In [1]:

```
import tensorflow as tf
```

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.0.0'
```

In [3]:

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

In [4]:

```
import os
```

In [5]:

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

In [36]:

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

In [7]:

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

Out[7]:

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

In [8]:

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

Out[8]:

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

In [9]:

```
y_train[0]
```

Out[9]:

```
1
```

In [10]:

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

Out[10]:

```
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 [11]:

```
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 [12]:

```
decoded_review
```

Out[12]:

```
"? 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 [13]:

```
import numpy as np
```

向量化

In [14]:

```
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 [15]:

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

In [16]:

```
x_train.shape
```

Out[16]:

```
(25000, 10000)
```

In [17]:

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

Out[17]:

```
(10000,)
```

In [18]:

```
x_train[0]
```

Out[18]:

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

In [19]:

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

In [20]:

```
y_train[0]
```

Out[20]:

```
1.0
```

留出验证集

In [21]:

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

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

In [22]:

```
from tensorflow.keras import regularizers
```

原始网络

In [23]:

```
model1 = models.Sequential()
model1.add(layers.Dense(16 ,activation='relu', input_shape=(10000, )))
model1.add(layers.Dense(16, activation='relu'))
model1.add(layers.Dense(1, activation='sigmoid'))
```

In [24]:

```
model1.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

In [25]:

```
history1 = model1.fit(x_train, y_train, epochs=20, batch_size=512, validation_data=(x_val,
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 2s 102us/sample - loss: 0.
5080 - accuracy: 0.7924 - val_loss: 0.3787 - val_accuracy: 0.8703
Epoch 2/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.3
037 - accuracy: 0.9024 - val_loss: 0.3046 - val_accuracy: 0.8870
Epoch 3/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.2
203 - accuracy: 0.9292 - val_loss: 0.2793 - val_accuracy: 0.8916
Epoch 4/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.1
721 - accuracy: 0.9443 - val_loss: 0.2945 - val_accuracy: 0.8817
Epoch 5/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.1
406 - accuracy: 0.9553 - val_loss: 0.2883 - val_accuracy: 0.8861
Epoch 6/20
15000/15000 [==============================] - 1s 49us/sample - loss: 0.1
149 - accuracy: 0.9646 - val_loss: 0.2961 - val_accuracy: 0.8851
Epoch 7/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
939 - accuracy: 0.9728 - val_loss: 0.3221 - val_accuracy: 0.8793
Epoch 8/20
15000/15000 [==============================] - 1s 46us/sample - loss: 0.0
750 - accuracy: 0.9807 - val_loss: 0.3719 - val_accuracy: 0.8676
Epoch 9/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
644 - accuracy: 0.9837 - val_loss: 0.3701 - val_accuracy: 0.8734
Epoch 10/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
503 - accuracy: 0.9875 - val_loss: 0.4188 - val_accuracy: 0.8714
Epoch 11/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
412 - accuracy: 0.9909 - val_loss: 0.4197 - val_accuracy: 0.8722
Epoch 12/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
322 - accuracy: 0.9937 - val_loss: 0.4446 - val_accuracy: 0.8775
Epoch 13/20
15000/15000 [==============================] - 1s 46us/sample - loss: 0.0
284 - accuracy: 0.9937 - val_loss: 0.4713 - val_accuracy: 0.8745
Epoch 14/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
216 - accuracy: 0.9956 - val_loss: 0.5040 - val_accuracy: 0.8741
Epoch 15/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
117 - accuracy: 0.9991 - val_loss: 0.5345 - val_accuracy: 0.8737
Epoch 16/20
15000/15000 [==============================] - 1s 46us/sample - loss: 0.0
130 - accuracy: 0.9983 - val_loss: 0.5962 - val_accuracy: 0.8696
Epoch 17/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
094 - accuracy: 0.9990 - val_loss: 0.6128 - val_accuracy: 0.8629
Epoch 18/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
052 - accuracy: 0.9998 - val_loss: 0.6329 - val_accuracy: 0.8700
Epoch 19/20
```

```
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
079 - accuracy: 0.9982 - val_loss: 0.6760 - val_accuracy: 0.8679
Epoch 20/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
026 - accuracy: 0.9999 - val_loss: 0.7031 - val_accuracy: 0.8665
```

In [26]:

```
history_dict1 = history1.history
history_dict1.keys()
```

Out[26]:

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

绘图

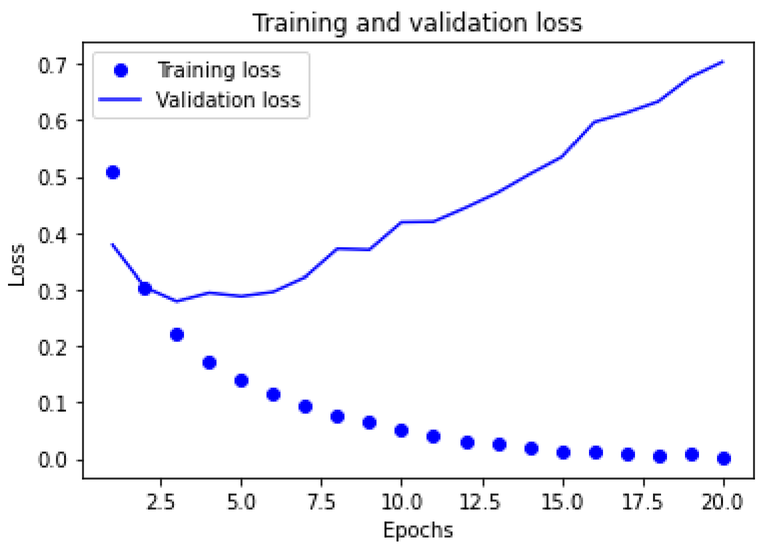
In [27]:

```
loss1 = history_dict1['loss']
val_loss1 = history_dict1['val_loss']

epochs1 = range(1, len(loss1) + 1)

plt.plot(epochs1, loss1, 'bo', label='Training loss')
plt.plot(epochs1, val_loss1, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

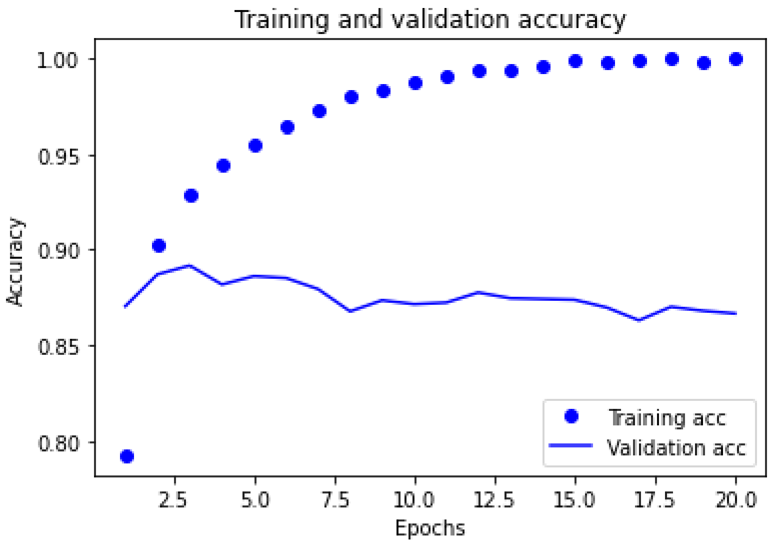


In [28]:

```
# plt.clf()     # 清除图像
acc1 = history_dict1['accuracy']
val_acc1 = history_dict1['val_accuracy']

plt.plot(epochs1, acc1, 'bo', label='Training acc')
plt.plot(epochs1, val_acc1, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [29]:

```
test_loss1, test_acc1 = model1.evaluate(x_test, y_test, verbose=0)
```

In [30]:

```
test_loss1, test_acc1
```

Out[30]:

```
(0.769314906027317, 0.85216)
```

In [31]:

```
model1.predict(x_test)
```

Out[31]:

```
array([[0.00692457],
       [1.        ],
       [0.99712336],
       ...,
       [0.00187099],
       [0.01051682],
       [0.45844334]], dtype=float32)
```

In [32]:

```
model2 = models.Sequential()
model2.add(layers.Dense(16, activation='relu' ,input_shape=(10000,)))
model2.add(layers.Dropout(0.5))
model2.add(layers.Dense(16, activation='relu'))
model2.add(layers.Dropout(0.5))
model2.add(layers.Dense(1, activation='sigmoid'))

model2.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

history2 =model2.fit(x_train, y_train, epochs=20, batch_size=512, validation_data=(x_val, y
test_loss2, test_acc2 = model2.evaluate(x_test, y_test, verbose=0)
test_loss2, test_acc2
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 1s 81us/sample - loss: 0.6
236 - accuracy: 0.6415 - val_loss: 0.5168 - val_accuracy: 0.8398
Epoch 2/20
15000/15000 [==============================] - 1s 49us/sample - loss: 0.5
006 - accuracy: 0.7711 - val_loss: 0.3936 - val_accuracy: 0.8745
Epoch 3/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.4
157 - accuracy: 0.8261 - val_loss: 0.3285 - val_accuracy: 0.8849
Epoch 4/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.3
570 - accuracy: 0.8643 - val_loss: 0.2954 - val_accuracy: 0.8882
Epoch 5/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.3
058 - accuracy: 0.8866 - val_loss: 0.2797 - val_accuracy: 0.8909
Epoch 6/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.2
672 - accuracy: 0.9088 - val_loss: 0.2773 - val_accuracy: 0.8904
Epoch 7/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.2
372 - accuracy: 0.9189 - val_loss: 0.2768 - val_accuracy: 0.8890
Epoch 8/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.2
137 - accuracy: 0.9306 - val_loss: 0.2878 - val_accuracy: 0.8885
Epoch 9/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.1
908 - accuracy: 0.9401 - val_loss: 0.2980 - val_accuracy: 0.8875
Epoch 10/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.1
681 - accuracy: 0.9474 - val_loss: 0.3277 - val_accuracy: 0.8874
Epoch 11/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.1
517 - accuracy: 0.9527 - val_loss: 0.3205 - val_accuracy: 0.8862
Epoch 12/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.1
390 - accuracy: 0.9563 - val_loss: 0.3634 - val_accuracy: 0.8883
Epoch 13/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.1
242 - accuracy: 0.9601 - val_loss: 0.3850 - val_accuracy: 0.8859
Epoch 14/20
15000/15000 [==============================] - 1s 49us/sample - loss: 0.1
114 - accuracy: 0.9635 - val_loss: 0.4183 - val_accuracy: 0.8860
Epoch 15/20
15000/15000 [==============================] - 1s 49us/sample - loss: 0.1
```

```
080 - accuracy: 0.9667 - val_loss: 0.4314 - val_accuracy: 0.8863
Epoch 16/20
15000/15000 [==============================] - 1s 49us/sample - loss: 0.0
935 - accuracy: 0.9719 - val_loss: 0.4654 - val_accuracy: 0.8856
Epoch 17/20
15000/15000 [==============================] - 1s 49us/sample - loss: 0.0
893 - accuracy: 0.9709 - val_loss: 0.5043 - val_accuracy: 0.8846
Epoch 18/20
15000/15000 [==============================] - 1s 49us/sample - loss: 0.0
895 - accuracy: 0.9717 - val_loss: 0.4989 - val_accuracy: 0.8834
Epoch 19/20
15000/15000 [==============================] - 1s 49us/sample - loss: 0.0
813 - accuracy: 0.9746 - val_loss: 0.5470 - val_accuracy: 0.8848
Epoch 20/20
15000/15000 [==============================] - 1s 49us/sample - loss: 0.0
811 - accuracy: 0.9765 - val_loss: 0.5725 - val_accuracy: 0.8849
```

Out[32]:

```
(0.6085672066307067, 0.874)
```

In [33]:

```python
history_dict2 = history2.history
```

添加L2正则化,比较原始网络和更大网络的验证损失和训练损失

In [34]:

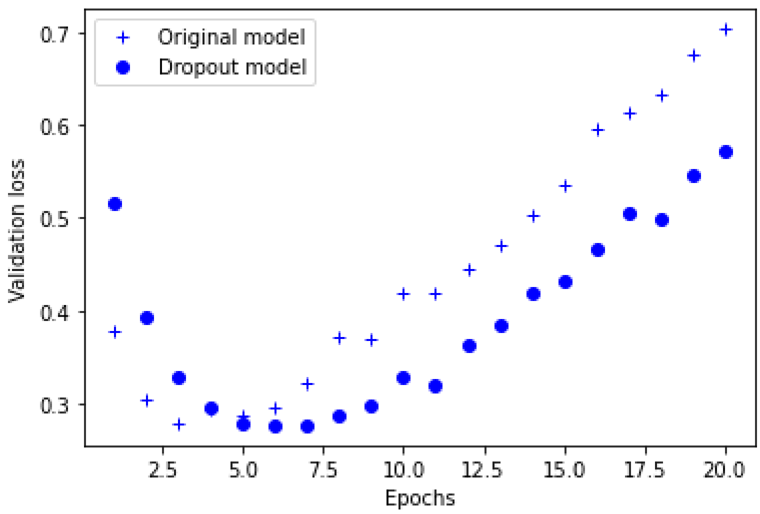
```python
loss2 = history_dict2['loss']
val_loss2 = history_dict2['val_loss']

epochs2 = range(1, len(loss2) + 1)

plt.plot(epochs1, val_loss1, 'b+', label='Original model')
plt.plot(epochs2, val_loss2, 'bo', label='Dropout model')

plt.xlabel('Epochs')
plt.ylabel('Validation loss')
plt.legend()

plt.show()
```



In [35]:

```
loss2 = history_dict2['loss']
val_loss2 = history_dict2['val_loss']

epochs3 = range(1, len(loss2) + 1)

plt.plot(epochs1, loss1, 'b+', label='Original model')
plt.plot(epochs2, loss2, 'bo', label='Dropout model')

plt.xlabel('Epochs')
plt.ylabel('Training Loss')
plt.legend()

plt.show()
```