《模式识别与机器学习A》实验报告

实验题目： 多层感知机实验

学号： 2021110683

姓名： 徐柯炎

# 实验报告内容

## 实验目的

自行构造一个多层感知机，完成对某种类型的样本数据的分类（如图像、文本等），也可以对人工自行构造的二维平面超过3类数据点（或者其它标准数据集）进行分类。

## 实验内容

1. 能给出与线性分类器（自行实现）做对比，并分析原因。
2. 用不同数据量，不同超参数，比较实验效果。
3. 不许用现成的平台，例如pytorch，tensorflow的自动微分工具。建议实验编程环境：1）安装anaconda以及必要的工具包；2）建议采用python作为主编程语言，也可以根据个人需要和兴趣采用其它编程语言；3）可以基于实验室机器，也可以使用自己的便携式计算机上完成该实验。
4. 实现实验结果的可视化。

## 实验环境

Windows10; python3.9;PyCharm 2021.2.2

## 实验过程、结果及分析（包括代码截图、运行结果截图及必要的理论支撑等）

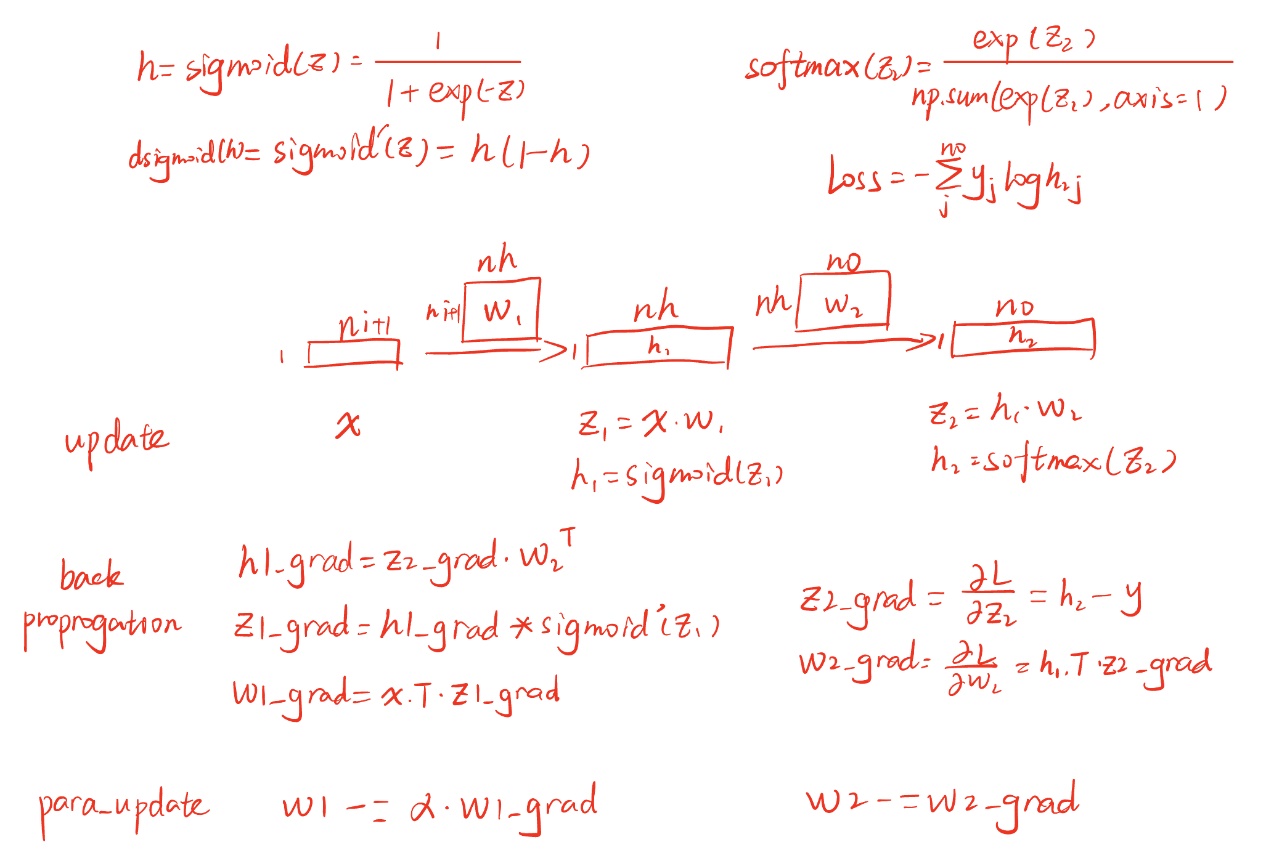
### 多层感知机的原理

多层感知机（Multilayer Perceptron，MLP）是一种前馈神经网络模型，用于解决各种机器学习问题，包括分类和回归问题。它是一种人工神经网络，由多个神经元层组成，包括输入层、一个或多个隐藏层和输出层。

以下是多层感知机的原理：

1. 输入层：多层感知机的输入层接收来自问题域的特征向量。每个输入特征通常表示为一个神经元。输入特征被传递到网络的下一层，即隐藏层。
2. 隐藏层：多层感知机可以有一个或多个隐藏层，每个隐藏层由多个神经元组成。每个神经元都连接到前一层的所有神经元，并且每个连接都有一个权重，用于调整输入的重要性。每个神经元还具有一个偏置项，用于调整神经元的激活阈值。隐藏层的目标是学习输入数据中的非线性关系，以便更好地进行特征提取和表示学习。
3. 输出层：输出层通常包含一个或多个神经元，每个神经元对应于模型的不同类别（分类问题）或一个连续的值（回归问题）。输出层的每个神经元会对来自隐藏层的信息进行加权和汇总，并应用一个激活函数，以产生最终的输出。
4. 激活函数：在每个神经元中，一个激活函数被用于引入非线性性。常用的激活函数包括Sigmoid、ReLU、Tanh等。激活函数允许神经网络捕捉复杂的非线性关系，提高了网络的表达能力。
5. 前向传播：输入特征从输入层传递到输出层的过程称为前向传播。在前向传播期间，每个神经元计算其加权输入，并通过激活函数产生输出。这一过程一直持续到达到输出层，从而产生网络的最终预测结果。
6. 反向传播：多层感知机使用反向传播算法进行训练，通过调整权重和偏置来最小化预测误差。在每个训练迭代中，通过计算损失函数的梯度，反向传播算法更新权重和偏置，以逐渐改进网络的性能。
7. 损失函数：损失函数用于衡量模型的预测与实际目标之间的差距。常见的损失函数包括均方误差（MSE）用于回归问题和交叉熵用于分类问题。

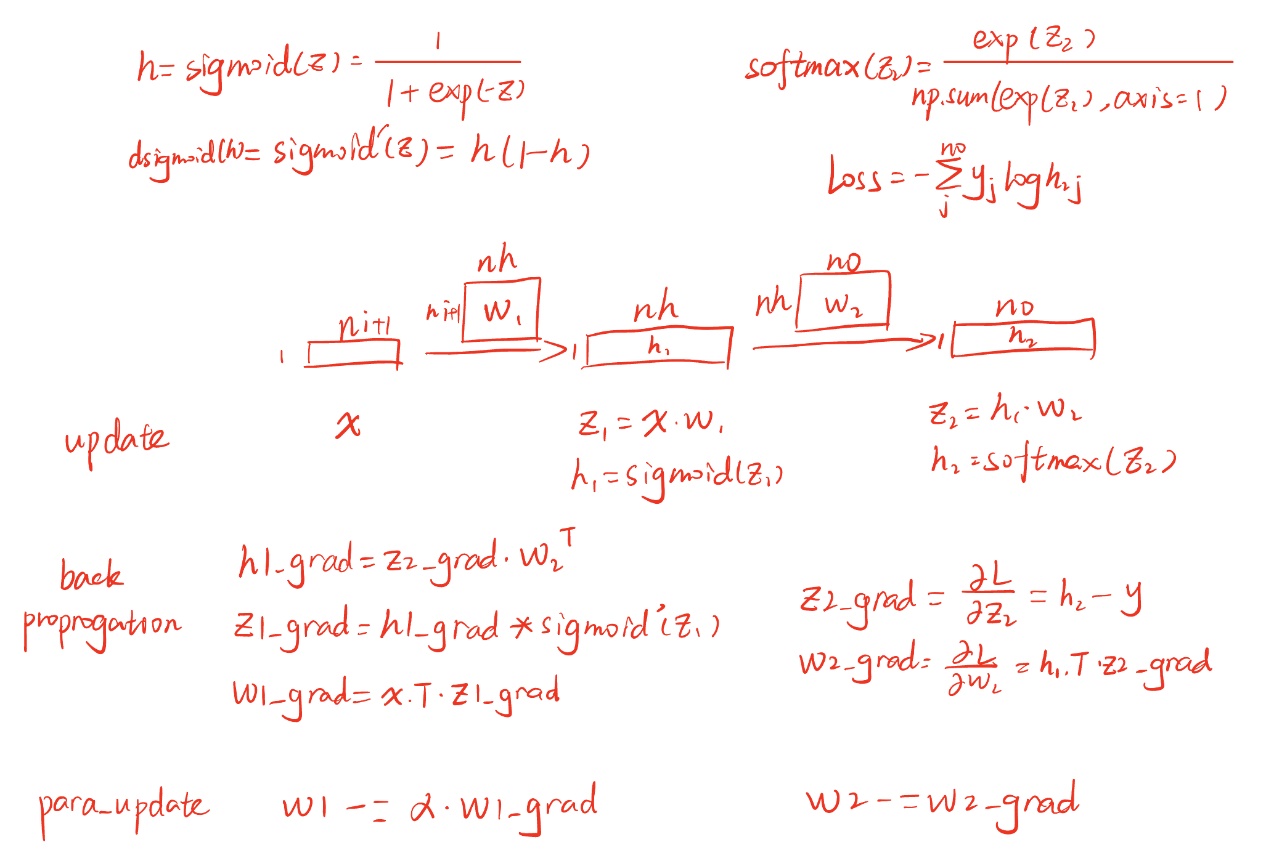
公式部分由下图所示：



### 实验过程

1. 多层感知机

多层感知机结构如下：

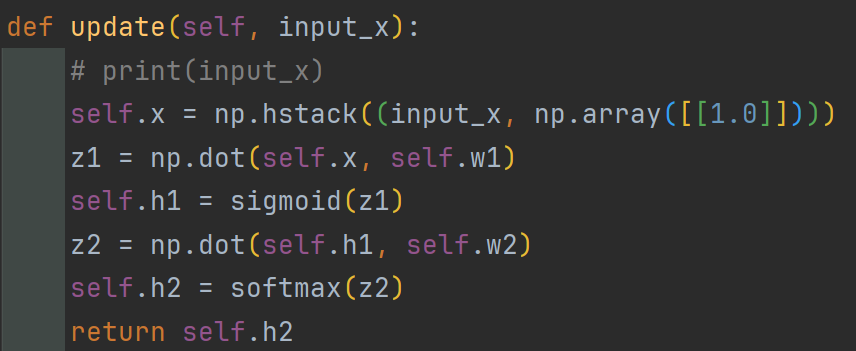


如上图所示，本次实验输入层由输入数据的阶数决定，隐藏层设为10个节点，输出层由分类的种类数量决定，采用的激活函数为sigmoid函数，由于是多分类，所以最后一层用的激活函数为softmax函数，损失函数在这里采用交叉熵损失，优化方法为随机梯度下降（SGD）。

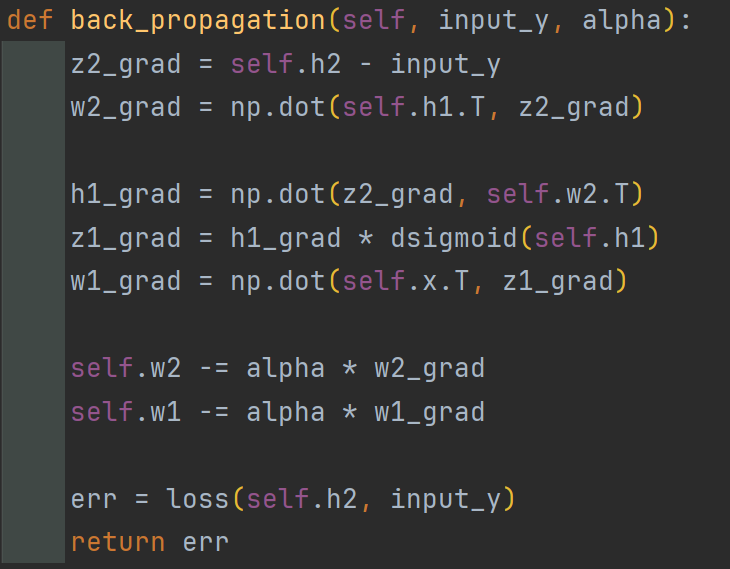
在这里主要实现的类为NN类，NN有如下参数：

NN有如下几个函数，具体如下：

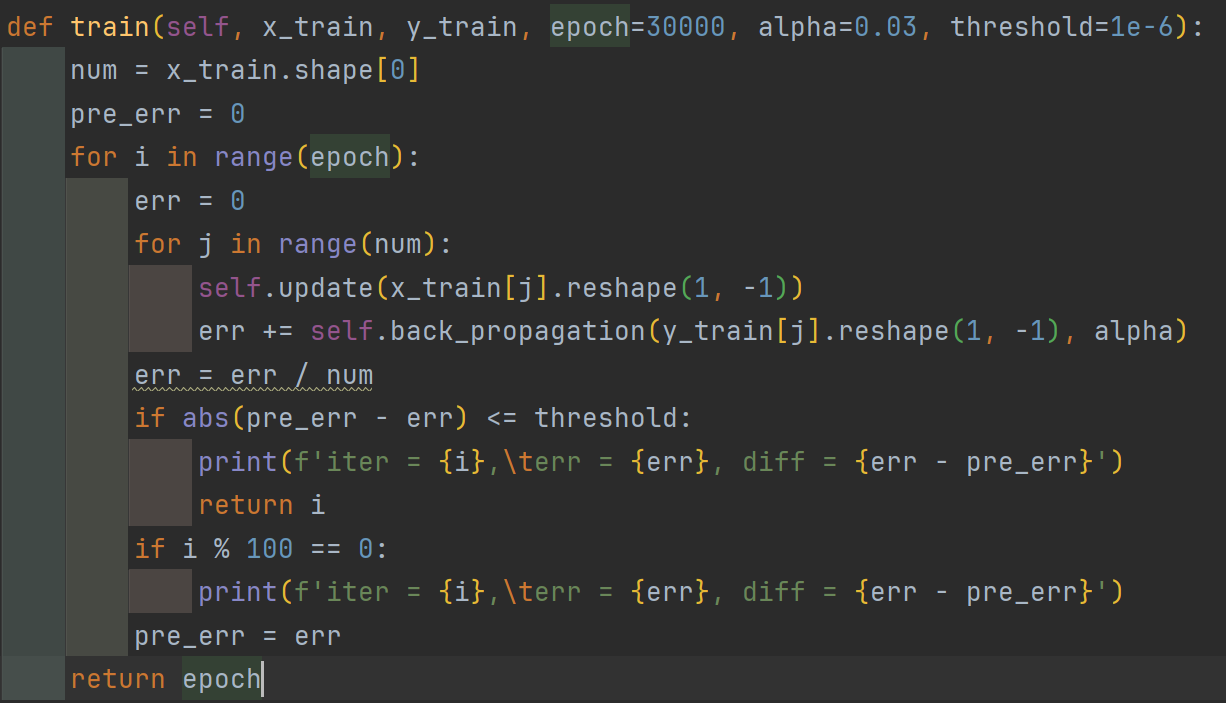
首先是update函数，用来进行前向传播：



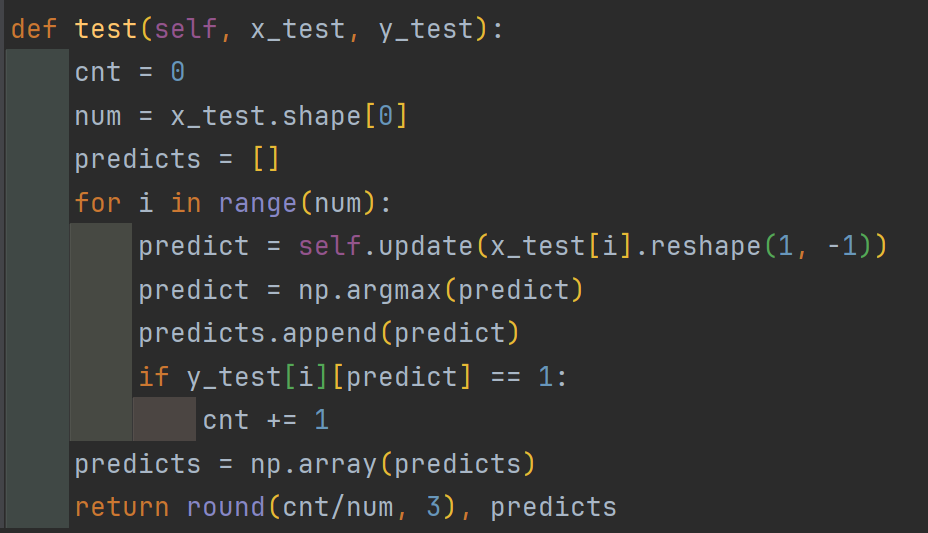
接着是backpropagation函数，用来进行反向传播：



然后是train函数，用来在训练集上训练模型：

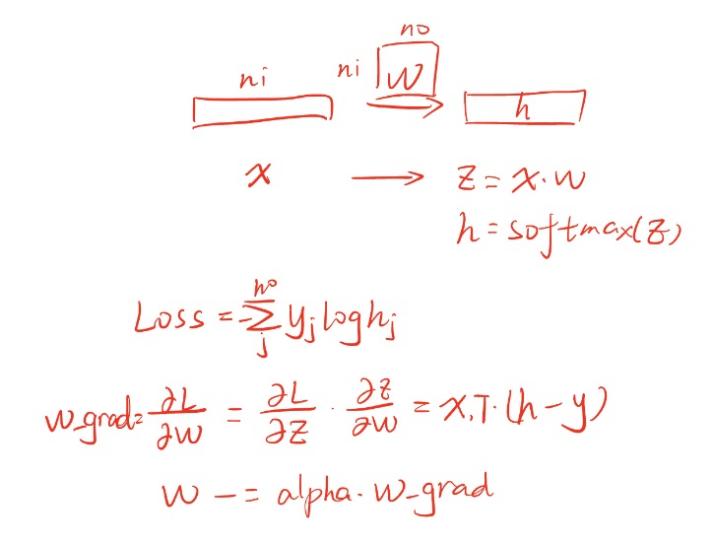


最后是test函数，用来在测试集上测试模型：



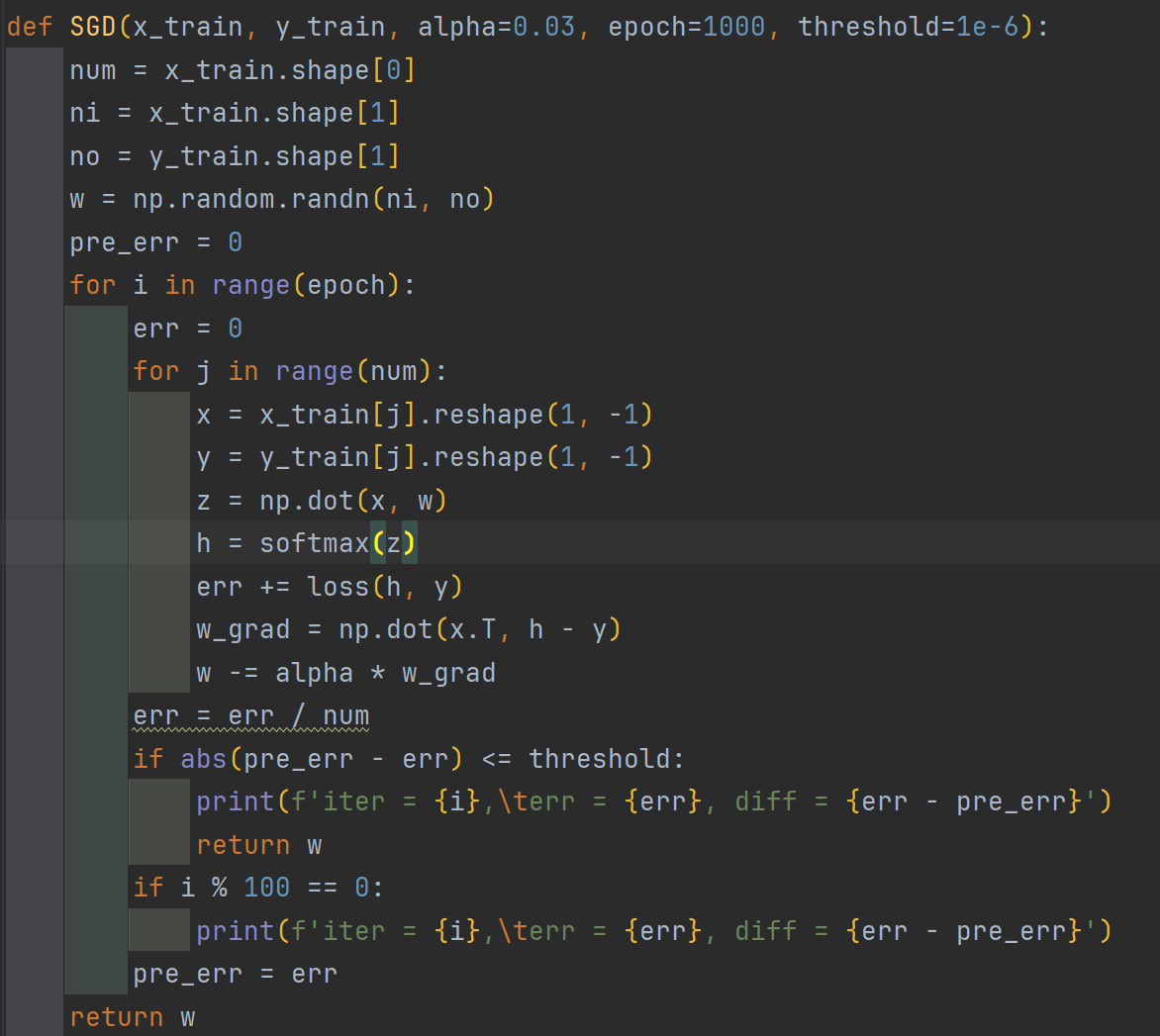
1. 线性分类器

线性分类器结构如下：

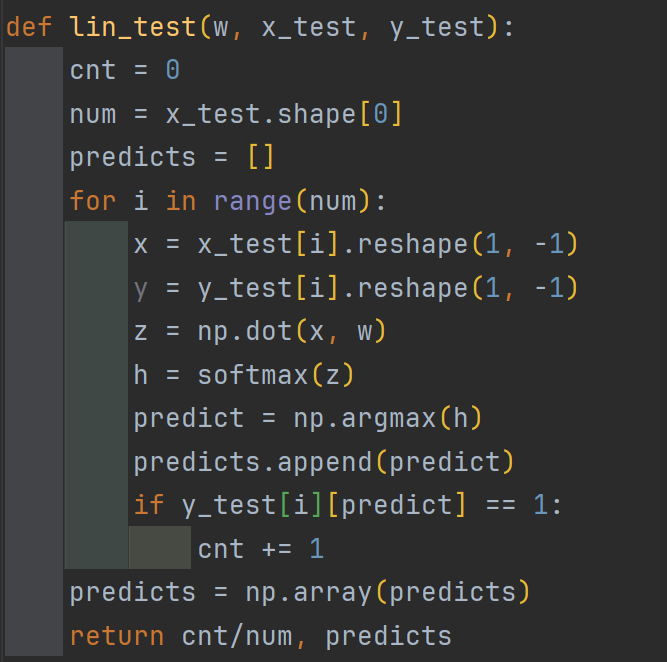


如上图所示，本次实验输入层由输入数据的阶数决定，输出层由分类的种类数量决定，由于是多分类，所以用的激活函数为softmax函数，损失函数在这里采用交叉熵损失，优化方法为随机梯度下降（SGD）。

在这里主要实现的函数为SGD函数，用来进行随机梯度下降：

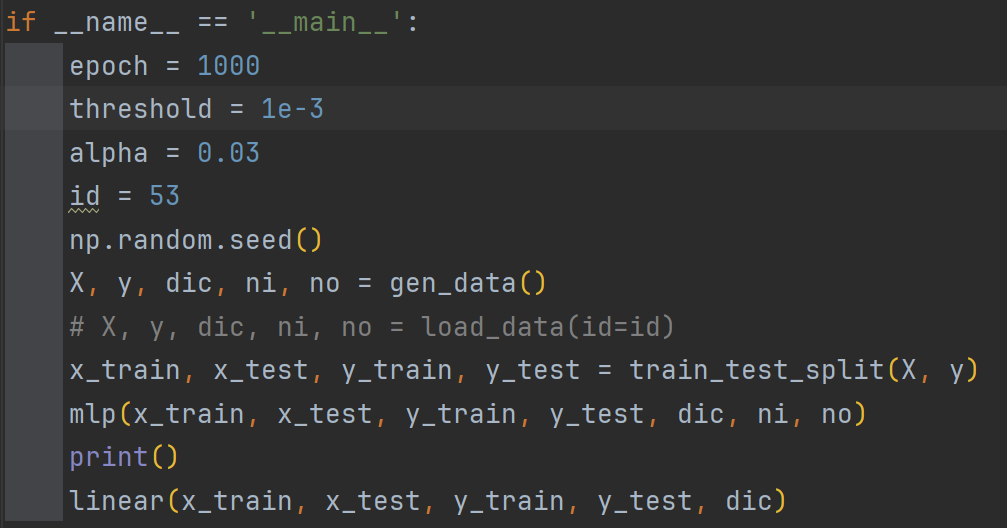


接着用lin\_test函数来进行测试：



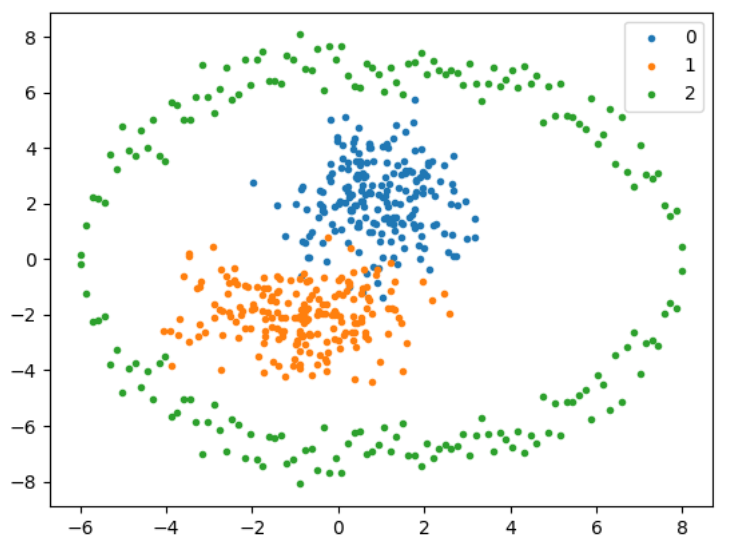
1. 主函数

主函数如下：

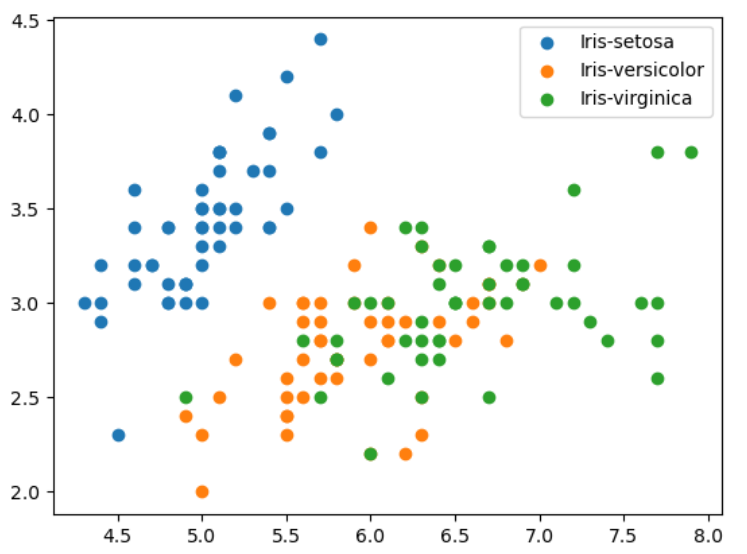


### 实验测试

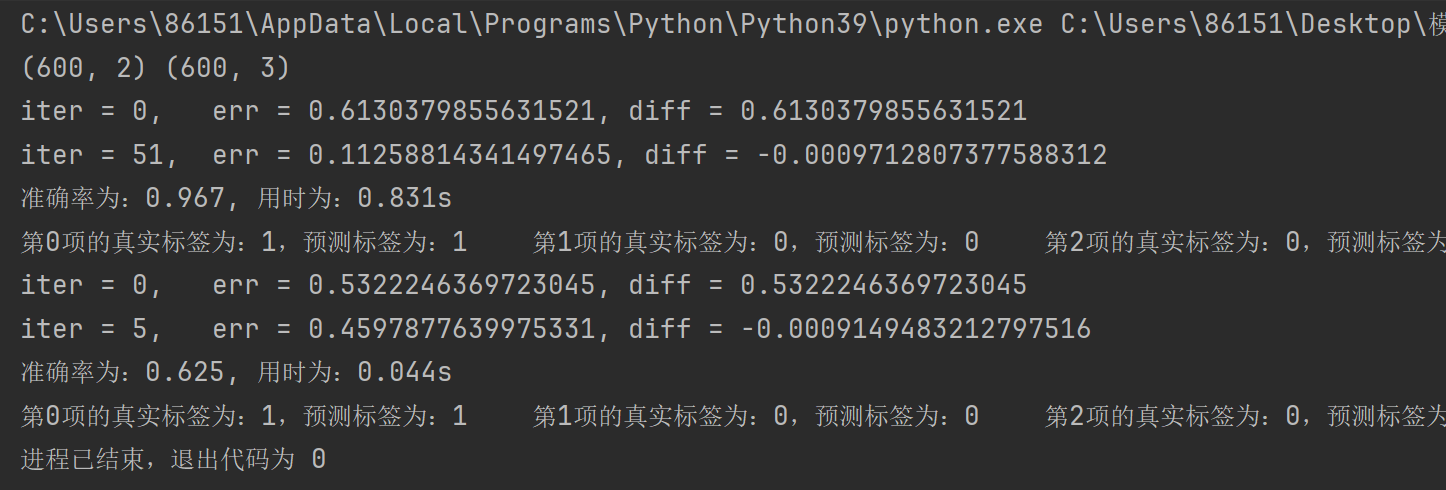
自己生成的数据集如下图所示：



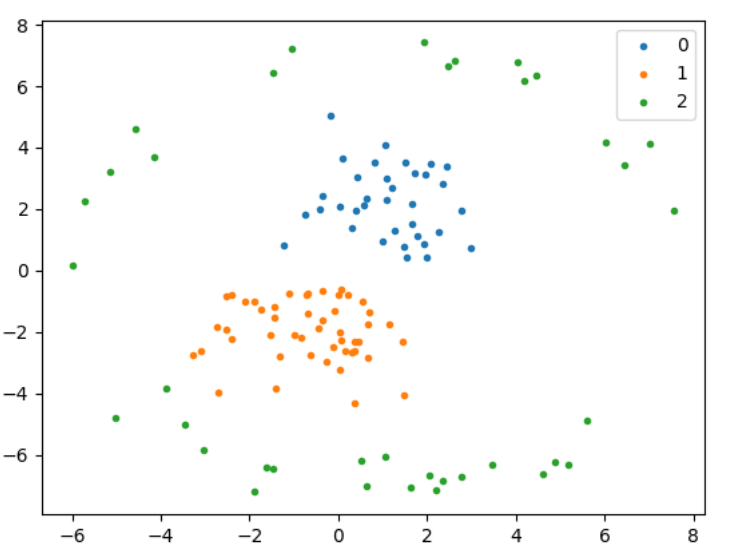
iris原始数据集如下图所示：



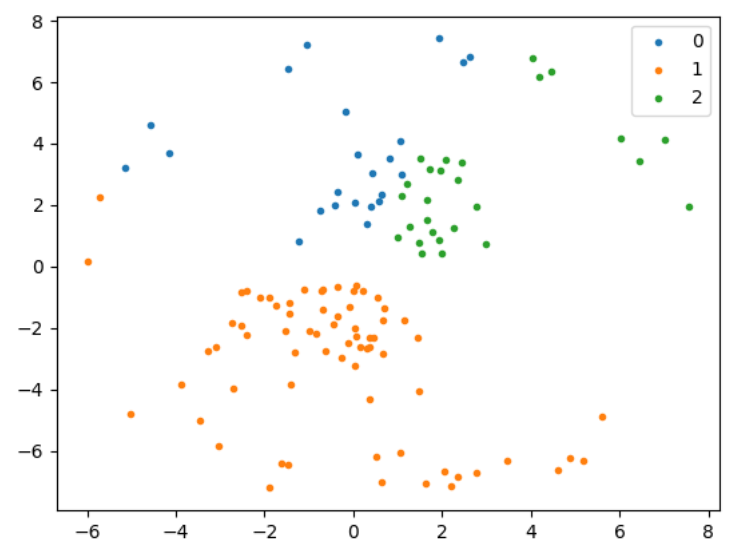
1. 在自己生成的数据集上



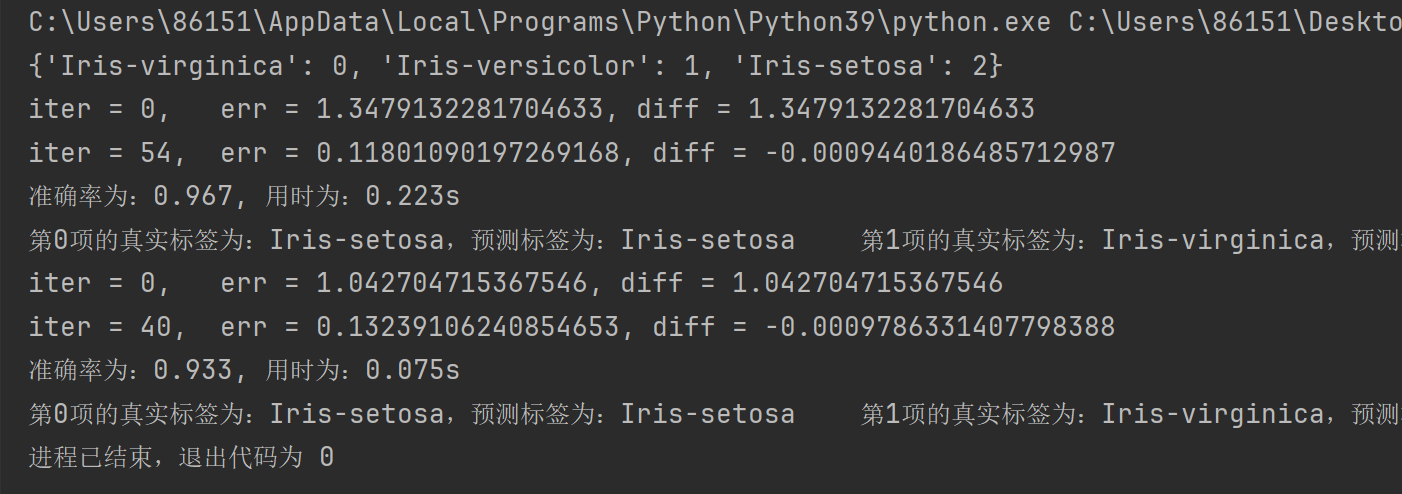
多层感知机分类可视化结果：



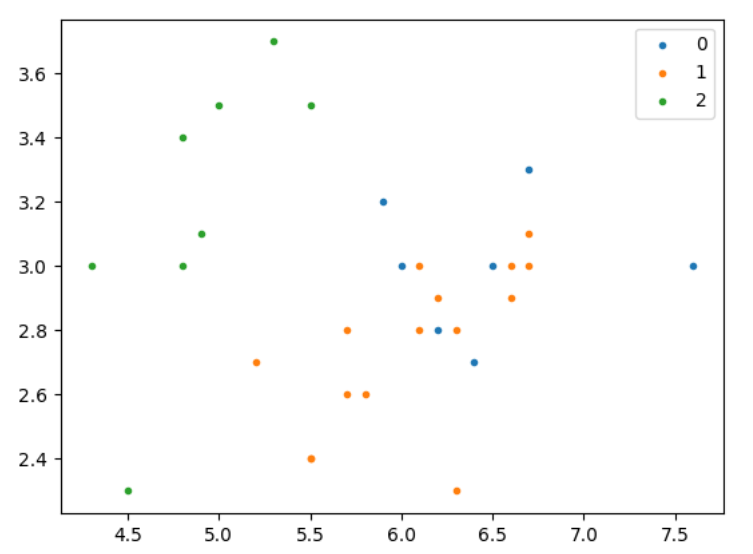
线性分类器可视化结果：



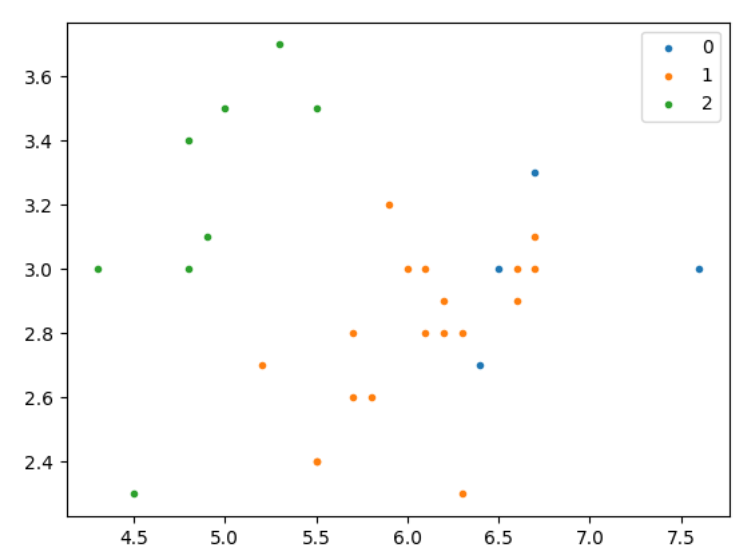
1. 在iris数据集上



多层感知机分类可视化结果：

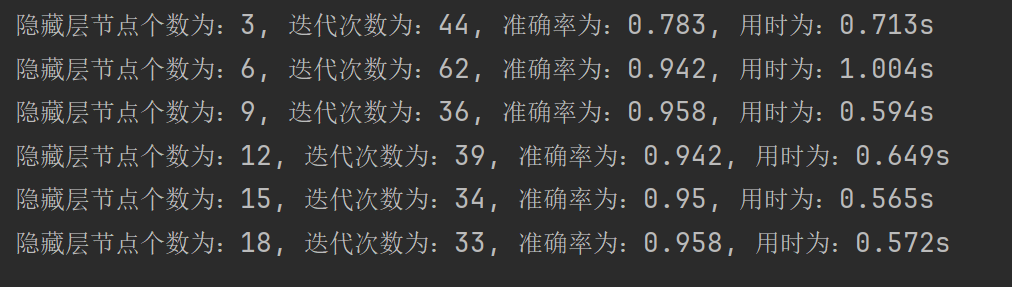


线性分类器可视化结果：



从上述实验中可得出结论，多层感知机相比线性分类器花费的时间虽然长了点，但是换来的是准确率的极大提升；线性分类器在线性可分的数据集上效果较好，但在非线性可分的数据集上的效果远不如多层感知机；多层感知机损失函数在收敛的时候的值远小于线性分类器，分类效果好。

1. 超参数测试
2. 隐藏层节点个数



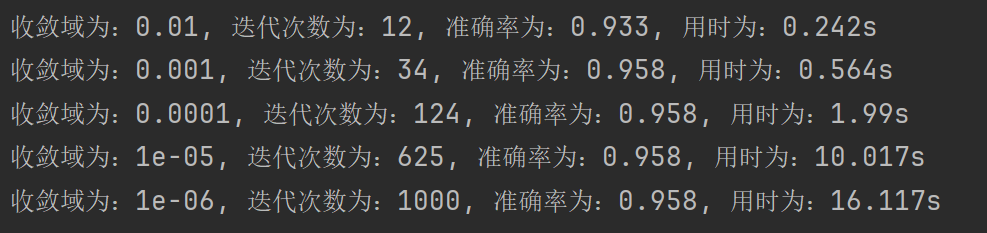
从上图可以看出，隐藏层节点不是越多越好，而是要通过测试来进行选取最佳的节点个数。在这里当隐藏层节点数为9的时候效果最好。

1. 学习率



从上图可以看出，学习率越小，迭代次数越多，训练时间越长，训练效果越好；学习率偏大的时候容易出现震荡的现象，虽然用时短了，但是训练效果偏差。在这里学习率取0.03最好。

1. 收敛条件



从上图可以看出，收敛域越小，迭代次数越多，训练时间越长，但是准确率并没有上升，这是因为产生了过拟合的现象，所以收敛域不是越小越好；但收敛域大了也会导致准确率下降。这里收敛域取0.001最好。

## 实验总体结论

1. 多层感知机相比线性分类器花费的时间虽然长了点，但是换来的是准确率的极大提升；线性分类器在线性可分的数据集上效果较好，但在非线性可分的数据集上的效果远不如多层感知机；多层感知机损失函数在收敛的时候的值远小于线性分类器，分类效果好。
2. 隐藏层节点不是越多越好，而是要通过测试来进行选取最佳的节点个数。
3. 学习率越小，迭代次数越多，训练时间越长，训练效果越好；学习率偏大的时候容易出现震荡的现象，虽然用时短了，但是训练效果偏差。
4. 收敛域越小，迭代次数越多，训练时间越长，但是准确率并没有上升，这是因为产生了过拟合的现象，所以收敛域不是越小越好；但收敛域大了也会导致准确率下降。

## 完整实验代码

main.py

1. """
2. @Filename: .py
3. @Author: Keyan Xu
4. @Time: 2023-10-18
5. """
6. **import** time
7. **from** multilayer\_perceptron **import** \*
8. **from** linear\_classfication **import** \*
9. **from** process\_data **import** \*
10. **import** random
11. **import** numpy as np
12. **from** ucimlrepo **import** fetch\_ucirepo

15. **def** mlp(x\_train, x\_test, y\_train, y\_test, dic, ni, no, nh=10, epoch=1000, threshold=1e-3, alpha=0.03):
16. start = time.time()
17. n = NN(ni, nh, no)
18. iter = n.train(x\_train, y\_train, epoch=epoch, threshold=threshold, alpha=alpha)
19. end = time.time()
20. cost = round(end - start, 3)
21. acc, predicts = n.test(x\_test, y\_test)
22. **print**(f'准确率为：{round(acc, 3)}, 用时为：{cost}s')
23. predict\_show(predicts, x\_test, y\_test, dic)
24. **print**()
25. **return** acc, cost, iter

28. **def** linear(x\_train, x\_test, y\_train, y\_test, dic):
29. start = time.time()
30. w = SGD(x\_train, y\_train, epoch=epoch, threshold=threshold, alpha=alpha)
31. end = time.time()
32. acc, predicts = lin\_test(w, x\_test, y\_test)
33. **print**(f'准确率为：{round(acc, 3)}, 用时为：{round(end - start, 3)}s')
34. predict\_show(predicts, x\_test, y\_test, dic)

37. **if** \_\_name\_\_ == '\_\_main\_\_':
38. epoch = 1000
39. threshold = 1e-3
40. alpha = 0.03
41. id = 53
42. np.random.seed()
44. X, y, dic, ni, no = gen\_data()
45. x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y)
46. mlp(x\_train, x\_test, y\_train, y\_test, dic, ni, no)
47. linear(x\_train, x\_test, y\_train, y\_test, dic)
49. **print**()
51. X, y, dic, ni, no = load\_data(id=id)
52. x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y)
53. linear(x\_train, x\_test, y\_train, y\_test, dic)
54. mlp(x\_train, x\_test, y\_train, y\_test, dic, ni, no)

process\_data.py

1. # --\*-- coding:utf-8 --\*--
2. """
3. @Filename: .py
4. @Author: Keyan Xu
5. @Time: 2023-10-20
6. """
7. **import** numpy as np
8. **from** matplotlib **import** pyplot as plt
9. **from** ucimlrepo **import** fetch\_ucirepo

12. **def** gen\_circle(num=200, r=7, a=1, b=0):
13. x11 = np.linspace(a - r, a + r, int(num / 2)).reshape(-1, 1)
14. noise = np.random.normal(0, 0.6, int(num / 2)).reshape(-1, 1)
15. x21 = (r \*\* 2 - (x11 - a) \*\* 2) \*\* 0.5 + b
16. x21 += noise
17. x1 = np.hstack((x11, x21))
18. x2 = np.hstack((x11, -x21))
19. X1 = np.vstack((x1, x2))
20. **return** X1

23. **def** gen\_data(num=200, plot=True, k=3):
24. mu = np.array([[1, 2], [-1, -2]])
25. cov = np.array([[[1, 0], [0, 2]], [[2, 0], [0, 1]]])
26. X1 = gen\_circle(num)
27. X2 = np.zeros(((k-1) \* num, 2))
28. **for** i **in** range(k-1):
29. X2[i \* num:(i + 1) \* num, :] = np.random.multivariate\_normal(mu[i], cov[i, :, :], num)
30. **if** plot:
31. **for** i **in** range(k-1):
32. plt.scatter(X2[i \* num:(i + 1) \* num, 0], X2[i \* num:(i + 1) \* num, 1], marker='.', label=i)
33. plt.scatter(X1[:, 0], X1[:, 1], label=2, marker='.')
34. plt.legend()
35. plt.show()
37. X = np.vstack((X2, X1))
38. dic = {}
39. y = np.zeros((k \* num, k))
40. # print(X.shape, y.shape)
41. **for** i **in** range(k):
42. **for** j **in** range(num):
43. y[i\*num+j, i] = 1
44. dic[i] = i
45. X, y = shuffle\_index(num\*k, X, y)
46. ni = 2
47. no = k
48. **return** X, y, dic, ni, no

51. **def** relabel\_y(y):
52. num = y.shape[0]
53. label\_dic = {}
54. cnt = 0
56. **for** i **in** range(num):
57. label = str(y[i, 0])
58. **if** label **not** **in** label\_dic:
59. label\_dic[label] = cnt
60. cnt += 1
61. **print**(label\_dic)
63. m = len(label\_dic)
64. index\_y = np.zeros((num, m))
65. **for** i **in** range(num):
66. label\_index = label\_dic[str(y[i, 0])]
67. index\_y[i, label\_index] = 1
68. # print(index\_y)
69. **return** index\_y, label\_dic

72. **def** train\_test\_split(X, y, k=0.8):
73. num = y.shape[0]
74. split = int(k \* num)
75. x\_train = X[:split]
76. y\_train = y[:split]
77. x\_test = X[split:]
78. y\_test = y[split:]
79. # x\_test = X
80. # y\_test = y
81. # print(x\_test, y\_test.shape[0])
82. **return** x\_train, x\_test, y\_train, y\_test

85. **def** load\_data(id=53):
86. # 从uci获取iris数据集
87. iris = fetch\_ucirepo(id=id)
88. # 数据（pd.dataframe格式）
89. X = np.array(iris.data.features)
90. y = np.array(iris.data.targets)
91. num = X.shape[0]
92. X, y = shuffle\_index(num, X, y)
93. y, dic = relabel\_y(y)
94. ni = X.shape[1]
95. no = y.shape[1]
96. **return** X, y, dic, ni, no

99. **def** shuffle\_index(num, X, y):
100. shuffled\_index = np.random.permutation(num)
101. X = X[shuffled\_index]
102. y = y[shuffled\_index]
103. **return** X, y

hyper\_para\_test.py

1. # --\*-- coding:utf-8 --\*--
2. """
3. @Filename: .py
4. @Author: Keyan Xu
5. @Time: 2023-10-20
6. """
7. **from** multilayer\_perceptron **import** \*
8. **from** linear\_classfication **import** \*
9. **from** process\_data **import** \*
10. **from** main **import** mlp
11. **import** random
12. **import** numpy as np

15. **def** test\_nh():
16. **for** nh **in** nh\_list:
17. acc\_i, cost\_i, iter\_i = mlp(x\_train, x\_test, y\_train, y\_test, dic, ni, no, nh)
18. acc.append(acc\_i)
19. cost.append(cost\_i)
20. iter.append(iter\_i)
21. **for** i **in** range(len(acc)):
22. **print**(f'隐藏层节点个数为：{nh\_list[i]}, 迭代次数为：{iter[i]}, '
23. f'准确率为：{acc[i]}, 用时为：{cost[i]}s')

26. **def** test\_alpha():
27. **for** alpha **in** alpha\_list:
28. acc\_i, cost\_i, iter\_i = mlp(x\_train, x\_test, y\_train, y\_test, dic, ni, no, alpha=alpha)
29. acc.append(acc\_i)
30. cost.append(cost\_i)
31. iter.append(iter\_i)
32. **for** i **in** range(len(acc)):
33. **print**(f'学习率为：{round(alpha\_list[i], 2)}, 迭代次数为：{iter[i]}, '
34. f'准确率为：{acc[i]}, 用时为：{cost[i]}s')

37. **def** test\_threshold():
38. **for** threshold **in** thre\_list:
39. acc\_i, cost\_i, iter\_i = mlp(x\_train, x\_test, y\_train, y\_test, dic, ni, no, threshold=threshold)
40. acc.append(acc\_i)
41. cost.append(cost\_i)
42. iter.append(iter\_i)
43. **for** i **in** range(len(acc)):
44. **print**(f'收敛域为：{thre\_list[i]}, 迭代次数为：{iter[i]}, '
45. f'准确率为：{acc[i]}, 用时为：{cost[i]}s')

48. **if** \_\_name\_\_ == '\_\_main\_\_':
49. np.random.seed()
50. X, y, dic, ni, no = gen\_data()
51. x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y)
52. nh\_list = np.arange(3, 20, 3)
53. alpha\_list = np.arange(0.01, 0.09, 0.01)
54. thre\_list = [1e-2, 1e-3, 1e-4, 1e-5, 1e-6]
55. acc = []
56. cost = []
57. iter = []
58. # test\_nh()
59. # test\_alpha()
60. test\_threshold()

multilayer\_perceptron.py

1. # --\*-- coding:utf-8 --\*--
2. """
3. @Filename: .py
4. @Author: Keyan Xu
5. @Time: 2023-10-19
6. """
7. **import** numpy as np
8. **from** matplotlib **import** pyplot as plt

11. **def** softmax(z):
12. # 计算总和
13. sum\_exp = np.sum(np.exp(z), axis=1, keepdims=True)
14. softmax\_z = np.exp(z) / sum\_exp
15. **return** softmax\_z

18. **def** loss(h, y):
19. l = -np.sum(y \* np.log(h), axis=1)
20. **return** l[0]

23. **def** sigmoid(z):
24. h = 1 / (1 + np.exp(-z))
25. **return** h

28. # sigmoid函数求导
29. **def** dsigmoid(h):
30. **return** h \* (1 - h)

33. **class** NN:
34. **def** \_\_init\_\_(self, ni, nh, no):
35. # 输入层、隐藏层、输出层的节点（数）
36. self.ni = ni + 1  # 增加一个偏差节点
37. self.nh = nh
38. self.no = no
40. # 激活神经网络的所有节点
41. self.x = np.ones((1, self.ni))
42. # print(self.x)
43. self.h1 = np.ones((1, self.nh))
44. self.h2 = np.ones((1, self.no))
46. # 建立权重（矩阵）, 设为随机值
47. self.w1 = np.random.randn(self.ni, self.nh)
48. self.w2 = np.random.randn(self.nh, self.no)
50. **def** update(self, input\_x):
51. # print(input\_x)
52. self.x = np.hstack((input\_x, np.array([[1.0]])))
53. z1 = np.dot(self.x, self.w1)
54. self.h1 = sigmoid(z1)
55. z2 = np.dot(self.h1, self.w2)
56. self.h2 = softmax(z2)
57. **return** self.h2
59. **def** back\_propagation(self, input\_y, alpha):
60. z2\_grad = self.h2 - input\_y
61. w2\_grad = np.dot(self.h1.T, z2\_grad)
63. h1\_grad = np.dot(z2\_grad, self.w2.T)
64. z1\_grad = h1\_grad \* dsigmoid(self.h1)
65. w1\_grad = np.dot(self.x.T, z1\_grad)
67. self.w2 -= alpha \* w2\_grad
68. self.w1 -= alpha \* w1\_grad
70. err = loss(self.h2, input\_y)
71. **return** err
73. **def** train(self, x\_train, y\_train, epoch=30000, alpha=0.03, threshold=1e-6):
74. num = x\_train.shape[0]
75. pre\_err = 0
76. **for** i **in** range(epoch):
77. err = 0
78. **for** j **in** range(num):
79. self.update(x\_train[j].reshape(1, -1))
80. err += self.back\_propagation(y\_train[j].reshape(1, -1), alpha)
81. err = err / num
82. **if** abs(pre\_err - err) <= threshold:
83. **print**(f'iter = {i},\terr = {err}, diff = {err - pre\_err}')
84. **return** i
85. **if** i % 100 == 0:
86. **print**(f'iter = {i},\terr = {err}, diff = {err - pre\_err}')
87. pre\_err = err
88. **return** epoch
90. **def** test(self, x\_test, y\_test):
91. cnt = 0
92. num = x\_test.shape[0]
93. predicts = []
94. **for** i **in** range(num):
95. predict = self.update(x\_test[i].reshape(1, -1))
96. predict = np.argmax(predict)
97. predicts.append(predict)
98. **if** y\_test[i][predict] == 1:
99. cnt += 1
100. predicts = np.array(predicts)
101. **return** round(cnt/num, 3), predicts

104. **def** predict\_show(predicts, x\_test, y\_test, dic):
105. num = predicts.shape[0]
106. k = y\_test.shape[1]
107. y = np.argmax(y\_test, axis=1)
108. # print(y)
109. predict\_label = []
110. true\_label = []
111. clusters = []
112. **for** i **in** range(k):
113. clusters.append([])
114. **for** i **in** range(num):
115. **for** key, value **in** dic.items():
116. **if** value == y[i]:
117. true\_label.append(key)
118. **if** value == predicts[i]:
119. predict\_label.append(key)
120. clusters[predicts[i]].append(x\_test[i])
121. **for** i **in** range(num):
122. **print**(f'第{i}项的真实标签为：{true\_label[i]}，预测标签为：{predict\_label[i]}', end='\t')
123. **print**()
124. **for** i **in** range(k):
125. cluster = np.array(clusters[i])
126. plt.scatter(cluster[:, 0], cluster[:, 1], label=i, marker='.')
127. plt.legend()
128. plt.show()
129. **return** predict\_label

linear\_classfication.py

1. # --\*-- coding:utf-8 --\*--
2. """
3. @Filename: .py
4. @Author: Keyan Xu
5. @Time: 2023-10-19
6. """
7. # --\*-- coding:utf-8 --\*--
8. """
9. @Filename: .py
10. @Author: Keyan Xu
11. @Time: 2023-10-19
12. """
13. **import** numpy as np
14. **from** multilayer\_perceptron **import** softmax, loss

17. **def** SGD(x\_train, y\_train, alpha=0.03, epoch=1000, threshold=1e-6):
18. num = x\_train.shape[0]
19. ni = x\_train.shape[1]
20. no = y\_train.shape[1]
21. w = np.random.randn(ni, no)
22. pre\_err = 0
23. **for** i **in** range(epoch):
24. err = 0
25. **for** j **in** range(num):
26. x = x\_train[j].reshape(1, -1)
27. y = y\_train[j].reshape(1, -1)
28. z = np.dot(x, w)
29. h = softmax(z)
30. err += loss(h, y)
31. w\_grad = np.dot(x.T, h - y)
32. w -= alpha \* w\_grad
33. err = err / num
34. **if** abs(pre\_err - err) <= threshold:
35. **print**(f'iter = {i},\terr = {err}, diff = {err - pre\_err}')
36. **return** w
37. **if** i % 100 == 0:
38. **print**(f'iter = {i},\terr = {err}, diff = {err - pre\_err}')
39. pre\_err = err
40. **return** w

43. **def** lin\_test(w, x\_test, y\_test):
44. cnt = 0
45. num = x\_test.shape[0]
46. predicts = []
47. **for** i **in** range(num):
48. x = x\_test[i].reshape(1, -1)
49. y = y\_test[i].reshape(1, -1)
50. z = np.dot(x, w)
51. h = softmax(z)
52. predict = np.argmax(h)
53. predicts.append(predict)
54. **if** y\_test[i][predict] == 1:
55. cnt += 1
56. predicts = np.array(predicts)
57. **return** cnt/num, predicts

## 参考文献

无