In [1]:

```
import tensorflow as tf
```

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.0.0'
```

In [35]:

```
from tensorflow.keras import layers, optimizers, metrics, datasets, Sequential, models
```

In [4]:

```
import os
```

In [5]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [9]:

```
(x_train, y_train), (x_test, y_test) = datasets.imdb.load_data(num_words=10000)
```

In [10]:

```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[10]:

```
((25000,), (25000,), (25000,), (25000,))
```

In [14]:

```
x_train[0][:10]
```

Out[14]:

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65]
```

In [16]:

```
y_train[0]
```

Out[16]:

```
1
```

In [17]:

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

Out[17]:

```
9999
```

数字和单词映射表，索引减3，因为0，1，2为padding、start of sequence、unknown保留的索引

In [19]:

```
word_index = datasets.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 x_train[0]])
```

In [20]:

```
decoded_review
```

Out[20]:

```
"? this film was just brilliant casting location scenery story direction eve
ryone's really 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 witty remarks throughout the film were gr
eat it was just brilliant so much that i bought the film as soon as it was r
eleased for ? and would recommend it to everyone to watch and the fly fishin
g was amazing really cried at the end it was so sad and you know what they s
ay 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 jus
t brilliant children are often left out of the ? list i think because the st
ars that play them all grown up are such a big profile for the whole film bu
t these children are amazing and should be praised for what they have done d
on't you think the whole story was so lovely because it was true and was som
eone's life after all that was shared with us all"
```

In [21]:

```
import numpy as np
```

向量化

In [22]:

```
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 [26]:

```
x_train = vectorize_sequences(x_train)
x_test = vectorize_sequences(x_test)
```

In [29]:

```
x_train.shape
```

Out[29]:

```
(25000, 10000)
```

In [30]:

```
x_train[0].shape
```

Out[30]:

```
(10000,)
```

In [31]:

```
x_train[0]
```

Out[31]:

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

In [32]:

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

In [33]:

```
y_train[0]
```

Out[33]:

```
1.0
```

In [36]:

```
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'))
```

In [37]:

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

In [42]:

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

In [38]:

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

In [40]:

```
# model.compile(optimizer=optimizers.RMSprop(lr=0.001), loss=losses.binary_crossentropy, me
```

留出验证集

In [41]:

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

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

In [43]:

```
history = model.fit(x_train, y_train, epochs=20, batch_size=512, validation_data=(x_val, y_
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 2s 122us/sample - loss: 0.
6140 - accuracy: 0.5672 - val_loss: 0.5530 - val_accuracy: 0.7330
Epoch 2/20
15000/15000 [==============================] - 1s 54us/sample - loss: 0.5
005 - accuracy: 0.8133 - val_loss: 0.4925 - val_accuracy: 0.8268
Epoch 3/20
15000/15000 [==============================] - 1s 54us/sample - loss: 0.4
313 - accuracy: 0.8904 - val_loss: 0.4632 - val_accuracy: 0.8236
Epoch 4/20
15000/15000 [==============================] - 1s 53us/sample - loss: 0.3
568 - accuracy: 0.9241 - val_loss: 0.3833 - val_accuracy: 0.8882
Epoch 5/20
15000/15000 [==============================] - 1s 54us/sample - loss: 0.2
721 - accuracy: 0.9447 - val_loss: 0.3422 - val_accuracy: 0.8810
Epoch 6/20
15000/15000 [==============================] - 1s 53us/sample - loss: 0.2
024 - accuracy: 0.9565 - val_loss: 0.3038 - val_accuracy: 0.8859
Epoch 7/20
15000/15000 [==============================] - 1s 53us/sample - loss: 0.1
558 - accuracy: 0.9657 - val_loss: 0.3056 - val_accuracy: 0.8831
Epoch 8/20
15000/15000 [==============================] - 1s 53us/sample - loss: 0.1
217 - accuracy: 0.9748 - val_loss: 0.3037 - val_accuracy: 0.8823
Epoch 9/20
15000/15000 [==============================] - 1s 53us/sample - loss: 0.0
974 - accuracy: 0.9793 - val_loss: 0.3241 - val_accuracy: 0.8804
Epoch 10/20
15000/15000 [==============================] - 1s 53us/sample - loss: 0.0
817 - accuracy: 0.9819 - val_loss: 0.3411 - val_accuracy: 0.8779
Epoch 11/20
15000/15000 [==============================] - 1s 53us/sample - loss: 0.0
644 - accuracy: 0.9879 - val_loss: 0.3544 - val_accuracy: 0.8785
Epoch 12/20
15000/15000 [==============================] - 1s 54us/sample - loss: 0.0
538 - accuracy: 0.9895 - val_loss: 0.4190 - val_accuracy: 0.8711
Epoch 13/20
15000/15000 [==============================] - 1s 54us/sample - loss: 0.0
452 - accuracy: 0.9921 - val_loss: 0.3974 - val_accuracy: 0.8774
Epoch 14/20
15000/15000 [==============================] - 1s 54us/sample - loss: 0.0
363 - accuracy: 0.9940 - val_loss: 0.4267 - val_accuracy: 0.8756
Epoch 15/20
15000/15000 [==============================] - 1s 53us/sample - loss: 0.0
302 - accuracy: 0.9954 - val_loss: 0.4549 - val_accuracy: 0.8735
Epoch 16/20
15000/15000 [==============================] - 1s 54us/sample - loss: 0.0
240 - accuracy: 0.9963 - val_loss: 0.4864 - val_accuracy: 0.8731
Epoch 17/20
15000/15000 [==============================] - 1s 54us/sample - loss: 0.0
188 - accuracy: 0.9974 - val_loss: 0.5344 - val_accuracy: 0.8687
Epoch 18/20
15000/15000 [==============================] - 1s 54us/sample - loss: 0.0
146 - accuracy: 0.9981 - val_loss: 0.5372 - val_accuracy: 0.8706
Epoch 19/20
```

```
15000/15000 [==============================] - 1s 55us/sample - loss: 0.0
122 - accuracy: 0.9983 - val_loss: 0.6691 - val_accuracy: 0.8610
Epoch 20/20
15000/15000 [==============================] - 1s 55us/sample - loss: 0.0
100 - accuracy: 0.9985 - val_loss: 0.6051 - val_accuracy: 0.8699
```

In [44]:

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

Out[44]:

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

绘图

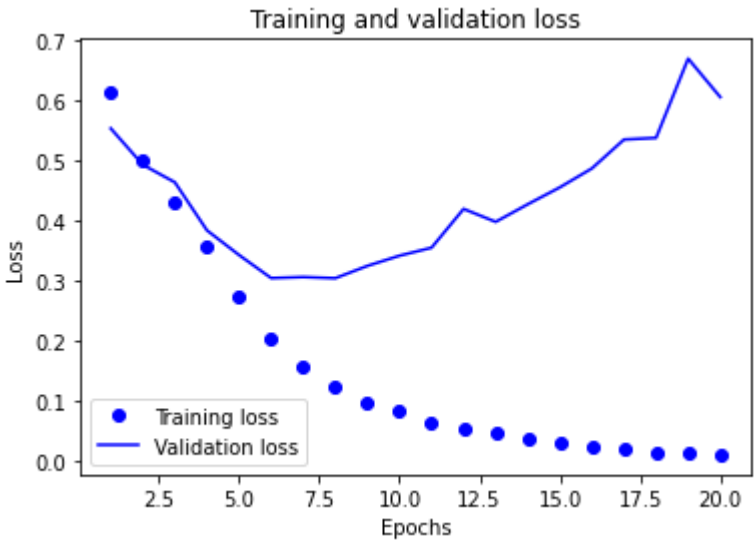
In [47]:

```
loss = history_dict['loss']
val_loss = history_dict['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()
```

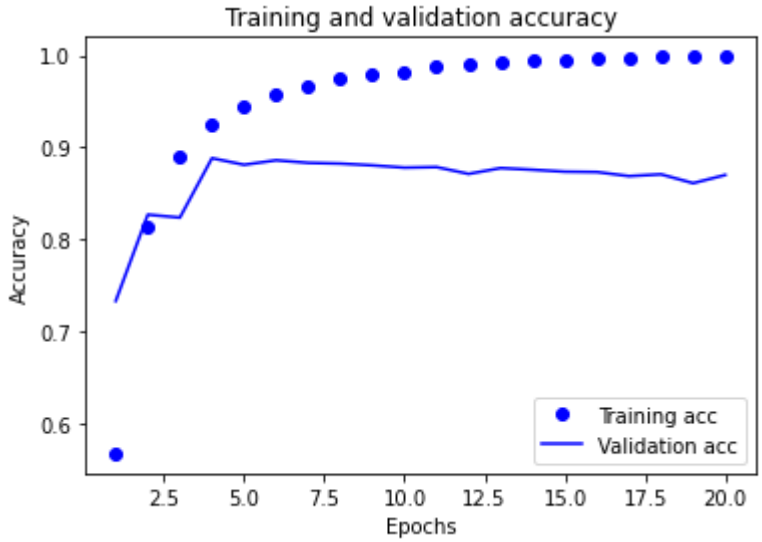


In [48]:

```
# plt.clf()     # 清除图像
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 [49]:

```
loss, acc = model.evaluate(x_test, y_test)
```

```
25000/1 [================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
=========================================================================
```

In [50]:

```
loss, acc
```

Out[50]:

```
(0.6686725060486793, 0.85684)
```

In [51]:

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

Out[51]:

```
array([[0.0075345 ],
       [0.9999391 ],
       [0.43583608],
       ...,
       [0.00217295],
       [0.00470984],
       [0.6878468 ]], dtype=float32)
```

early stopping，只训练4个周期

In [ ]:

```
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)
loss, acc = model.evaluate(x_test, y_test)
loss, acc
```