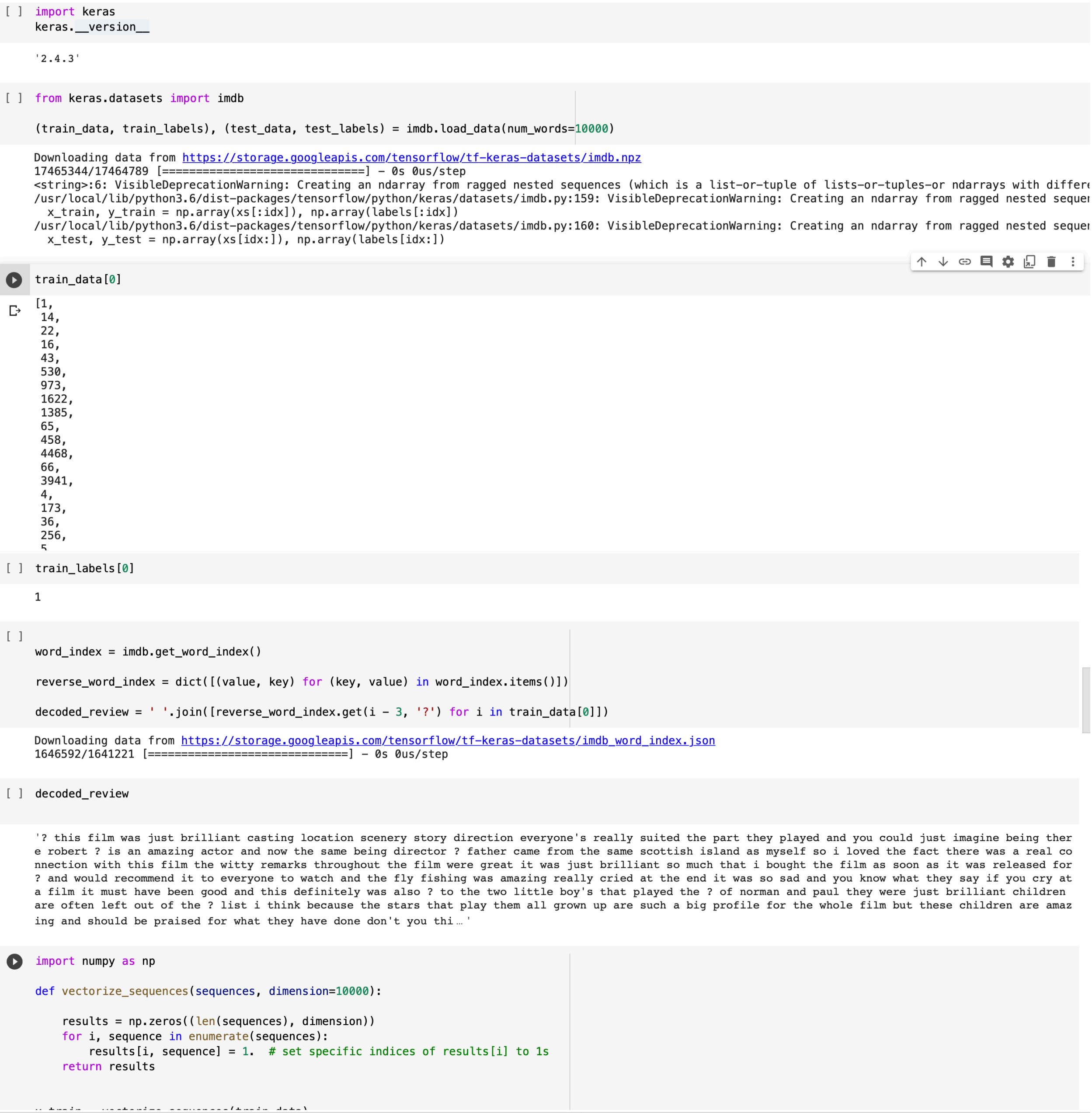


```
import keras
keras.__version__
```

```
'2.4.3'
```

```
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-datasets/imdb.npz
17465344/17464789 [==============================] - 0s 0us/step
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with differe
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/datasets/imdb.py:159: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequer
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/datasets/imdb.py:160: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequer
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

```
train_data[0]
```

```
[1,
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
 1385,
 65,
 458,
 4468,
 66,
 3941,
 4,
 173,
 36,
 256,
 5
```

```
train_labels[0]
```

```
1
```

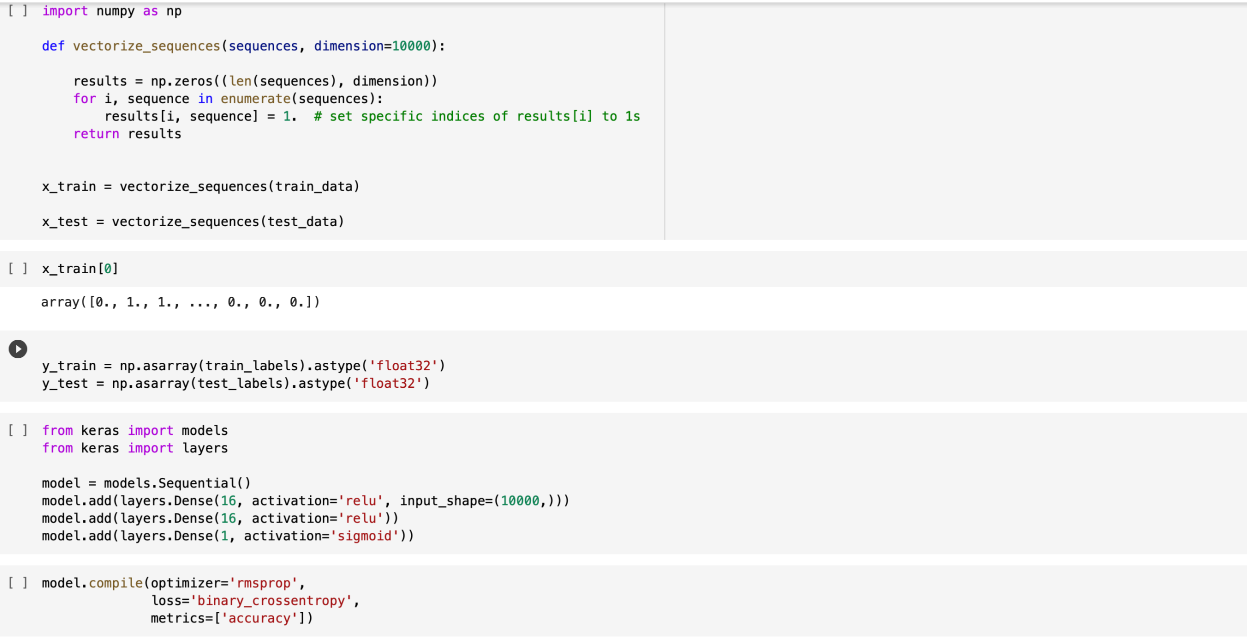
```

word_index = imdb.get_word_index()

reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[0]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1646592/1641221 [==============================] - 0s 0us/step
```

```
decoded_review
```

```
'? this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being ther
e 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 co
nnection with this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for
? and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at
a film it must have been good and this definitely was also ? to the two little boy's that played the ? of norman and paul they were just brilliant children
are often left out of the ? list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amaz
ing and should be praised for what they have done don't you thi…'
```

```
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.  # set specific indices of results[i] to 1s
    return results


x_train = vectorize_sequences(train_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.  # set specific indices of results[i] to 1s
    return results


x_train = vectorize_sequences(train_data)

x_test = vectorize_sequences(test_data)
```

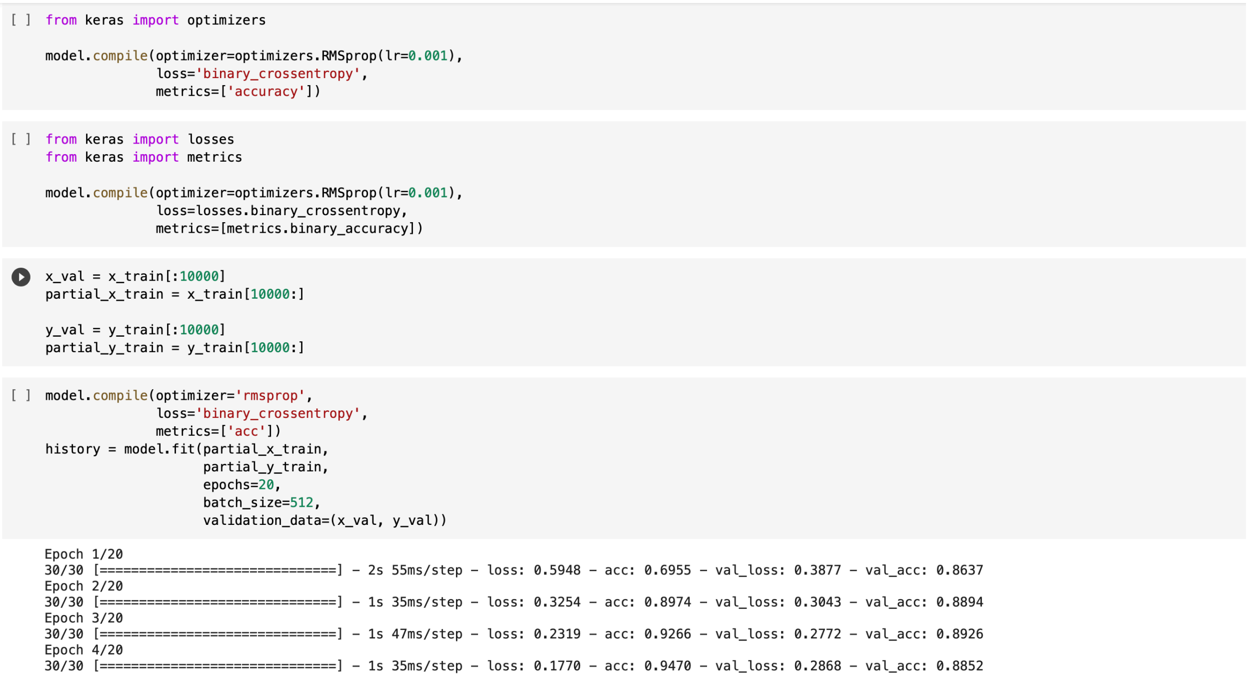
```
x_train[0]
```

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

```

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

```
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'))
```

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



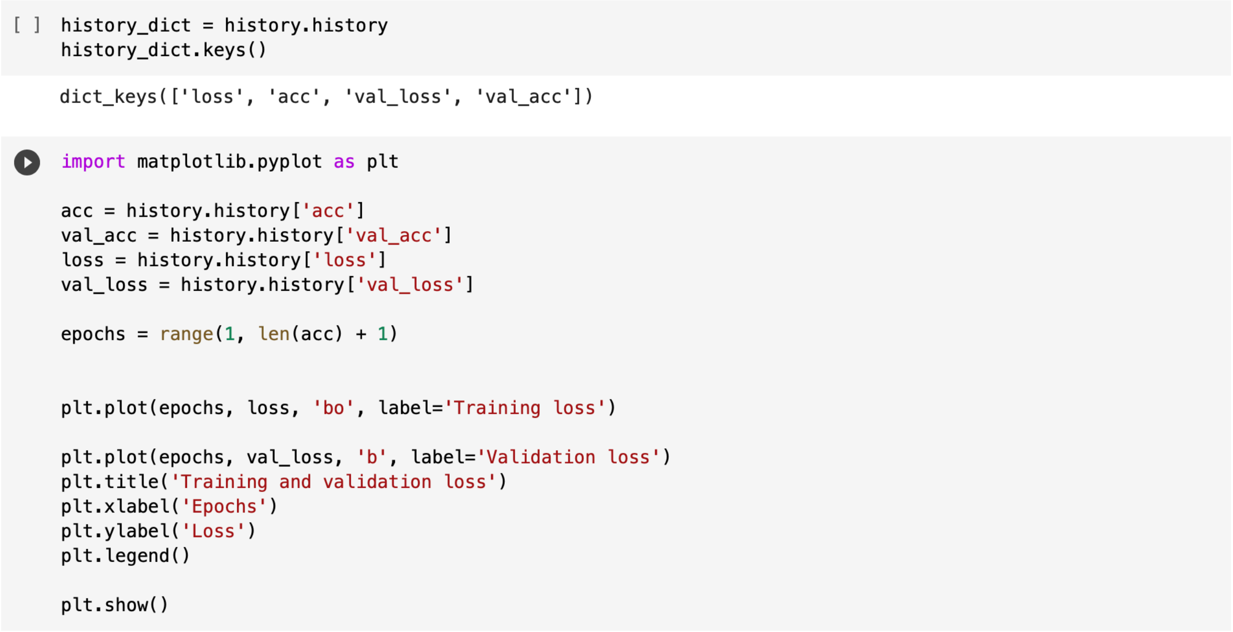
```python
from keras import optimizers

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

```python
from keras import losses
from keras import metrics

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

```python
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```python
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))
```

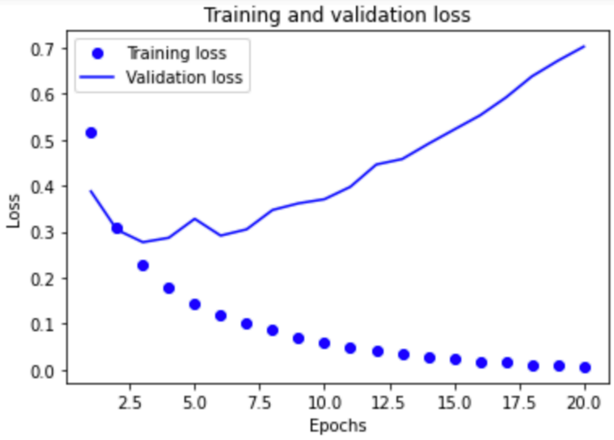
```
Epoch 1/20
30/30 [==============================] - 2s 55ms/step - loss: 0.5948 - acc: 0.6955 - val_loss: 0.3877 - val_acc: 0.8637
Epoch 2/20
30/30 [==============================] - 1s 35ms/step - loss: 0.3254 - acc: 0.8974 - val_loss: 0.3043 - val_acc: 0.8894
Epoch 3/20
30/30 [==============================] - 1s 47ms/step - loss: 0.2319 - acc: 0.9266 - val_loss: 0.2772 - val_acc: 0.8926
Epoch 4/20
30/30 [==============================] - 1s 35ms/step - loss: 0.1770 - acc: 0.9470 - val_loss: 0.2868 - val_acc: 0.8852
```



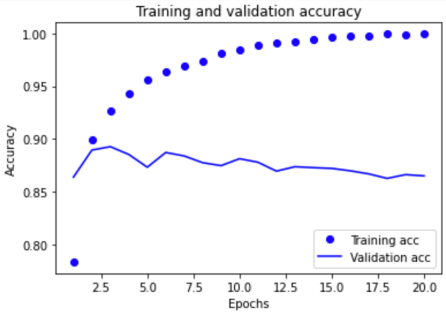
```python
history_dict = history.history
history_dict.keys()
```

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

```python
import matplotlib.pyplot as plt

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

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

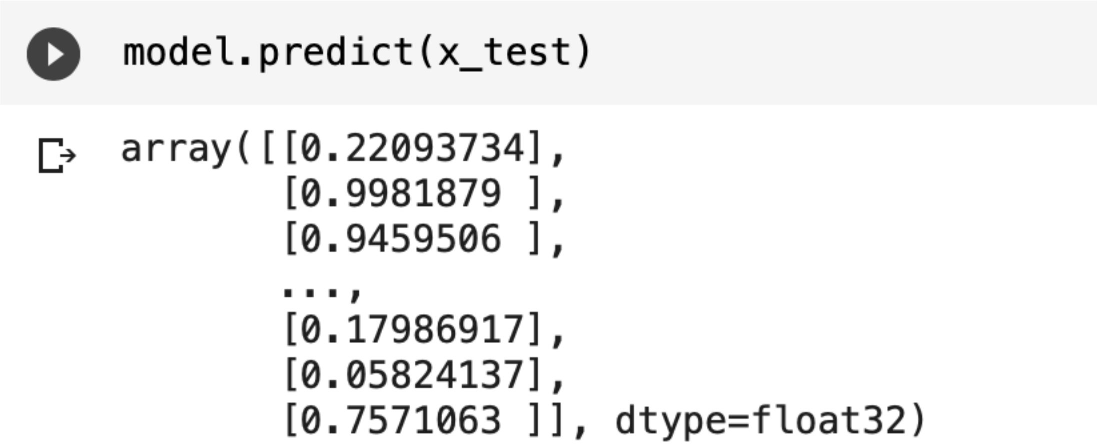


```
plt.clf()
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



```
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)
results
```

```
Epoch 1/4
49/49 [==============================] - 2s 27ms/step - loss: 0.5391 - accuracy: 0.7429
Epoch 2/4
49/49 [==============================] - 1s 27ms/step - loss: 0.2648 - accuracy: 0.9136
Epoch 3/4
49/49 [==============================] - 1s 30ms/step - loss: 0.1966 - accuracy: 0.9353
Epoch 4/4
49/49 [==============================] - 1s 27ms/step - loss: 0.1615 - accuracy: 0.9444
782/782 [==============================] - 2s 2ms/step - loss: 0.3035 - accuracy: 0.8797
[0.3035437762737274, 0.8797199726104736]
```



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

```
array([[0.22093734],
       [0.9981879 ],
       [0.9459506 ],
       ...,
       [0.17986917],
       [0.05824137],
       [0.7571063 ]], dtype=float32)
```