第 3 次作业报告

SA19225404 吴语港

1、 尝试对 3.5 节中的解决方案做如下两种修改。验证修改后的方案，打印出验证集和训练集上的学习曲线；然后，观察分析学习曲线，找到最佳的 Epoch 取值；最后，设置最佳 Epoch 值后，在训练集上从头开始训练一个模型，评估该训练好的模型在测试集上的性能。

1）只修改隐藏层的单元数。尝试使用更多或更少的隐藏单元，比如 32 个、128 个等。

答:

64 个隐藏单元:（epochs=20）

代码的截图:

由于官网下载速度太慢了，拷贝了本地的数据集放在源程序文件夹处，并在 load 数据时注明本地 path 路径

```
[2] from keras.datasets import reuters

    (train_data, train_labels), (test_data, test_labels) = reuters.load_data(path="D:/Users/WYG/Desktop/科软/2学习/专业课_大数据与人工智能/AI□/作业/作业3/reuters.npz",
    num_words=10000)
```

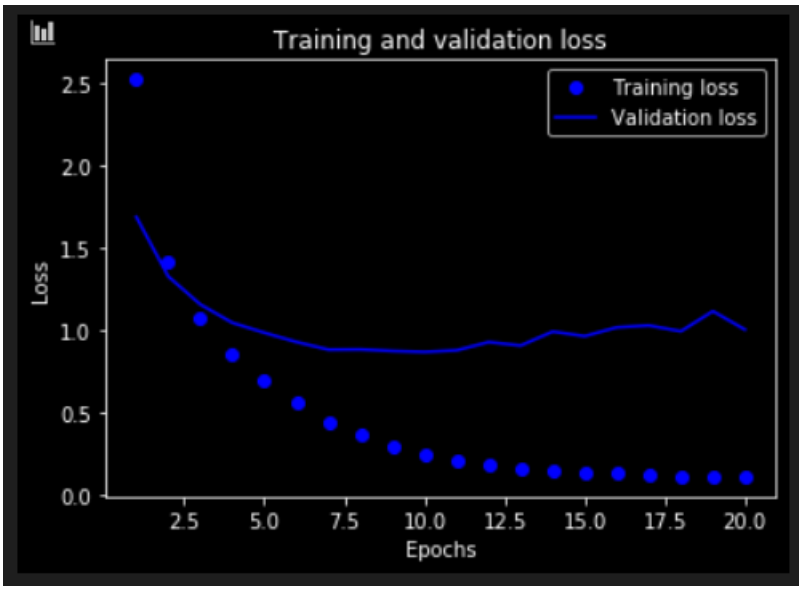
具有两层的神经网络，隐藏层 64 个神经元，激活函数为 relu，层与层之间连接采用全连接 dense。

```
[12] from keras import models
     from keras import layers

     model = models.Sequential()
     model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
     model.add(layers.Dense(64, activation='relu'))
     model.add(layers.Dense(46, activation='softmax'))
```

开始了 20 轮次的神经网络训练

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

WARNING:tensorflow:From D:\Users\WYG\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please use
tf.compat.v1.global_variables instead.

Train on 7982 samples, validate on 1000 samples
Epoch 1/20
7982/7982 [==============================] - 2s 245us/step - loss: 2.5260 - accuracy: 0.5192 - val_loss: 1.6920 - val_accuracy: 0.6220
Epoch 2/20
7982/7982 [==============================] - 2s 188us/step - loss: 1.4178 - accuracy: 0.6982 - val_loss: 1.3297 - val_accuracy: 0.7070
Epoch 3/20
7982/7982 [==============================] - 1s 184us/step - loss: 1.0819 - accuracy: 0.7686 - val_loss: 1.1599 - val_accuracy: 0.7430
Epoch 4/20
7982/7982 [==============================] - 1s 178us/step - loss: 0.8628 - accuracy: 0.8163 - val_loss: 1.0485 - val_accuracy: 0.7710
Epoch 5/20
7982/7982 [==============================] - 2s 191us/step - loss: 0.6990 - accuracy: 0.8495 - val_loss: 0.9880 - val_accuracy: 0.7880
Epoch 6/20
7982/7982 [==============================] - 1s 185us/step - loss: 0.5607 - accuracy: 0.8797 - val_loss: 0.9318 - val_accuracy: 0.7980
Epoch 7/20
7982/7982 [==============================] - 2s 190us/step - loss: 0.4470 - accuracy: 0.9049 - val_loss: 0.8855 - val_accuracy: 0.8180
Epoch 8/20
7982/7982 [==============================] - 2s 192us/step - loss: 0.3658 - accuracy: 0.9214 - val_loss: 0.8867 - val_accuracy: 0.8150
Epoch 9/20
7982/7982 [==============================] - 2s 190us/step - loss: 0.3022 - accuracy: 0.9347 - val_loss: 0.8769 - val_accuracy: 0.8240
Epoch 10/20
7982/7982 [==============================] - 2s 189us/step - loss: 0.2535 - accuracy: 0.9412 - val_loss: 0.8714 - val_accuracy: 0.8250
Epoch 11/20
7982/7982 [==============================] - 2s 191us/step - loss: 0.2160 - accuracy: 0.9481 - val_loss: 0.8811 - val_accuracy: 0.8170
Epoch 12/20
7982/7982 [==============================] - 1s 183us/step - loss: 0.1884 - accuracy: 0.9504 - val_loss: 0.9311 - val_accuracy: 0.8150
Epoch 13/20
7982/7982 [==============================] - 2s 190us/step - loss: 0.1677 - accuracy: 0.9525 - val_loss: 0.9100 - val_accuracy: 0.8260
Epoch 14/20
7982/7982 [==============================] - 1s 186us/step - loss: 0.1530 - accuracy: 0.9540 - val_loss: 0.9950 - val_accuracy: 0.8010
Epoch 15/20
7982/7982 [==============================] - 2s 190us/step - loss: 0.1396 - accuracy: 0.9548 - val_loss: 0.9672 - val_accuracy: 0.8160
Epoch 16/20
7982/7982 [==============================] - 1s 185us/step - loss: 0.1371 - accuracy: 0.9549 - val_loss: 1.0206 - val_accuracy: 0.7980
Epoch 17/20
7982/7982 [==============================] - 1s 186us/step - loss: 0.1263 - accuracy: 0.9553 - val_loss: 1.0322 - val_accuracy: 0.7950
```

学习曲线的截图:

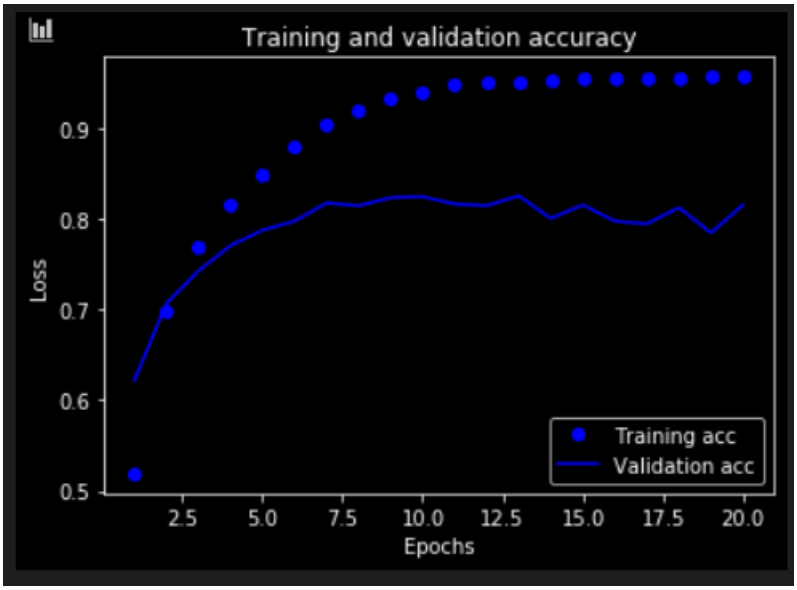


有点稍微过拟合，不是很明显

测试集上的性能:

程序源码出现问题，经过查找发现是 history 的参数不应该缩写为 acc 而应该是 accuracy

```
[19] plt.clf()   # clear figure

     acc = history.history['accuracy']
     val_acc = history.history['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('Loss')
     plt.legend()

     plt.show()
```



验证集的准确度大概 80%左右，最佳 epochs 大概是 10 左右

32 个隐藏单元：（epochs=20）

代码的截图：

修正了第一个参数（隐藏层神经元个数）为 32

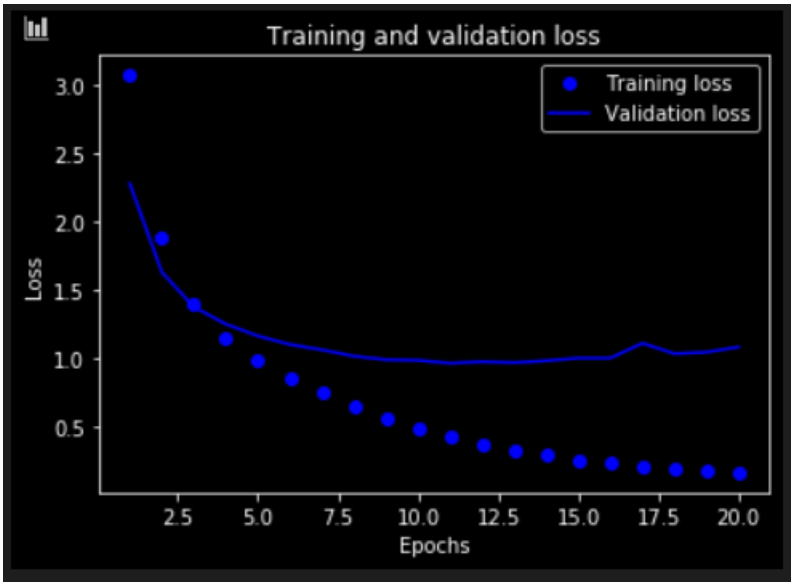
```
[20] from keras import models
     from keras import layers

     model = models.Sequential()
     model.add(layers.Dense(32, activation='relu', input_shape=(10000,)))
     model.add(layers.Dense(32, activation='relu'))
     model.add(layers.Dense(46, activation='softmax'))
```

开始训练

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

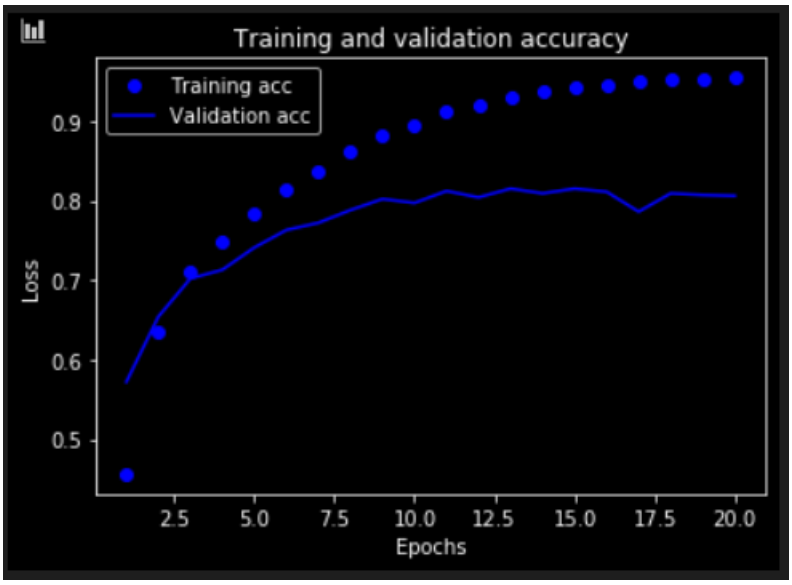
```
Train on 7982 samples, validate on 1000 samples
Epoch 1/20
7982/7982 [==============================] - 2s 200us/step - loss: 3.0767 - accuracy: 0.4564 - val_loss: 2.2811 - val_accuracy: 0.5720
Epoch 2/20
7982/7982 [==============================] - 1s 167us/step - loss: 1.8834 - accuracy: 0.6346 - val_loss: 1.6299 - val_accuracy: 0.6540
Epoch 3/20
7982/7982 [==============================] - 1s 165us/step - loss: 1.3943 - accuracy: 0.7097 - val_loss: 1.3752 - val_accuracy: 0.7020
Epoch 4/20
7982/7982 [==============================] - 1s 160us/step - loss: 1.1521 - accuracy: 0.7487 - val_loss: 1.2489 - val_accuracy: 0.7130
Epoch 5/20
7982/7982 [==============================] - 1s 173us/step - loss: 0.9863 - accuracy: 0.7840 - val_loss: 1.1617 - val_accuracy: 0.7410
Epoch 6/20
7982/7982 [==============================] - 1s 175us/step - loss: 0.8544 - accuracy: 0.8130 - val_loss: 1.0993 - val_accuracy: 0.7630
Epoch 7/20
7982/7982 [==============================] - 1s 167us/step - loss: 0.7410 - accuracy: 0.8361 - val_loss: 1.0602 - val_accuracy: 0.7720
Epoch 8/20
7982/7982 [==============================] - 1s 171us/step - loss: 0.6441 - accuracy: 0.8609 - val_loss: 1.0136 - val_accuracy: 0.7880
Epoch 9/20
7982/7982 [==============================] - 1s 171us/step - loss: 0.5579 - accuracy: 0.8817 - val_loss: 0.9874 - val_accuracy: 0.8020
Epoch 10/20
7982/7982 [==============================] - 1s 178us/step - loss: 0.4830 - accuracy: 0.8956 - val_loss: 0.9835 - val_accuracy: 0.7970
Epoch 11/20
7982/7982 [==============================] - 1s 174us/step - loss: 0.4214 - accuracy: 0.9113 - val_loss: 0.9613 - val_accuracy: 0.8120
Epoch 12/20
7982/7982 [==============================] - 1s 175us/step - loss: 0.3671 - accuracy: 0.9201 - val_loss: 0.9730 - val_accuracy: 0.8040
Epoch 13/20
7982/7982 [==============================] - 1s 171us/step - loss: 0.3235 - accuracy: 0.9300 - val_loss: 0.9657 - val_accuracy: 0.8150
Epoch 14/20
7982/7982 [==============================] - 1s 172us/step - loss: 0.2848 - accuracy: 0.9366 - val_loss: 0.9802 - val_accuracy: 0.8090
Epoch 15/20
7982/7982 [==============================] - 1s 171us/step - loss: 0.2525 - accuracy: 0.9424 - val_loss: 0.9993 - val_accuracy: 0.8150
Epoch 16/20
7982/7982 [==============================] - 1s 172us/step - loss: 0.2270 - accuracy: 0.9449 - val_loss: 0.9996 - val_accuracy: 0.8110
Epoch 17/20
7982/7982 [==============================] - 1s 173us/step - loss: 0.2040 - accuracy: 0.9488 - val_loss: 1.1088 - val_accuracy: 0.7860
Epoch 18/20
7982/7982 [==============================] - 1s 170us/step - loss: 0.1888 - accuracy: 0.9526 - val_loss: 1.0318 - val_accuracy: 0.8090
```

学习曲线的截图:



测试集的效果好了一丢丢

测试集上的性能:



准确度小幅度上升，最佳 epochs 大概 15 左右

128 个隐藏单元：（epochs=20）

代码的截图：

隐藏层神经元个数改为 128 个

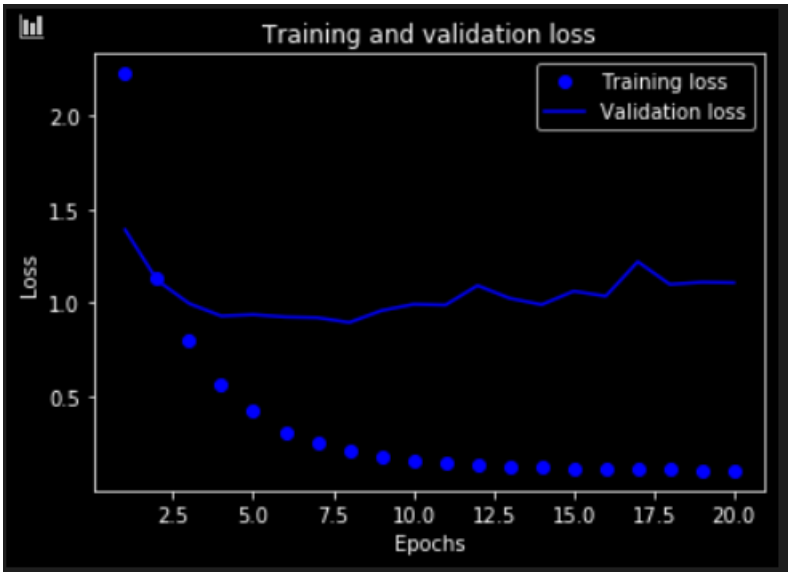
```
[26] from keras import models
     from keras import layers

     model = models.Sequential()
     model.add(layers.Dense(128, activation='relu', input_shape=(10000,)))
     model.add(layers.Dense(128, activation='relu'))
     model.add(layers.Dense(46, activation='softmax'))
```

开始训练

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

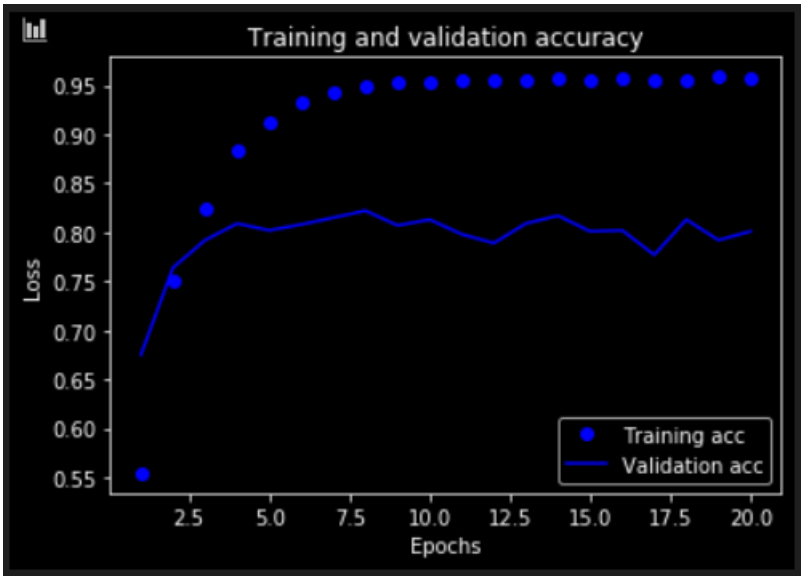
```
Train on 7982 samples, validate on 1000 samples
Epoch 1/20
7982/7982 [==============================] - 2s 264us/step - loss: 2.2216 - accuracy: 0.5542 - val_loss: 1.3906 - val_accuracy: 0.6750
Epoch 2/20
7982/7982 [==============================] - 2s 218us/step - loss: 1.1298 - accuracy: 0.7506 - val_loss: 1.1207 - val_accuracy: 0.7640
Epoch 3/20
7982/7982 [==============================] - 2s 223us/step - loss: 0.7972 - accuracy: 0.8242 - val_loss: 0.9970 - val_accuracy: 0.7920
Epoch 4/20
7982/7982 [==============================] - 2s 221us/step - loss: 0.5630 - accuracy: 0.8842 - val_loss: 0.9306 - val_accuracy: 0.8090
Epoch 5/20
7982/7982 [==============================] - 2s 223us/step - loss: 0.4246 - accuracy: 0.9117 - val_loss: 0.9378 - val_accuracy: 0.8020
Epoch 6/20
7982/7982 [==============================] - 2s 235us/step - loss: 0.3131 - accuracy: 0.9327 - val_loss: 0.9254 - val_accuracy: 0.8080
Epoch 7/20
7982/7982 [==============================] - 2s 220us/step - loss: 0.2520 - accuracy: 0.9426 - val_loss: 0.9217 - val_accuracy: 0.8150
Epoch 8/20
7982/7982 [==============================] - 2s 228us/step - loss: 0.2095 - accuracy: 0.9494 - val_loss: 0.8957 - val_accuracy: 0.8220
Epoch 9/20
7982/7982 [==============================] - 2s 228us/step - loss: 0.1788 - accuracy: 0.9536 - val_loss: 0.9591 - val_accuracy: 0.8070
Epoch 10/20
7982/7982 [==============================] - 2s 225us/step - loss: 0.1602 - accuracy: 0.9529 - val_loss: 0.9929 - val_accuracy: 0.8130
Epoch 11/20
7982/7982 [==============================] - 2s 227us/step - loss: 0.1477 - accuracy: 0.9550 - val_loss: 0.9891 - val_accuracy: 0.7980
Epoch 12/20
7982/7982 [==============================] - 2s 228us/step - loss: 0.1401 - accuracy: 0.9548 - val_loss: 1.0929 - val_accuracy: 0.7890
Epoch 13/20
7982/7982 [==============================] - 2s 227us/step - loss: 0.1327 - accuracy: 0.9544 - val_loss: 1.0249 - val_accuracy: 0.8090
Epoch 14/20
7982/7982 [==============================] - 2s 226us/step - loss: 0.1258 - accuracy: 0.9580 - val_loss: 0.9907 - val_accuracy: 0.8170
Epoch 15/20
7982/7982 [==============================] - 2s 222us/step - loss: 0.1214 - accuracy: 0.9558 - val_loss: 1.0627 - val_accuracy: 0.8010
Epoch 16/20
7982/7982 [==============================] - 2s 220us/step - loss: 0.1151 - accuracy: 0.9573 - val_loss: 1.0361 - val_accuracy: 0.8020
Epoch 17/20
7982/7982 [==============================] - 2s 225us/step - loss: 0.1137 - accuracy: 0.9555 - val_loss: 1.2198 - val_accuracy: 0.7770
Epoch 18/20
7982/7982 [==============================] - 2s 218us/step - loss: 0.1143 - accuracy: 0.9558 - val_loss: 1.0989 - val_accuracy: 0.8130
Epoch 19/20
```

学习曲线的截图:



有轻微过拟合，不过最佳性能好了一些

测试集上的性能:



最佳 epochs 大概在 7.5 左右，最佳性能有小幅度下降

对截图中的性能评测结果进行分析，说明其原因。(提示：对比分析：修改后，在测试集上的 Accuracy 是变好了还是变坏了？为什么？)

修改后发现在一定程度上增加隐藏层神经元个数有助于捕获更多有用的信息，从而增加准确度，在测试集上的 Accuracy 是变好了一些，不过如果无限制增加隐藏层神经元个数会导致模型太过于复杂从而导致过拟合，影响准确度，需要不断调节这些超参数，让准确率尽可能高一些。

2）只修改隐藏层的数量。前面使用了两个隐藏层，现在尝试使用一个或三个隐藏层。

答：

一个隐藏层：（epochs=20）

代码的截图：

注释掉一层

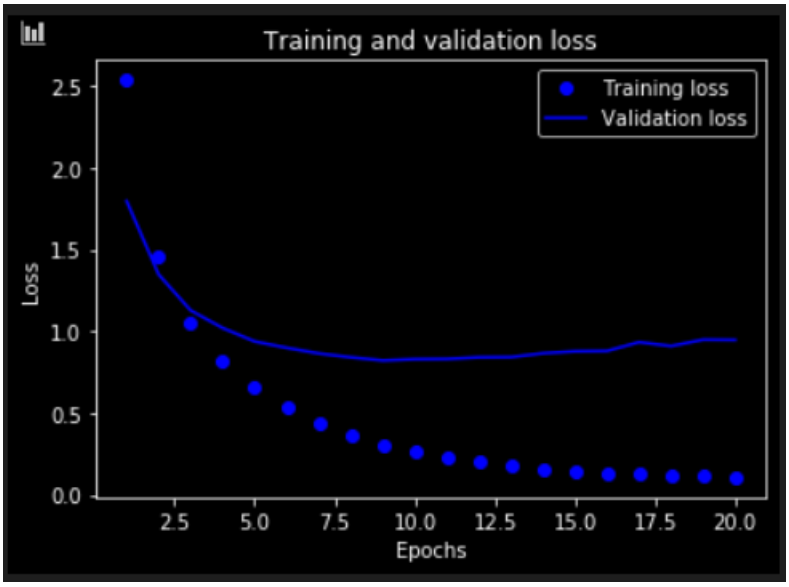
```
[32] from keras import models
     from keras import layers

     model = models.Sequential()
     model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
     # model.add(layers.Dense(64, activation='relu'))
     model.add(layers.Dense(46, activation='softmax'))
```

开始训练：

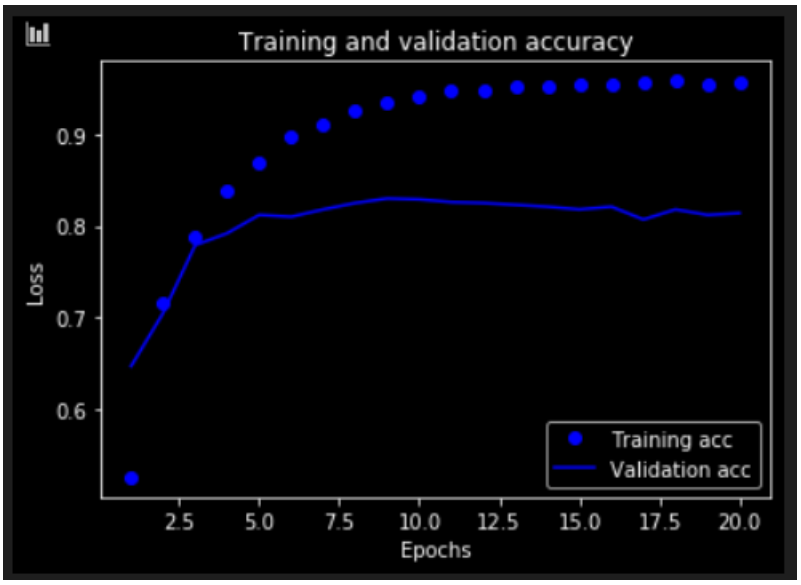
```
[35] history = model.fit(partial_x_train,
                         partial_y_train,
                         epochs=20,
                         batch_size=512,
                         validation_data=(x_val, y_val))
```

```
Train on 7982 samples, validate on 1000 samples
Epoch 1/20
7982/7982 [==============================] - 2s 213us/step - loss: 2.5349 - accuracy: 0.5258 - val_loss: 1.7961 - val_accuracy: 0.6470
Epoch 2/20
7982/7982 [==============================] - 1s 185us/step - loss: 1.4578 - accuracy: 0.7170 - val_loss: 1.3469 - val_accuracy: 0.7050
Epoch 3/20
7982/7982 [==============================] - 2s 190us/step - loss: 1.0575 - accuracy: 0.7884 - val_loss: 1.1273 - val_accuracy: 0.7790
Epoch 4/20
7982/7982 [==============================] - 2s 189us/step - loss: 0.8184 - accuracy: 0.8379 - val_loss: 1.0197 - val_accuracy: 0.7920
Epoch 5/20
7982/7982 [==============================] - 2s 191us/step - loss: 0.6542 - accuracy: 0.8698 - val_loss: 0.9374 - val_accuracy: 0.8120
Epoch 6/20
7982/7982 [==============================] - 2s 188us/step - loss: 0.5326 - accuracy: 0.8970 - val_loss: 0.8986 - val_accuracy: 0.8100
Epoch 7/20
7982/7982 [==============================] - 2s 196us/step - loss: 0.4396 - accuracy: 0.9116 - val_loss: 0.8641 - val_accuracy: 0.8180
Epoch 8/20
7982/7982 [==============================] - 2s 198us/step - loss: 0.3648 - accuracy: 0.9262 - val_loss: 0.8407 - val_accuracy: 0.8250
Epoch 9/20
7982/7982 [==============================] - 2s 204us/step - loss: 0.3086 - accuracy: 0.9351 - val_loss: 0.8214 - val_accuracy: 0.8300
Epoch 10/20
7982/7982 [==============================] - 2s 221us/step - loss: 0.2631 - accuracy: 0.9416 - val_loss: 0.8292 - val_accuracy: 0.8290
Epoch 11/20
7982/7982 [==============================] - 2s 205us/step - loss: 0.2261 - accuracy: 0.9471 - val_loss: 0.8308 - val_accuracy: 0.8260
Epoch 12/20
7982/7982 [==============================] - 1s 178us/step - loss: 0.2006 - accuracy: 0.9489 - val_loss: 0.8407 - val_accuracy: 0.8250
Epoch 13/20
7982/7982 [==============================] - 1s 171us/step - loss: 0.1769 - accuracy: 0.9530 - val_loss: 0.8419 - val_accuracy: 0.8230
Epoch 14/20
7982/7982 [==============================] - 1s 182us/step - loss: 0.1603 - accuracy: 0.9528 - val_loss: 0.8651 - val_accuracy: 0.8210
Epoch 15/20
7982/7982 [==============================] - 1s 172us/step - loss: 0.1481 - accuracy: 0.9539 - val_loss: 0.8771 - val_accuracy: 0.8180
Epoch 16/20
7982/7982 [==============================] - 1s 174us/step - loss: 0.1350 - accuracy: 0.9554 - val_loss: 0.8795 - val_accuracy: 0.8210
Epoch 17/20
7982/7982 [==============================] - 1s 173us/step - loss: 0.1252 - accuracy: 0.9564 - val_loss: 0.9332 - val_accuracy: 0.8070
Epoch 18/20
7982/7982 [==============================] - 1s 174us/step - loss: 0.1178 - accuracy: 0.9583 - val_loss: 0.9092 - val_accuracy: 0.8180
```

学习曲线的截图:



测试集上的性能:



性能较两层的模型有小幅度上升，最佳 epochs 大概为 10

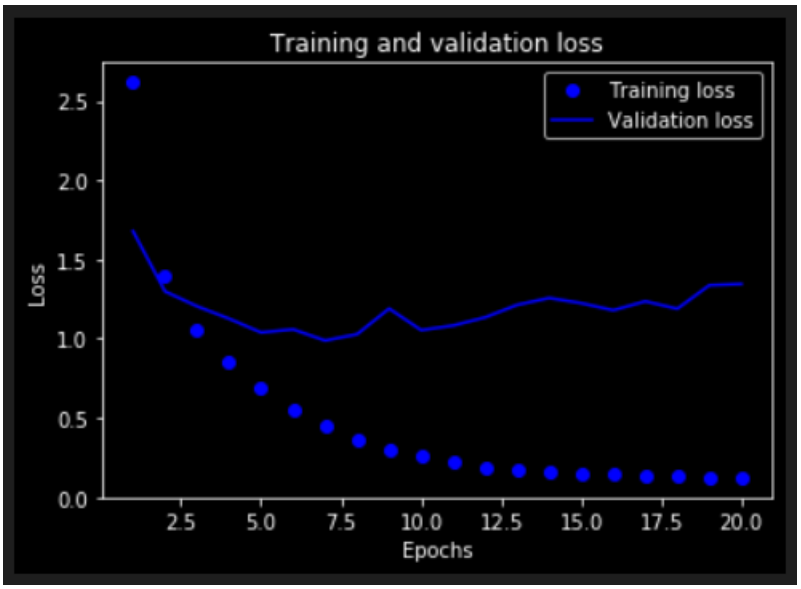
三个隐藏层：（epochs=20）

代码的截图：

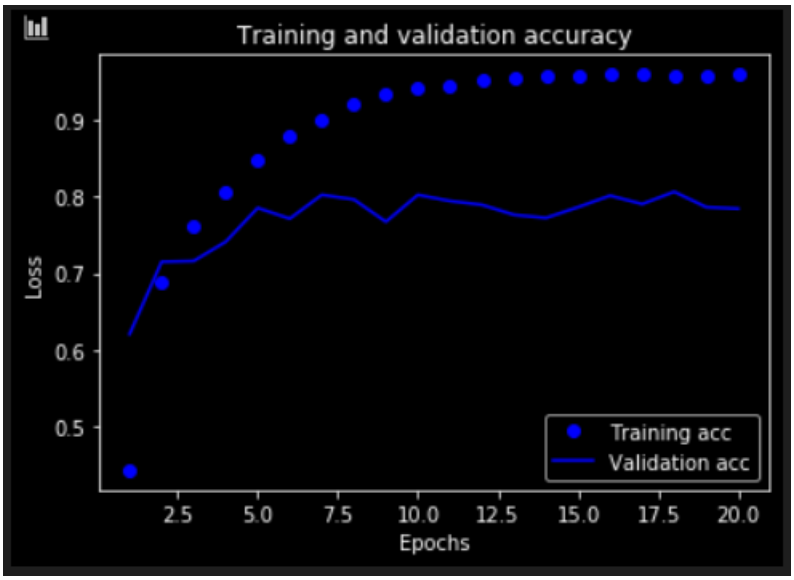
```
[38] from keras import models
     from keras import layers

     model = models.Sequential()
     model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
     model.add(layers.Dense(64, activation='relu'))
     model.add(layers.Dense(64, activation='relu'))
     model.add(layers.Dense(46, activation='softmax'))
```

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

Train on 7982 samples, validate on 1000 samples
Epoch 1/20
7982/7982 [==============================] - 2s 235us/step - loss: 2.6186 - accuracy: 0.4442 - val_loss: 1.6798 - val_accuracy: 0.6210
Epoch 2/20
7982/7982 [==============================] - 2s 194us/step - loss: 1.3928 - accuracy: 0.6894 - val_loss: 1.2987 - val_accuracy: 0.7150
Epoch 3/20
7982/7982 [==============================] - 2s 189us/step - loss: 1.0564 - accuracy: 0.7602 - val_loss: 1.2049 - val_accuracy: 0.7160
Epoch 4/20
7982/7982 [==============================] - 2s 190us/step - loss: 0.8485 - accuracy: 0.8064 - val_loss: 1.1266 - val_accuracy: 0.7410
Epoch 5/20
7982/7982 [==============================] - 2s 189us/step - loss: 0.6846 - accuracy: 0.8470 - val_loss: 1.0394 - val_accuracy: 0.7850
Epoch 6/20
7982/7982 [==============================] - 1s 187us/step - loss: 0.5462 - accuracy: 0.8792 - val_loss: 1.0592 - val_accuracy: 0.7710
Epoch 7/20
7982/7982 [==============================] - 1s 186us/step - loss: 0.4565 - accuracy: 0.8986 - val_loss: 0.9884 - val_accuracy: 0.8020
Epoch 8/20
7982/7982 [==============================] - 2s 196us/step - loss: 0.3637 - accuracy: 0.9194 - val_loss: 1.0284 - val_accuracy: 0.7960
Epoch 9/20
7982/7982 [==============================] - 2s 190us/step - loss: 0.2961 - accuracy: 0.9335 - val_loss: 1.1903 - val_accuracy: 0.7670
Epoch 10/20
7982/7982 [==============================] - 2s 190us/step - loss: 0.2576 - accuracy: 0.9419 - val_loss: 1.0542 - val_accuracy: 0.8020
Epoch 11/20
7982/7982 [==============================] - 2s 192us/step - loss: 0.2278 - accuracy: 0.9442 - val_loss: 1.0832 - val_accuracy: 0.7940
Epoch 12/20
7982/7982 [==============================] - 1s 188us/step - loss: 0.1878 - accuracy: 0.9504 - val_loss: 1.1349 - val_accuracy: 0.7890
Epoch 13/20
7982/7982 [==============================] - 2s 192us/step - loss: 0.1706 - accuracy: 0.9543 - val_loss: 1.2136 - val_accuracy: 0.7760
Epoch 14/20
7982/7982 [==============================] - 1s 187us/step - loss: 0.1597 - accuracy: 0.9555 - val_loss: 1.2571 - val_accuracy: 0.7720
Epoch 15/20
7982/7982 [==============================] - 2s 191us/step - loss: 0.1536 - accuracy: 0.9555 - val_loss: 1.2243 - val_accuracy: 0.7860
Epoch 16/20
7982/7982 [==============================] - 1s 187us/step - loss: 0.1431 - accuracy: 0.9585 - val_loss: 1.1797 - val_accuracy: 0.8010
Epoch 17/20
7982/7982 [==============================] - 2s 202us/step - loss: 0.1322 - accuracy: 0.9582 - val_loss: 1.2361 - val_accuracy: 0.7900
Epoch 18/20
7982/7982 [==============================] - 2s 190us/step - loss: 0.1323 - accuracy: 0.9565 - val_loss: 1.1890 - val_accuracy: 0.8060
```

学习曲线的截图：



测试集上的性能:



性能较两层的模型有小幅度上升，最佳 epochs 大概为 10

代码的截图。(代码修改之处用红色框圈出来。)

运行结果的截图。包含学习曲线的截图 和 测试集上的性能评测结果的截图。

对截图中的性能评测结果进行分析，说明其原因。(提示：对比分析：修改后，在测试集上的 Accuracy 是变好了还是变坏了？为什么？)

修改后发现在虽然做实验感觉一定程度上增加隐藏层个数有助于捕获更多有用的信息，从而增加准确度，但是做完了这个实验发现，其实一层的模型在测试集上的 Accuracy 居然更好了一些，可能是增加隐藏层个数会导致模型太过于复杂从而导致过拟合，影响准确度，需要不断调节这些超参数，找到最佳层数，让准确率尽可能高一些。

2、 尝试对 3.4 节中的解决方案做如下三种修改。验证修改后的方案，打印出验证集和训练集上的学习曲线；然后，观察分析学习曲线，找到最佳的 Epoch 取值；最后，设置最佳 Epoch 值后，在训练集上从头开始训练一个模型，评估该训练好的模型在测试集上的性能。

1） 只修改隐藏层的数量。前面使用了两个隐藏层。你可以尝试使用一个或三个隐藏层，然后观察对验证精度和测试精度的影响。

答:

两个隐藏层：（epochs=20）

代码的截图:

神经网络

由于官网下载速度太慢了，拷贝了本地的数据集放在源程序文件夹处，并在 load 数据时注明本地 path 路径

```
[2] from keras.datasets import imdb
    # 加载 IMDB 数据集
    from keras.datasets import imdb
    #加载路透社数据集
    (train_data, train_labels), (test_data, test_labels) = imdb.load_data(path="D:/Users/WYG/Desktop/科软/2学习/专业课_大数据与人工智能/AI□/作业/作业3/imdb.npz",
    num_words=10000)
```

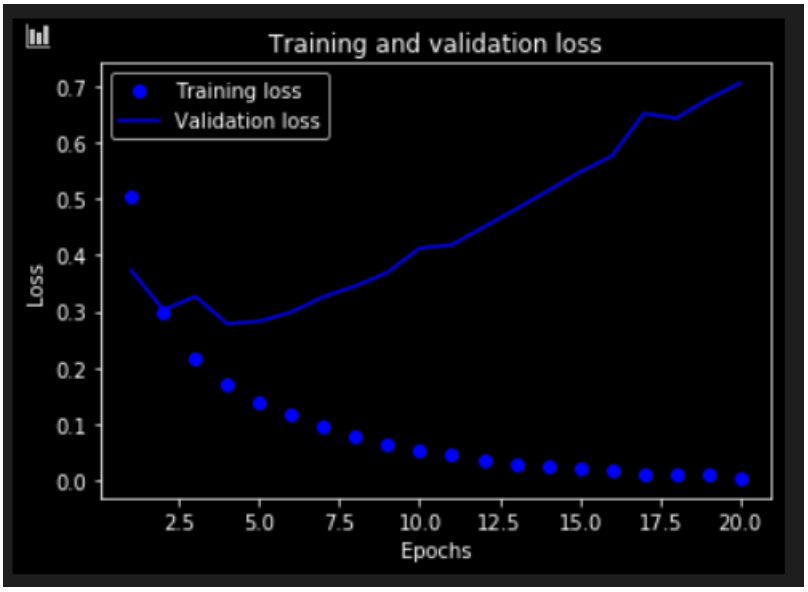
具有两层的神经网络，隐藏层 16 个神经元，激活函数为 relu，层与层之间连接采用全连接 dense。

```
[30] 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'))
```

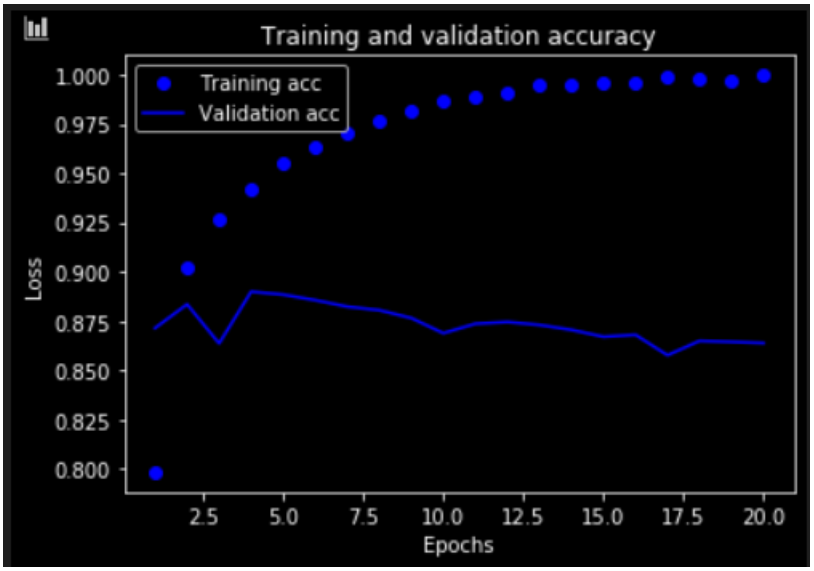
训练模型参数

```
[69] # 训练模型
     history = model.fit(partial_x_train,
                         partial_y_train,
                         epochs=20,
                         batch_size=512,
                         validation_data=(x_val, y_val))

Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 4s 265us/step - loss: 0.4963 - binary_accuracy: 0.7881 - val_loss: 0.3725 - val_binary_accuracy: 0.8693
Epoch 2/20
15000/15000 [==============================] - 3s 220us/step - loss: 0.2951 - binary_accuracy: 0.9013 - val_loss: 0.2962 - val_binary_accuracy: 0.8880
Epoch 3/20
15000/15000 [==============================] - 4s 250us/step - loss: 0.2174 - binary_accuracy: 0.9282 - val_loss: 0.3091 - val_binary_accuracy: 0.8758
Epoch 4/20
15000/15000 [==============================] - 4s 257us/step - loss: 0.1676 - binary_accuracy: 0.9464 - val_loss: 0.3189 - val_binary_accuracy: 0.8717
Epoch 5/20
15000/15000 [==============================] - 3s 198us/step - loss: 0.1427 - binary_accuracy: 0.9541 - val_loss: 0.3050 - val_binary_accuracy: 0.8801
Epoch 6/20
15000/15000 [==============================] - 3s 223us/step - loss: 0.1162 - binary_accuracy: 0.9637 - val_loss: 0.2946 - val_binary_accuracy: 0.8857
Epoch 7/20
15000/15000 [==============================] - 4s 252us/step - loss: 0.0939 - binary_accuracy: 0.9735 - val_loss: 0.3119 - val_binary_accuracy: 0.8830
Epoch 8/20
15000/15000 [==============================] - 3s 227us/step - loss: 0.0801 - binary_accuracy: 0.9781 - val_loss: 0.3378 - val_binary_accuracy: 0.8805
Epoch 9/20
15000/15000 [==============================] - 4s 252us/step - loss: 0.0623 - binary_accuracy: 0.9838 - val_loss: 0.3556 - val_binary_accuracy: 0.8795
Epoch 10/20
15000/15000 [==============================] - 4s 235us/step - loss: 0.0525 - binary_accuracy: 0.9871 - val_loss: 0.3906 - val_binary_accuracy: 0.8762
Epoch 11/20
15000/15000 [==============================] - 4s 245us/step - loss: 0.0394 - binary_accuracy: 0.9921 - val_loss: 0.4274 - val_binary_accuracy: 0.8689
Epoch 12/20
15000/15000 [==============================] - 4s 236us/step - loss: 0.0321 - binary_accuracy: 0.9935 - val_loss: 0.4923 - val_binary_accuracy: 0.8657
Epoch 13/20
15000/15000 [==============================] - 4s 244us/step - loss: 0.0270 - binary_accuracy: 0.9953 - val_loss: 0.4721 - val_binary_accuracy: 0.8735
Epoch 14/20
15000/15000 [==============================] - 4s 244us/step - loss: 0.0214 - binary_accuracy: 0.9963 - val_loss: 0.5022 - val_binary_accuracy: 0.8719
Epoch 15/20
15000/15000 [==============================] - 4s 242us/step - loss: 0.0160 - binary_accuracy: 0.9976 - val_loss: 0.5345 - val_binary_accuracy: 0.8716
Epoch 16/20
15000/15000 [==============================] - 3s 225us/step - loss: 0.0151 - binary_accuracy: 0.9973 - val_loss: 0.5672 - val_binary_accuracy: 0.8702
Epoch 17/20
15000/15000 [==============================] - 4s 262us/step - loss: 0.0071 - binary_accuracy: 0.9998 - val_loss: 0.6062 - val_binary_accuracy: 0.8664
Epoch 18/20
15000/15000 [==============================] - 4s 238us/step - loss: 0.0091 - binary_accuracy: 0.9987 - val_loss: 0.6325 - val_binary_accuracy: 0.8691
```

学习曲线的截图:



测试集上的性能评测结果的截图:



出现了过拟合，现在提前终止，，最佳 epochs 大概为 5，改 epochs=5

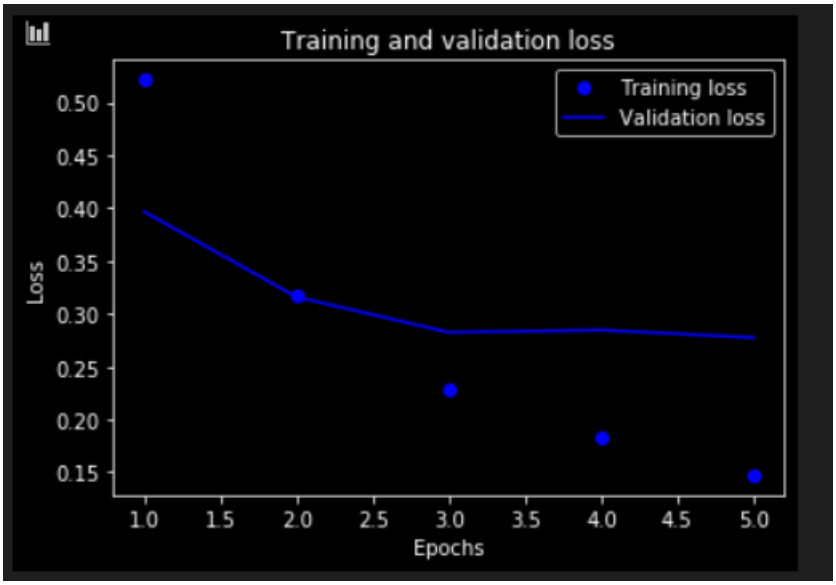
两个隐藏层：（epochs=5**）**

代码的截图:

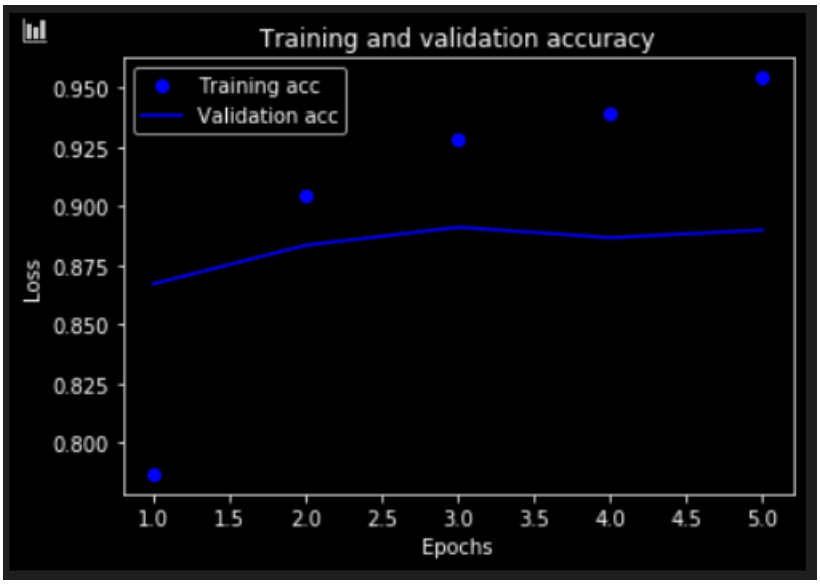
```
[32] # 训练模型
     history = model.fit(partial_x_train,
                         partial_y_train,
                         epochs=5,
                         batch_size=512,
                         validation_data=(x_val, y_val))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/5
15000/15000 [==============================] - 4s 251us/step - loss: 0.5215 -
binary_accuracy: 0.7869 - val_loss: 0.3967 - val_binary_accuracy: 0.8673
Epoch 2/5
15000/15000 [==============================] - 4s 260us/step - loss: 0.3166 -
binary_accuracy: 0.9042 - val_loss: 0.3160 - val_binary_accuracy: 0.8836
Epoch 3/5
15000/15000 [==============================] - 3s 231us/step - loss: 0.2285 -
binary_accuracy: 0.9284 - val_loss: 0.2825 - val_binary_accuracy: 0.8912
Epoch 4/5
15000/15000 [==============================] - 4s 266us/step - loss: 0.1825 -
binary_accuracy: 0.9397 - val_loss: 0.2847 - val_binary_accuracy: 0.8868
Epoch 5/5
15000/15000 [==============================] - 4s 259us/step - loss: 0.1473 -
binary_accuracy: 0.9544 - val_loss: 0.2775 - val_binary_accuracy: 0.8900
```

学习曲线的截图:



测试集上的性能评测结果的截图:



提前终止是应对过拟合的一个好办法，既提高了准确率，又节省了计算量

一个隐藏层：（epochs=20）

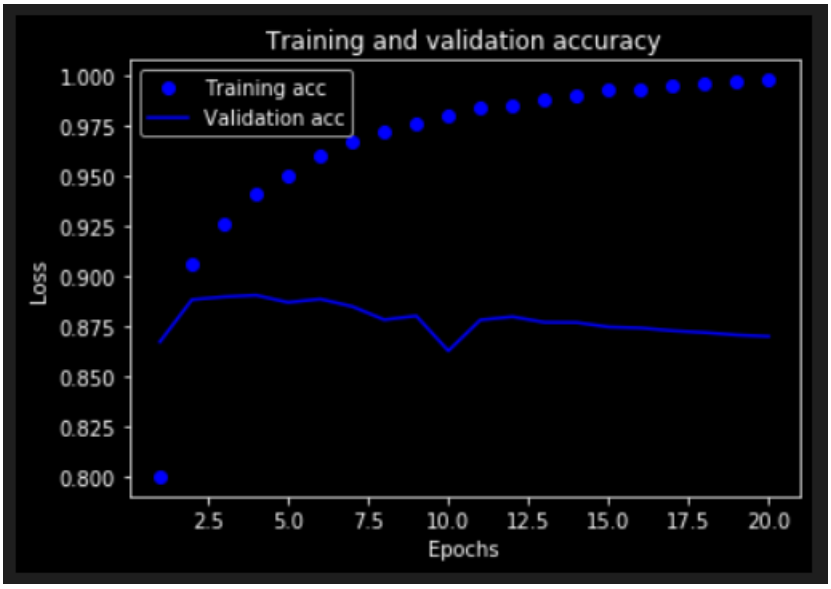
代码的截图：

```
[36] 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'))
```

```
[60] # 训练模型
     history = model.fit(partial_x_train,
                         partial_y_train,
                         epochs=20,
                         batch_size=512,
                         validation_data=(x_val, y_val))

Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 4s 261us/step - loss: 0.4890 - binary_accuracy: 0.8001 - val_loss: 0.3795 - val_binary_accuracy: 0.8671
Epoch 2/20
15000/15000 [==============================] - 4s 236us/step - loss: 0.3041 - binary_accuracy: 0.9063 - val_loss: 0.3098 - val_binary_accuracy: 0.8881
Epoch 3/20
15000/15000 [==============================] - 4s 238us/step - loss: 0.2367 - binary_accuracy: 0.9261 - val_loss: 0.2853 - val_binary_accuracy: 0.8895
Epoch 4/20
15000/15000 [==============================] - 4s 254us/step - loss: 0.1938 - binary_accuracy: 0.9407 - val_loss: 0.2747 - val_binary_accuracy: 0.8903
Epoch 5/20
15000/15000 [==============================] - 3s 225us/step - loss: 0.1654 - binary_accuracy: 0.9501 - val_loss: 0.2823 - val_binary_accuracy: 0.8867
Epoch 6/20
15000/15000 [==============================] - 4s 236us/step - loss: 0.1418 - binary_accuracy: 0.9597 - val_loss: 0.2763 - val_binary_accuracy: 0.8884
Epoch 7/20
15000/15000 [==============================] - 3s 213us/step - loss: 0.1223 - binary_accuracy: 0.9663 - val_loss: 0.2851 - val_binary_accuracy: 0.8847
Epoch 8/20
15000/15000 [==============================] - 3s 219us/step - loss: 0.1072 - binary_accuracy: 0.9722 - val_loss: 0.3072 - val_binary_accuracy: 0.8781
Epoch 9/20
15000/15000 [==============================] - 4s 254us/step - loss: 0.0949 - binary_accuracy: 0.9755 - val_loss: 0.3054 - val_binary_accuracy: 0.8800
Epoch 10/20
15000/15000 [==============================] - 4s 256us/step - loss: 0.0832 - binary_accuracy: 0.9796 - val_loss: 0.3738 - val_binary_accuracy: 0.8626
Epoch 11/20
15000/15000 [==============================] - 4s 234us/step - loss: 0.0733 - binary_accuracy: 0.9837 - val_loss: 0.3273 - val_binary_accuracy: 0.8780
Epoch 12/20
15000/15000 [==============================] - 4s 255us/step - loss: 0.0658 - binary_accuracy: 0.9851 - val_loss: 0.3394 - val_binary_accuracy: 0.8796
Epoch 13/20
15000/15000 [==============================] - 4s 251us/step - loss: 0.0582 - binary_accuracy: 0.9879 - val_loss: 0.3542 - val_binary_accuracy: 0.8768
Epoch 14/20
15000/15000 [==============================] - 4s 255us/step - loss: 0.0512 - binary_accuracy: 0.9902 - val_loss: 0.3693 - val_binary_accuracy: 0.8767
Epoch 15/20
15000/15000 [==============================] - 3s 208us/step - loss: 0.0438 - binary_accuracy: 0.9927 - val_loss: 0.3876 - val_binary_accuracy: 0.8745
Epoch 16/20
15000/15000 [==============================] - 4s 236us/step - loss: 0.0399 - binary_accuracy: 0.9927 - val_loss: 0.4037 - val_binary_accuracy: 0.8740
Epoch 17/20
15000/15000 [==============================] - 3s 200us/step - loss: 0.0343 - binary_accuracy: 0.9948 - val_loss: 0.4289 - val_binary_accuracy: 0.8726
```

学习曲线的截图：

测试集上的性能评测结果的截图:



过拟合好了一些，验证集的准确度也好了一些，最佳 epochs 大概为 4

三个隐藏层：（epochs=20）

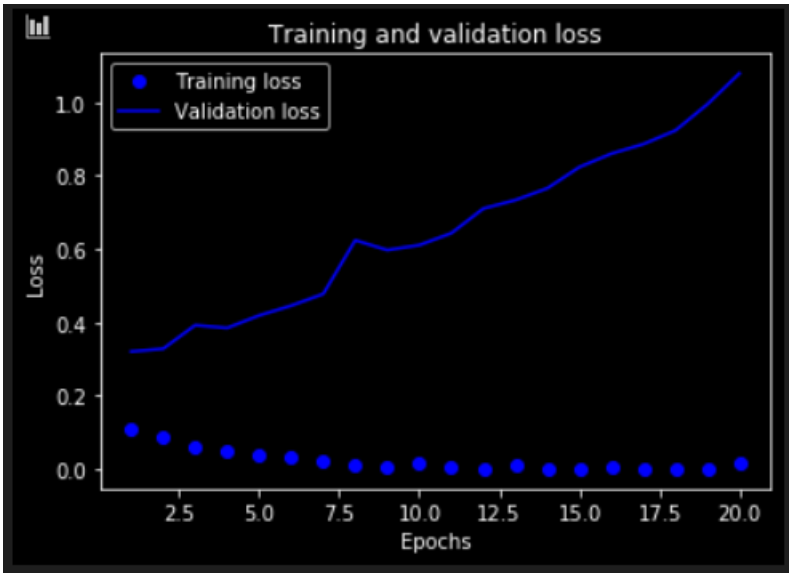
代码的截图:

```
[36] 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(16, activation='relu'))
     model.add(layers.Dense(1, activation='sigmoid'))
```

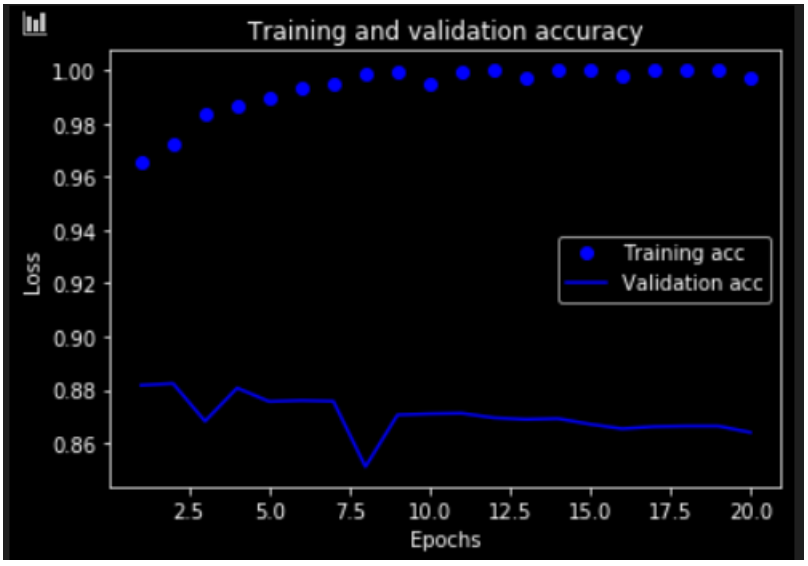
```
[51] # 训练模型
     history = model.fit(partial_x_train,
                         partial_y_train,
                         epochs=20,
                         batch_size=512,
                         validation_data=(x_val, y_val))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 4s 247us/step - loss: 0.1070 - binary_accuracy: 0.9657 - val_loss: 0.3201 - val_binary_accuracy: 0.8817
Epoch 2/20
15000/15000 [==============================] - 4s 261us/step - loss: 0.0881 - binary_accuracy: 0.9723 - val_loss: 0.3277 - val_binary_accuracy: 0.8824
Epoch 3/20
15000/15000 [==============================] - 4s 257us/step - loss: 0.0624 - binary_accuracy: 0.9833 - val_loss: 0.3920 - val_binary_accuracy: 0.8682
Epoch 4/20
15000/15000 [==============================] - 4s 238us/step - loss: 0.0502 - binary_accuracy: 0.9868 - val_loss: 0.3844 - val_binary_accuracy: 0.8807
Epoch 5/20
15000/15000 [==============================] - 4s 292us/step - loss: 0.0405 - binary_accuracy: 0.9897 - val_loss: 0.4186 - val_binary_accuracy: 0.8756
Epoch 6/20
15000/15000 [==============================] - 4s 295us/step - loss: 0.0302 - binary_accuracy: 0.9931 - val_loss: 0.4454 - val_binary_accuracy: 0.8760
Epoch 7/20
15000/15000 [==============================] - 4s 264us/step - loss: 0.0229 - binary_accuracy: 0.9947 - val_loss: 0.4773 - val_binary_accuracy: 0.8757
Epoch 8/20
15000/15000 [==============================] - 4s 263us/step - loss: 0.0121 - binary_accuracy: 0.9988 - val_loss: 0.6238 - val_binary_accuracy: 0.8511
Epoch 9/20
15000/15000 [==============================] - 4s 261us/step - loss: 0.0077 - binary_accuracy: 0.9997 - val_loss: 0.5970 - val_binary_accuracy: 0.8706
Epoch 10/20
15000/15000 [==============================] - 4s 284us/step - loss: 0.0158 - binary_accuracy: 0.9953 - val_loss: 0.6105 - val_binary_accuracy: 0.8710
Epoch 11/20
15000/15000 [==============================] - 4s 274us/step - loss: 0.0028 - binary_accuracy: 0.9998 - val_loss: 0.6430 - val_binary_accuracy: 0.8712
Epoch 12/20
15000/15000 [==============================] - 4s 294us/step - loss: 0.0020 - binary_accuracy: 0.9999 - val_loss: 0.7103 - val_binary_accuracy: 0.8695
Epoch 13/20
15000/15000 [==============================] - 4s 275us/step - loss: 0.0100 - binary_accuracy: 0.9969 - val_loss: 0.7333 - val_binary_accuracy: 0.8689
Epoch 14/20
15000/15000 [==============================] - 4s 251us/step - loss: 9.8719e-04 - binary_accuracy: 0.9999 - val_loss: 0.7657 - val_binary_accuracy: 0.8692
Epoch 15/20
15000/15000 [==============================] - 4s 271us/step - loss: 7.4403e-04 - binary_accuracy: 0.9999 - val_loss: 0.8234 - val_binary_accuracy: 0.8671
Epoch 16/20
15000/15000 [==============================] - 4s 256us/step - loss: 0.0067 - binary_accuracy: 0.9979 - val_loss: 0.8601 - val_binary_accuracy: 0.8654
Epoch 17/20
```

学习曲线的截图:



测试集上的性能评测结果的截图:



过拟合十分严重，准确度也十分差，最佳 epochs 大概为 1

对截图中的性能评测结果进行分析，说明其原因。(提示：对比分析：修改后，在测试集上的 Accuracy 是变好了还是变坏了？为什么？)

修改后发现在虽然做实验感觉一定程度上增加隐藏层个数有助于捕获更多有用的信息，从而增加准确度，但是做完了这个实验发现，其实一层的模型在测试集上的 Accuracy 居然更好了一些，可能是增加隐藏层个数会导致模型太过于复杂从而导致过拟合，影响准确度，需要不断调节这些超参数，找到最佳层数，让准确率尽可能高一些。

2) 只修改隐藏层的单元数。尝试用更多或更少的隐藏单元，比如 32 个、64 个等。

答:

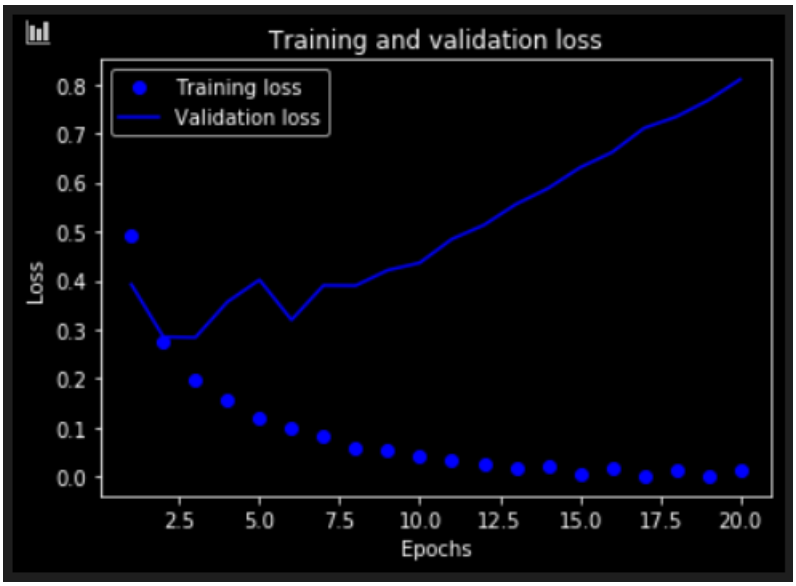
32 个隐藏单元：

代码的截图:

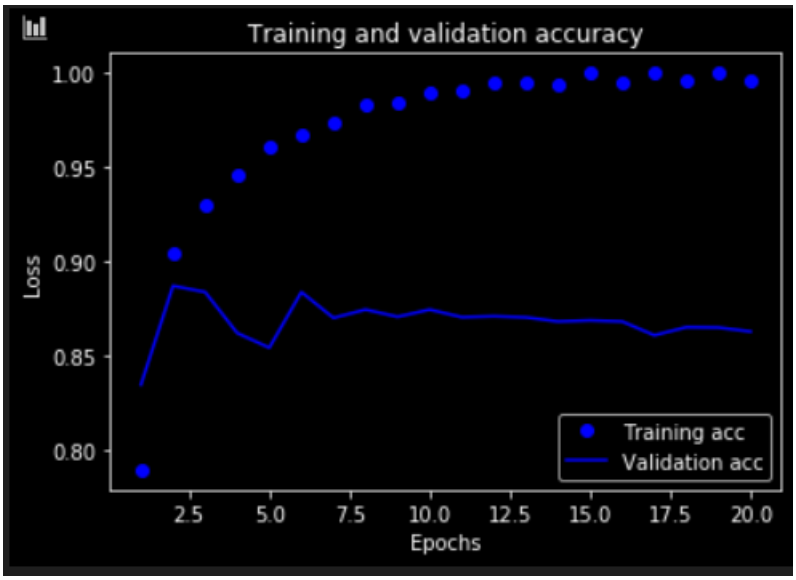
```
[70] from keras import models
     from keras import layers

     model = models.Sequential()
     model.add(layers.Dense(32, activation='relu', input_shape=(10000,)))
     model.add(layers.Dense(32, activation='relu'))
     model.add(layers.Dense(1, activation='sigmoid'))
```

```
[75] # 训练模型
     history = model.fit(partial_x_train,
                         partial_y_train,
                         epochs=20,
                         batch_size=512,
                         validation_data=(x_val, y_val))

Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 4s 242us/step - loss: 0.4906 - binary_accuracy: 0.7897 - val_loss: 0.3924 - val_binary_accuracy: 0.8348
Epoch 2/20
15000/15000 [==============================] - 4s 235us/step - loss: 0.2758 - binary_accuracy: 0.9042 - val_loss: 0.2854 - val_binary_accuracy: 0.8872
Epoch 3/20
15000/15000 [==============================] - 4s 249us/step - loss: 0.1985 - binary_accuracy: 0.9296 - val_loss: 0.2840 - val_binary_accuracy: 0.8839
Epoch 4/20
15000/15000 [==============================] - 4s 300us/step - loss: 0.1561 - binary_accuracy: 0.9459 - val_loss: 0.3565 - val_binary_accuracy: 0.8622
Epoch 5/20
15000/15000 [==============================] - 4s 264us/step - loss: 0.1199 - binary_accuracy: 0.9607 - val_loss: 0.4015 - val_binary_accuracy: 0.8544
Epoch 6/20
15000/15000 [==============================] - 3s 224us/step - loss: 0.0994 - binary_accuracy: 0.9675 - val_loss: 0.3198 - val_binary_accuracy: 0.8839
Epoch 7/20
15000/15000 [==============================] - 4s 240us/step - loss: 0.0817 - binary_accuracy: 0.9735 - val_loss: 0.3902 - val_binary_accuracy: 0.8702
Epoch 8/20
15000/15000 [==============================] - 4s 247us/step - loss: 0.0607 - binary_accuracy: 0.9828 - val_loss: 0.3897 - val_binary_accuracy: 0.8746
Epoch 9/20
15000/15000 [==============================] - 3s 229us/step - loss: 0.0542 - binary_accuracy: 0.9840 - val_loss: 0.4210 - val_binary_accuracy: 0.8708
Epoch 10/20
15000/15000 [==============================] - 3s 229us/step - loss: 0.0417 - binary_accuracy: 0.9893 - val_loss: 0.4361 - val_binary_accuracy: 0.8746
Epoch 11/20
15000/15000 [==============================] - 3s 227us/step - loss: 0.0332 - binary_accuracy: 0.9910 - val_loss: 0.4844 - val_binary_accuracy: 0.8706
Epoch 12/20
15000/15000 [==============================] - 4s 269us/step - loss: 0.0242 - binary_accuracy: 0.9952 - val_loss: 0.5130 - val_binary_accuracy: 0.8711
Epoch 13/20
15000/15000 [==============================] - 3s 224us/step - loss: 0.0196 - binary_accuracy: 0.9953 - val_loss: 0.5555 - val_binary_accuracy: 0.8705
Epoch 14/20
15000/15000 [==============================] - 4s 243us/step - loss: 0.0223 - binary_accuracy: 0.9937 - val_loss: 0.5877 - val_binary_accuracy: 0.8683
Epoch 15/20
15000/15000 [==============================] - 3s 227us/step - loss: 0.0065 - binary_accuracy: 0.9997 - val_loss: 0.6302 - val_binary_accuracy: 0.8689
Epoch 16/20
15000/15000 [==============================] - 4s 245us/step - loss: 0.0170 - binary_accuracy: 0.9947 - val_loss: 0.6613 - val_binary_accuracy: 0.8683
Epoch 17/20
15000/15000 [==============================] - 4s 263us/step - loss: 0.0034 - binary_accuracy: 0.9999 - val_loss: 0.7107 - val_binary_accuracy: 0.8610
```

学习曲线的截图:



测试集上的性能评测结果的截图:



性能并没有多大改善，训练曲线有锯齿状现象，最佳 epochs 大概为 2

64 个隐藏单元：

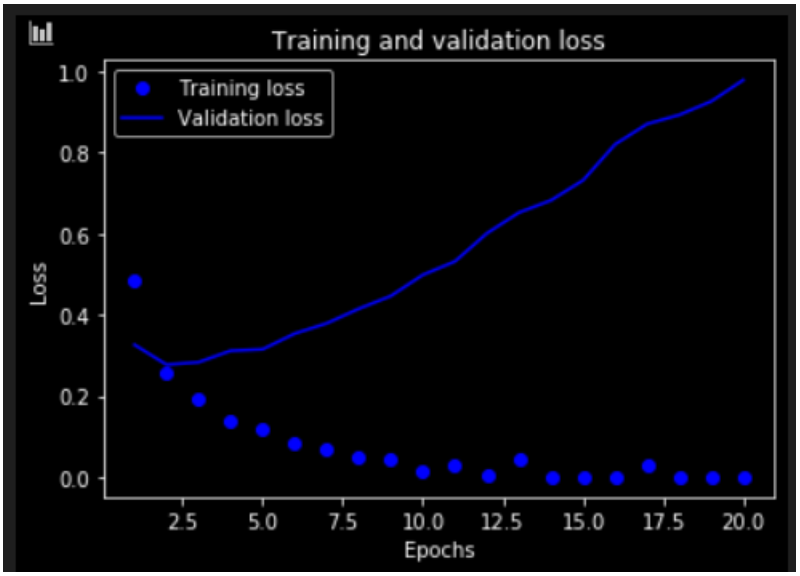
代码的截图:

```
[70] from keras import models
     from keras import layers

     model = models.Sequential()
     model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
     model.add(layers.Dense(64, activation='relu'))
     model.add(layers.Dense(1, activation='sigmoid'))
```

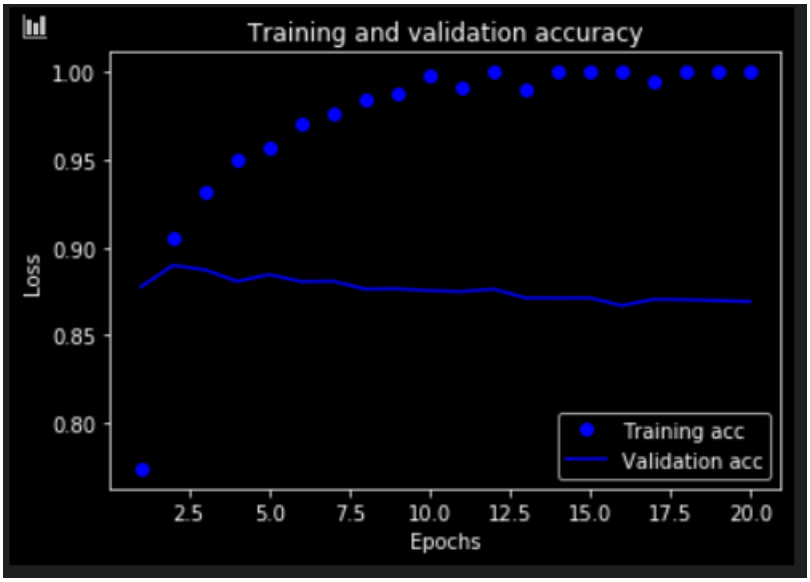
```
[84] # 训练模型
     history = model.fit(partial_x_train,
                         partial_y_train,
                         epochs=20,
                         batch_size=512,
                         validation_data=(x_val, y_val))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 5s 302us/step - loss: 0.4827 - binary_accuracy: 0.7740 - val_loss: 0.3263 - val_binary_accuracy: 0.8776
Epoch 2/20
15000/15000 [==============================] - 4s 273us/step - loss: 0.2577 - binary_accuracy: 0.9053 - val_loss: 0.2772 - val_binary_accuracy: 0.8898
Epoch 3/20
15000/15000 [==============================] - 4s 262us/step - loss: 0.1908 - binary_accuracy: 0.9313 - val_loss: 0.2831 - val_binary_accuracy: 0.8873
Epoch 4/20
15000/15000 [==============================] - 4s 266us/step - loss: 0.1389 - binary_accuracy: 0.9504 - val_loss: 0.3114 - val_binary_accuracy: 0.8807
Epoch 5/20
15000/15000 [==============================] - 4s 268us/step - loss: 0.1184 - binary_accuracy: 0.9565 - val_loss: 0.3150 - val_binary_accuracy: 0.8846
Epoch 6/20
15000/15000 [==============================] - 4s 270us/step - loss: 0.0848 - binary_accuracy: 0.9703 - val_loss: 0.3536 - val_binary_accuracy: 0.8806
Epoch 7/20
15000/15000 [==============================] - 4s 272us/step - loss: 0.0694 - binary_accuracy: 0.9762 - val_loss: 0.3790 - val_binary_accuracy: 0.8808
Epoch 8/20
15000/15000 [==============================] - 4s 279us/step - loss: 0.0508 - binary_accuracy: 0.9845 - val_loss: 0.4149 - val_binary_accuracy: 0.8763
Epoch 9/20
15000/15000 [==============================] - 4s 272us/step - loss: 0.0423 - binary_accuracy: 0.9883 - val_loss: 0.4463 - val_binary_accuracy: 0.8765
Epoch 10/20
15000/15000 [==============================] - 4s 271us/step - loss: 0.0130 - binary_accuracy: 0.9983 - val_loss: 0.4984 - val_binary_accuracy: 0.8754
Epoch 11/20
15000/15000 [==============================] - 4s 268us/step - loss: 0.0287 - binary_accuracy: 0.9917 - val_loss: 0.5315 - val_binary_accuracy: 0.8749
Epoch 12/20
15000/15000 [==============================] - 4s 280us/step - loss: 0.0049 - binary_accuracy: 0.9997 - val_loss: 0.6009 - val_binary_accuracy: 0.8763
Epoch 13/20
15000/15000 [==============================] - 4s 280us/step - loss: 0.0452 - binary_accuracy: 0.9905 - val_loss: 0.6520 - val_binary_accuracy: 0.8713
Epoch 14/20
15000/15000 [==============================] - 4s 278us/step - loss: 0.0015 - binary_accuracy: 0.9999 - val_loss: 0.6822 - val_binary_accuracy: 0.8712
Epoch 15/20
15000/15000 [==============================] - 4s 274us/step - loss: 0.0010 - binary_accuracy: 0.9999 - val_loss: 0.7316 - val_binary_accuracy: 0.8713
Epoch 16/20
15000/15000 [==============================] - 4s 266us/step - loss: 6.3949e-04 - binary_accuracy: 1.0000 - val_loss: 0.8203 - val_binary_accuracy: 0.8669
Epoch 17/20
15000/15000 [==============================] - 4s 265us/step - loss: 0.0288 - binary_accuracy: 0.9941 - val_loss: 0.8705 - val_binary_accuracy: 0.8705
```

学习曲线的截图:



测试集上的性能评测结果的截图:



性能变化不大，最佳 epochs 大概为 2，训练曲线有锯齿状现象消失了，神奇，是因为神经元多，考虑周全了吗。

对截图中的性能评测结果进行分析，说明其原因。(提示：对比分析：修改后，在测试集上的 Accuracy 是变好了还是变坏了？为什么？)

修改后发现在一定程度上增加隐藏层神经元个数有助于捕获更多有用的信息，从而增加准确度，在测试集上的 Accuracy 是变好了一些，不过如果无限制增加隐藏层神经元个数会导致模型太过于复杂从而导致过拟合，影响准确度，需要不断调节这些超参数，让准确率尽可能高一些。

3） 只修改损失函数。尝试使用 mse 损失函数代替 binary_crossentropy。

答:

代码的截图:

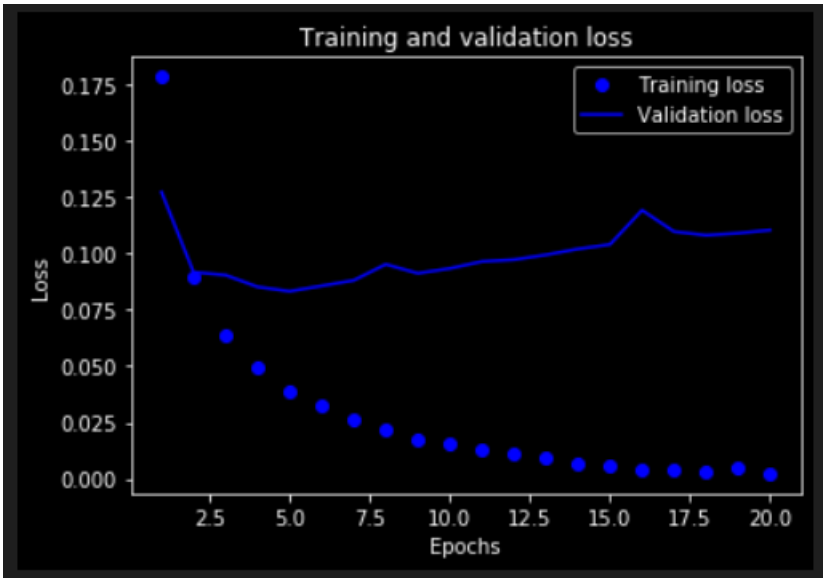
```
[91] from keras import optimizers
     # 配置优化器
     model.compile(optimizer=optimizers.RMSprop(lr=0.001),
                   loss='mse',
                   metrics=['accuracy'])
```

```
[92] from keras import losses
     from keras import metrics
     # 使用自定义的损失和指标
     model.compile(optimizer=optimizers.RMSprop(lr=0.001),
                   loss=losses.mse,
                   metrics=[metrics.binary_accuracy])
```

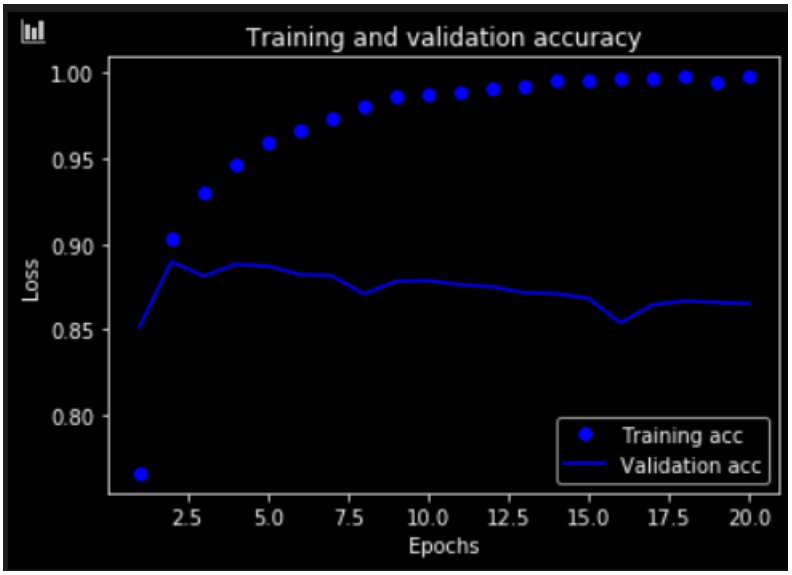
```
[94] # 训练模型
     history = model.fit(partial_x_train,
                         partial_y_train,
                         epochs=20,
                         batch_size=512,
                         validation_data=(x_val, y_val))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 4s 299us/step - loss: 0.1785 - binary_accuracy: 0.7660 - val_loss: 0.1274 - val_binary_accuracy: 0.8513
Epoch 2/20
15000/15000 [==============================] - 4s 255us/step - loss: 0.0900 - binary_accuracy: 0.9033 - val_loss: 0.0918 - val_binary_accuracy: 0.8897
Epoch 3/20
15000/15000 [==============================] - 4s 242us/step - loss: 0.0641 - binary_accuracy: 0.9297 - val_loss: 0.0905 - val_binary_accuracy: 0.8812
Epoch 4/20
15000/15000 [==============================] - 4s 248us/step - loss: 0.0495 - binary_accuracy: 0.9469 - val_loss: 0.0852 - val_binary_accuracy: 0.8881
Epoch 5/20
15000/15000 [==============================] - 4s 258us/step - loss: 0.0393 - binary_accuracy: 0.9597 - val_loss: 0.0833 - val_binary_accuracy: 0.8871
Epoch 6/20
15000/15000 [==============================] - 4s 272us/step - loss: 0.0327 - binary_accuracy: 0.9666 - val_loss: 0.0857 - val_binary_accuracy: 0.8822
Epoch 7/20
15000/15000 [==============================] - 4s 274us/step - loss: 0.0266 - binary_accuracy: 0.9741 - val_loss: 0.0881 - val_binary_accuracy: 0.8814
Epoch 8/20
15000/15000 [==============================] - 4s 263us/step - loss: 0.0222 - binary_accuracy: 0.9803 - val_loss: 0.0953 - val_binary_accuracy: 0.8708
Epoch 9/20
15000/15000 [==============================] - 4s 267us/step - loss: 0.0172 - binary_accuracy: 0.9864 - val_loss: 0.0913 - val_binary_accuracy: 0.8782
Epoch 10/20
15000/15000 [==============================] - 4s 273us/step - loss: 0.0159 - binary_accuracy: 0.9871 - val_loss: 0.0934 - val_binary_accuracy: 0.8786
Epoch 11/20
15000/15000 [==============================] - 4s 274us/step - loss: 0.0127 - binary_accuracy: 0.9887 - val_loss: 0.0965 - val_binary_accuracy: 0.8763
Epoch 12/20
15000/15000 [==============================] - 4s 263us/step - loss: 0.0110 - binary_accuracy: 0.9908 - val_loss: 0.0973 - val_binary_accuracy: 0.8750
Epoch 13/20
15000/15000 [==============================] - 4s 271us/step - loss: 0.0095 - binary_accuracy: 0.9920 - val_loss: 0.0995 - val_binary_accuracy: 0.8715
Epoch 14/20
15000/15000 [==============================] - 4s 265us/step - loss: 0.0065 - binary_accuracy: 0.9955 - val_loss: 0.1021 - val_binary_accuracy: 0.8710
Epoch 15/20
15000/15000 [==============================] - 4s 266us/step - loss: 0.0063 - binary_accuracy: 0.9953 - val_loss: 0.1041 - val_binary_accuracy: 0.8682
Epoch 16/20
15000/15000 [==============================] - 4s 276us/step - loss: 0.0043 - binary_accuracy: 0.9968 - val_loss: 0.1193 - val_binary_accuracy: 0.8539
Epoch 17/20
15000/15000 [==============================] - 4s 251us/step - loss: 0.0043 - binary_accuracy: 0.9965 - val_loss: 0.1098 - val_binary_accuracy: 0.8645
Epoch 18/20
15000/15000 [==============================] - 4s 240us/step - loss: 0.0031 - binary_accuracy: 0.9976 - val_loss: 0.1082 - val_binary_accuracy: 0.8667
```

学习曲线的截图:



测试集上的性能评测结果的截图:



验证集的预测准确度下降了，最佳 epochs 大概为 3

对截图中的性能评测结果进行分析，说明其原因。(提示：对比分析：修改后，在测试集上的 Accuracy 是变好了还是变坏了？为什么？)

常用的损失函数有均方误差：MSE

是最经典也是最简单的损失函数，几乎万能，但是不太准确。

binary_crossentropy 交叉熵损失函数，一般用于二分类：

这个是针对概率之间的损失函数，你会发现只有 yi 和 y^i 是相等时，loss 才为 0，否则 loss 就是为一个正数。而且，概率相差越大，loss 就越大。这个神奇的度量概率距离的方式称为交叉熵。

categorical_crossentropy 分类交叉熵函数：

n 是样本数，m 是分类数，注意，这是一个多输出的 loss 的函数，所以它的 loss 计算也是多个的。

在本例中使用 MSE 损失函数的效果比 binary_crossentropy 交叉熵损失函数的效果差了一些，看来还是要专业的损失函数对应专业的模型。

4） 只修改激活函数。尝试使用 tanh 激活（这种激活在神经网络早期非常流行）代替 relu。

答：

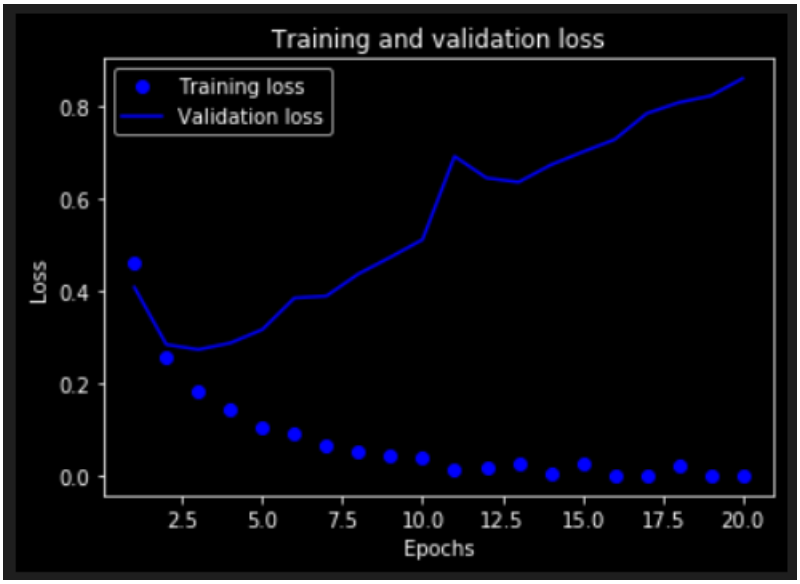
代码的截图：

```
[98] from keras import models
     from keras import layers

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

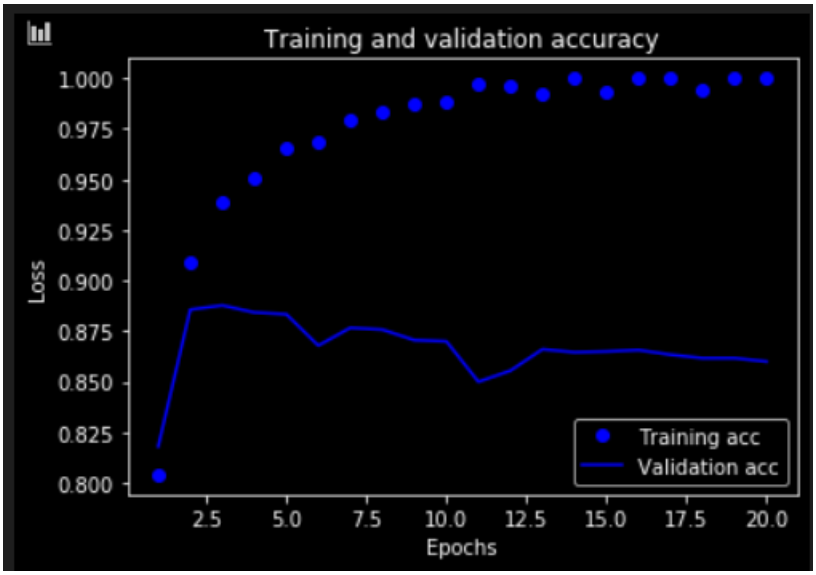
```
[103]# 训练模型
     history = model.fit(partial_x_train,
                         partial_y_train,
                         epochs=20,
                         batch_size=512,
                         validation_data=(x_val, y_val))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 4s 274us/step - loss: 0.4604 - binary_accuracy: 0.8040 - val_loss: 0.4085 - val_binary_accuracy: 0.8179
Epoch 2/20
15000/15000 [==============================] - 4s 237us/step - loss: 0.2588 - binary_accuracy: 0.9093 - val_loss: 0.2841 - val_binary_accuracy: 0.8856
Epoch 3/20
15000/15000 [==============================] - 4s 237us/step - loss: 0.1821 - binary_accuracy: 0.9391 - val_loss: 0.2732 - val_binary_accuracy: 0.8877
Epoch 4/20
15000/15000 [==============================] - 4s 243us/step - loss: 0.1425 - binary_accuracy: 0.9506 - val_loss: 0.2875 - val_binary_accuracy: 0.8842
Epoch 5/20
15000/15000 [==============================] - 4s 249us/step - loss: 0.1054 - binary_accuracy: 0.9654 - val_loss: 0.3167 - val_binary_accuracy: 0.8833
Epoch 6/20
15000/15000 [==============================] - 4s 251us/step - loss: 0.0899 - binary_accuracy: 0.9683 - val_loss: 0.3850 - val_binary_accuracy: 0.8678
Epoch 7/20
15000/15000 [==============================] - 4s 261us/step - loss: 0.0652 - binary_accuracy: 0.9798 - val_loss: 0.3893 - val_binary_accuracy: 0.8766
Epoch 8/20
15000/15000 [==============================] - 4s 243us/step - loss: 0.0539 - binary_accuracy: 0.9831 - val_loss: 0.4372 - val_binary_accuracy: 0.8757
Epoch 9/20
15000/15000 [==============================] - 4s 248us/step - loss: 0.0426 - binary_accuracy: 0.9873 - val_loss: 0.4735 - val_binary_accuracy: 0.8705
Epoch 10/20
15000/15000 [==============================] - 4s 245us/step - loss: 0.0378 - binary_accuracy: 0.9883 - val_loss: 0.5115 - val_binary_accuracy: 0.8699
Epoch 11/20
15000/15000 [==============================] - 4s 243us/step - loss: 0.0147 - binary_accuracy: 0.9971 - val_loss: 0.6921 - val_binary_accuracy: 0.8499
Epoch 12/20
15000/15000 [==============================] - 4s 246us/step - loss: 0.0158 - binary_accuracy: 0.9960 - val_loss: 0.6451 - val_binary_accuracy: 0.8554
Epoch 13/20
15000/15000 [==============================] - 4s 247us/step - loss: 0.0270 - binary_accuracy: 0.9925 - val_loss: 0.6361 - val_binary_accuracy: 0.8660
Epoch 14/20
15000/15000 [==============================] - 4s 252us/step - loss: 0.0039 - binary_accuracy: 0.9998 - val_loss: 0.6732 - val_binary_accuracy: 0.8645
Epoch 15/20
15000/15000 [==============================] - 4s 245us/step - loss: 0.0248 - binary_accuracy: 0.9935 - val_loss: 0.7016 - val_binary_accuracy: 0.8649
Epoch 16/20
15000/15000 [==============================] - 4s 244us/step - loss: 0.0017 - binary_accuracy: 0.9999 - val_loss: 0.7284 - val_binary_accuracy: 0.8656
Epoch 17/20
15000/15000 [==============================] - 4s 246us/step - loss: 0.0012 - binary_accuracy: 0.9999 - val_loss: 0.7848 - val_binary_accuracy: 0.8633
```

学习曲线的截图:



测试集上的性能评测结果的截图:



效果变差了，最佳 epochs 大概为 3

对截图中的性能评测结果进行分析，说明其原因。(提示：对比分析：修改后，在测试集上的 Accuracy 是变好了还是变坏了？为什么？)

修改后在测试集上的效果变差了，可能是因为 tanh 存在梯度消失和指数计算问题

附：各种激活函数对比

早期研究神经网络主要采用 sigmoid 函数或者 tanh 函数，输出有界，很容易充当下一层的输入。

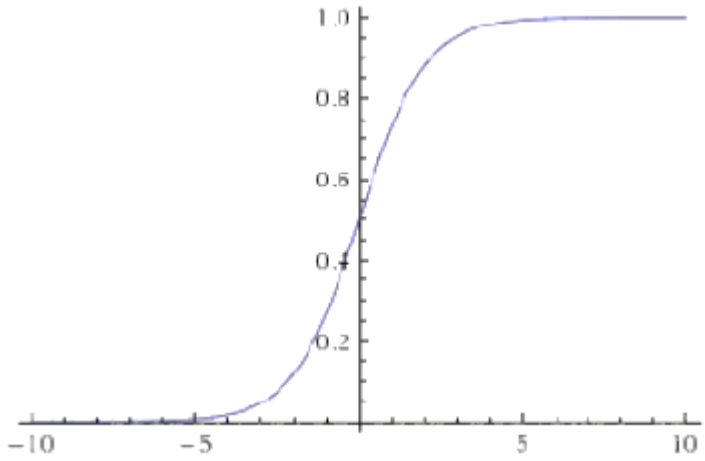
近些年 Relu 函数及其改进型（如 Leaky-ReLU、P-ReLU、R-ReLU 等）在多层神经网络中应用比较多。下面我们来总结下这些激活函数：

Sigmoid 函数

Sigmoid 是常用的非线性的激活函数，它的数学形式如下：

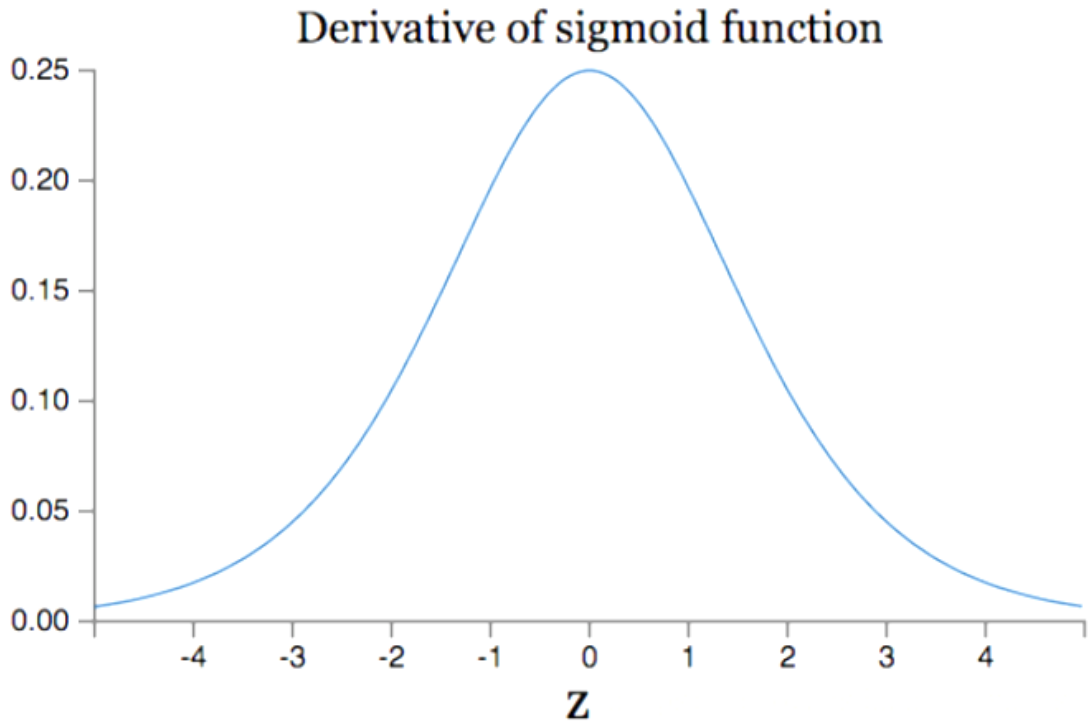
$$f(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid 的几何图像如下：



特点：

它能够把输入的连续实值变换为 0 和 1 之间的输出，特别的，如果是非常大的负数，那么输出就是 0；如果是非常大的正数，输出就是 1.
缺点：
sigmoid 函数曾经被使用的很多，不过近年来，用它的人越来越少了。主要是因为它固有的一些 缺点。
缺点 1：在深度神经网络中梯度反向传递时导致梯度爆炸和梯度消失，其中梯度爆炸发生的概率非常小，而梯度消失发生的概率比较大。首先来看 Sigmoid 函数的导数，如下图所示：

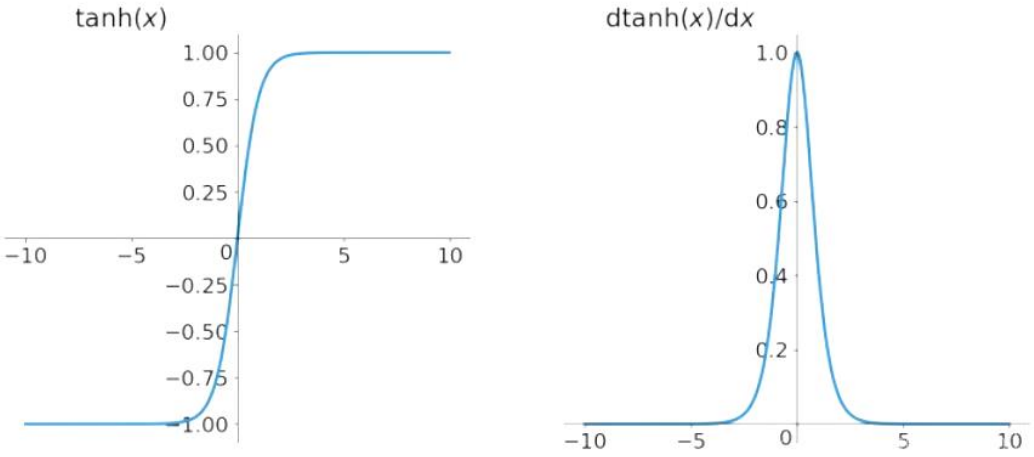


如果我们初始化神经网络的权值为 [0,1][0,1][0,1] 之间的随机值，由反向传播算法的数学推导可知，梯度从后向前传播时，每传递一层梯度值都会减小为原来的 0.25 倍，如果神经网络隐层特别多，那么梯度在穿过多层后将变得非常小接近于 0，即出现梯度消失现象；当网络权值初始化为 (1,+∞)(1,+∞)(1,+∞) 区间内的值，则会出现梯度爆炸情况。
缺点 2：Sigmoid 的 output 不是 0 均值（即 zero-centered）。这是不可取的，因为这会导致后一层的神经元将得到上一层输出的非 0 均值的信号作为输入。产生的一个结果就是：如 x>0, f=wTx+bx>0, \ f= wˆTx+bx>0, f=wTx+b, 那么对 w 求局部梯度则都为正，这样在反向传播的过程中 w 要么都往正方向更新，要么都往负方向更新，导致有一种捆绑的效果，使得收敛缓慢。 当然了，如果按 batch 去训练，那么那个 batch 可能得到不同的信号，所以这个问题还是可以缓解一下的。因此，非 0 均值这个问题虽然会产生一些不好的影响，不过跟上面提到的梯度消失问题相比还是要好很多的。
缺点 3：其解析式中含有幂运算，计算机求解时相对来讲比较耗时。对于规模比较大的深度网络，这会较大地增加训练时间。

tanh 函数
tanh 函数解析式：

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

tanh 函数及其导数的几何图像如下图：

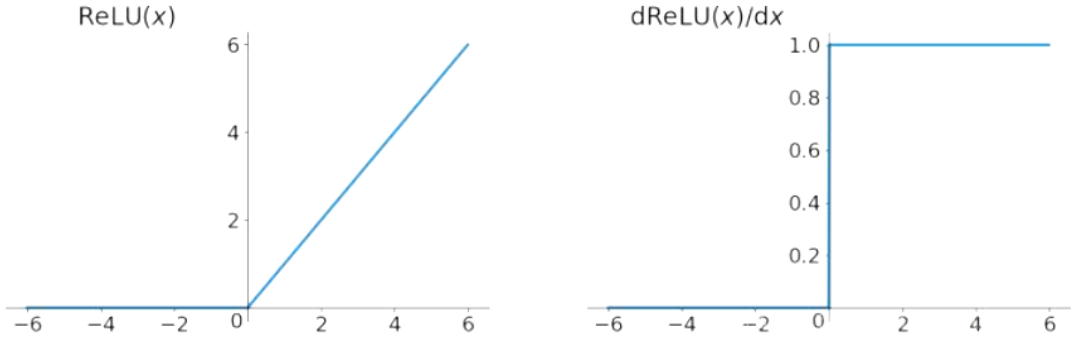


tanh 读作 Hyperbolic Tangent，它解决了 Sigmoid 函数的不是 zero-centered 输出问题，然而，梯度消失（gradient vanishing）的问题和幂运算的问题仍然存在。

Relu 函数
Relu 函数的解析式：

$$Relu = max(0, x)$$

Relu 函数及其导数的图像如下图所示：



ReLU 函数其实就是一个取最大值函数，注意这并不是全区间可导的，但是我们可以取 sub-gradient，如上图所示。ReLU 虽然简单，但却是近几年的重要成果，有以下几大优点：
1） 解决了 gradient vanishing 问题 (在正区间)
2）计算速度非常快，只需要判断输入是否大于 0
3）收敛速度远快于 sigmoid 和 tanh

ReLU 也有几个需要特别注意的问题：
1）ReLU 的输出不是 zero-centered
2）Dead ReLU Problem，指的是某些神经元可能永远不会被激活，导致相应的参

数永远不能被更新。有两个主要原因可能导致这种情况产生：(1) 非常不幸的参数初始化，这种情况比较少见 (2) learning rate 太高导致在训练过程中参数更新太大，不幸使网络进入这种状态。解决方法是可以采用 Xavier 初始化方法，以及避免将 learning rate 设置太大或使用 adagrad 等自动调节 learning rate 的算法。

尽管存在这两个问题，ReLU 目前仍是最常用的 activation function

损失函数：

常用的损失函数有均方误差：MSE

$$loss = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\frac{\partial loss}{\partial y} = 2\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

是最经典也是最简单的损失函数，几乎万能，但是不太准确。

binary_crossentropy交叉熵损失函数，一般用于二分类：

$$loss = -\sum_{i=1}^{n}\hat{y}_i log y_i + (1-\hat{y}_i)log(1-\hat{y}_i)$$

$$\frac{\partial loss}{\partial y} = -\sum_{i=1}^{n}\frac{\hat{y}_i}{y_i} - \frac{1-\hat{y}_i}{1-y_i}$$

这个是针对概率之间的损失函数，你会发现只有y_i和$\hat{y}_i$是相等时，loss才为0，否则loss就是为一个正数。而且，概率相差越大，loss就越大。这个神奇的度量概率距离的方式称为交叉熵。

categorical_crossentropy分类交叉熵函数：

$$loss = -\sum_{i=1}^{n}\hat{y}_{i1} log y_{i1} + \hat{y}_{i2} log y_{i2} + \cdots + \hat{y}_{im} log y_{im}$$

n是样本数，m是分类数，注意，这是一个多输出的loss的函数
，所以它的loss计算也是多个的。

$$\frac{\partial loss}{\partial y_{i1}} = -\sum_{i=1}^{n}\frac{\hat{y}_{i1}}{y_{i1}}$$

$$\frac{\partial loss}{\partial y_{i2}} = -\sum_{i=1}^{n}\frac{\hat{y}_{i2}}{y_{i2}}$$

$$\cdots$$

$$\frac{\partial loss}{\partial y_{im}} = -\sum_{i=1}^{n}\frac{\hat{y}_{im}}{y_{im}}$$