模式识别编程实验报告

专业班级： 未来技术 2104 班

学　号： U202115058

姓　名： 陶鸿远

报告日期： 2023 年 11 月 17 日

注：本实验所有代码可见

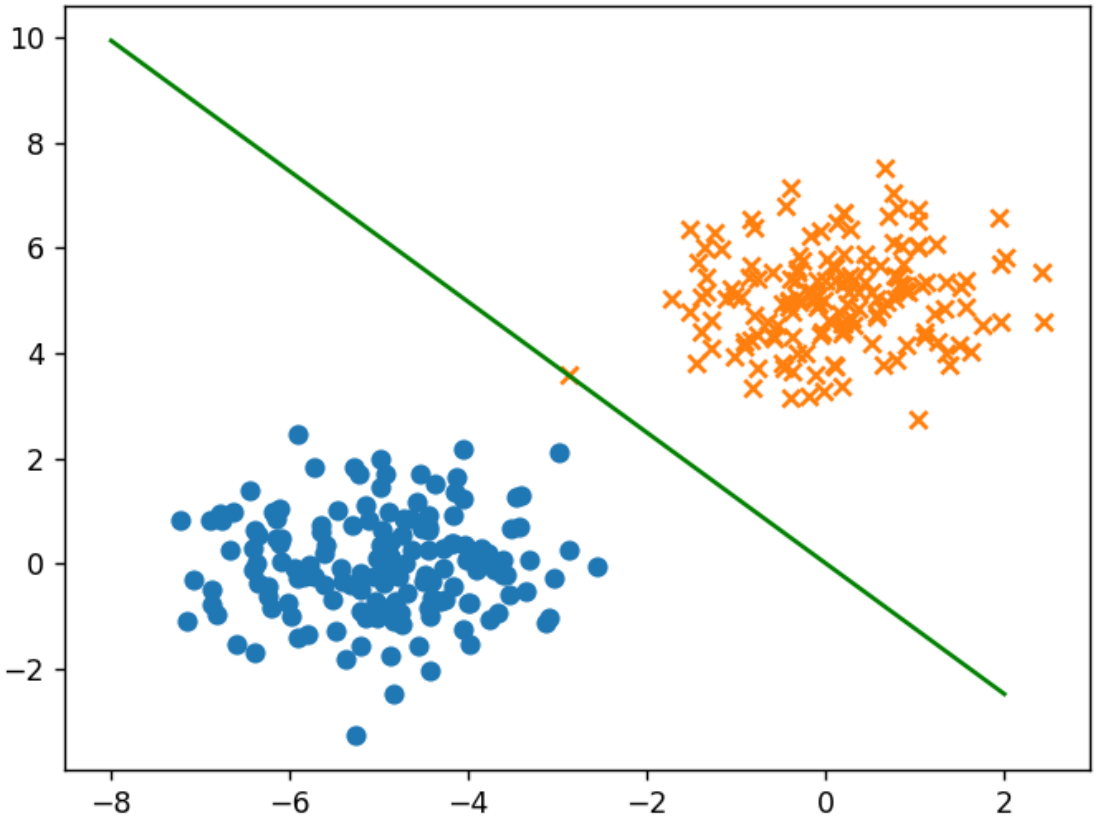
https://github.com/Hongyuan-Tao/Pattern-Recognition-And-Machine-Learning

Lecture2（感知机算法）：

1、如程序PLAandPocket.py中PLA和Pocket函数所示

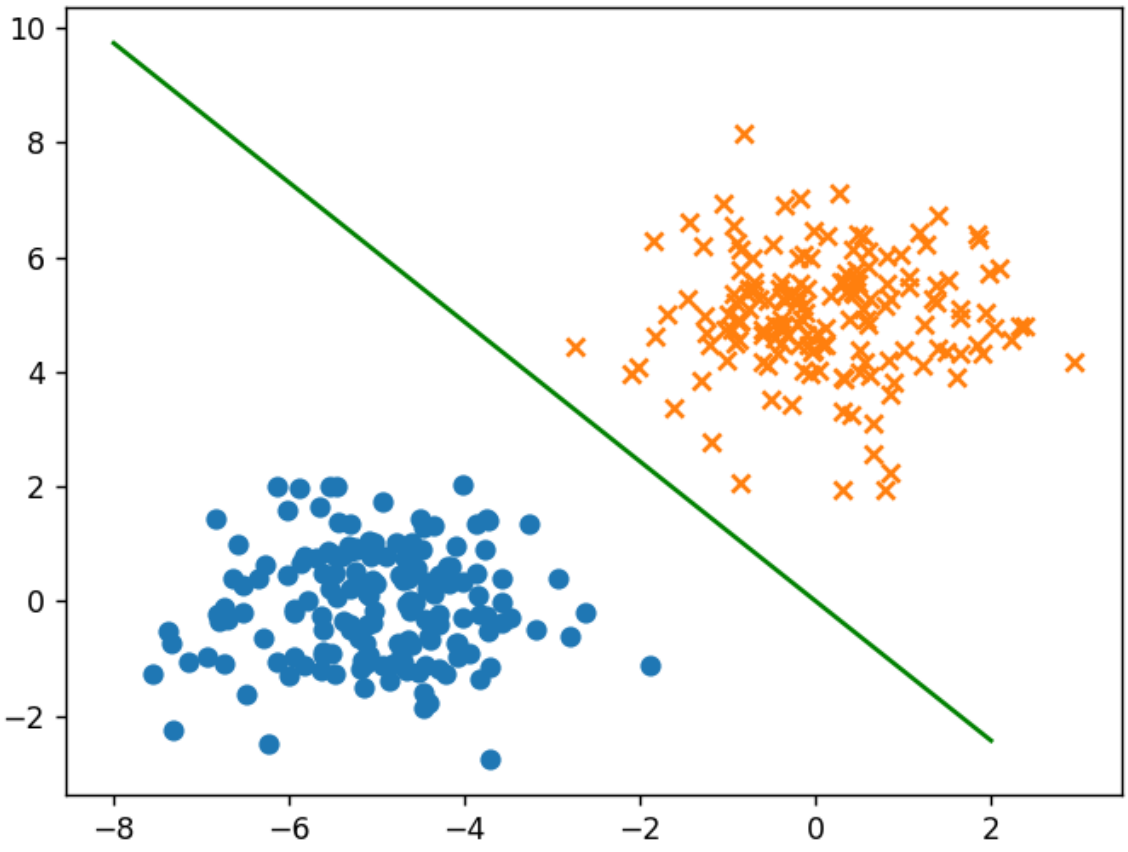
2、PLA:

```
生成数据集: [[-6.38993151 -1.69156339  1.        ]
 [-3.54371144 -0.5830198   1.        ]
 [-4.26765222 -0.69521965  1.        ]
 ...
 [-0.58339504  4.26962143 -1.        ]
 [ 0.26893695  3.86227642 -1.        ]
 [-0.83921265  4.23696374 -1.        ]]
W: [-6.2021366  -4.99212743]
b: 0.0
count 2
0.002959012985229492 s
训练集分类准确率: 1.0
测试集分类准确率: 1.0
```



Pocket:

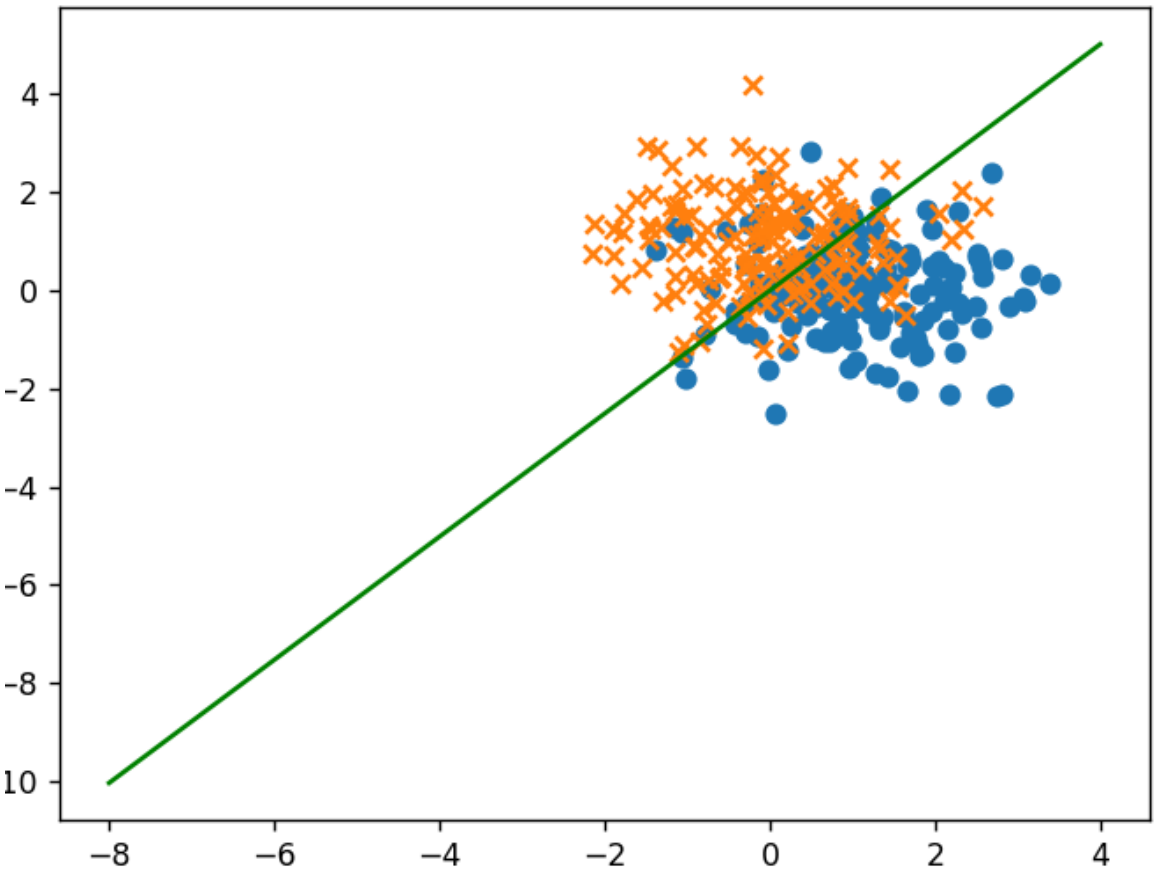
```
生成数据集: [[-5.48511596  0.49605235  1.        ]
 [-4.6405052   0.39200793  1.        ]
 [-5.30438422  0.85997873  1.        ]
 ...
 [ 1.06931431  5.48861631 -1.        ]
 [ 2.06875026  4.6722137  -1.        ]
 [ 0.31130234  1.93929649 -1.        ]]
W: [ 1.41142393 -3.79126822]
b: -1.0
0.0059697628021240234 s
训练集分类准确率: 1.0
测试集分类准确率: 1.0
```



3、PLA算法无法得出结果

Pocket:

```
生成数据集: [[ 0.68920654  0.28215435  1.        ]
 [ 0.99576377  1.49270854  1.        ]
 [ 1.61096564 -0.55377333  1.        ]
 ...
 [-1.15125378  0.78527363 -1.        ]
 [-0.01157509 -0.64601332 -1.        ]
 [ 0.49904407  1.76670186 -1.        ]]
W: [ 0.90811003 -0.63114889]
b: 0.0
0.22539782524108887 s
训练集分类准确率: 0.775
测试集分类准确率: 0.7875
```



4、过多增加pocket算法的迭代次数无法提升性能

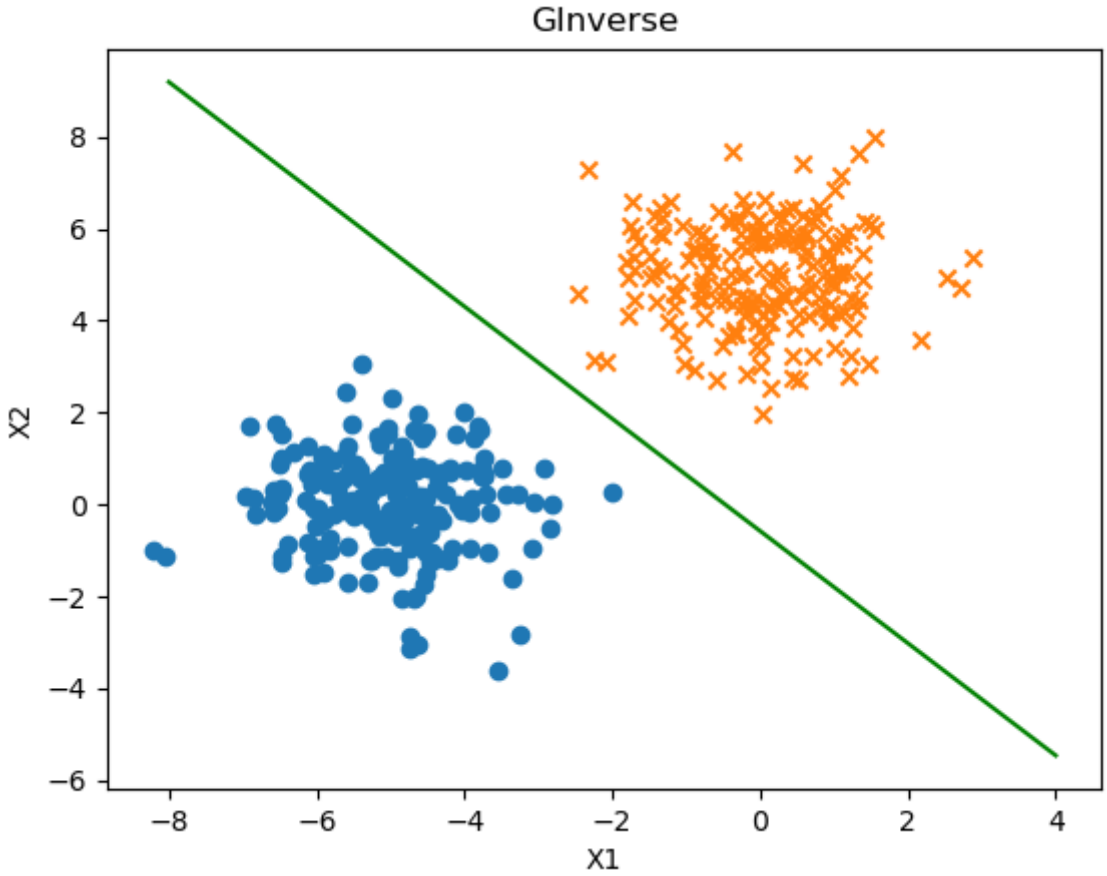
x1与x2的均值越分开，同时其协方差越小，其可分性越好

Lecture3（线性回归）：

1、见代码文件GradientDescent.py中的函数GInverse()与GD()

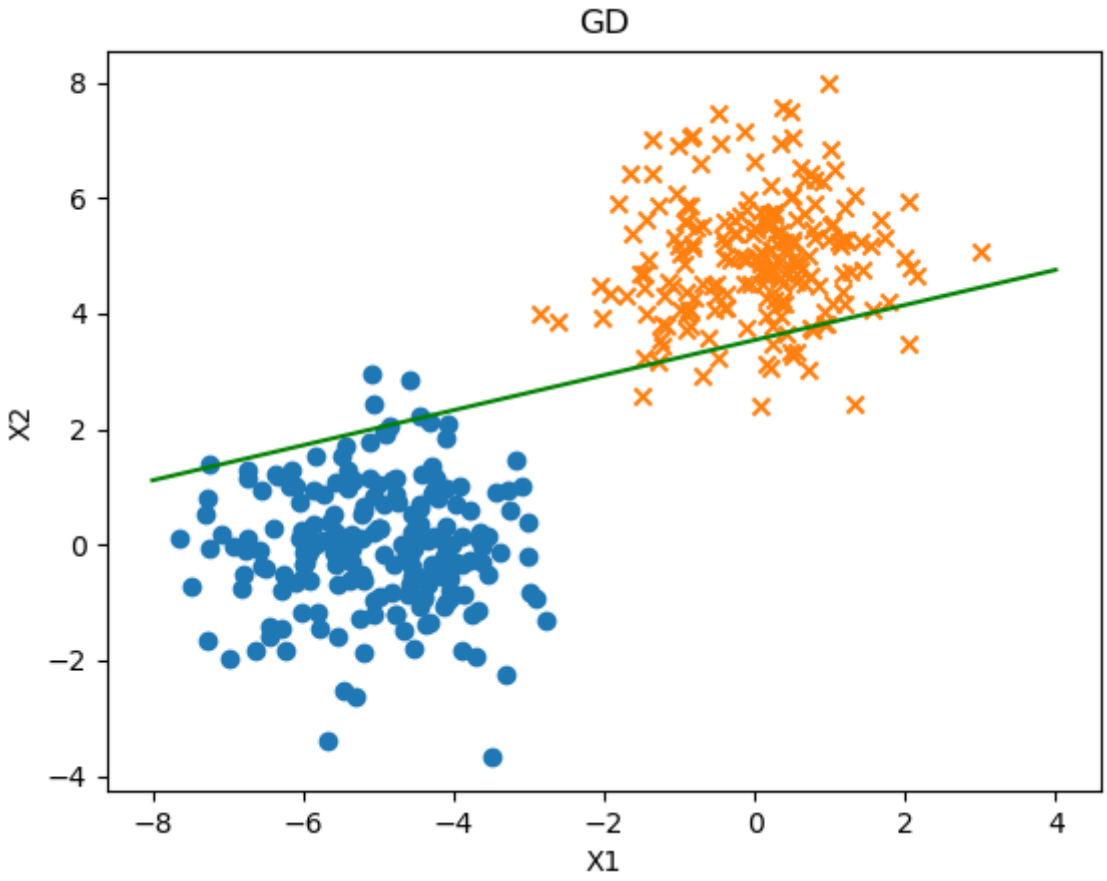
2、广义逆法：

```
生成数据集: [[-6.95675602  0.16948929  1.
 [-5.25301305  0.49213068  1.        ]
 [-3.76502649  0.81173035  1.        ]
 ...
 [ 2.51177111  4.94749949 -1.        ]
 [-0.0268919   5.67736544 -1.        ]
 [ 0.94748708  4.9454531  -1.        ]]
[-0.20277849 -0.16615728 -0.09575341]
耗时: 0.0020301342010498047 s
训练集分类准确率: 1.0
测试集分类准确率: 1.0
```

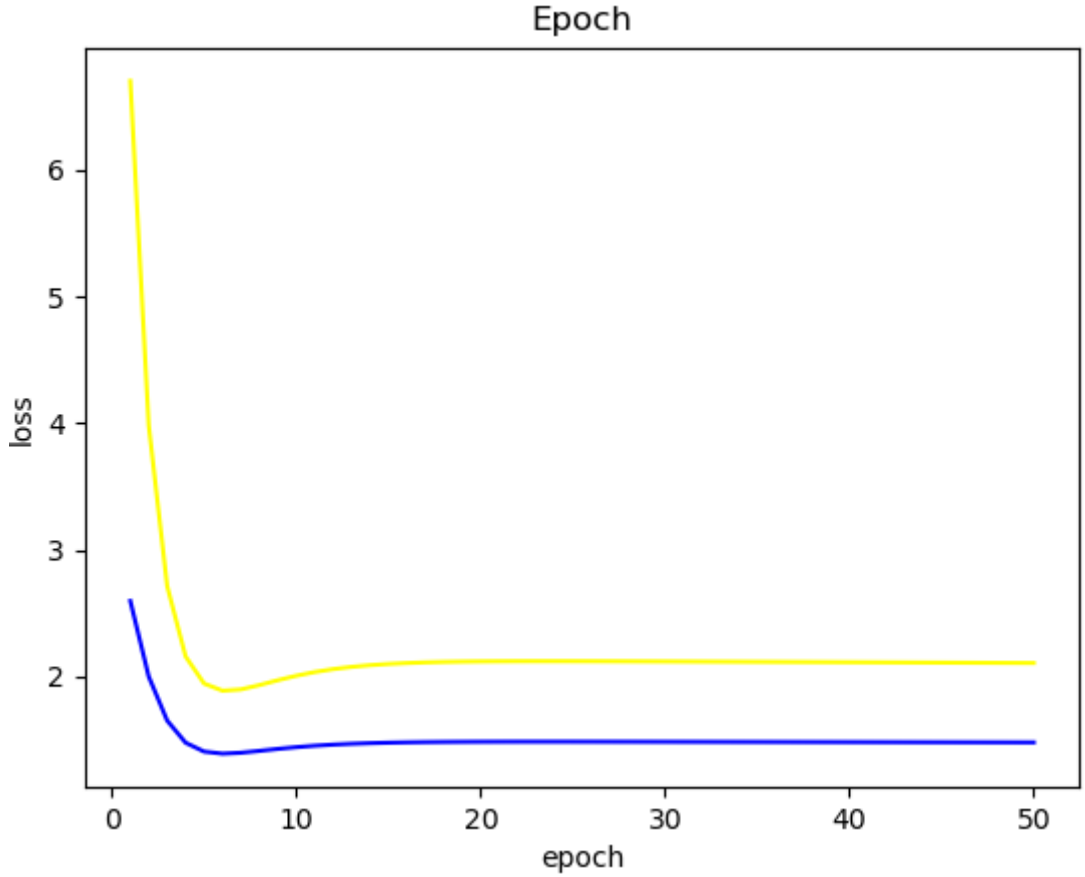


梯度下降法：epoch=50时:

```
W: [ 0.16197141 -0.53337153  1.88999169]
训练集损失函数: 1.487914145613151
测试集损失函数: 1.4839018244978677
耗时: 1.129492998123169 s
训练集分类准确率: 0.940625
测试集分类准确率: 0.9625
```



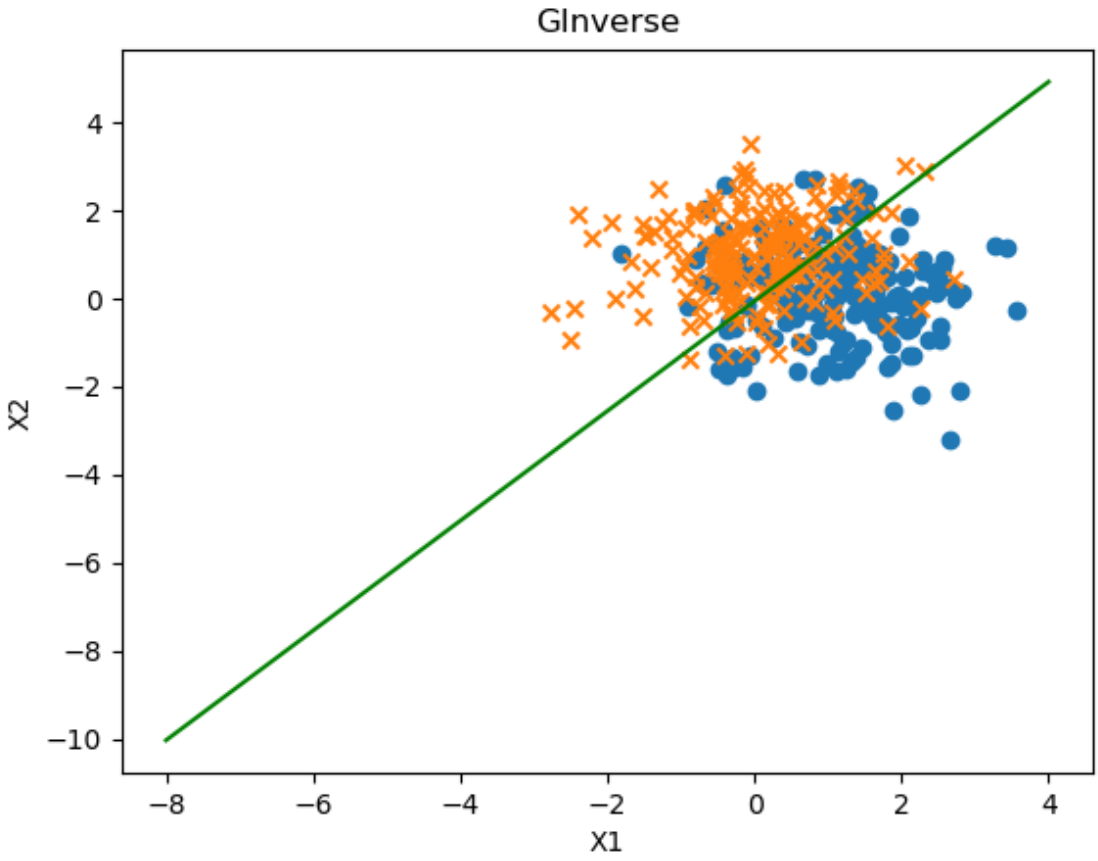
训练集损失与测试集损失随epoch的变化曲线:



![](C:\Users\DELL\AppData\Roaming\Typora\typora-user-images\image-20231003021803947.png![image-20231008130558746](C:\Users\DELL\AppData\Roaming\Typora\typora-user-images\image-20231008130558746.png

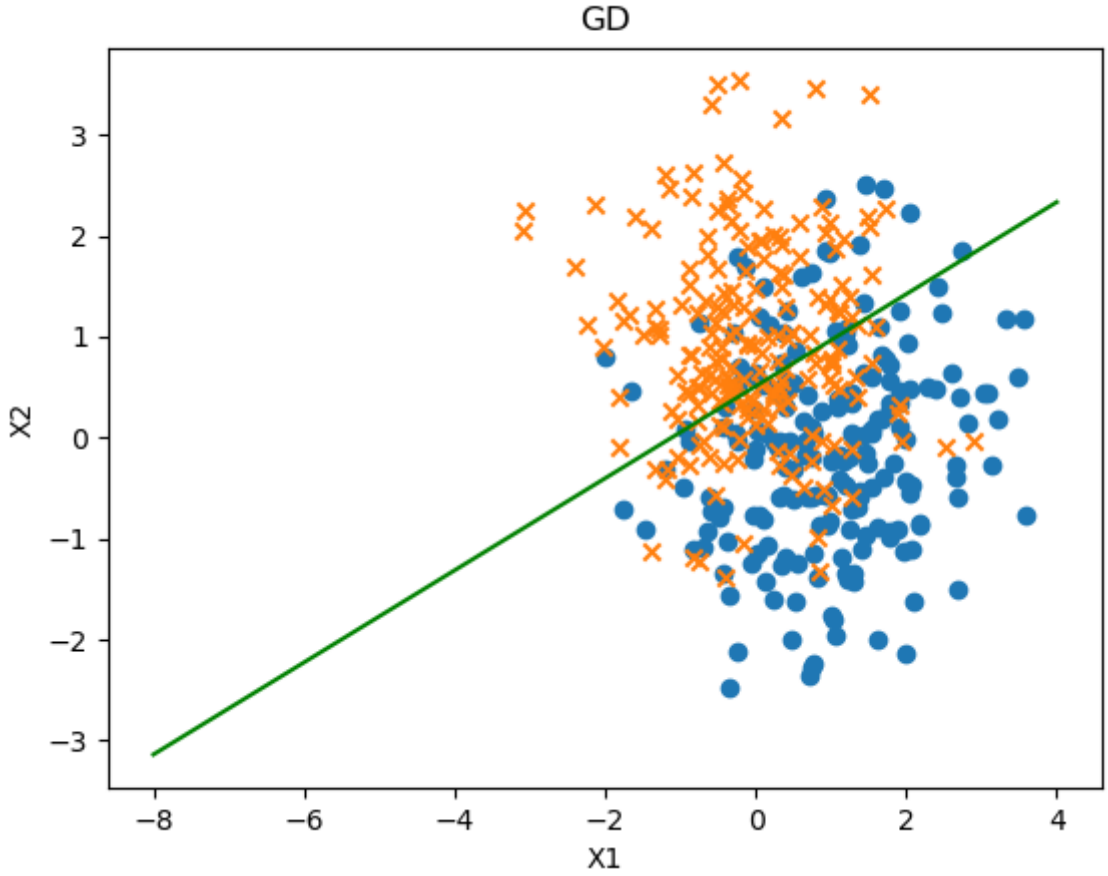
3、广义逆法：

```
生成数据集: [[ 0.70426024 -1.06137361  1.        ]
 [ 1.62252998  0.04296114  1.        ]
 [ 2.80836003  0.12391411  1.        ]
 ...
 [-0.52842288  1.23118205 -1.        ]
 [ 0.37686012  0.21420823 -1.        ]
 [-1.6913932   0.85207638 -1.        ]]
[ 0.37733183 -0.30260107 -0.01837985]
耗时: 0.002002716064453125 s
训练集分类准确率: 0.76875
测试集分类准确率: 0.75
```

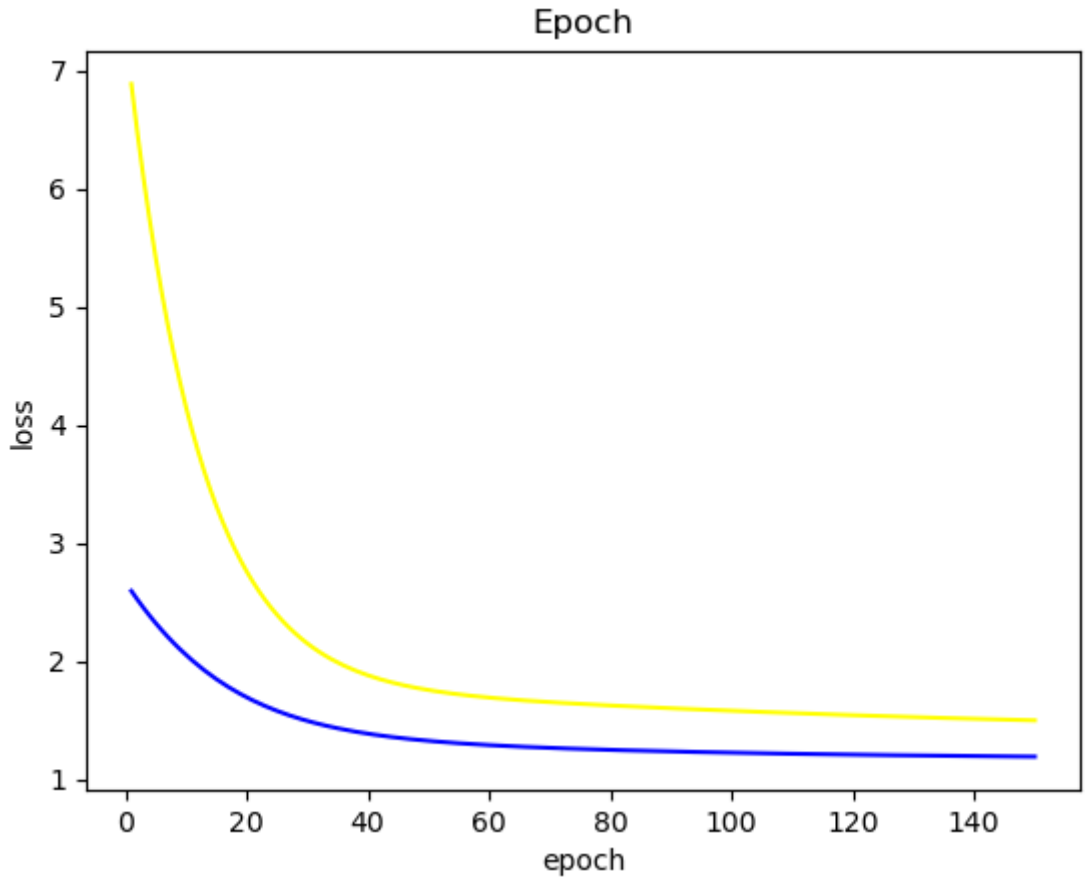


梯度下降法(epoch=150):

```
W: [ 0.20805413 -0.45635146  0.23216762]
训练集损失函数: 1.1819619336373846
测试集损失函数: 1.1432849368471796
耗时: 10.597349405288696 s
训练集分类准确率: 0.7375
测试集分类准确率: 0.7625
```

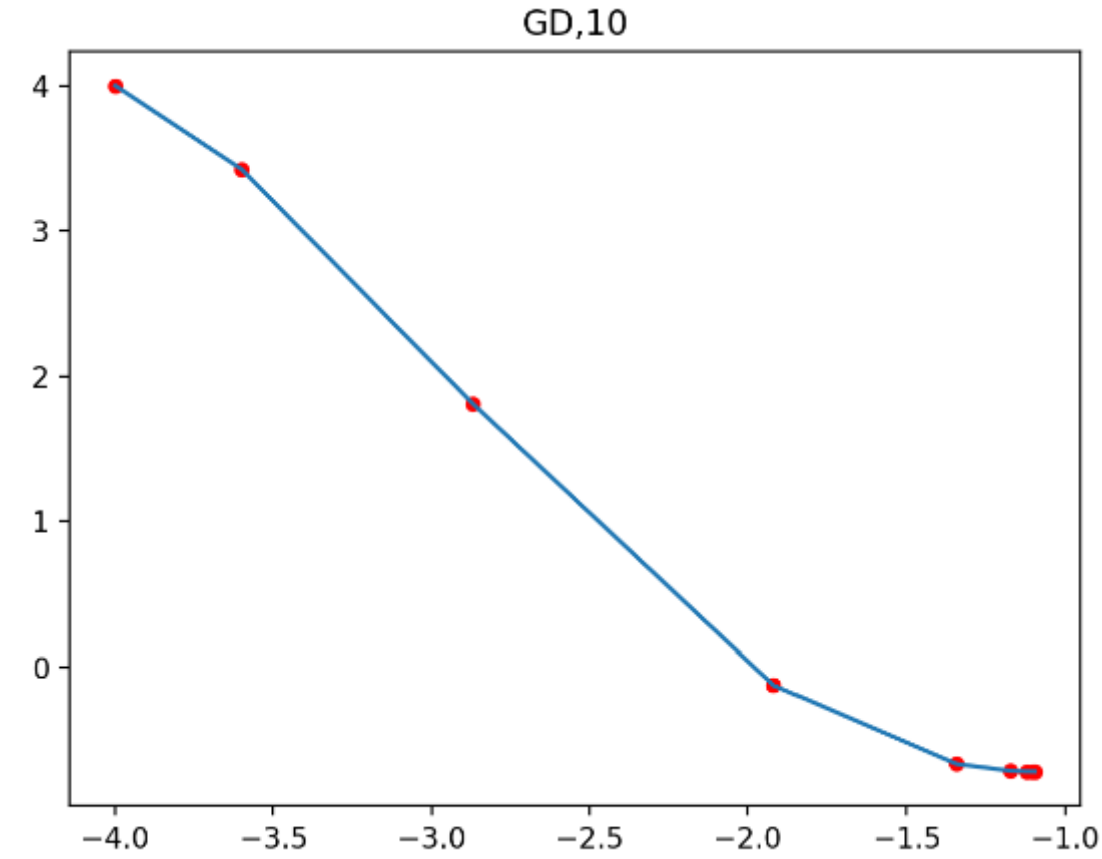


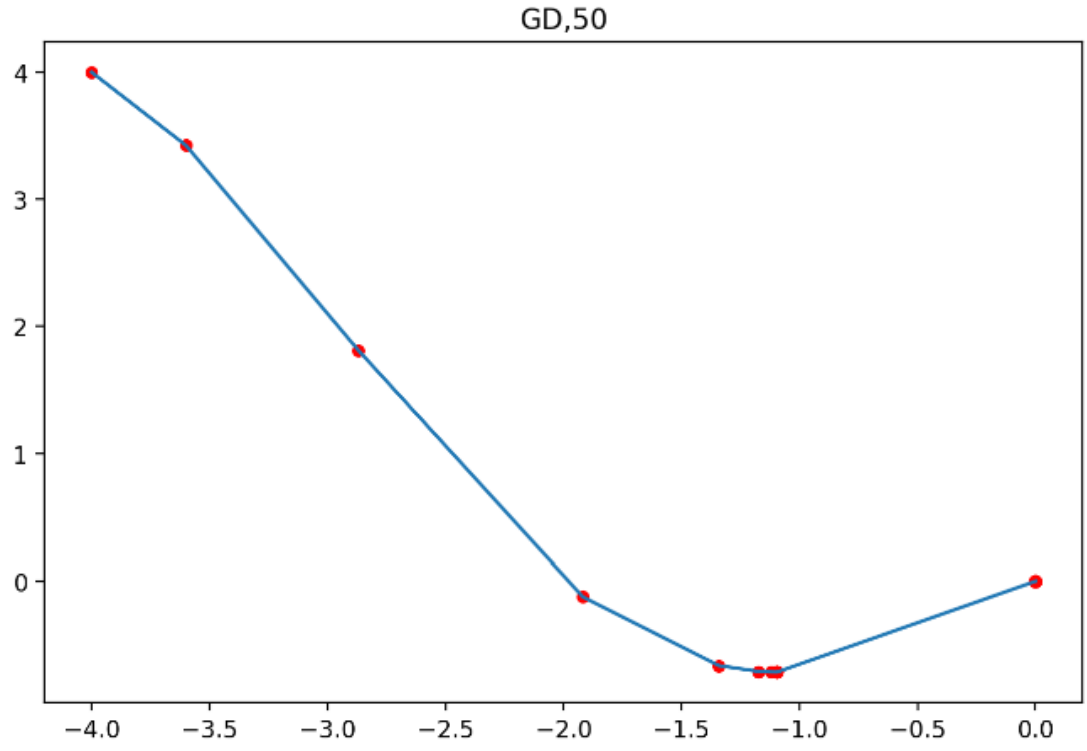
训练集损失与测试集损失随epoch的变化曲线: ![image-20231003022247548](C:\Users\DELL\AppData\Roaming\Typora\typora-user-images\image-20231003022247548.png



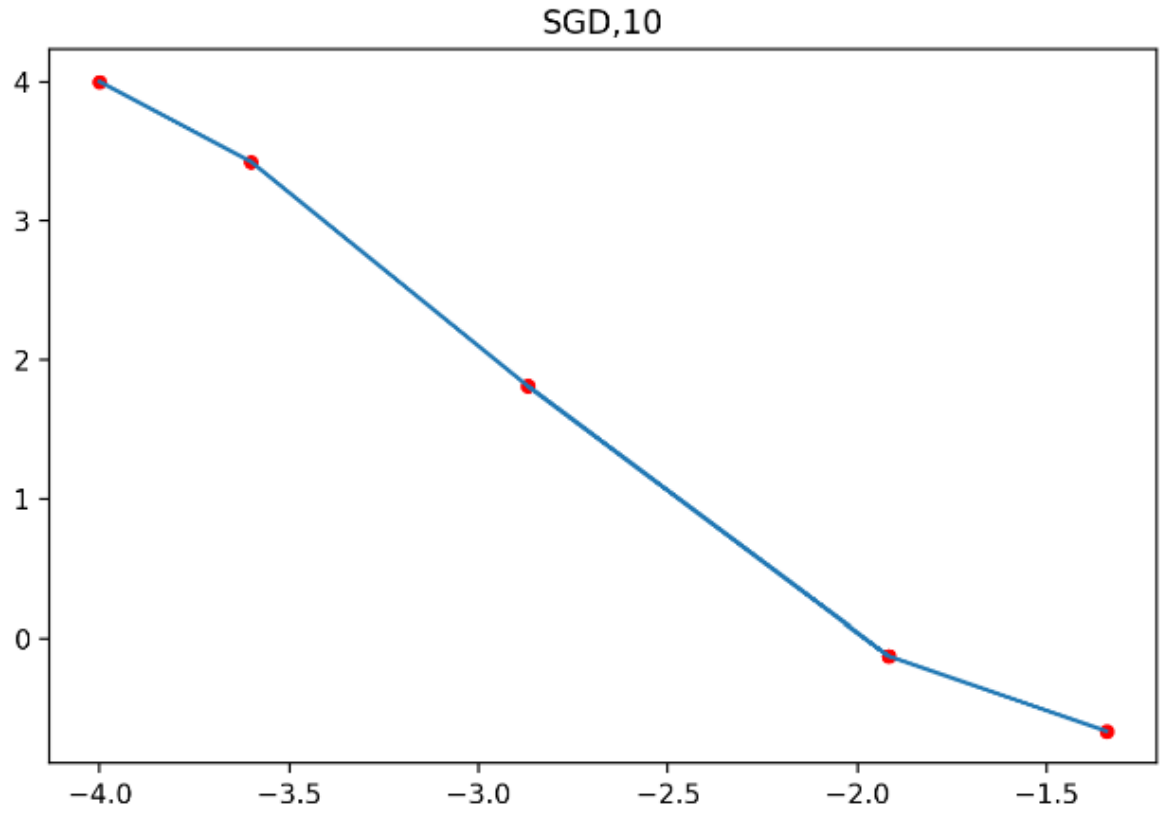
5、不同算法的变化结果：

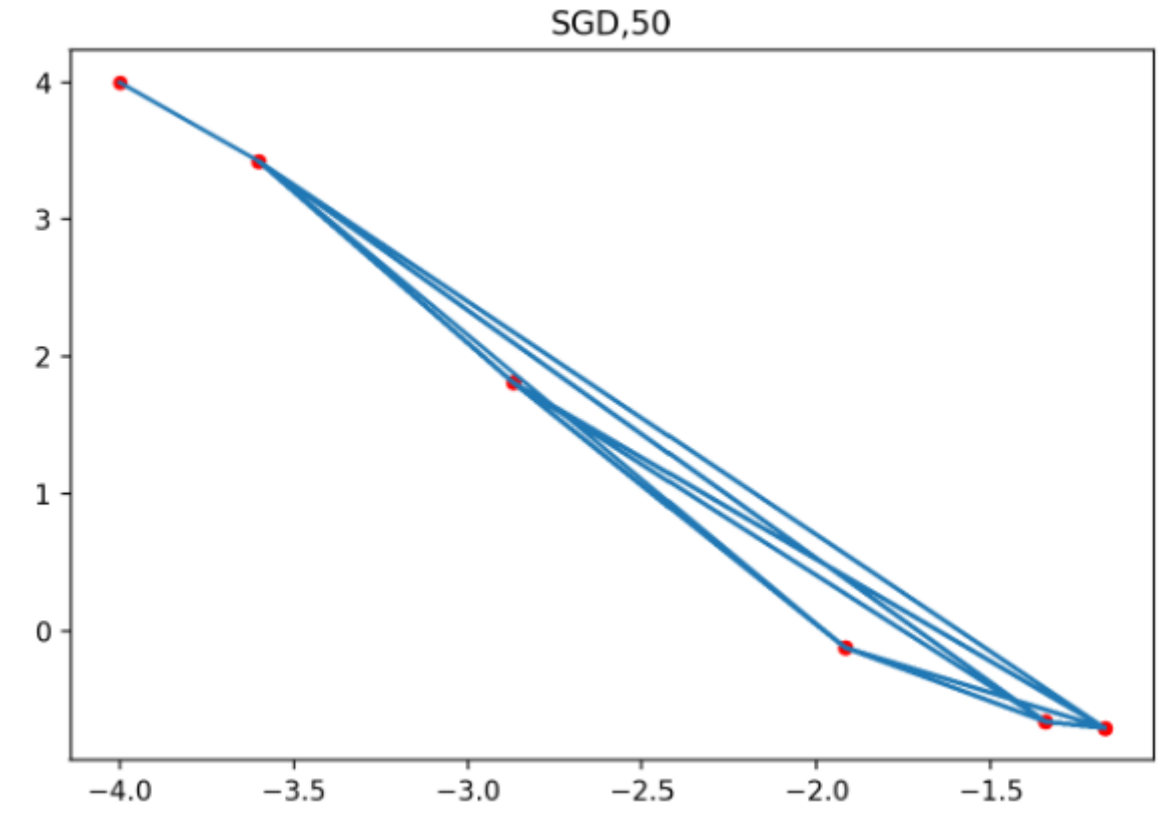
梯度下降法：


GD,10
4
3
2
1
0
−4.0
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0

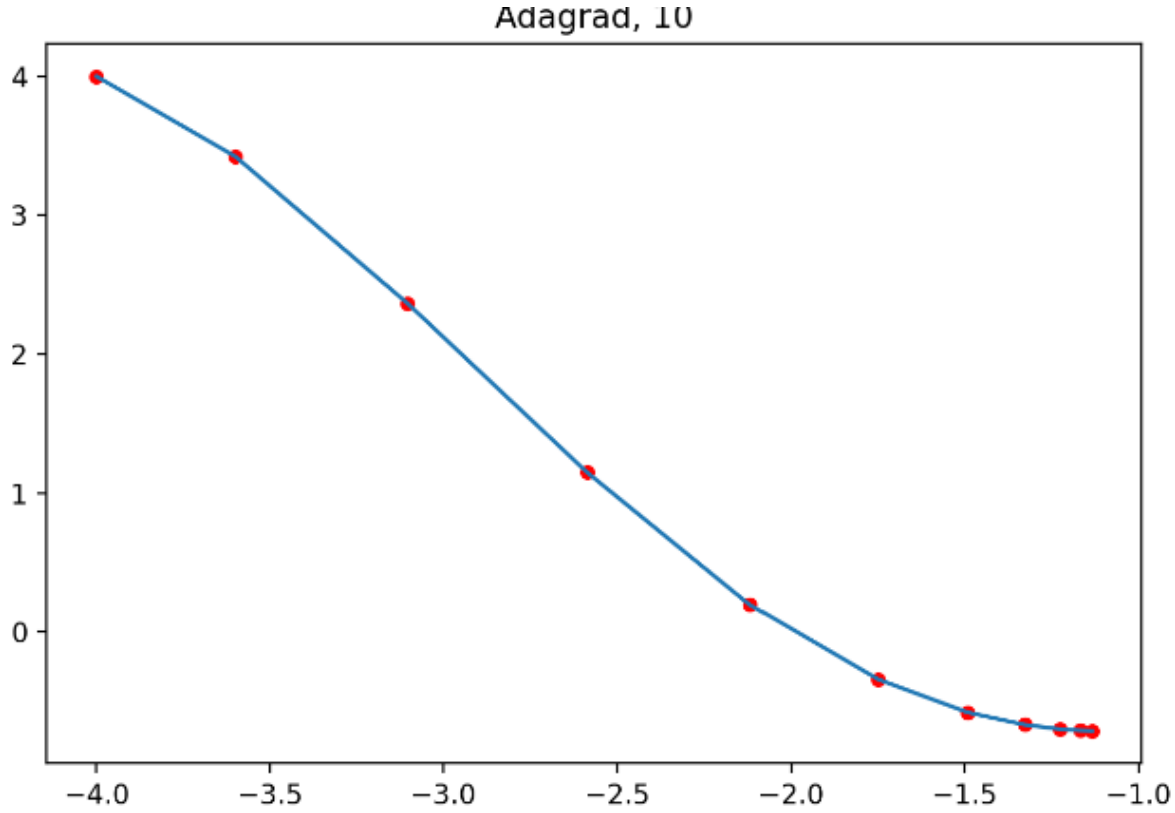

GD,50
4
3
2
1
0
−4.0
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0

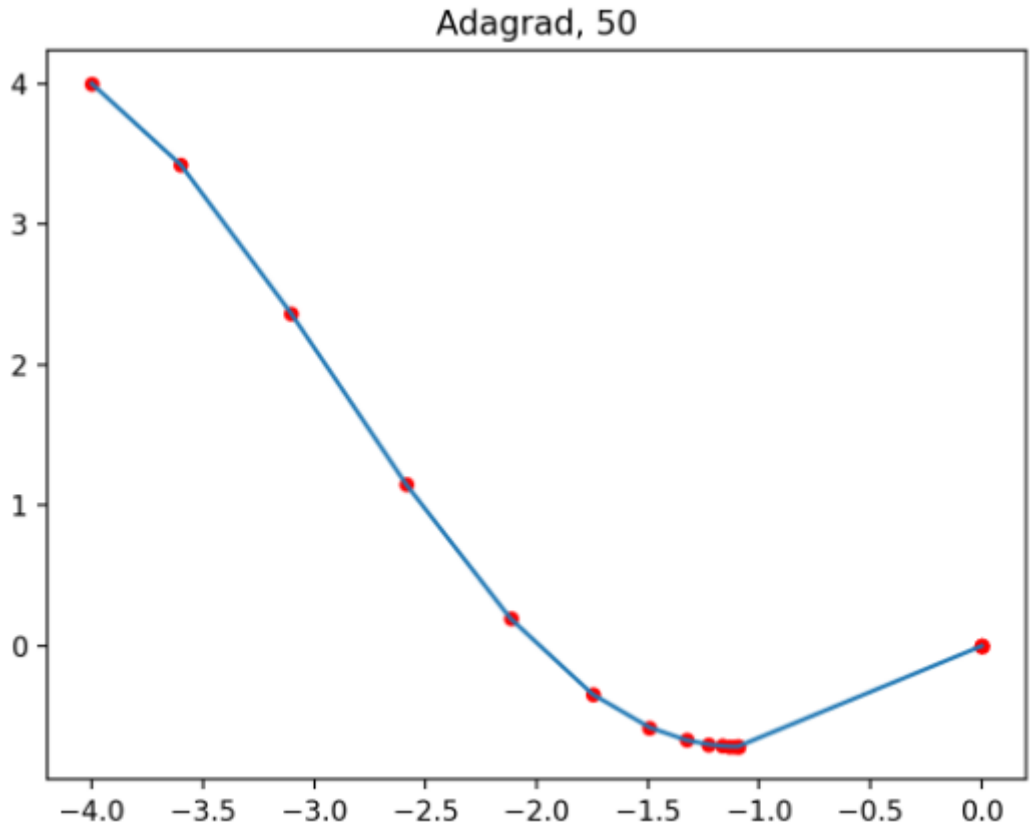
随机梯度下降法：



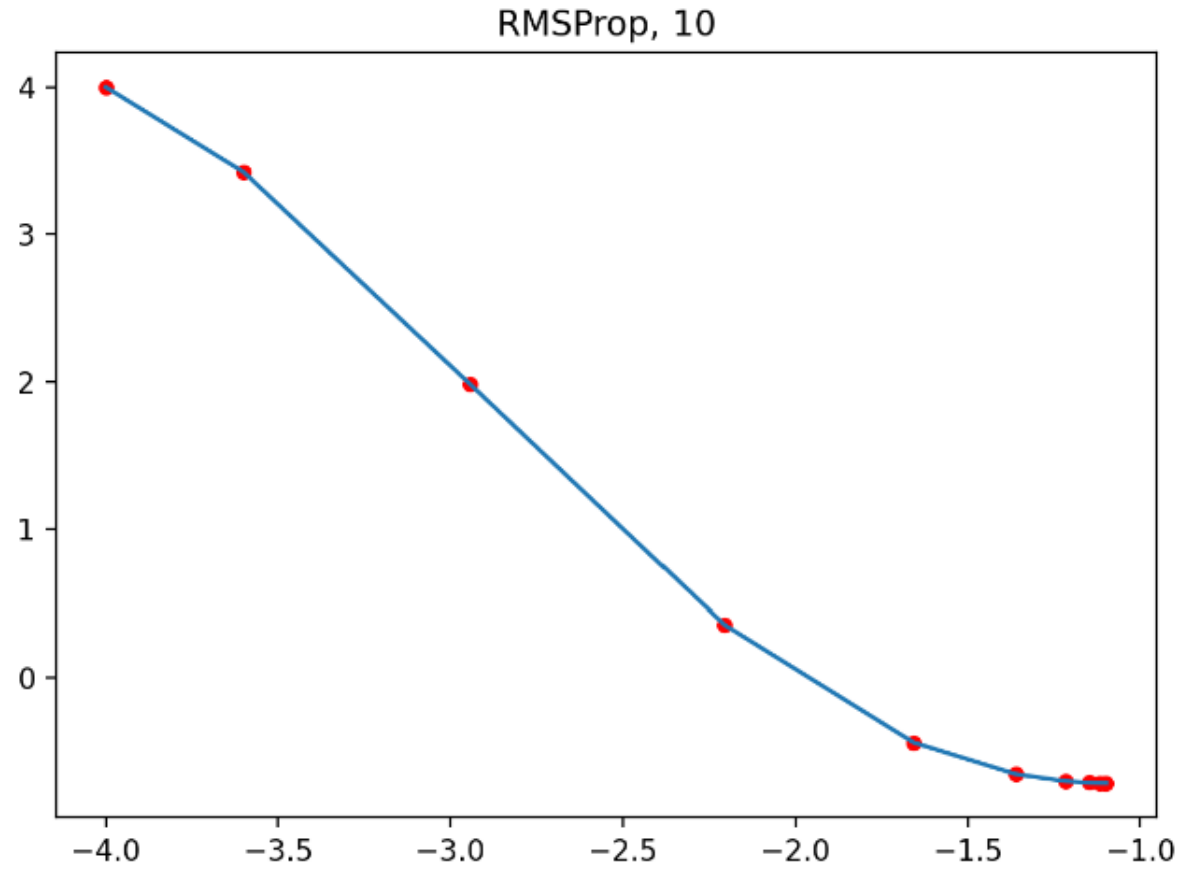


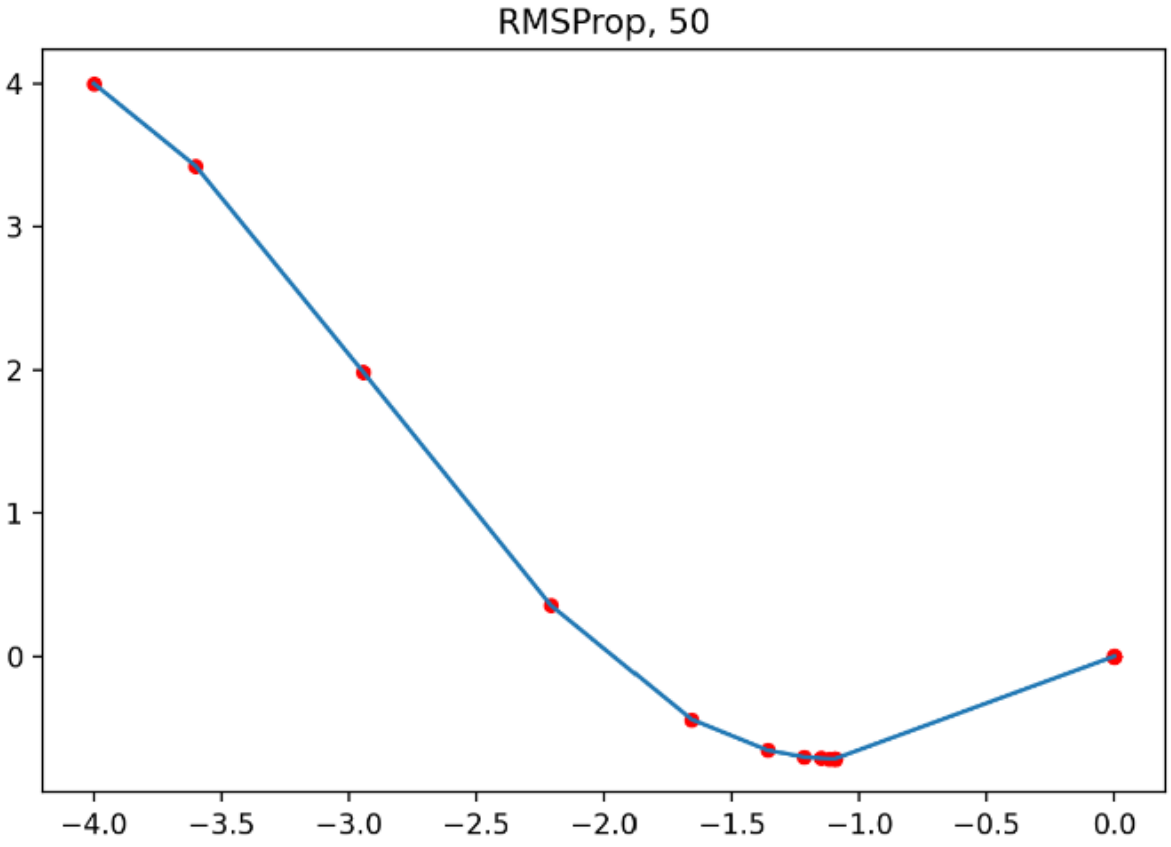
Adagrad:


Adagrad, 10
4
3
2
1
0
−4.0
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0

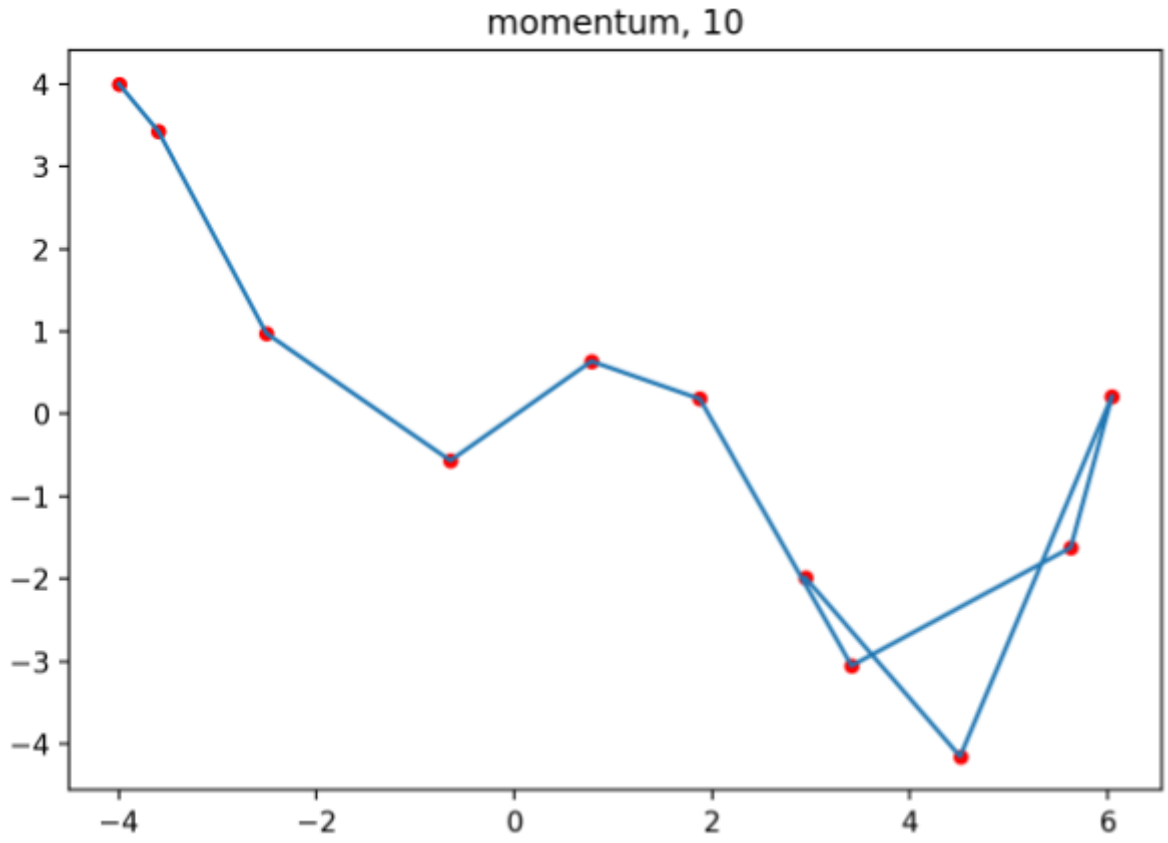

Adagrad, 50
4
3
2
1
0
−4.0
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0

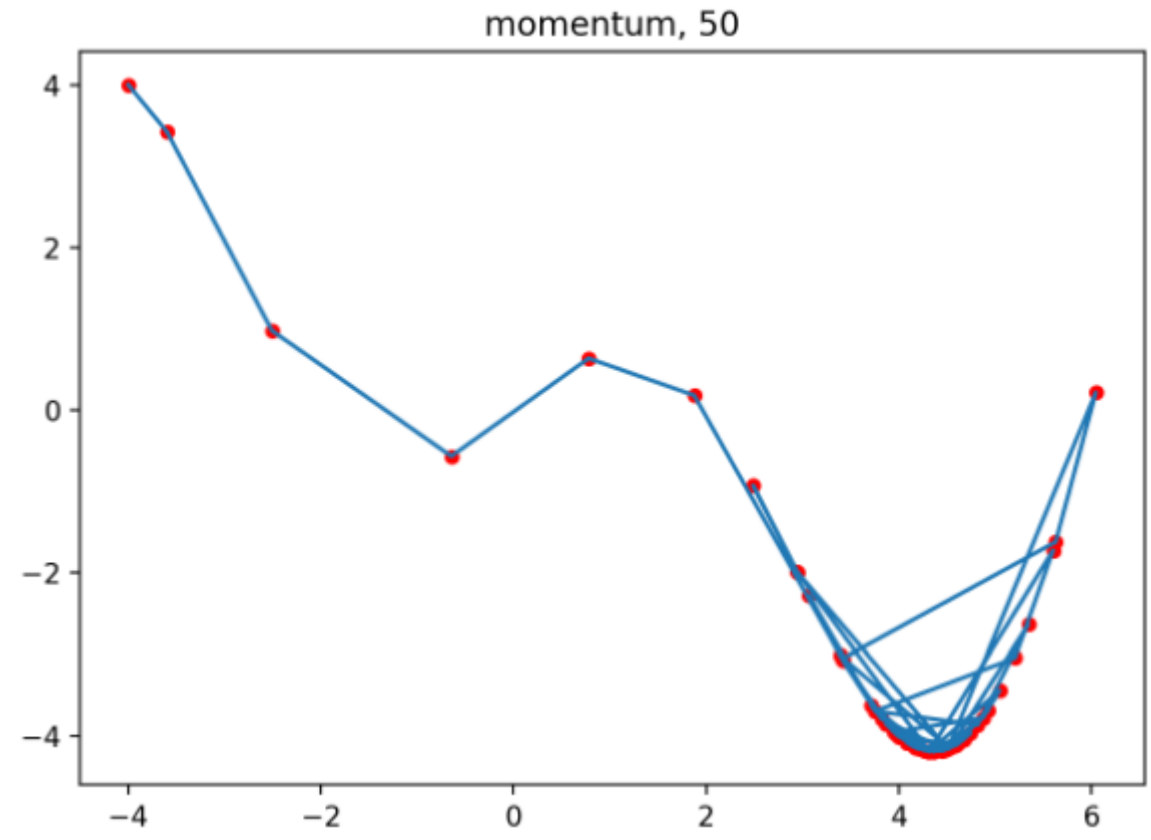
RMSProp:



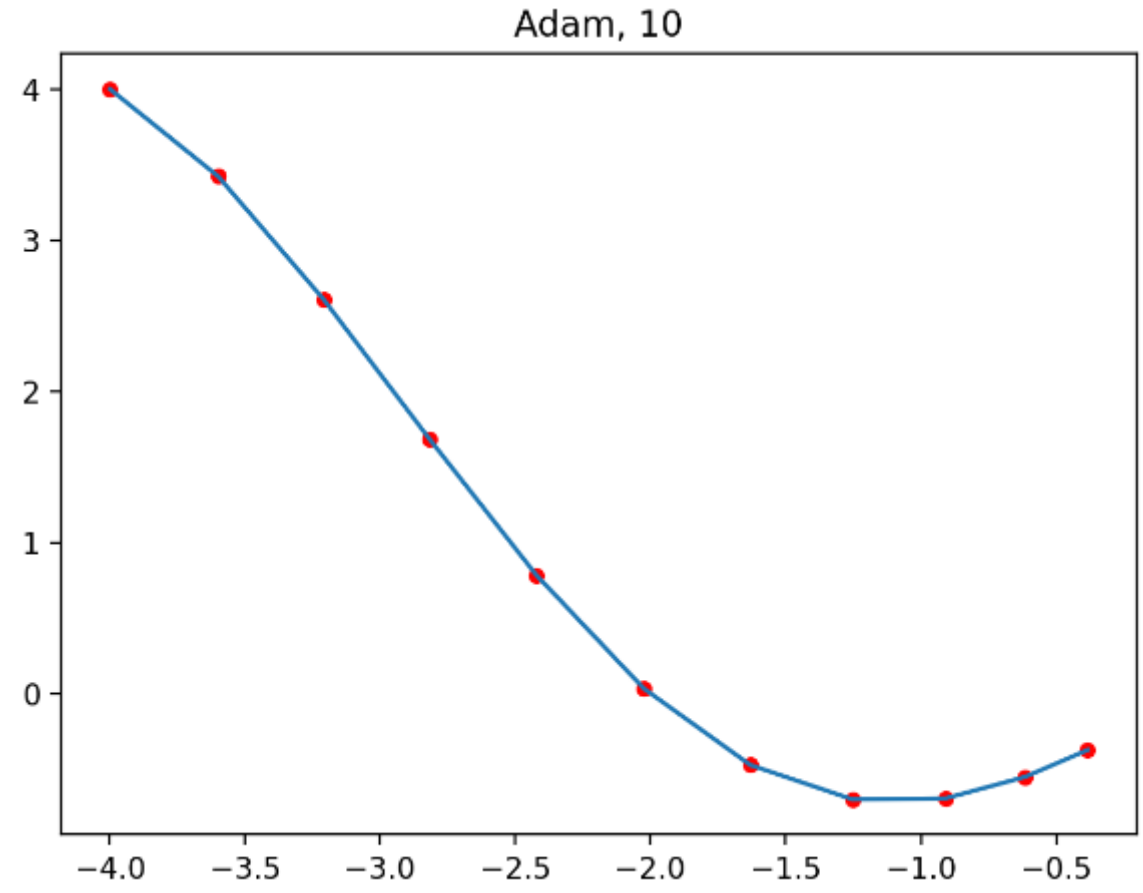


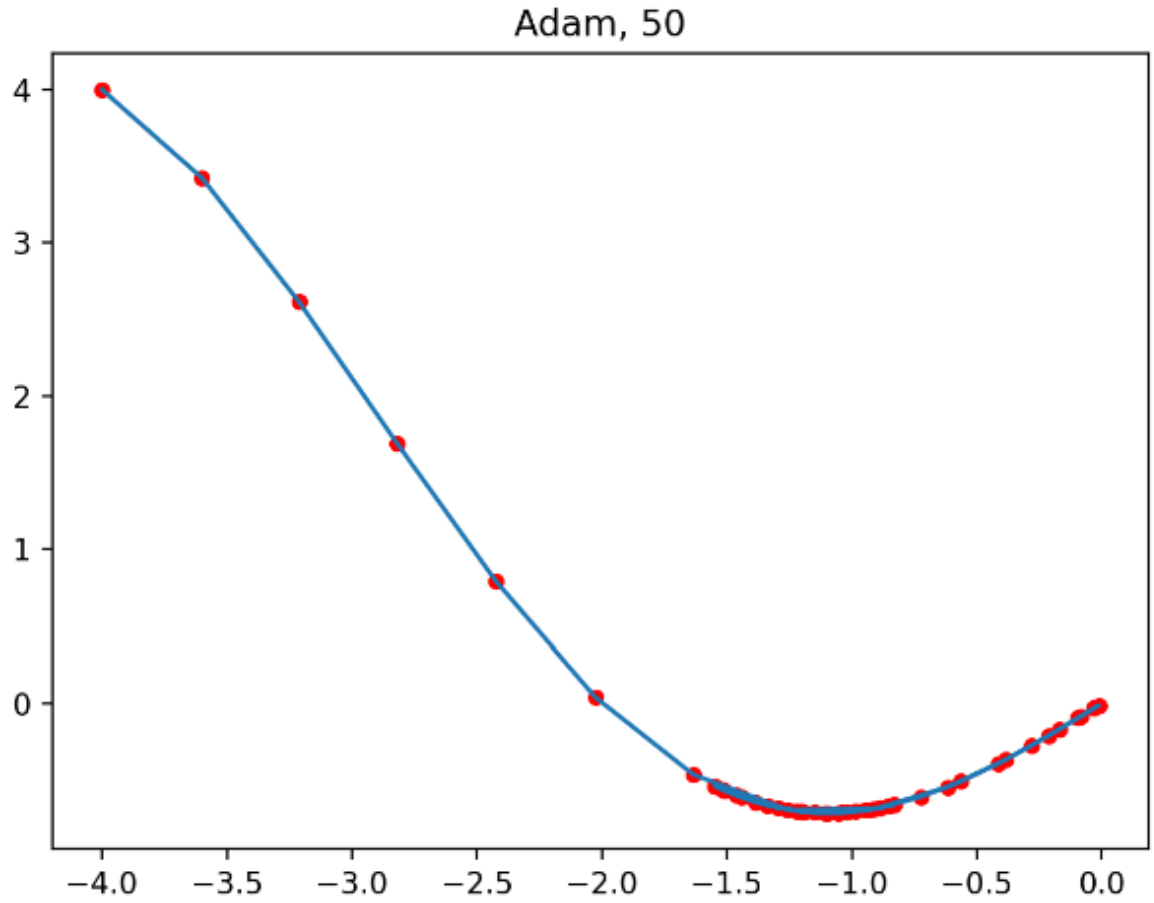
动量法（Momentum）：





Adam:


Adam, 10
4
3
2
1
0
−4.0
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5

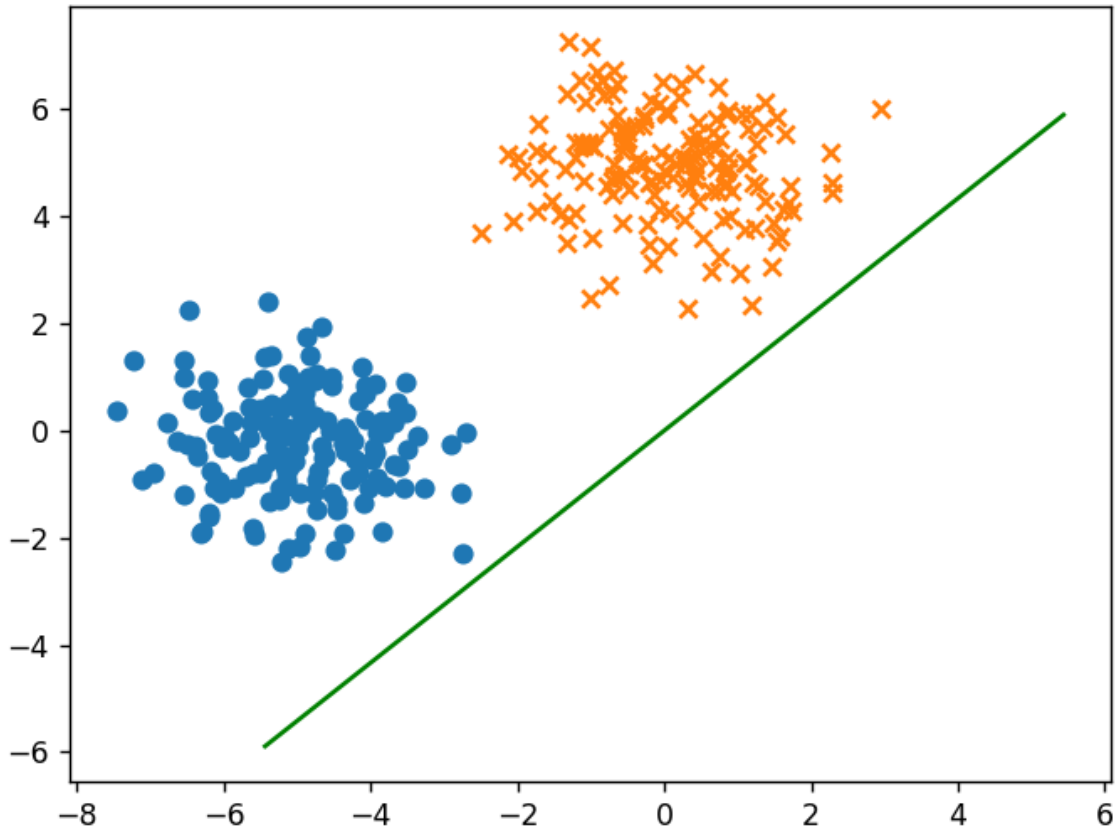

Adam, 50
4
3
2
1
0
−4.0
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0

Lecture4-1（Fisher线性判别）：

1、见代码文件Fisher.py中的函数fisher()与predict()

2、

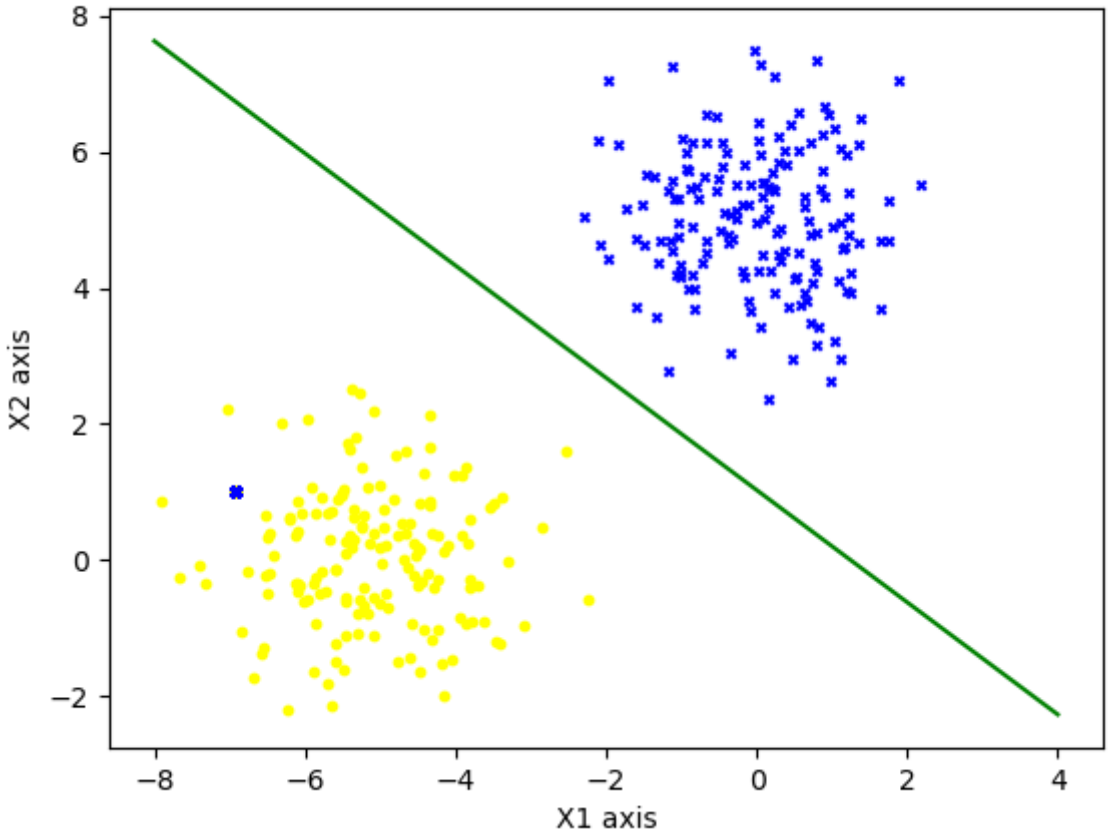
```
生成数据集: [[-6.55299706  0.99846154  1.        ]
 [-5.13744008 -1.04008932  1.        ]
 [-6.17953585 -0.75385586  1.        ]
 ...
 [-1.99574922  5.10808674 -1.        ]
 [-0.56924375  3.89041239 -1.        ]
 [ 1.13969594  5.64476592 -1.        ]]
类型:  <class 'numpy.ndarray'>
最佳投影向量: [-0.01812006 -0.01964221]
分类阈值: -0.0014052345016553475
在训练集上的分类准确率为: 1.0
在测试集上的分类准确率为: 1.0
```



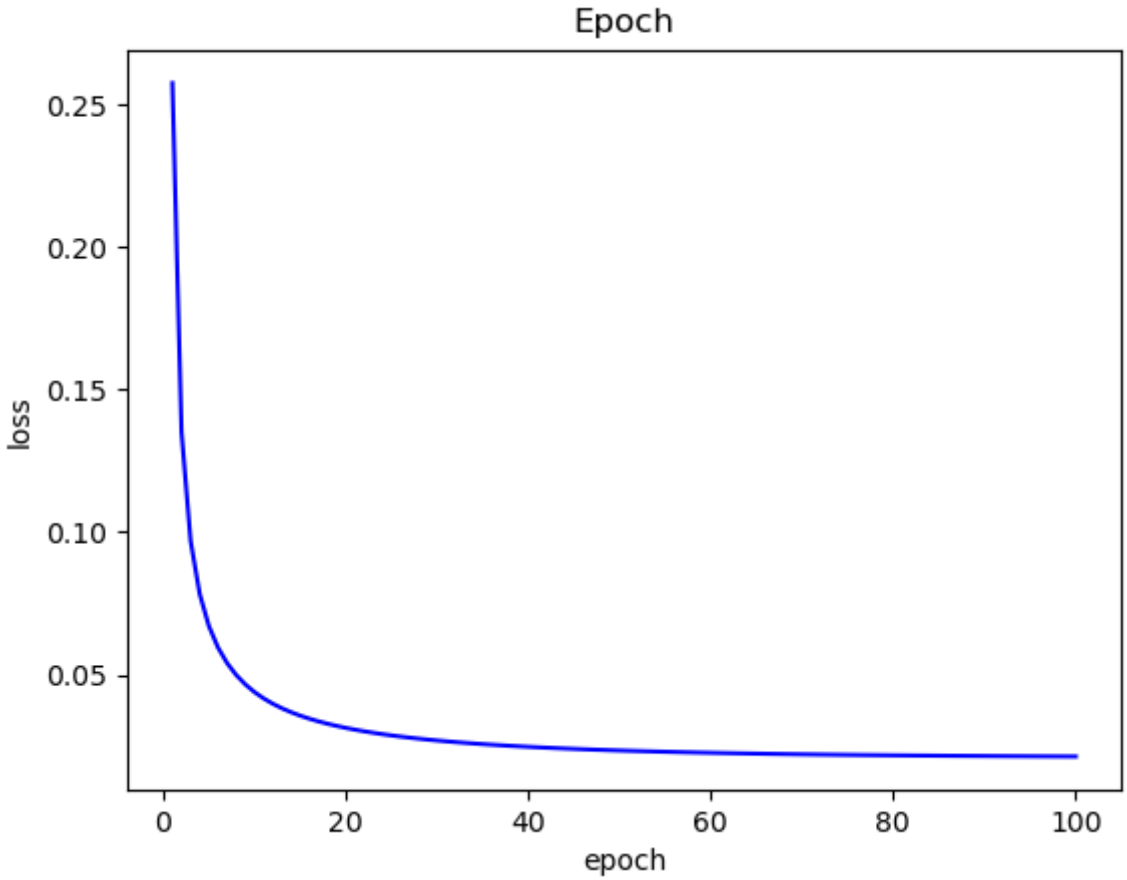
Lecture4-2（逻辑斯蒂回归）：

1、见代码文件Logistic.py内的函数SGD。

2、



```
数据点: -5.082438797608212 0.8334802825538884 真实类别: [1.] 计算类别: 1 概率值: 0.9818114951794473
数据点: -5.772598958776367 0.9458639135274352 真实类别: [1.] 计算类别: 1 概率值: 0.9879196908606043
数据点: -4.257023999337464 0.7755354896695198 真实类别: [1.] 计算类别: 1 概率值: 0.9683941179802275
数据点: -4.980955645009449 0.6037761133675454 真实类别: [1.] 计算类别: 1 概率值: 0.9840348885177581
数据点: -4.635614419537781 -1.3892300205411732 真实类别: [1.] 计算类别: 1 概率值: 0.9965758217447358
数据点: -5.147365034248389 1.2594196946057206 真实类别: [1.] 计算类别: 1 概率值: 0.9746726745519639
数据点: -4.972331827619395 0.21331119992917202 真实类别: [1.] 计算类别: 1 概率值: 0.9886781007032581
数据点: -3.9994636110749804 1.1684312964756562 真实类别: [1.] 计算类别: 1 概率值: 0.9464500325745031
数据点: -4.041931813733918 -1.5072532462904535 真实类别: [1.] 计算类别: 1 概率值: 0.9952056985530027
数据点: -4.010596878491258 -0.1314652790900866 真实类别: [1.] 计算类别: 1 概率值: 0.9830726756585872
数据点: -4.857167670013449 2.4064597628486646 真实类别: [1.] 计算类别: 1 概率值: 0.9160777009890345
数据点: -4.516803399352791 -0.8700105164065465 真实类别: [1.] 计算类别: 1 概率值: 0.9940151377028733
数据点: -7.289197522640739 0.5866740767239247 真实类别: [1.] 计算类别: 1 概率值: 0.9971786917199906
数据点: -5.133941318783918 -1.8031644605371908 真实类别: [1.] 计算类别: 1 概率值: 0.998379380227356
数据点: -4.9523068344412815 -0.08265155009505408 真实类别: [1.] 计算类别: 1 概率值: 0.9911948498835613
数据点: -4.174800435402759 0.8284691827769635 真实类别: [1.] 计算类别: 1 概率值: 0.9648562646275112
数据点: -5.742562488341206 1.3989518545688144 真实类别: [1.] 计算类别: 1 概率值: 0.9814716198233385
数据点: -2.004842103595118 -0.253377893057851 真实类别: [1.] 计算类别: 1 概率值: 0.9351387561177097
数据点: -4.406331497057808 -0.40269302412634433 真实类别: [1.] 计算类别: 1 概率值: 0.9900972898414141
数据点: -4.327822984405973 1.5504172561995564 真实类别: [1.] 计算类别: 1 概率值: 0.9410995989235603
数据点: -3.519028179373069 1.0975889471287252 真实类别: [1.] 计算类别: 1 概率值: 0.929314546733761
数据点: -3.134739493168724 -1.0004530330655164 真实类别: [1.] 计算类别: 1 概率值: 0.9851480721920968
数据点: -7.483330228335294 1.2574782002172955 真实类别: [1.] 计算类别: 1 概率值: 0.995520274252808
数据点: -5.535501279096166 -0.5707433716517446 真实类别: [1.] 计算类别: 1 概率值: 0.9963321263135214
数据点: -4.73236581912802 0.6249539386059606 真实类别: [1.] 计算类别: 1 概率值: 0.9804606923707542
数据点: -6.338440734985751 0.3963170410211942 真实类别: [1.] 计算类别: 1 概率值: 0.9951684970603293
数据点: -5.544124846877673 -0.457524785778184 真实类别: [1.] 计算类别: 1 概率值: 0.9959623477480174
数据点: -5.702037726897173 -0.3398048889681044 真实类别: [1.] 计算类别: 1 概率值: 0.9960080828936145
数据点: -3.709779246197387 1.3722905632051088 真实类别: [1.] 计算类别: 1 概率值: 0.9219837594358352
数据点: -4.5843138771335195 0.34919262472729323 真实类别: [1.] 计算类别: 1 概率值: 0.9829628314889282
数据点: -5.7346335714216075 0.6578734081354721 真实类别: [1.] 计算类别: 1 概率值: 0.9904087902399246
数据点: -6.0832688430780975 1.6597287024543563 真实类别: [1.] 计算类别: 1 概率值: 0.9818052025527166
数据点: -4.6654040896117825 0.6009756129758765 真实类别: [1.] 计算类别: 1 概率值: 0.9799086696208604
数据点: -6.1217996635877014 -0.49084471663214047 真实类别: [1.] 计算类别: 1 概率值: 0.9974561253065534
数据点: -6.369360595711045 -1.937088810900264 真实类别: [1.] 计算类别: 1 概率值: 0.9994312277789209
数据点: -4.082991962245137 -0.3868008963295351 真实类别: [1.] 计算类别: 1 概率值: 0.9872335096116719
数据点: -3.705846742289825 -0.8541850637201754 真实类别: [1.] 计算类别: 1 概率值: 0.9889039454668889
数据点: -4.985484841696257 -0.7232110602786869 真实类别: [1.] 计算类别: 1 概率值: 0.9951820307573562
数据点: -0.42984840078648745 4.314563206461582 真实类别: [0.] 计算类别: 0 概率值: 0.9733664697807481
数据点: -0.08786545755948205 5.094055218740585 真实类别: [0.] 计算类别: 0 概率值: 0.9896812061695637
数据点: 1.1232138797334772 5.255950377726406 真实类别: [0.] 计算类别: 0 概率值: 0.9963838386312939
数据点: 0.5187954422063538 6.373511255929149 真实类别: [0.] 计算类别: 0 概率值: 0.9979360603947655
数据点: 0.5030652846302315 3.0457443380525206 真实类别: [0.] 计算类别: 0 概率值: 0.9587157812644288
数据点: 1.9299682443481663 5.542364688732313 真实类别: [0.] 计算类别: 0 概率值: 0.9984746147963821
数据点: -0.7049643625664485 4.924606620199083 真实类别: [0.] 计算类别: 0 概率值: 0.9810472081643058
数据点: -1.1115861110129486 5.2164112374995435 真实类别: [0.] 计算类别: 0 概率值: 0.980293995926454
数据点: -0.048739481389295396 4.29343041507631 真实类别: [0.] 计算类别: 0 概率值: 0.9794710266051431
数据点: -0.2941960466828457 5.751183744401182 真实类别: [0.] 计算类别: 0 概率值: 0.9933460379040839
数据点: -1.198257259215314 5.933998853051266 真实类别: [0.] 计算类别: 0 概率值: 0.9889482649792446
数据点: 0.06811324897171993 4.323938161941664 真实类别: [0.] 计算类别: 0 概率值: 0.9816665920477476
数据点: 0.1776216075069517 5.156199818825839 真实类别: [0.] 计算类别: 0 概率值: 0.9919896445160995
数据点: -1.5527709663507738 4.286521813161135 真实类别: [0.] 计算类别: 0 概率值: 0.9388333591293296
数据点: -1.7099291137725479 5.450982776551628 真实类别: [0.] 计算类别: 0 概率值: 0.9751905260086414
数据点: -0.6153105796061 5.872731376076321 真实类别: [0.] 计算类别: 0 概率值: 0.9924266965128594
数据点: 0.8238882470822332 5.7897149291571175 真实类别: [0.] 计算类别: 0 概率值: 0.9972111267291643
```



Lecture5（支撑向量机）：

1、见代码文件SVM.py中的函数Dual_SVM、Kernel_SVM

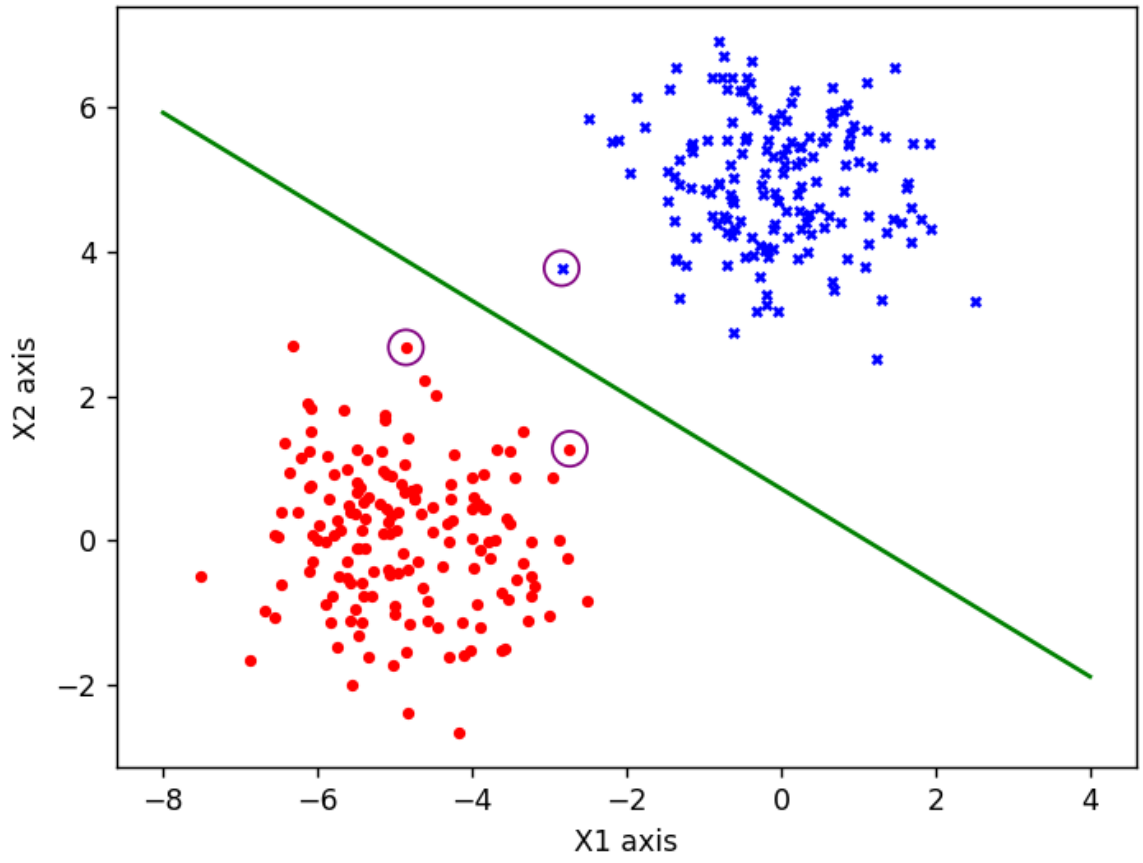
2、数据生成见代码文件SVM.py中的函数DATA

均值向量为(-5,0)(0,5)时：

生成数据集：

```
生成数据集: [[-5.75633488  0.2780014   1.        ]
 [-5.5780553   0.39399782  1.        ]
 [-5.05860219 -0.46664276  1.        ]
 ...
 [ 0.05880747  5.83408442 -1.        ]
 [ 2.05597182  5.13623444 -1.        ]
 [ 0.38309998  4.25561865 -1.        ]]
```

图示：

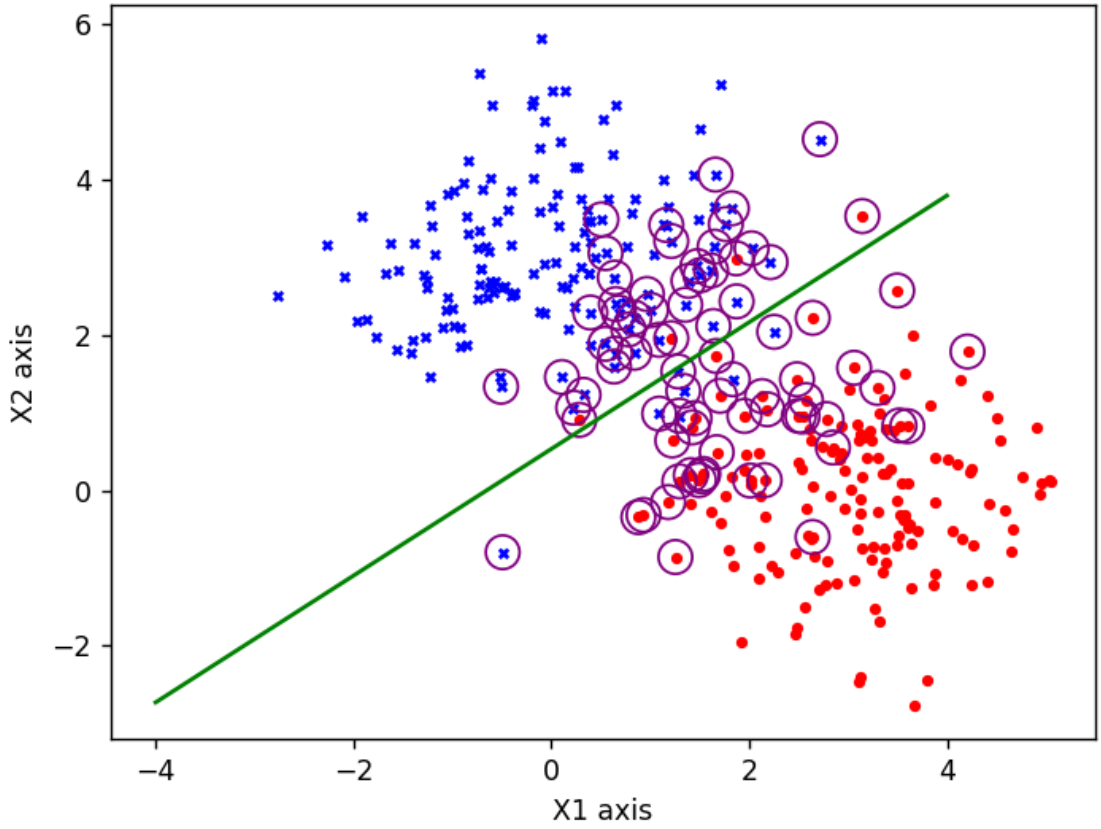


均值向量为(3,0)(0,3)时：

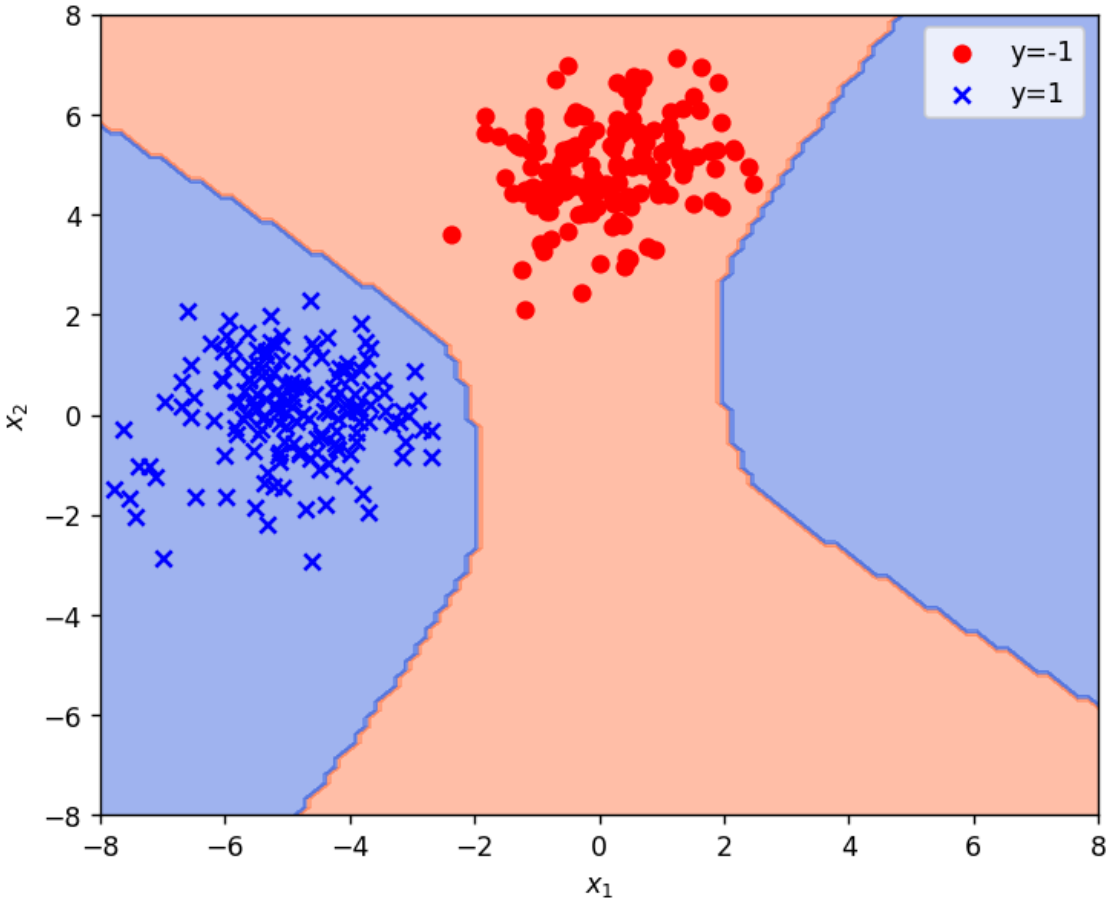
生成数据集：

```
生成数据集: [[-1.56934444  1.12805754  1.        ]
 [-3.74885618 -0.24597254  1.        ]
 [-3.64893547  0.08641762  1.        ]
 ...
 [-0.58315937  4.13975806 -1.        ]
 [-0.08931694  3.16266402 -1.        ]
 [ 0.79695897  3.00932367 -1.        ]]
```

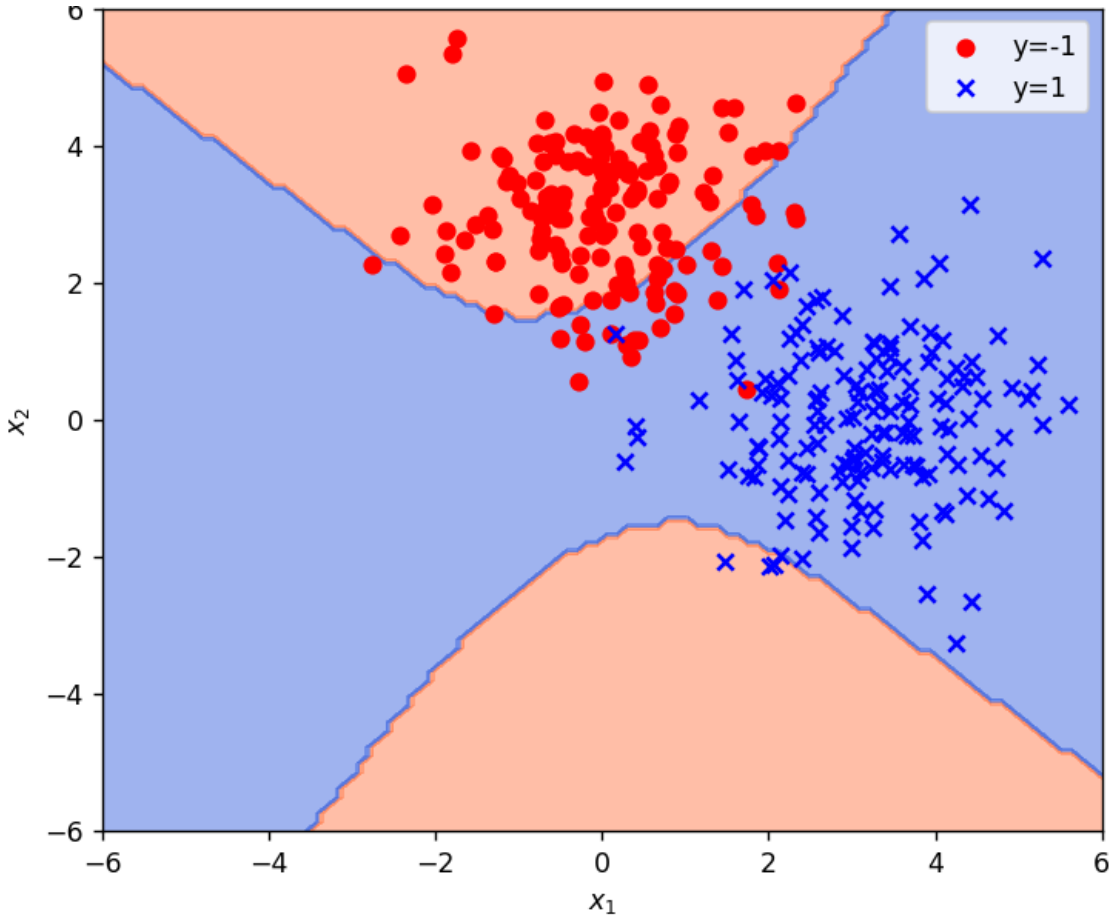
图示：



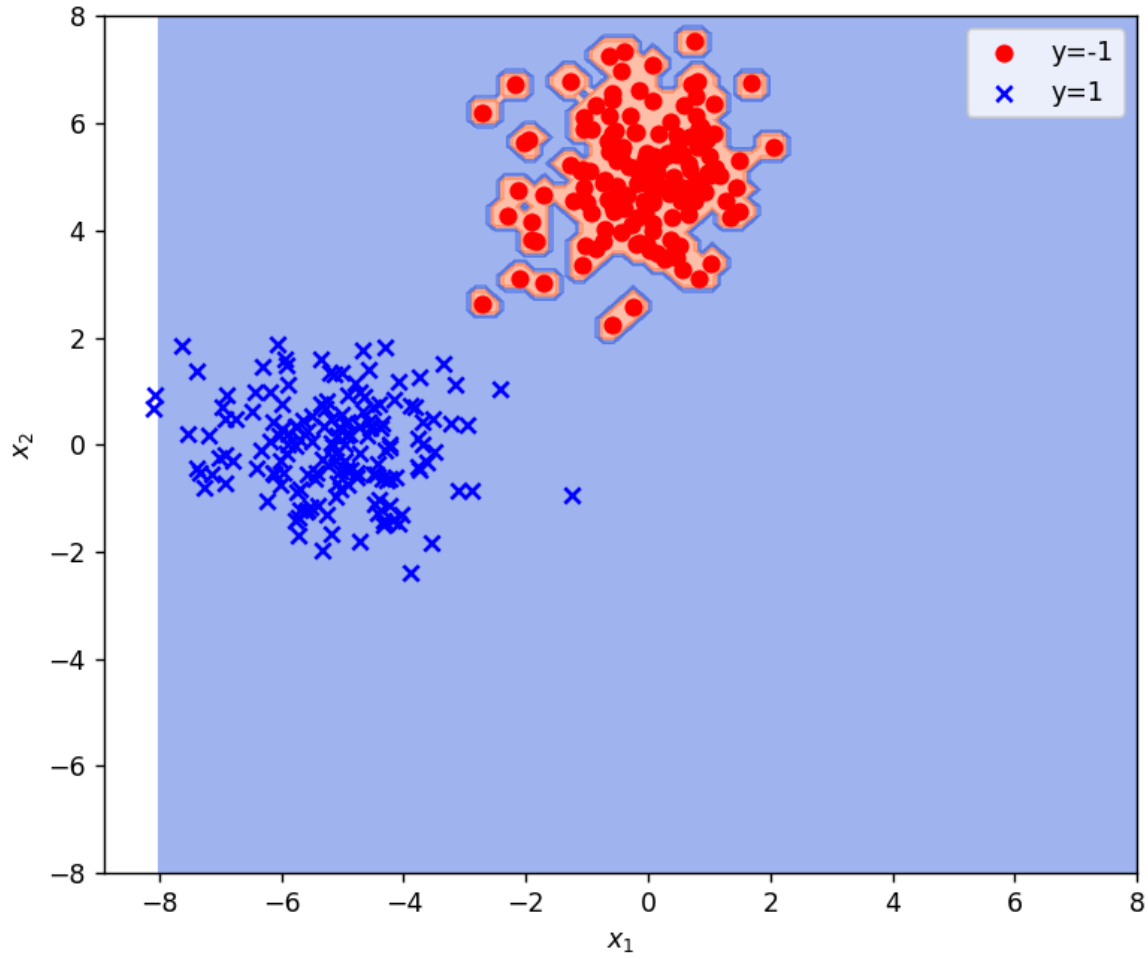
Kernel-SVM（四次多项式核）： 均值向量为(-5,0)(0,5)时：



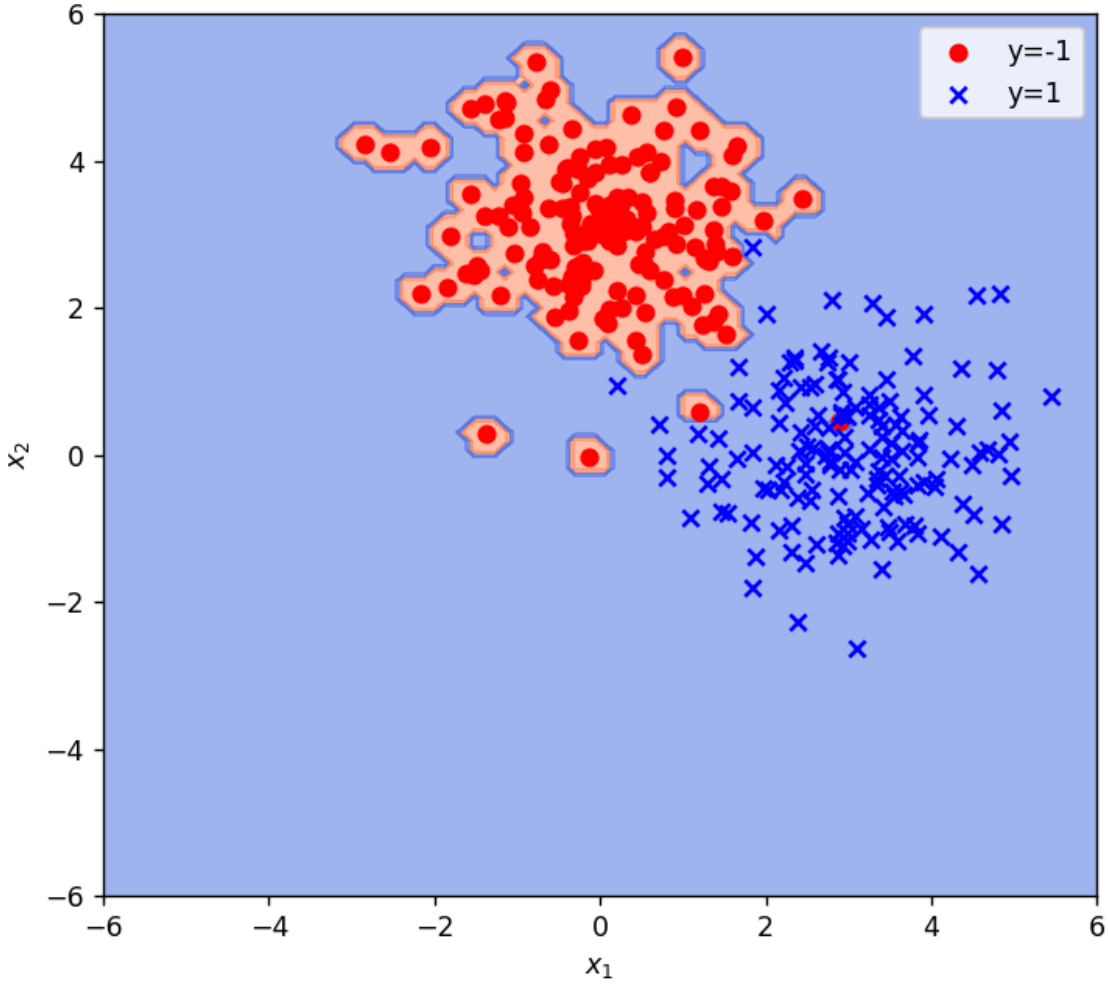
均值向量为(3,0)(0,3)时：



Kernel-SVM（高斯核）： 均值向量为(-5,0)(0,5)时：

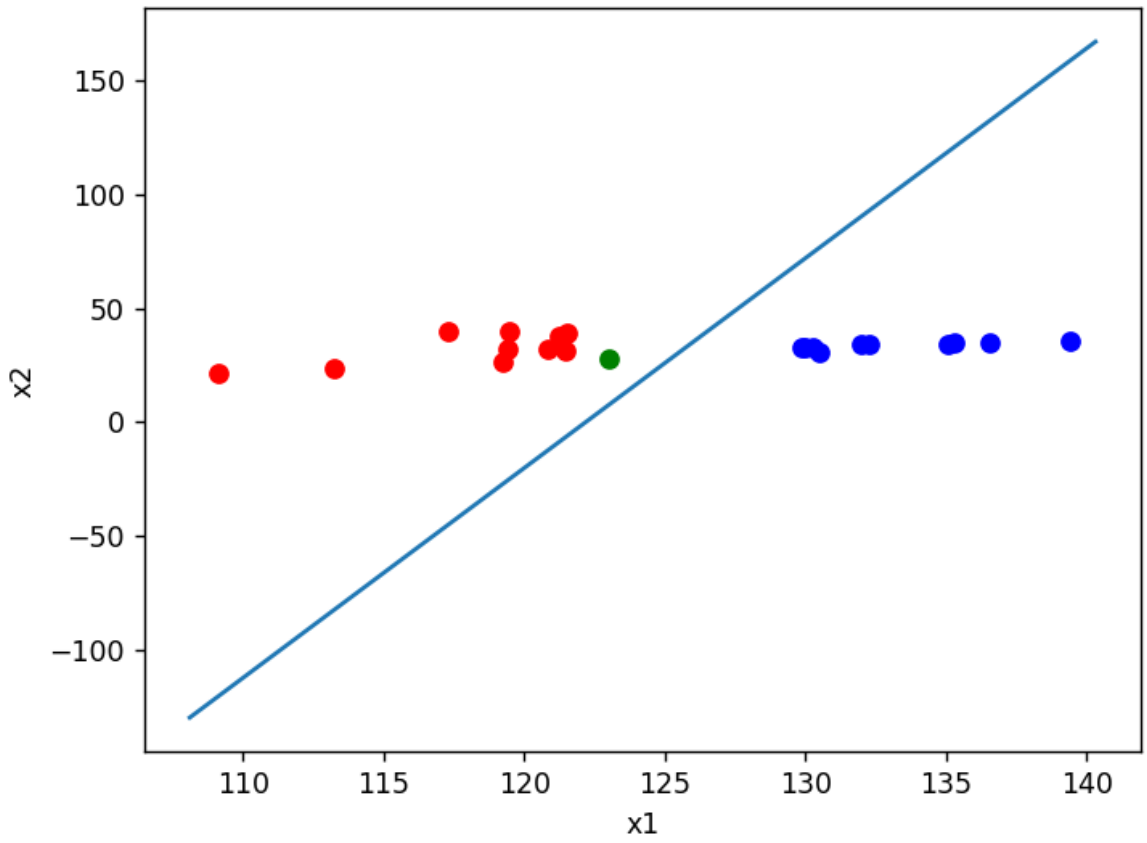


均值向量为(3,0)(0,3)时：



3、钓鱼岛：

将对偶支撑向量机的数据改成中日城市数据，训练后，将钓鱼岛数据进行分类，得出钓鱼岛属于中国。如图所示，红色类为中国，蓝色类为日本，绿色点为钓鱼岛：



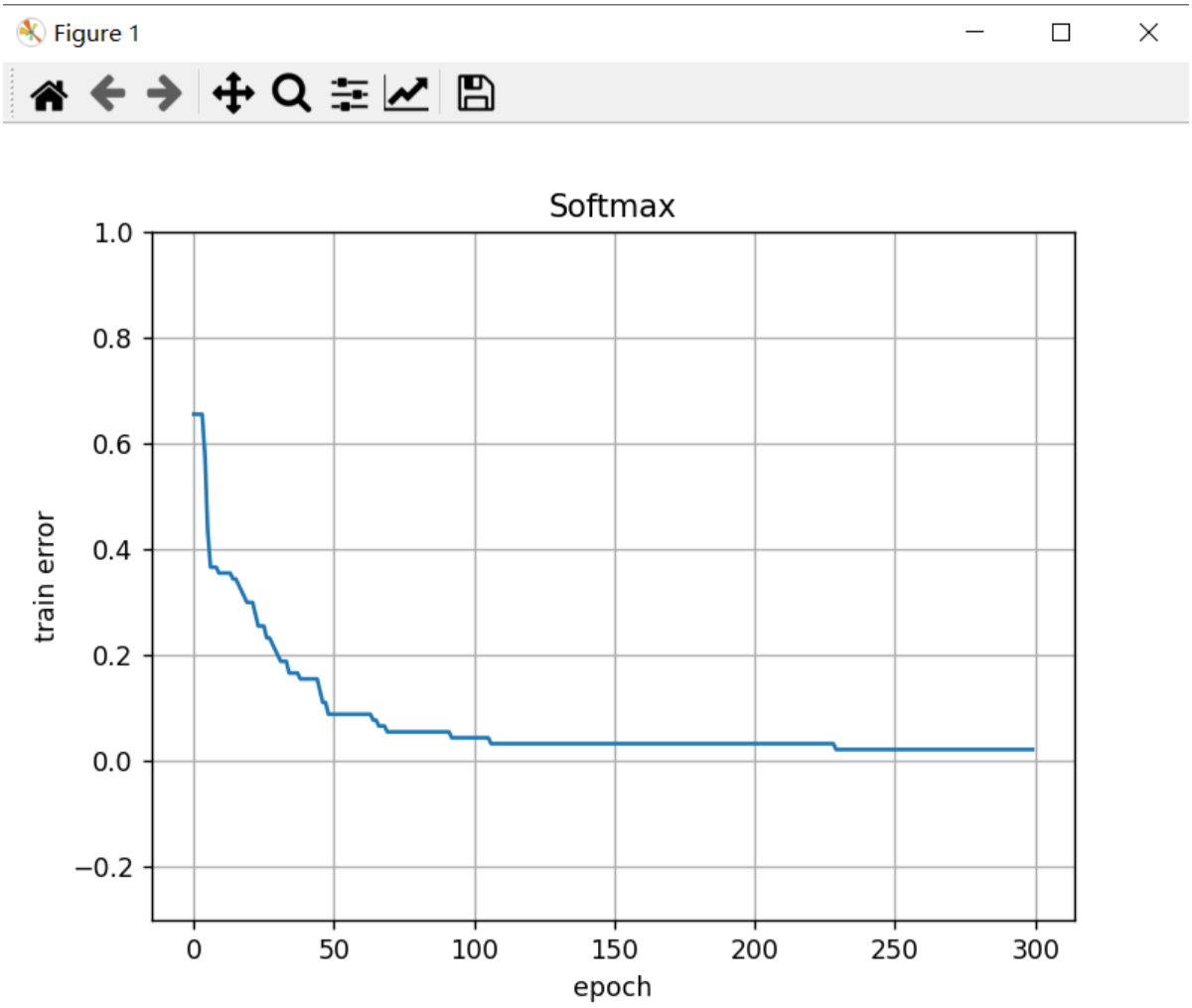
Lecture6（多类分类）：

1、(a):见代码文件OVOPocket.py：

```
PS C:\Users\DELL> & C:/ProgramData/anaconda3/python.exe c:/Users/DELL/Desktop/Pattern-Recognition-And-Machine-Learning/OVOPOCKET.py
W: [  3.6  11.7 -14.9  -5.6]
b: 1.0
W: [  2.7   9.7 -11.7  -5.5]
b: 1.0
W: [ 0.4  2.  -2.4 -0.2]
b: 0.0
训练集预测准确率为:  0.9111111111111111
测试集预测准确率为:  0.9166666666666666
PS C:\Users\DELL>
```

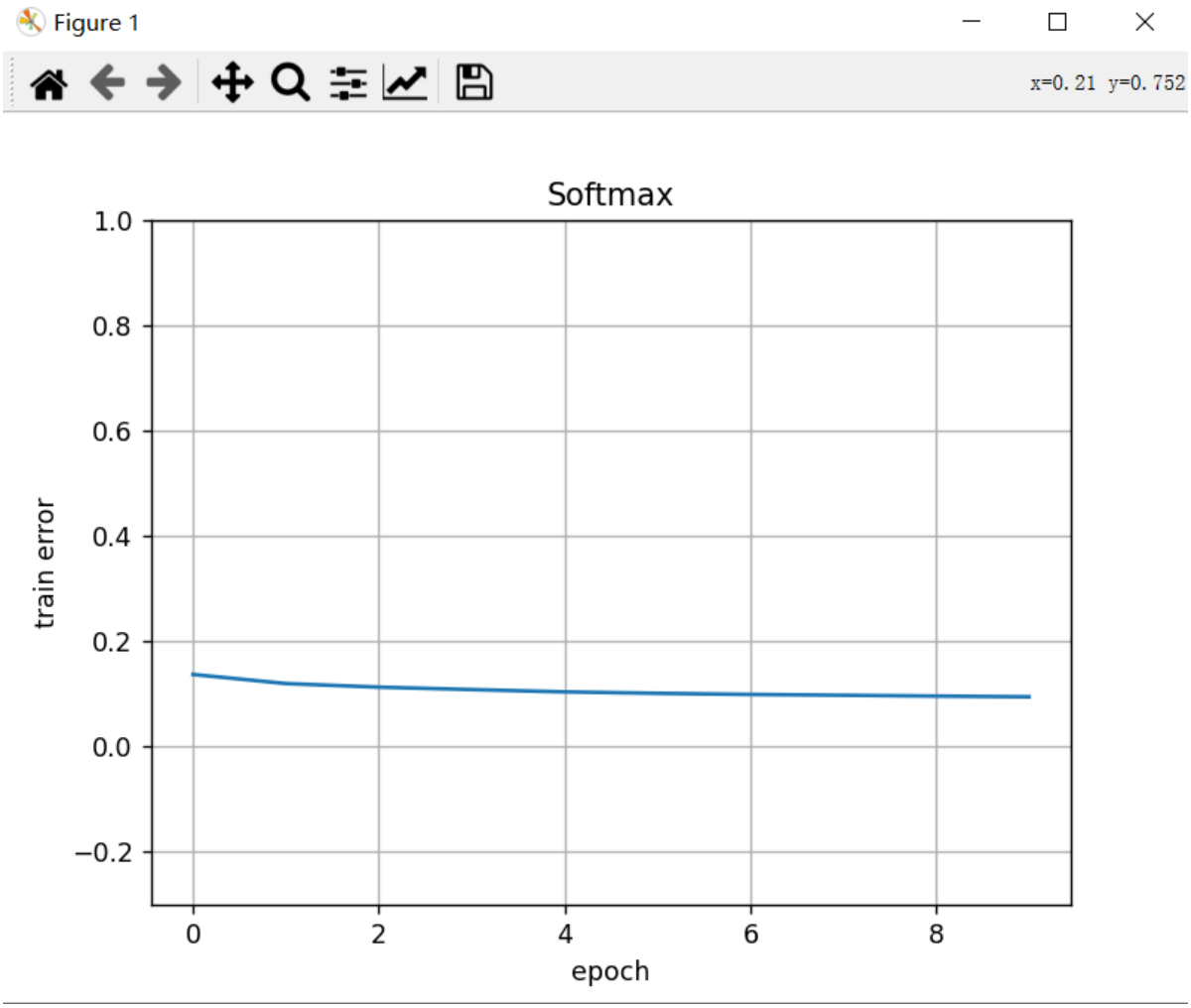
(b):见代码文件Softmax-Iris.py：

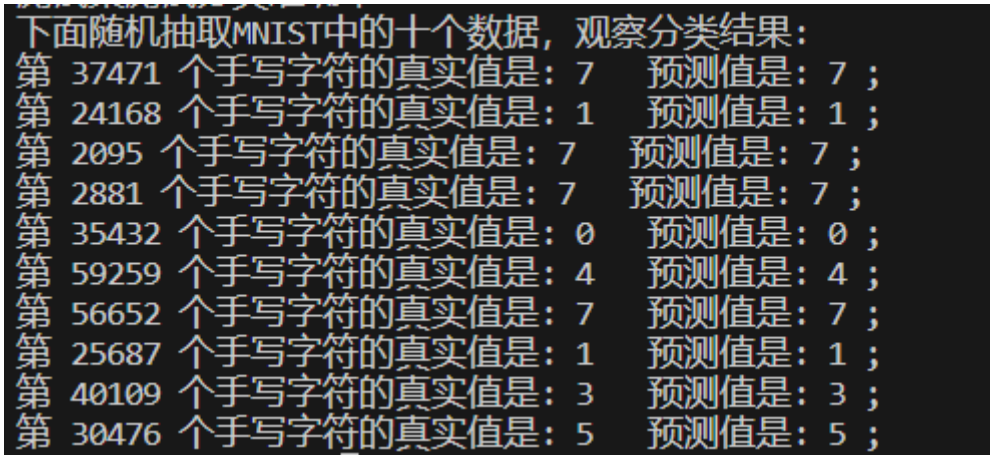
```
PS C:\Users\DELL> & C:/ProgramData/anaconda3/python.exe c:/Users/DELL/Desktop/Pattern-Recognition-And-Machine-Learning/Softmax-Iris.py
耗时:  1.8211085796356201 s
[[ 1.37727173  2.02183271 -0.37738173  0.35193286]
 [ 1.27737735  0.76316254  1.14230329  0.6867577 ]
 [ 0.34535092  0.21500474  2.23507844  1.96130944]]
训练集测试分类准确率:  0.9777777777777777
测试集测试分类准确率:  0.9666666666666667
```



2、见代码文件Softmax-MNIST.py：

```
PS C:\Users\DELL> & C:/ProgramData/anaconda3/python.exe c:/Users/DELL/Desktop/Pattern-Recognition-And-Machine-Learning/Softmax-MNIST.py
x_shape:  (60000, 28, 28)
y_shape:  (60000,)
x_test_shape:  (10000, 28, 28)
y_test_shape:  (10000,)
1
100%|████████████████████████████████████████████████████████████████████████████████| 10/10 [01:23<00:00,  8.30s/it]
耗时:  83.04125809669495 s
[[-1.23611836e-02  1.69504199e-02  6.51871555e-03 ... -1.07446715e-02
   6.45103401e-03 -5.56960129e-03]
 [ 2.06666600e-03 -1.68584136e-02  2.14859125e-02 ...  5.13243789e-03
   5.09176003e-03 -3.96222761e-03]
 [-6.64083917e-03  7.11372733e-03 -2.58894339e-02 ... -1.63225302e-02
  -1.29199247e-02  2.24033619e-03]
 ...
 [-1.42068708e-02 -3.66747072e-03 -8.36659057e-05 ... -2.32774457e-03
   5.61803057e-03  1.73093468e-04]
 [-2.07109917e-02  2.88242671e-03  1.05031641e-04 ...  2.31130869e-04
  -7.60128773e-03 -3.62323379e-04]
 [ 1.21163721e-02  1.02818542e-02  6.17361640e-03 ...  3.05746142e-03
   1.08008806e-02 -2.25733407e-03]]
训练集测试分类准确率:  0.90505
测试集测试分类准确率:  0.9104
```





Lecture7（神经网络与深度学习）：

1、见代码文件NN-Iris.py：

网络1：设计了一个两层的的神经网络进行分类，第一层使用ReLU函数做激活函数，两层网络的大小分别为(4,10)、(10,3)，最后一层使用Softmax函数做激活函数，学习率为0.02，num_epochs=100，下面展示分类效果：

```
耗时:  4.130415678024292 s
第一层网络参数为:  OrderedDict([('weight', tensor([[-0.4725, -0.3978,  0.1171, -0.1366],
        [-0.3732, -0.5735,  1.1813,  2.1240],
        [-0.2485, -0.4724, -0.1900, -0.1538],
        [ 1.8978,  2.4894, -2.5734, -3.5348],
        [ 0.1669, -0.1456, -0.4103, -0.2964],
        [ 0.5205,  1.1156, -0.9817, -0.9971],
        [-0.1879, -0.4565, -0.1521,  0.1314],
        [ 0.0192, -0.0374,  0.2314, -0.1045],
        [-0.3120, -0.0141, -0.2300,  0.2974],
        [-0.6111, -0.4328,  1.2196,  1.5446]])), ('bias', tensor([ 0.3226, -0.9810,  0.2884,  1.1039, -0.0201,  0.4511,  0.0706, -0.2951,
         0.4725, -0.0951]))])第二层网络参数为:  OrderedDict([('weight', tensor([[ 0.1048, -1.6729, -0.1307,  0.9818, -0.2552,  0.2736, -0.2051, -0.2173,
          0.2373, -1.3679],
        [ 0.1043,  0.4174, -0.1280,  0.4751,  0.2866, -0.0059,  0.1946,  0.2574,
          0.1584,  0.3057],
        [-0.1434,  1.2874, -0.1769, -1.0684,  0.2727, -0.6116, -0.2264,  0.3091,
          0.1993,  0.9359]])), ('bias', tensor([-0.3965,  1.3936, -1.6447]))])
预测准确率为: 0.9666666666666667
```

网络2：设计了一个三层的的神经网络进行分类，第一层使用ReLU函数做激活函数，两层网络的大小分别为(4,10),(10,20),(10,3)，最后一层使用Softmax函数做激活函数，学习率为0.02，num_epochs=100，下面展示分类效果：

```
PS C:\Users\DELL> & C:/ProgramData/anaconda3/python.exe c:/Users/DELL/Desktop/Pattern-Recognition-And-Machine-Learning/NN-Iris.py
2023-11-17 13:16:54.760916: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
ource Control (Ctrl+Shift+G) 47 s
网络参数为:  OrderedDict([('0.weight', tensor([[ 0.1068, -0.3759, -0.2028, -0.4079],
        [-0.2531,  0.4481, -0.4444, -0.3760],
        [-0.3973,  0.4424,  0.1850, -0.2623],
        [-0.6421, -0.6999,  1.4910,  2.0585],
        [-0.2987,  0.3924, -0.2685,  0.1422],
        [ 1.8071,  2.5937, -2.6898, -3.2684],
        [ 1.0638,  0.4638, -1.2590, -1.0143],
        [-0.1298, -0.2473, -0.2800,  0.0848],
        [-0.2634,  0.0162,  0.2360, -0.0361],
        [-0.8227, -1.0081,  1.3967,  1.9871]])), ('0.bias', tensor([ 0.4595,  0.3813,  0.0696, -0.1351, -0.2314,  1.2405, -0.0145, -0.4693,
         0.2447, -0.5559])), ('2.weight', tensor([[-0.1143,  0.0961, -0.2936, -0.2689,  0.2225, -0.2365, -0.0085, -0.0286,
          0.0866,  0.0771],
        [-0.1269,  0.1830,  0.0988, -0.3024,  0.2629,  0.1411,  0.0446,  0.2423,
         -0.0230,  0.0954],
        [-0.0988, -0.2918, -0.1534, -0.1309, -0.1427, -0.1866,  0.0093, -0.1222,
          0.0652,  0.1849],
        [ 0.0099,  0.1682,  0.2717, -0.1760,  0.2874,  0.1639,  0.1446,  0.1667,
         -0.1886,  0.1205],
        [ 0.0334,  0.2979, -0.2014, -0.0827, -0.3081, -0.1413,  0.2517,  0.1764,
          0.0621, -0.1495],
        [ 0.2428, -0.2213, -0.1988,  0.2627,  0.2095, -0.4707, -0.1964, -0.2411,
          0.0959, -0.1507],
        [ 0.2186, -0.0164,  0.2274, -0.5168, -0.2799,  0.5217,  0.3356, -0.1873,
         -0.1511,  0.0717],
        [-0.2026, -0.1786, -0.2973, -0.2929, -0.2550,  0.9016,  0.3695,  0.3065,
         -0.1327, -0.3356],
        [ 0.2212, -0.2713, -0.0082, -0.7617,  0.2876,  0.2932,  0.2346, -0.1560,
          0.1489,  0.2902],
        [ 0.1466,  0.2838,  0.2978, -0.2439,  0.2863,  0.0693, -0.2556, -0.1263,
          0.0614, -0.1696],
        [ 0.1576,  0.0866,  0.0233, -0.3829, -0.0480,  0.8338,  0.1351,  0.0430,
          0.2925, -0.3247],
        [-0.1333,  0.1000, -0.2032,  0.4141,  0.1844, -0.6699, -0.0195,  0.2673,
         -0.2075,  0.3556],
        [-0.1929, -0.2071,  0.2955,  0.0083,  0.0315,  0.2922,  0.2897,  0.2194,
         -0.1622, -0.4601],
        [ 0.0623,  0.1972, -0.0801, -0.1250,  0.2189, -0.0436, -0.0710, -0.1913,
          0.1314,  0.2680],
        [-0.2292,  0.3400,  0.2964, -0.6694,  0.2664,  0.3290,  0.0840, -0.0582,
         -0.1816, -0.1291],
```

```
         -0.6993, -1.3850,  0.4587,  0.1523],
        [-0.1499, -0.3122,  0.0795,  0.0513,  0.1737, -0.2976, -0.1598,  0.3140,
         -0.6111, -0.0507,  0.2263, -0.4245,  0.3435,  0.0868, -0.4697, -0.0901,
          0.3984,  0.3112, -0.4188,  0.0410],
        [ 0.0744, -0.1961, -0.1031, -0.1307, -0.1794, -0.0457, -0.5498, -0.5144,
         -0.2220,  0.1017, -0.5331,  0.2478,  0.0031,  0.0339, -0.3572,  0.2423,
          0.1824,  0.9257,  0.0648, -0.1170]])), ('4.bias', tensor([-0.6372,  1.2775, -0.2294]))])
预测准确率为: 0.9666666666666667
```

网络3：将网络2中的激活函数由ReLU()换成Sigmoid()，下面展示分类效果：

```
PS C:\Users\DELL> & C:/ProgramData/anaconda3/python.exe c:/Users/DELL/Desktop/Pattern-Recognition-And-Machine-Learning/NN-Iris.py
2023-11-17 13:19:06.665255: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
耗时:  5.118422985076904 s
网络参数为:  OrderedDict([('0.weight', tensor([[ 0.3153, -0.6293,  0.0924,  0.4473],
        [-0.7426, -1.1196,  1.2403,  1.6521],
        [ 0.5374,  0.8568, -1.2865, -0.6488],
        [ 0.5741,  1.1938, -1.2619, -1.2188],
        [ 0.0259, -1.3203,  1.2286,  0.4208],
        [ 0.2927,  0.9715, -0.7127, -1.2979],
        [ 0.8382,  0.9260, -1.2135, -1.7290],
        [-0.5396, -0.2993,  0.6882,  1.1940],
        [ 0.3932, -0.1456,  0.1603, -0.0352],
        [-0.3266,  0.5611, -0.3233,  0.1699]])), ('0.bias', tensor([-0.3704, -0.6529,  0.3360,  0.6110,  0.1602,  0.0722,  0.6428, -0.0313,
        -0.0283, -0.3129])), ('2.weight', tensor([[-0.3505, -1.2179,  1.1527,  1.2997, -0.4480,  0.9223,  1.4081, -0.6179,
          0.1428,  0.2119],
        [-0.1358, -0.5245,  0.3453,  0.5615, -0.5754,  0.5938,  0.4264, -0.2672,
         -0.3147,  0.2980],
        [ 0.3647,  0.5989, -0.6318, -0.2368,  0.4967, -0.0997, -0.7605,  0.4022,
          0.1104, -0.3456],
        [-0.0054,  0.4940, -0.4405, -0.4324,  0.0925, -0.0840, -0.6423,  0.4360,
          0.1512,  0.2034],
        [-0.0546,  0.0671, -0.1249, -0.2900,  0.5271, -0.2336, -0.3820,  0.1008,
         -0.1243, -0.1034],
        [ 0.0645,  0.4963, -0.5545, -0.3185,  0.6040, -0.4813, -0.1950,  0.2783,
         -0.0788,  0.0386],
        [ 0.2227,  0.5128, -0.0155, -0.4372,  0.2256, -0.1175, -0.2571,  0.4121,
          0.0404, -0.2226],
        [ 0.3331,  1.0502, -0.5840, -0.8245,  1.2205, -0.6668, -0.7179,  0.5148,
          0.2431, -0.2154],
        [-0.2160, -1.0383,  0.6946,  0.9580, -0.3739,  0.8365,  1.0756, -0.5538,
          0.2292, -0.0511],
        [ 0.0604,  0.9743, -0.7505, -0.8740,  1.1365, -0.4900, -0.6446,  0.7709,
          0.5401, -0.3607],
        [-0.2261, -0.2412,  0.1801,  0.4079, -0.5749, -0.0221,  0.0615, -0.2113,
          0.1621,  0.0837],
        [-0.0037, -0.1339, -0.1858, -0.3290, -0.1812,  0.0438,  0.2462,  0.1627,
          0.2104,  0.1627],
        [ 0.0739,  0.6445, -0.7183, -0.7726,  0.9509, -0.7329, -0.8232,  0.4266,
          0.1237, -0.1777],
        [ 0.2187,  0.1439, -0.3131, -0.2680, -0.0217, -0.1211, -0.1664, -0.1150,
         -0.2605, -0.3156],
        [ 0.0223, -0.6735,  0.3079,  0.2264, -0.5833,  0.3042,  0.1851, -0.0693,
          0.1080, -0.2163],
        [-0.1730, -0.9192,  1.0475,  1.0648, -0.7305,  0.6765,  0.9044, -0.4374,
```

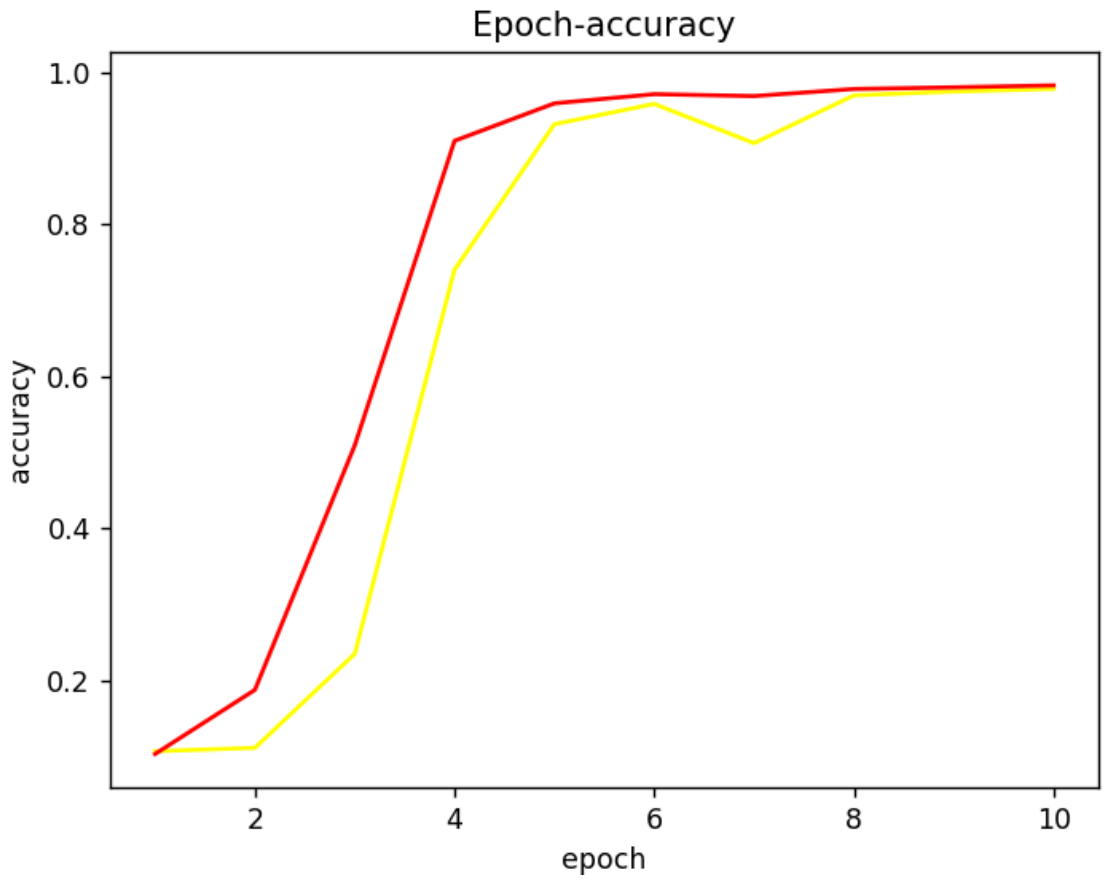
```
        -0.1793, -0.0344],
        [-0.1488,  0.4601, -0.1614, -0.1630,  0.6186, -0.5162, -0.6177,  0.1988,
          0.1031, -0.1721],
        [ 0.0365, -0.6953,  0.1798,  0.6183, -0.5007,  0.4094,  0.8413, -0.4076,
          0.1049,  0.3416],
        [ 0.1838,  0.3848, -0.0732, -0.3041, -0.0713,  0.0624, -0.1542,  0.1854,
         -0.1977,  0.0751],
        [ 0.0657,  0.0523, -0.6062, -0.4062,  0.6986, -0.4085, -0.4352,  0.3375,
          0.1785, -0.2820]])), ('2.bias', tensor([ 0.0048,  0.0143,  0.1717, -0.2072, -0.0243,  0.0569, -0.3576,  0.3044,
        -0.0216, -0.0062, -0.0451,  0.0174,  0.2039, -0.2277, -0.2187,  0.2006,
        -0.0458,  0.1436, -0.0481,  0.3308])), ('4.weight', tensor([[ 1.9910,  1.0211, -1.1351, -0.6906, -0.5127, -0.9262, -0.6190, -2.1861,
          1.2397, -2.1896,  0.6708, -0.2199, -1.6025, -0.2541,  0.5766,  1.7615,
         -0.8351,  0.8893, -0.1504, -0.8916],
        [ 0.7866, -0.0975,  0.2724, -0.1824,  0.1731,  0.0358, -0.0348,  0.5510,
          0.5415,  0.4115, -0.1512,  0.1003,  0.1140,  0.1378, -0.1042,  0.3830,
          0.0385,  0.3946,  0.0328,  0.4882],
        [-2.6260, -1.0420,  0.7551,  0.7236,  0.4020,  0.6522,  0.4822,  1.6732,
         -2.0639,  1.6744, -0.3238, -0.2684,  1.6791,  0.2961, -0.7846, -2.0670,
          0.6593, -1.2231,  0.3412,  0.7651]])), ('4.bias', tensor([ 0.1767,  0.1364, -0.5173])))])
预测准确率为: 0.9833333333333333
```

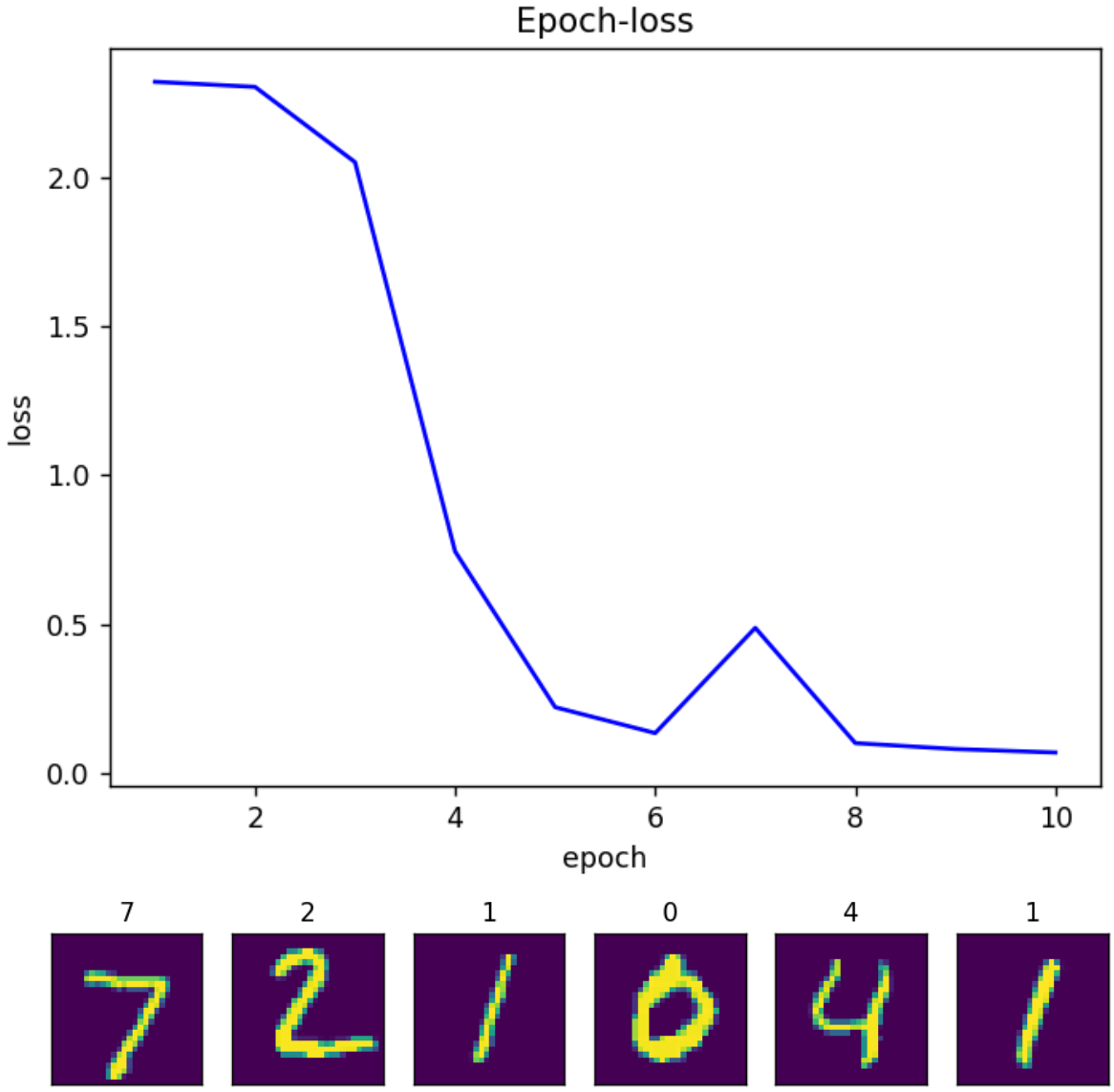
可以观察到，从网络1到网络3，分类准确率不断提升

2、见代码文件LeNet-MNIST.py,代码调用GPU运算：

结果如图所示：

```
PS C:\Users\DELL> & C:/ProgramData/anaconda3/python.exe c:/Users/DELL/Desktop/Pattern-Recognition-And-Machine-Learning/LeNet-MNIST.py
2023-11-17 13:46:09.101996: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
  0%|                                                                                                                                                  | 0/10 [00:00<?, ?it/s]a
cc_train: 0.1065
 10%|                                                                                                                                       | 1/10 [00:18<02:43, 18.12s/it]a
cc_train: 0.1109
 20%|                                                                                                                                       | 2/10 [00:34<02:16, 17.05s/it]a
cc_train: 0.23456666666666667
 30%|                                                                                                                                       | 3/10 [00:50<01:55, 16.49s/it]a
cc_train: 0.7403166666666666
 40%|                                                                                                                                       | 4/10 [01:06<01:37, 16.25s/it]a
cc_train: 0.931733333333333
 50%|                                                                                                                                       | 5/10 [01:22<01:21, 16.25s/it]a
cc_train: 0.9584833333333334
 60%|                                                                                                                                       | 6/10 [01:38<01:04, 16.11s/it]a
cc_train: 0.9068166666666667
 70%|                                                                                                                                       | 7/10 [01:53<00:47, 15.90s/it]a
cc_train: 0.9696666666666667
 80%|                                                                                                                                       | 8/10 [02:09<00:31, 15.72s/it]a
cc_train: 0.9748666666666667
 90%|                                                                                                                                       | 9/10 [02:24<00:15, 15.69s/it]a
cc_train: 0.9782333333333333
100%|                                                                                                                                      | 10/10 [02:40<00:00, 16.03s/it]
耗时:  160.33472394943237 s
```




Epoch-loss
loss
epoch
2.0
1.5
1.0
0.5
0.0
2
4
6
8
10

7
2
1
0
4
1