《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

1. 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。

2. 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。

3. 实验环境

Windows10; python3.9;PyCharm 2021.2.2

4. 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）

(1) 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：

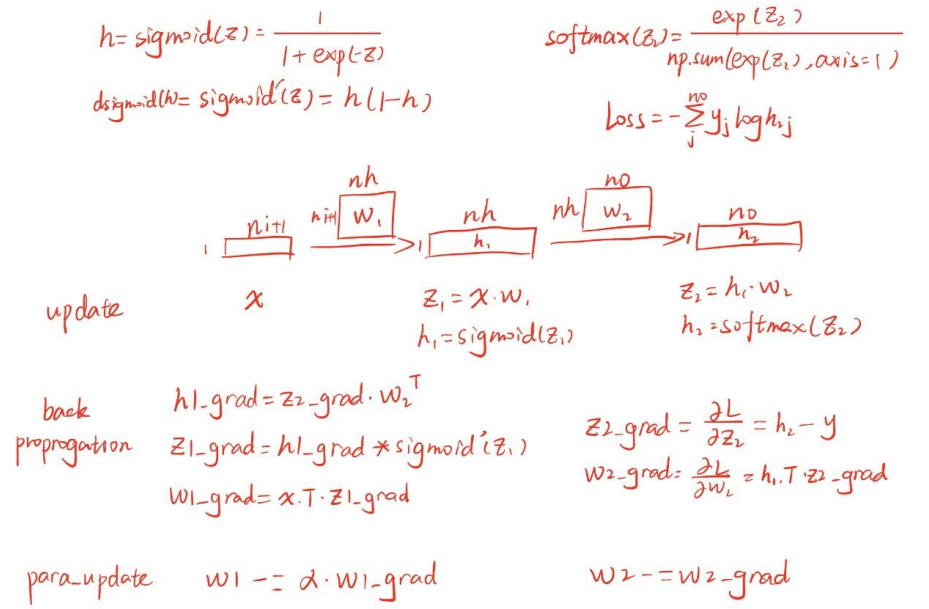
1) 输入层：多层感知机的输入层接收来自问题域的特征向量。每个输入特征通常表示为一个神经元。输入特征被传递到网络的下一层，即隐藏层。
2) 隐藏层：多层感知机可以有一个或多个隐藏层，每个隐藏层由多个神经元组成。每个神经元都连接到前一层的所有神经元，并且每个连接都有一个权重，用于调整输入的重要性。每个神经元还具有一个偏置项，用于调整神经元的激活阈值。隐藏层的目标是学习输入数据中的非线性关系，以便更好地进行特征提取和表示学习。
3) 输出层：输出层通常包含一个或多个神经元，每个神经元对应于模型的不同类别（分类问题）或一个连续的值（回归问题）。输出层的每个神经元会对来自隐藏层的信息进行加权和汇总，并应用一个激活函数，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。

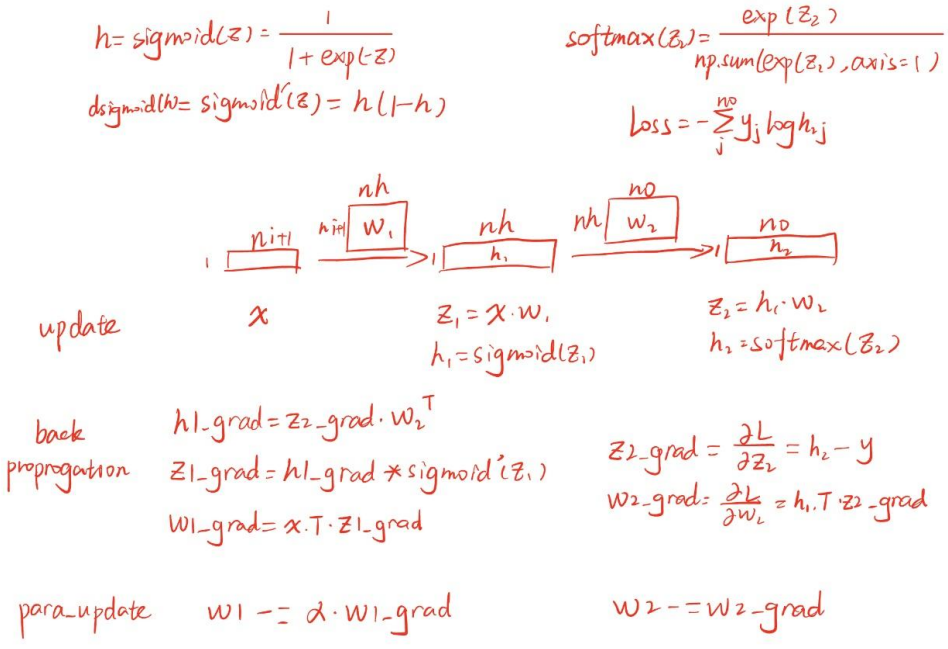
公式部分由下图所示：



(2) 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：

```
def update(self, input_x):
    # print(input_x)
    self.x = np.hstack((input_x, np.array([[1.0]])))
    z1 = np.dot(self.x, self.w1)
    self.h1 = sigmoid(z1)
    z2 = np.dot(self.h1, self.w2)
    self.h2 = softmax(z2)
    return self.h2
```

接着是 backpropagation 函数，用来进行反向传播：

```
def back_propagation(self, input_y, alpha):
    z2_grad = self.h2 - input_y
    w2_grad = np.dot(self.h1.T, z2_grad)

    h1_grad = np.dot(z2_grad, self.w2.T)
    z1_grad = h1_grad * dsigmoid(self.h1)
    w1_grad = np.dot(self.x.T, z1_grad)

    self.w2 -= alpha * w2_grad
    self.w1 -= alpha * w1_grad

    err = loss(self.h2, input_y)
    return err
```

然后是 train 函数，用来在训练集上训练模型：

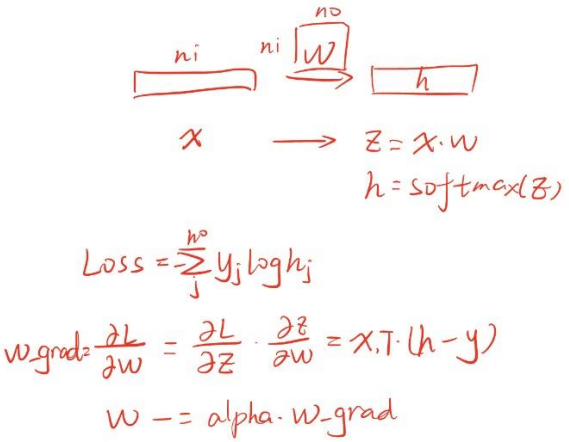
```
def train(self, x_train, y_train, epoch=30000, alpha=0.03, threshold=1e-6):
    num = x_train.shape[0]
    pre_err = 0
    for i in range(epoch):
        err = 0
        for j in range(num):
            self.update(x_train[j].reshape(1, -1))
            err += self.back_propagation(y_train[j].reshape(1, -1), alpha)
        err = err / num
        if abs(pre_err - err) <= threshold:
            print(f'iter = {i},\terr = {err}, diff = {err - pre_err}')
            return i
        if i % 100 == 0:
            print(f'iter = {i},\terr = {err}, diff = {err - pre_err}')
        pre_err = err
    return epoch
```

最后是 test 函数，用来在测试集上测试模型：

```
def test(self, x_test, y_test):
    cnt = 0
    num = x_test.shape[0]
    predicts = []
    for i in range(num):
        predict = self.update(x_test[i].reshape(1, -1))
        predict = np.argmax(predict)
        predicts.append(predict)
        if y_test[i][predict] == 1:
            cnt += 1
    predicts = np.array(predicts)
    return round(cnt/num, 3), predicts
```

2) 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：

```python
def SGD(x_train, y_train, alpha=0.03, epoch=1000, threshold=1e-6):
    num = x_train.shape[0]
    ni = x_train.shape[1]
    no = y_train.shape[1]
    w = np.random.randn(ni, no)
    pre_err = 0
    for i in range(epoch):
        err = 0
        for j in range(num):
            x = x_train[j].reshape(1, -1)
            y = y_train[j].reshape(1, -1)
            z = np.dot(x, w)
            h = softmax(z)
            err += loss(h, y)
            w_grad = np.dot(x.T, h - y)
            w -= alpha * w_grad
        err = err / num
        if abs(pre_err - err) <= threshold:
            print(f'iter = {i},\terr = {err}, diff = {err - pre_err}')
            return w
        if i % 100 == 0:
            print(f'iter = {i},\terr = {err}, diff = {err - pre_err}')
        pre_err = err
    return w
```

接着用 lin_test 函数来进行测试：

```python
def lin_test(w, x_test, y_test):
    cnt = 0
    num = x_test.shape[0]
    predicts = []
    for i in range(num):
        x = x_test[i].reshape(1, -1)
        y = y_test[i].reshape(1, -1)
        z = np.dot(x, w)
        h = softmax(z)
        predict = np.argmax(h)
        predicts.append(predict)
        if y_test[i][predict] == 1:
            cnt += 1
    predicts = np.array(predicts)
    return cnt/num, predicts
```

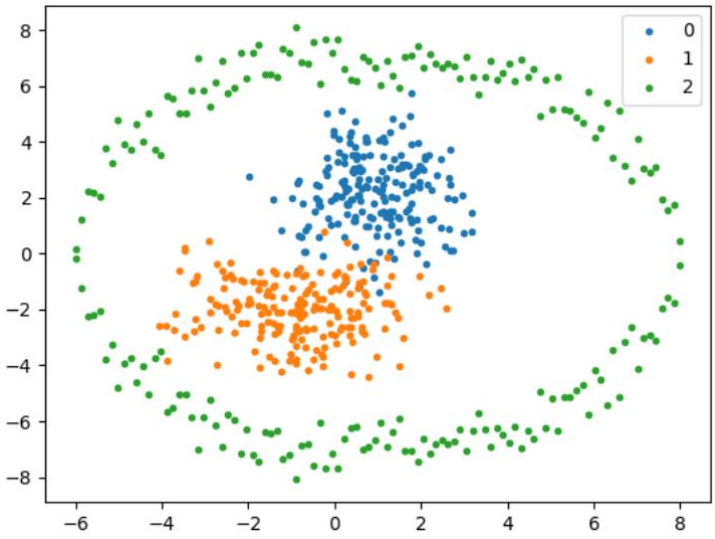
3) 主函数

主函数如下：

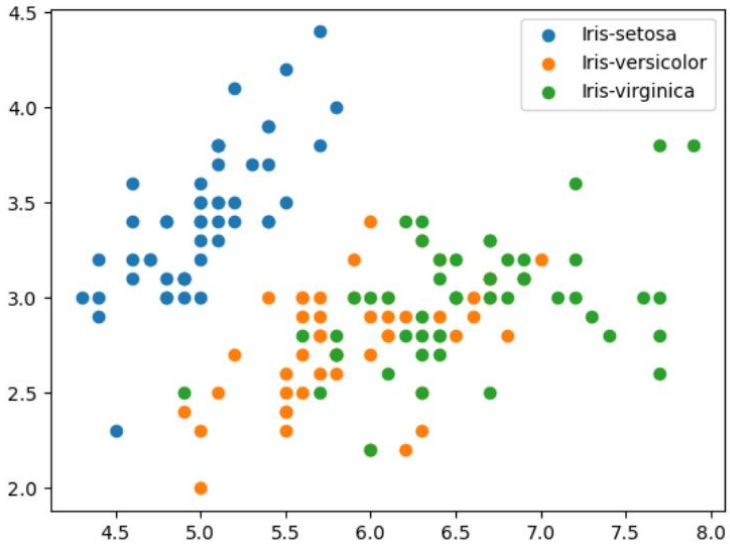
```python
if __name__ == '__main__':
    epoch = 1000
    threshold = 1e-3
    alpha = 0.03
    id = 53
    np.random.seed()
    X, y, dic, ni, no = gen_data()
    # X, y, dic, ni, no = load_data(id=id)
    x_train, x_test, y_train, y_test = train_test_split(X, y)
    mlp(x_train, x_test, y_train, y_test, dic, ni, no)
    print()
    linear(x_train, x_test, y_train, y_test, dic)
```

(3) 实验测试

自己生成的数据集如下图所示：



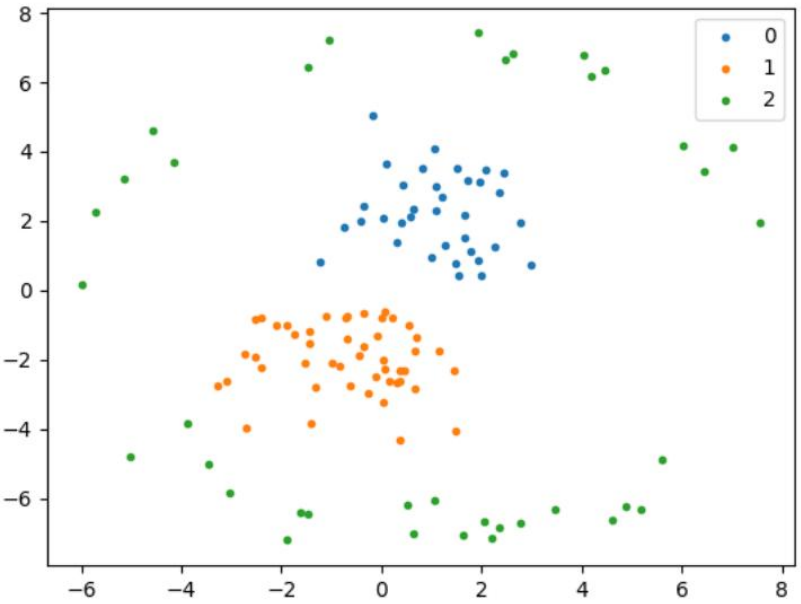
iris 原始数据集如下图所示：



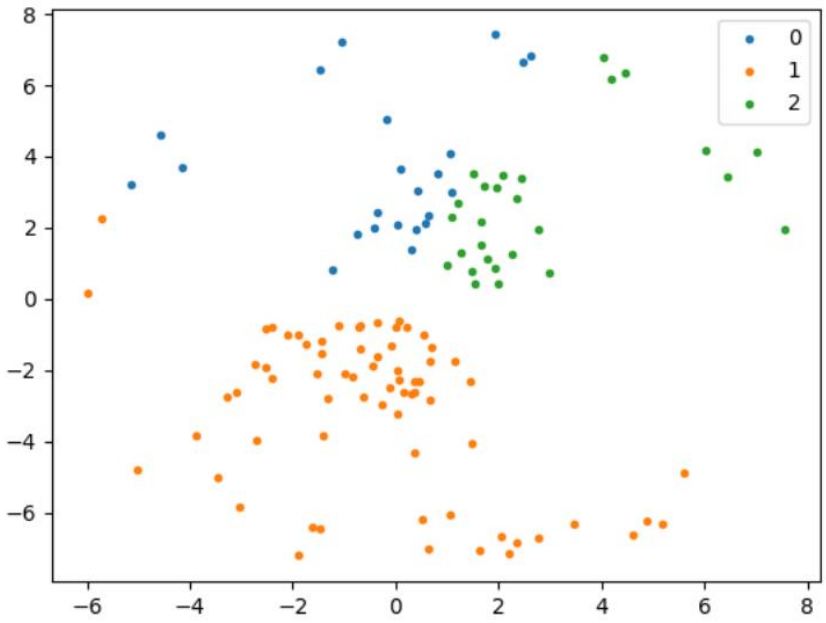
1) 在自己生成的数据集上

```
C:\Users\86151\AppData\Local\Programs\Python\Python39\python.exe C:\Users\86151\Desktop\模
(600, 2) (600, 3)
iter = 0,   err = 0.6130379855631521, diff = 0.6130379855631521
iter = 51,  err = 0.11258814341497465, diff = -0.0009712807377588312
准确率为：0.967，用时为：0.831s
第0项的真实标签为：1，预测标签为：1    第1项的真实标签为：0，预测标签为：0    第2项的真实标签为：0，预测标签为
iter = 0,   err = 0.5322246369723045, diff = 0.5322246369723045
iter = 5,   err = 0.4597877639975331, diff = -0.0009149483212797516
准确率为：0.625，用时为：0.044s
第0项的真实标签为：1，预测标签为：1    第1项的真实标签为：0，预测标签为：0    第2项的真实标签为：0，预测标签为
进程已结束，退出代码为 0
```

多层感知机分类可视化结果：



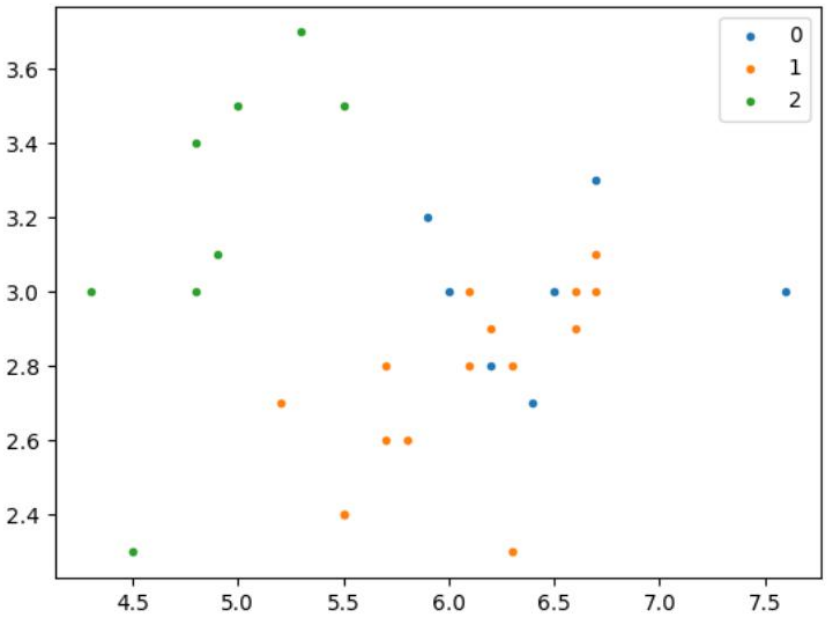
线性分类器可视化结果：



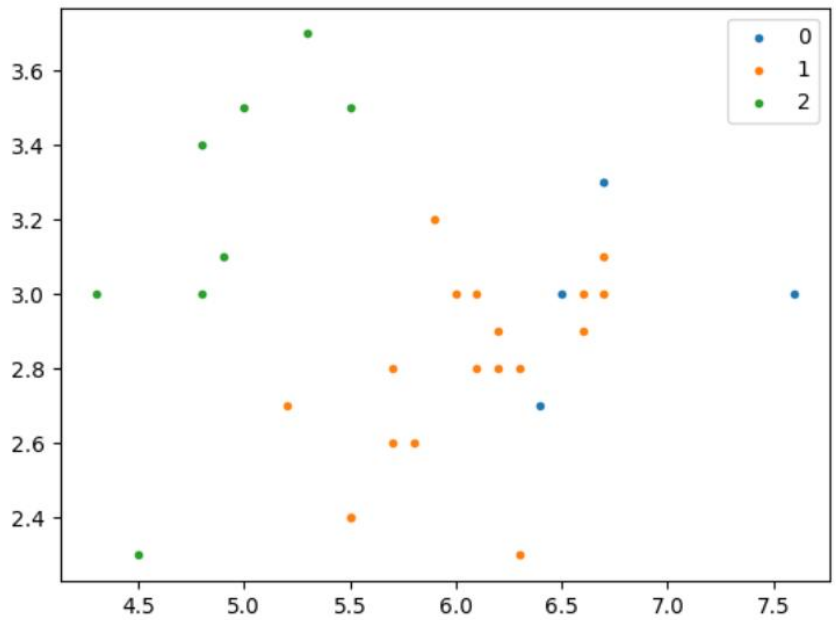
2) 在 iris 数据集上

```
C:\Users\86151\AppData\Local\Programs\Python\Python39\python.exe C:\Users\86151\Deskto
{'Iris-virginica': 0, 'Iris-versicolor': 1, 'Iris-setosa': 2}
iter = 0,   err = 1.3479132281704633, diff = 1.3479132281704633
iter = 54,  err = 0.11801090197269168, diff = -0.0009440186485712987
准确率为：0.967，用时为：0.223s
第0项的真实标签为：Iris-setosa，预测标签为：Iris-setosa    第1项的真实标签为：Iris-virginica，预测
iter = 0,   err = 1.042704715367546, diff = 1.042704715367546
iter = 40,  err = 0.13239106240854653, diff = -0.0009786331407798388
准确率为：0.933，用时为：0.075s
第0项的真实标签为：Iris-setosa，预测标签为：Iris-setosa    第1项的真实标签为：Iris-virginica，预测
进程已结束，退出代码为 0
```

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。

3) 超参数测试

a) 隐藏层节点个数

```
隐藏层节点个数为：3，迭代次数为：44，准确率为：0.783，用时为：0.713s
隐藏层节点个数为：6，迭代次数为：62，准确率为：0.942，用时为：1.004s
隐藏层节点个数为：9，迭代次数为：36，准确率为：0.958，用时为：0.594s
隐藏层节点个数为：12，迭代次数为：39，准确率为：0.942，用时为：0.649s
隐藏层节点个数为：15，迭代次数为：34，准确率为：0.95，用时为：0.565s
隐藏层节点个数为：18，迭代次数为：33，准确率为：0.958，用时为：0.572s
```

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。

b) 学习率

```
学习率为：0.01，迭代次数为：79，准确率为：0.992，用时为：1.306s
学习率为：0.02，迭代次数为：56，准确率为：0.983，用时为：0.928s
学习率为：0.03，迭代次数为：37，准确率为：0.992，用时为：0.621s
学习率为：0.04，迭代次数为：45，准确率为：0.975，用时为：0.751s
学习率为：0.05，迭代次数为：35，准确率为：0.95，用时为：0.591s
学习率为：0.06，迭代次数为：29，准确率为：0.975，用时为：0.506s
学习率为：0.07，迭代次数为：34，准确率为：0.992，用时为：0.594s
学习率为：0.08，迭代次数为：20，准确率为：0.95，用时为：0.344s
```

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。

c) 收敛条件

```
收敛域为：0.01，迭代次数为：12，准确率为：0.933，用时为：0.242s
收敛域为：0.001，迭代次数为：34，准确率为：0.958，用时为：0.564s
收敛域为：0.0001，迭代次数为：124，准确率为：0.958，用时为：1.99s
收敛域为：1e-05，迭代次数为：625，准确率为：0.958，用时为：10.017s
收敛域为：1e-06，迭代次数为：1000，准确率为：0.958，用时为：16.117s
```

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。

5. 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。

6. 完整实验代码

main.py

```
"""
@Filename: .py
@Author: Keyan Xu
@Time: 2023-10-18
"""
import time
from multilayer_perceptron import *
from linear_classfication import *
from process_data import *
import random
import numpy as np
from ucimlrepo import fetch_ucirepo


def mlp(x_train, x_test, y_train, y_test, dic, ni, no, nh=10, epoch=1000, threshold=1e-3, alpha=0.03):
    start = time.time()
    n = NN(ni, nh, no)
    iter = n.train(x_train, y_train, epoch=epoch, threshold=threshold, alpha=alpha)
    end = time.time()
    cost = round(end - start, 3)
    acc, predicts = n.test(x_test, y_test)
    print(f'准确率为：{round(acc, 3)}, 用时为：{cost}s')
    predict_show(predicts, x_test, y_test, dic)
    print()
    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=thresh
    old, 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)
35.
36.
37. if __name__ == '__main__':
38.     epoch = 1000
39.     threshold = 1e-3
40.     alpha = 0.03
41.     id = 53
42.     np.random.seed()
43.
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)
48.
49.     print()
50.
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
10.
11.
```

```
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
21.
22.
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.multivaria
    te_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()
36.
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
49.
50.
```

```
51. def relabel_y(y):
52.     num = y.shape[0]
53.     label_dic = {}
54.     cnt = 0
55.
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)
62.
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
70.
71.
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
83.
84.
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]
```

```
    no = y.shape[1]
    return X, y, dic, ni, no


def shuffle_index(num, X, y):
     shuffled_index = np.random.permutation(num)
     X = X[shuffled_index]
     y = y[shuffled_index]
     return X, y
```

hyper_para_test.py

```
# --*-- coding:utf-8 --*--
"""
@Filename: .py
@Author: Keyan Xu
@Time: 2023-10-20
"""
from multilayer_perceptron import *
from linear_classfication import *
from process_data import *
from main import mlp
import random
import numpy as np


def test_nh():
    for nh in nh_list:
        acc_i, cost_i, iter_i = mlp(x_train, x_test, y_train, y_test, dic, ni, no, nh)
        acc.append(acc_i)
        cost.append(cost_i)
        iter.append(iter_i)
    for i in range(len(acc)):
        print(f'隐藏层节点个数为：{nh_list[i]}, 迭代次数为：{iter[i]}, '
              f'准确率为：{acc[i]}, 用时为：{cost[i]}s')


def test_alpha():
    for alpha in alpha_list:
        acc_i, cost_i, iter_i = mlp(x_train, x_test, y_train, y_test, dic, ni, no, alpha=alpha)
        acc.append(acc_i)
        cost.append(cost_i)
```

```
        iter.append(iter_i)
    for i in range(len(acc)):
        print(f'学习率为：{round(alpha_list[i], 2)}, 迭代次数为：{iter[i]}, '
              f'准确率为：{acc[i]}, 用时为：{cost[i]}s')


def test_threshold():
    for threshold in thre_list:
        acc_i, cost_i, iter_i = mlp(x_train, x_test, y_train, y_test, dic, ni, no, threshold=threshold)
        acc.append(acc_i)
        cost.append(cost_i)
        iter.append(iter_i)
    for i in range(len(acc)):
        print(f'收敛域为：{thre_list[i]}, 迭代次数为：{iter[i]}, '
              f'准确率为：{acc[i]}, 用时为：{cost[i]}s')


if __name__ == '__main__':
    np.random.seed()
    X, y, dic, ni, no = gen_data()
    x_train, x_test, y_train, y_test = train_test_split(X, y)
    nh_list = np.arange(3, 20, 3)
    alpha_list = np.arange(0.01, 0.09, 0.01)
    thre_list = [1e-2, 1e-3, 1e-4, 1e-5, 1e-6]
    acc = []
    cost = []
    iter = []
    # test_nh()
    # test_alpha()
    test_threshold()
```

multilayer_perceptron.py

```
# --*-- coding:utf-8 --*--
"""
@Filename: .py
@Author: Keyan Xu
@Time: 2023-10-19
"""
import numpy as np
from matplotlib import pyplot as plt
```

```
9.
10.
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
16.
17.
18. def loss(h, y):
19.     l = -np.sum(y * np.log(h), axis=1)
20.     return l[0]
21.
22.
23. def sigmoid(z):
24.     h = 1 / (1 + np.exp(-z))
25.     return h
26.
27.
28. # sigmoid 函数求导
29. def dsigmoid(h):
30.     return h * (1 - h)
31.
32.
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
39.
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))
45.
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)
49.
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
58.
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)
62.
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)
66.
67.         self.w2 -= alpha * w2_grad
68.         self.w1 -= alpha * w1_grad
69.
70.         err = loss(self.h2, input_y)
71.         return err
72.
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].res
    hape(1, -1), alpha)
81.             err = err / num
82.             if abs(pre_err - err) <= threshold:
83.                 print(f'iter = {i},\terr = {err}, diff = {e
    rr - pre_err}')
84.                 return i
85.             if i % 100 == 0:
86.                 print(f'iter = {i},\terr = {err}, diff = {e
    rr - pre_err}')
87.             pre_err = err
88.         return epoch
89.
90.     def test(self, x_test, y_test):
91.         cnt = 0
92.         num = x_test.shape[0]
```

```
        predicts = []
        for i in range(num):
            predict = self.update(x_test[i].reshape(1, -1))

            predict = np.argmax(predict)
            predicts.append(predict)
            if y_test[i][predict] == 1:
                cnt += 1
        predicts = np.array(predicts)
        return round(cnt/num, 3), predicts


def predict_show(predicts, x_test, y_test, dic):
    num = predicts.shape[0]
    k = y_test.shape[1]
    y = np.argmax(y_test, axis=1)
    # print(y)
    predict_label = []
    true_label = []
    clusters = []
    for i in range(k):
        clusters.append([])
    for i in range(num):
        for key, value in dic.items():
            if value == y[i]:
                true_label.append(key)
            if value == predicts[i]:
                predict_label.append(key)
                clusters[predicts[i]].append(x_test[i])
    for i in range(num):
        print(f'第{i}项的真实标签为：{true_label[i]}，预测标签为：{predict_label[i]}', end='\t')
    print()
    for i in range(k):
        cluster = np.array(clusters[i])
        plt.scatter(cluster[:, 0], cluster[:, 1], label=i, marker='.')
    plt.legend()
    plt.show()
    return predict_label
```

linear_classfication.py

```
# --*-- coding:utf-8 --*--
"""
```

```
@Filename: .py
@Author: Keyan Xu
@Time: 2023-10-19
"""
# --*-- coding:utf-8 --*--
"""
@Filename: .py
@Author: Keyan Xu
@Time: 2023-10-19
"""
import numpy as np
from multilayer_perceptron import softmax, loss


def SGD(x_train, y_train, alpha=0.03, epoch=1000, threshold=1e-6):
    num = x_train.shape[0]
    ni = x_train.shape[1]
    no = y_train.shape[1]
    w = np.random.randn(ni, no)
    pre_err = 0
    for i in range(epoch):
        err = 0
        for j in range(num):
            x = x_train[j].reshape(1, -1)
            y = y_train[j].reshape(1, -1)
            z = np.dot(x, w)
            h = softmax(z)
            err += loss(h, y)
            w_grad = np.dot(x.T, h - y)
            w -= alpha * w_grad
        err = err / num
        if abs(pre_err - err) <= threshold:
            print(f'iter = {i},\terr = {err}, diff = {err - pre_err}')
            return w
        if i % 100 == 0:
            print(f'iter = {i},\terr = {err}, diff = {err - pre_err}')
        pre_err = err
    return w


def lin_test(w, x_test, y_test):
```

```
    cnt = 0
    num = x_test.shape[0]
    predicts = []
    for i in range(num):
        x = x_test[i].reshape(1, -1)
        y = y_test[i].reshape(1, -1)
        z = np.dot(x, w)
        h = softmax(z)
        predict = np.argmax(h)
        predicts.append(predict)
        if y_test[i][predict] == 1:
            cnt += 1
    predicts = np.array(predicts)
    return cnt/num, predicts
```

7. 参考文献

无