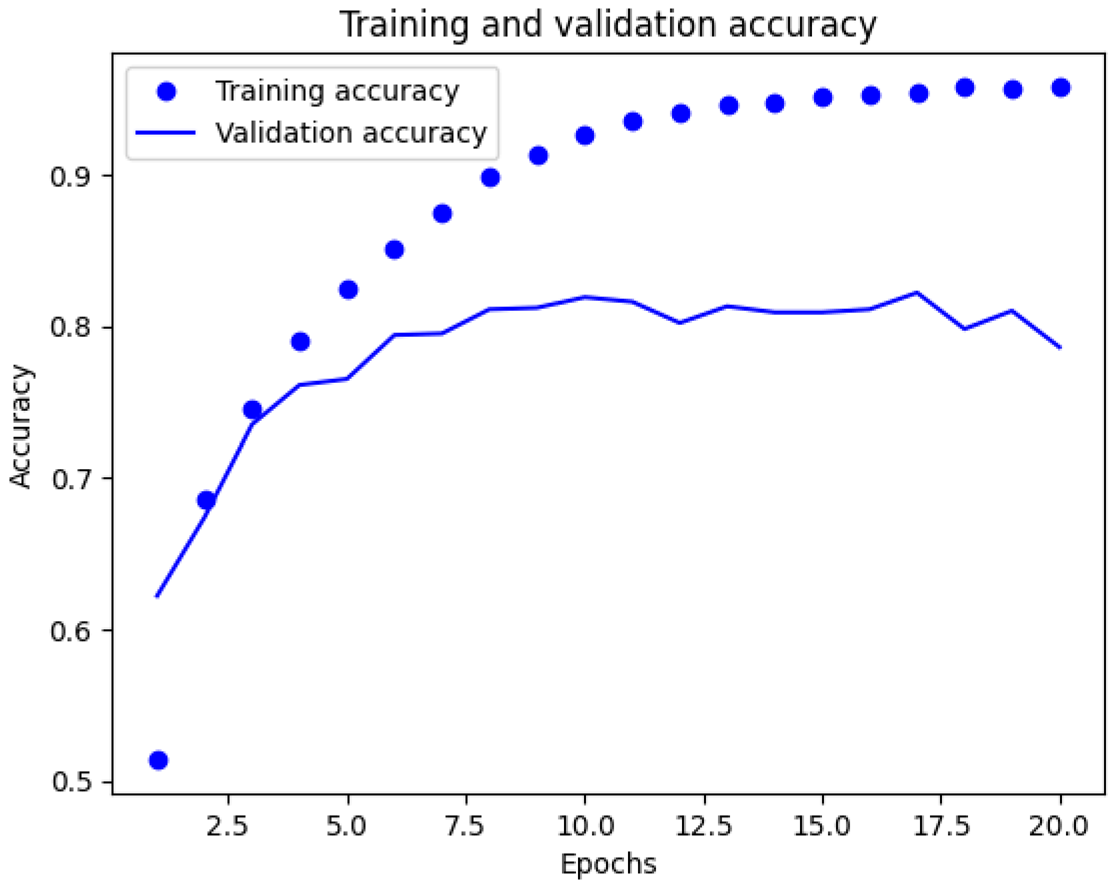
```
val_loss = history.history["val_loss"]
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



In [ ]:

```
#Plotting the training and 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.legend()
plt.show()
```



In [ ]:
```
#Retraining a model from scratch

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)
results

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()
```

```
Epoch 1/9
18/18 [==============================] - 2s 56ms/step - loss: 2.6564 - accuracy:
0.5148
Epoch 2/9
18/18 [==============================] - 1s 49ms/step - loss: 1.4806 - accuracy:
0.6819
Epoch 3/9
18/18 [==============================] - 1s 51ms/step - loss: 1.1260 - accuracy:
0.7503
Epoch 4/9
18/18 [==============================] - 1s 60ms/step - loss: 0.9199 - accuracy:
0.8012
Epoch 5/9
18/18 [==============================] - 1s 48ms/step - loss: 0.7598 - accuracy:
0.8342
Epoch 6/9
18/18 [==============================] - 1s 50ms/step - loss: 0.6312 - accuracy:
0.8642
Epoch 7/9
18/18 [==============================] - 1s 46ms/step - loss: 0.5254 - accuracy:
0.8868
Epoch 8/9
18/18 [==============================] - 1s 62ms/step - loss: 0.4435 - accuracy:
0.9063
Epoch 9/9
18/18 [==============================] - 1s 75ms/step - loss: 0.3725 - accuracy:
0.9194
71/71 [==============================] - 1s 5ms/step - loss: 0.9168 - accuracy: 0.
7890
```

Out[ ]:

```
0.19323241317898487
```

In [ ]:

```
#Generating predictions on new data
predictions = model.predict(x_test)
predictions[0].shape
np.sum(predictions[0])
np.argmax(predictions[0])
```

```
71/71 [==============================] - 0s 4ms/step
```

Out[ ]:

```
3
```

In [ ]:

```
#A different way to handle the labels and the loss
y_train = np.array(train_labels)
y_test = np.array(test_labels)
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

In [ ]:

```
#The importance of having sufficiently large intermediate layers
#A model with an information bottleneck

model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(4, 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=20,
```

```
          batch_size=128,
          validation_data=(x_val, y_val))
```

```
Epoch 1/20
63/63 [==============================] - 2s 23ms/step - loss: 2.6702 - accuracy:
0.2752 - val_loss: 2.0430 - val_accuracy: 0.5140
Epoch 2/20
63/63 [==============================] - 1s 19ms/step - loss: 1.7775 - accuracy:
0.5678 - val_loss: 1.6408 - val_accuracy: 0.5790
Epoch 3/20
63/63 [==============================] - 1s 19ms/step - loss: 1.4859 - accuracy:
0.5986 - val_loss: 1.5342 - val_accuracy: 0.5890
Epoch 4/20
63/63 [==============================] - 1s 18ms/step - loss: 1.3407 - accuracy:
0.6250 - val_loss: 1.4276 - val_accuracy: 0.6150
Epoch 5/20
63/63 [==============================] - 1s 17ms/step - loss: 1.2326 - accuracy:
0.6555 - val_loss: 1.3840 - val_accuracy: 0.6370
Epoch 6/20
63/63 [==============================] - 1s 23ms/step - loss: 1.1496 - accuracy:
0.6788 - val_loss: 1.3693 - val_accuracy: 0.6650
Epoch 7/20
63/63 [==============================] - 2s 27ms/step - loss: 1.0819 - accuracy:
0.6956 - val_loss: 1.3408 - val_accuracy: 0.6520
Epoch 8/20
63/63 [==============================] - 1s 19ms/step - loss: 1.0185 - accuracy:
0.7209 - val_loss: 1.3346 - val_accuracy: 0.6840
Epoch 9/20
63/63 [==============================] - 1s 19ms/step - loss: 0.9660 - accuracy:
0.7453 - val_loss: 1.3311 - val_accuracy: 0.6870
Epoch 10/20
63/63 [==============================] - 1s 17ms/step - loss: 0.9163 - accuracy:
0.7615 - val_loss: 1.3240 - val_accuracy: 0.6880
Epoch 11/20
63/63 [==============================] - 1s 17ms/step - loss: 0.8759 - accuracy:
0.7667 - val_loss: 1.3629 - val_accuracy: 0.6850
Epoch 12/20
63/63 [==============================] - 1s 18ms/step - loss: 0.8401 - accuracy:
0.7762 - val_loss: 1.3709 - val_accuracy: 0.6860
Epoch 13/20
63/63 [==============================] - 1s 17ms/step - loss: 0.8076 - accuracy:
0.7849 - val_loss: 1.3758 - val_accuracy: 0.6860
Epoch 14/20
63/63 [==============================] - 1s 17ms/step - loss: 0.7794 - accuracy:
0.7883 - val_loss: 1.4144 - val_accuracy: 0.6850
Epoch 15/20
63/63 [==============================] - 1s 18ms/step - loss: 0.7538 - accuracy:
0.7929 - val_loss: 1.4538 - val_accuracy: 0.6800
Epoch 16/20
63/63 [==============================] - 1s 18ms/step - loss: 0.7292 - accuracy:
0.7977 - val_loss: 1.5199 - val_accuracy: 0.6740
Epoch 17/20
63/63 [==============================] - 2s 26ms/step - loss: 0.7077 - accuracy:
0.7993 - val_loss: 1.4769 - val_accuracy: 0.6880
Epoch 18/20
63/63 [==============================] - 2s 27ms/step - loss: 0.6892 - accuracy:
0.8013 - val_loss: 1.5256 - val_accuracy: 0.6750
Epoch 19/20
63/63 [==============================] - 1s 17ms/step - loss: 0.6699 - accuracy:
0.8028 - val_loss: 1.5625 - val_accuracy: 0.6880
Epoch 20/20
63/63 [==============================] - 1s 17ms/step - loss: 0.6560 - accuracy:
0.8091 - val_loss: 1.6277 - val_accuracy: 0.6910
```

Out[ ]: <keras.src.callbacks.History at 0x7ec1ae173c70>

In [ ]:
```
#Further experiments
#Wrapping up
#Predicting house prices: A regression example
#The Boston Housing Price dataset
#Loading the Boston housing dataset

from tensorflow.keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()
train_data.shape
test_data.shape
train_targets
```

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

Out[ ]:
```
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 [ ]:
```
#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
```

In [ ]:
```
#Building your model
#Model definition

def build_model():
    model = keras.Sequential([
        layers.Dense(64, activation="relu"),
        layers.Dense(64, activation="relu"),
        layers.Dense(1)
    ])
    model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
    return model
```

In [ ]:
```
#Validating your approach using K-fold validation
#K-fold validation
!pip install numpy
import numpy as np


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)

import numpy as np
from tensorflow import keras
from tensorflow.keras import layers

# Rest of your code...

def build_model():
    model = keras.Sequential([
        layers.Dense(64, activation="relu"),
        layers.Dense(64, activation="relu"),
        layers.Dense(1)  # Adjust the number of units according to your problem
    ])

    model.compile(optimizer='adam', loss='mse', metrics=['mae'])

    return model
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(1.25.2)
Processing fold #0
Processing fold #1
Processing fold #2
Processing fold #3
```

In [ ]:
```
#Saving the validation logs at each fold

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 [ ]:
```
#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)]
```

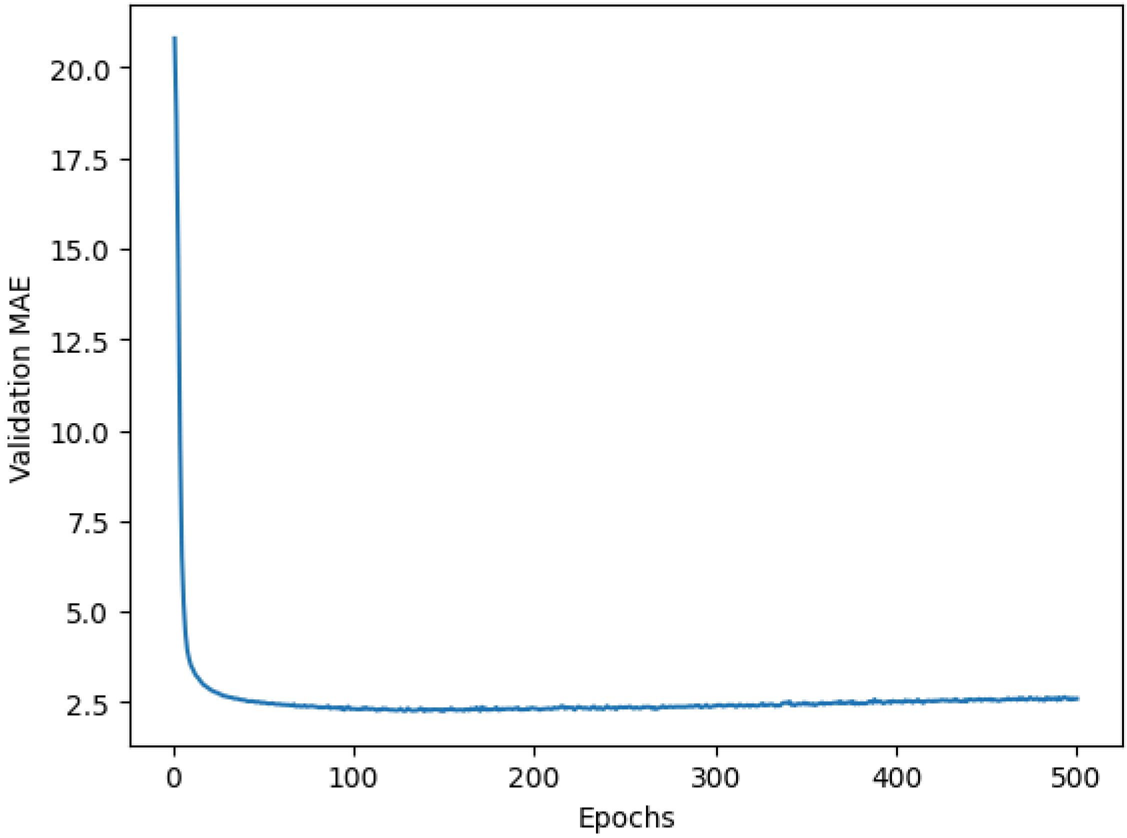
In [ ]:
```
#Plotting validation scores

import matplotlib.pyplot as plt

# Assuming you have defined `average_mae_history` before this point

%matplotlib inline

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



In [ ]:
```
#Training the final model

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)
test_mae_score
```

```
4/4 [==============================] - 0s 3ms/step - loss: 15.5534 - mae: 2.5204
```

Out[ ]:
```
2.520387649536133
```

In [ ]:
```
#Generating predictions on new data
predictions = model.predict(test_data)
predictions[0]
```

```
4/4 [==============================] - 0s 4ms/step
```

Out[ ]:
```
array([7.7675962], dtype=float32)
```

In [10]:
```
%%shell
jupyter nbconvert --to html /%%shell
jupyter nbconvert --to html //content//sample_data//neural networks.ipynb
```