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 101us/sample - loss: 0.
5187 - accuracy: 0.7887 - val_loss: 0.3843 - val_accuracy: 0.8702
Epoch 2/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.3
097 - accuracy: 0.9007 - val_loss: 0.3081 - val_accuracy: 0.8851
Epoch 3/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.2
278 - accuracy: 0.9261 - val_loss: 0.2987 - val_accuracy: 0.8814
Epoch 4/20
15000/15000 [==============================] - 1s 49us/sample - loss: 0.1
783 - accuracy: 0.9434 - val_loss: 0.2742 - val_accuracy: 0.8907
Epoch 5/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.1
455 - accuracy: 0.9533 - val_loss: 0.2875 - val_accuracy: 0.8846
Epoch 6/20
15000/15000 [==============================] - 1s 48us/sample - loss: 0.1
193 - accuracy: 0.9636 - val_loss: 0.3084 - val_accuracy: 0.8831
Epoch 7/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
970 - accuracy: 0.9704 - val_loss: 0.3391 - val_accuracy: 0.8729
Epoch 8/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
814 - accuracy: 0.9755 - val_loss: 0.3267 - val_accuracy: 0.8819
Epoch 9/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
646 - accuracy: 0.9828 - val_loss: 0.3486 - val_accuracy: 0.8825
Epoch 10/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
522 - accuracy: 0.9881 - val_loss: 0.3748 - val_accuracy: 0.8805
Epoch 11/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
413 - accuracy: 0.9910 - val_loss: 0.4065 - val_accuracy: 0.8778
Epoch 12/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
310 - accuracy: 0.9940 - val_loss: 0.4368 - val_accuracy: 0.8778
Epoch 13/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
248 - accuracy: 0.9951 - val_loss: 0.4701 - val_accuracy: 0.8743
Epoch 14/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
173 - accuracy: 0.9977 - val_loss: 0.5202 - val_accuracy: 0.8684
Epoch 15/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
130 - accuracy: 0.9985 - val_loss: 0.6520 - val_accuracy: 0.8519
Epoch 16/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
096 - accuracy: 0.9992 - val_loss: 0.5867 - val_accuracy: 0.8698
Epoch 17/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
093 - accuracy: 0.9987 - val_loss: 0.6251 - val_accuracy: 0.8688
Epoch 18/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
049 - accuracy: 0.9997 - val_loss: 0.6466 - val_accuracy: 0.8667
Epoch 19/20
```

```
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
051 - accuracy: 0.9993 - val_loss: 0.6904 - val_accuracy: 0.8673
Epoch 20/20
15000/15000 [==============================] - 1s 47us/sample - loss: 0.0
022 - accuracy: 0.9999 - val_loss: 0.7179 - val_accuracy: 0.8679
```

In [26]:

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

Out[26]:

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

绘图

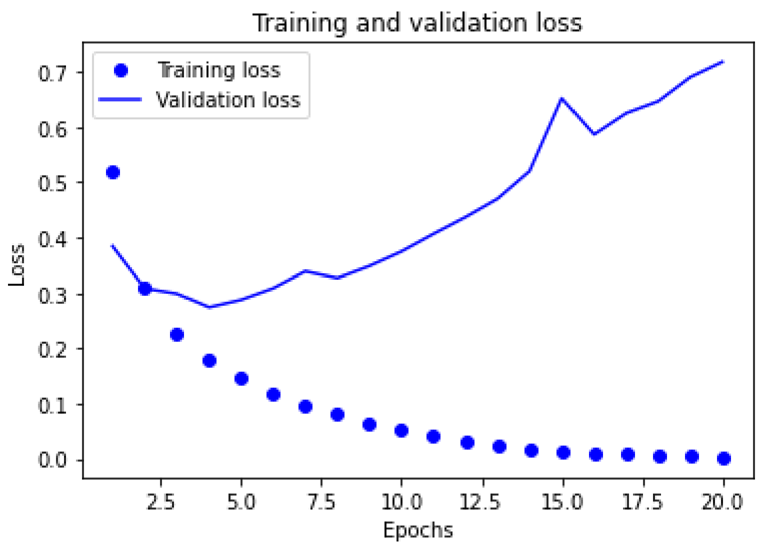
In [27]:

```
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()
```

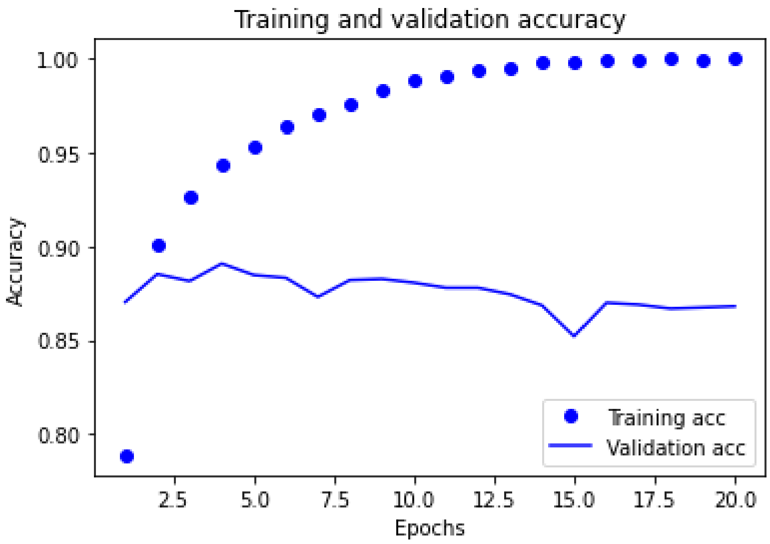


In [28]:

```
# 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]:

```
(0.783345184173584, 0.85196)
```

In [31]:

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

Out[31]:

```
array([[0.00308475],
       [0.9999999 ],
       [0.9244658 ],
       ...,
       [0.00168303],
       [0.00499073],
       [0.9267075 ]], dtype=float32)
```

In [32]:

```
model2 = models.Sequential()
model2.add(layers.Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.001) ,i
model2.add(layers.Dense(16, activation='relu'))
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 83us/sample - loss: 0.5
726 - accuracy: 0.7813 - val_loss: 0.4535 - val_accuracy: 0.8606
Epoch 2/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.3
810 - accuracy: 0.8886 - val_loss: 0.3830 - val_accuracy: 0.8630
Epoch 3/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.3
024 - accuracy: 0.9129 - val_loss: 0.3421 - val_accuracy: 0.8782
Epoch 4/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.2
618 - accuracy: 0.9237 - val_loss: 0.3195 - val_accuracy: 0.8894
Epoch 5/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.2
349 - accuracy: 0.9353 - val_loss: 0.3289 - val_accuracy: 0.8825
Epoch 6/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.2
180 - accuracy: 0.9426 - val_loss: 0.3233 - val_accuracy: 0.8858
Epoch 7/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.2
024 - accuracy: 0.9482 - val_loss: 0.3258 - val_accuracy: 0.8838
Epoch 8/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.1
913 - accuracy: 0.9535 - val_loss: 0.3484 - val_accuracy: 0.8826
Epoch 9/20
15000/15000 [==============================] - 1s 52us/sample - loss: 0.1
818 - accuracy: 0.9555 - val_loss: 0.3406 - val_accuracy: 0.8826
Epoch 10/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.1
710 - accuracy: 0.9618 - val_loss: 0.3680 - val_accuracy: 0.8791
Epoch 11/20
15000/15000 [==============================] - 1s 52us/sample - loss: 0.1
716 - accuracy: 0.9603 - val_loss: 0.3681 - val_accuracy: 0.8820
Epoch 12/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.1
620 - accuracy: 0.9633 - val_loss: 0.3660 - val_accuracy: 0.8820
Epoch 13/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.1
572 - accuracy: 0.9665 - val_loss: 0.3772 - val_accuracy: 0.8728
Epoch 14/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.1
542 - accuracy: 0.9675 - val_loss: 0.3765 - val_accuracy: 0.8764
Epoch 15/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.1
420 - accuracy: 0.9727 - val_loss: 0.4041 - val_accuracy: 0.8646
Epoch 16/20
```

```
15000/15000 [==============================] - 1s 51us/sample - loss: 0.1
424 - accuracy: 0.9722 - val_loss: 0.4117 - val_accuracy: 0.8730
Epoch 17/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.1
383 - accuracy: 0.9730 - val_loss: 0.4545 - val_accuracy: 0.8561
Epoch 18/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.1
383 - accuracy: 0.9719 - val_loss: 0.4167 - val_accuracy: 0.8757
Epoch 19/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.1
281 - accuracy: 0.9781 - val_loss: 0.4127 - val_accuracy: 0.8720
Epoch 20/20
15000/15000 [==============================] - 1s 51us/sample - loss: 0.1
303 - accuracy: 0.9759 - val_loss: 0.4167 - val_accuracy: 0.8720
```

Out[32]:

```
(0.43747030277252197, 0.86184)
```

In [33]:

```
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]:

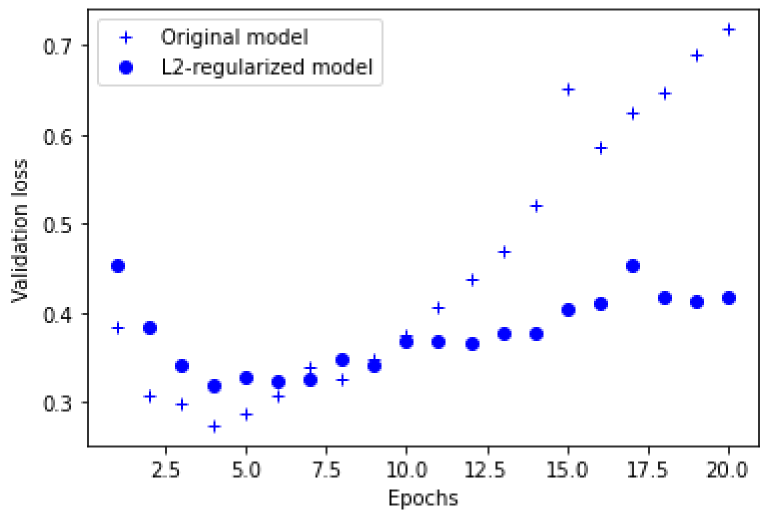
```
loss2 = history_dict2['loss']
val_loss2 = history_dict2['val_loss']

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

plt.plot(epochs1, val_loss1, 'b+', label='Original model')
plt.plot(epochs2, val_loss2, 'bo', label='L2-regularized 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='L2-regularized model')

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

plt.show()
```