认知科学与类脑计算

实验报告二

日期：2019/5/17

班级：16 人工智能

姓名：贾乘兴

学号：201600301304

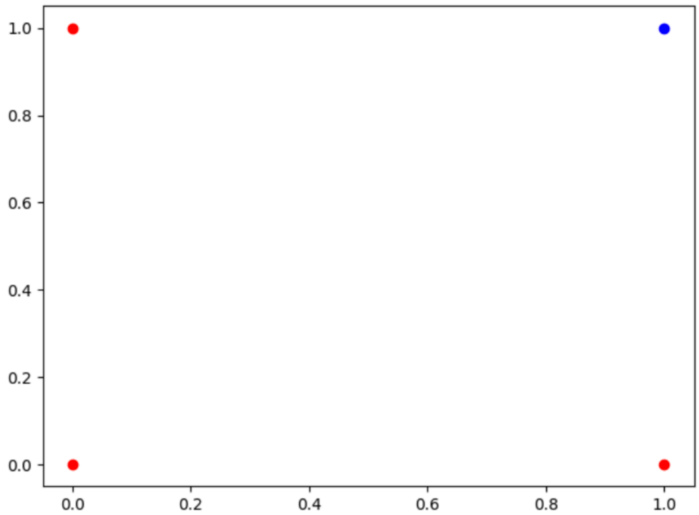
一.需求分析：

感知器，也可翻译为感知机，是 Frank Rosenblatt 在 1957 年就职于 Cornell 航空实验室(Cornell Aeronautical Laboratory)时所发明的一种人工神经网络。它可以被视为一种最简单形式的前馈式人工神经网络，是一种二元线性分类器。感知器是生物神经细胞的简单抽象，神经细胞结构大致可分为：树突、突触、细胞体及轴突。单个神经细胞可被视为一种只有两种状态的机器——激动时为‘是’，而未激动时为‘否’。神经细胞的状态取决于从其它的神经细胞收到的输入信号量，及突触的强度（抑制或加强）。当信号量总和超过了某个阈值时，细胞体就会激动，产生电脉冲。电脉冲沿着轴突并通过突触传递到其它神经元。为了模拟神经细胞行为，与之对应的感知机基础概念被提出，如权量（突触）、偏置（阈值）及激活函数（细胞体）。本次实验的目的是加深对感知器模型的理解，能够使用感知器模型解决简单的分类问题。根据感知器的相关知识，使用 Python 语言实现一个简单的感知器模型，该模型能够实现简单的二分类任务（与或非运算任选一）。

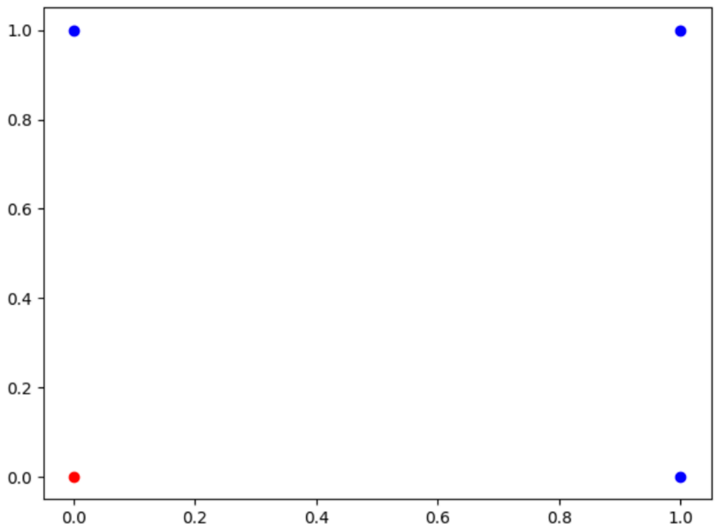
二.概要设计：

基于与或非运算的真值表构建训练数据，将训练数据存在列表中，创建简单的感知机网络，并使用测试数据集测试网络的运算输出。

与运算：二维数据点，二分类



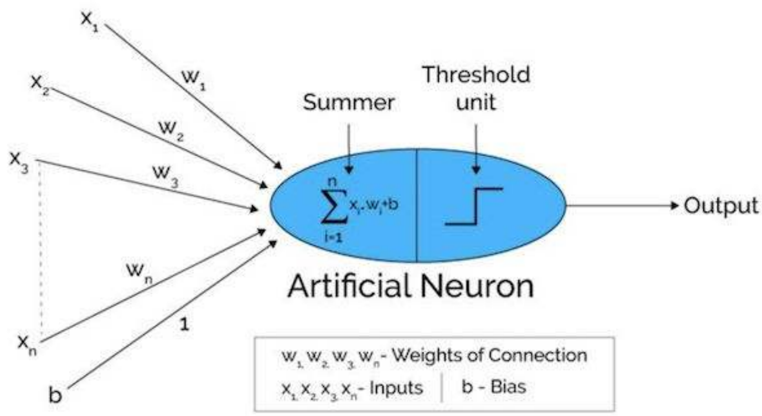
或运算：二维数据点，二分类



非运算：一维数据点，二分类

三.详细设计：

1.单层感知机模型



对于第 t 个样本，计算公式如下

$$y^t = \operatorname{sgn}\left(\sum_{i=1}^{n} w_i x_i^t + b\right)$$

$$\operatorname{sgn}(x) = \begin{cases} 1, x > 0 \\ 0, x \le 0 \end{cases}$$

在训练过程中，我们对参数 w 和 b 进行更新，更新的法则是，如果对该样本预测正确，则不对参数进行更新，如果对该样本预测错误，则我们根据梯度方向对每个参数进行更行。

$$bias = t - y$$

$$w_i := w_i + \eta \cdot bias \cdot \frac{\partial \sum_{j=1}^{n} w_j x_j + b}{\partial w_i}$$

$$b := b + \eta \cdot bias \cdot \frac{\partial \sum_{j=1}^{n} w_j x_j + b}{\partial b}$$

整理得到最终的表达式为

$$w_i := w_i + \eta \cdot bias \cdot x_i$$
$$b := b + \eta \cdot bias$$

该方法可视为简单的 bp 算法，如果我们在最后一层不使用二值化的激活函数，则可进行反向传播，定义均方误差为损失函数，可以得到与上式类似的结果

2.数据集设计

与运算：

input:　[[1,1],[1,0],[0,1],[0,1]]

label:　[1,0,0,0]

或运算：

input:　[[1,1],[1,0],[0,1],[0,1]]

label:　[1,1,1,0]

非运算：

input:　[1,0]

label:　[0,1]

3.收敛性

我们已知数据集线性可分，则必然存在一个超平面 Wopt，可以将所有的样本正确的分类，对于我们的目标 t 和计算得到的结果 y，都存在其乘积大于 0

取乘积中最小的为 r，且令 R=最大范数的样本，则可证明迭代次数 k 满足以下

$$k \le \left(\frac{R}{r}\right)^2$$
$$r = \min\left\{t_i\left(w_{opt}x_i\right)\right\}$$
$$R = \max\left\|x_i\right\|$$

最终得到以下不等式

$$|knr| \le w_k \cdot w_{opt} \le \left\|w_k\right\| \cdot \left\|w_{opt}\right\| \le \sqrt{k}nR$$

且我们可以证明 w 在更新过程中的单调性，再加之有界性，我们可以证明其收敛性

四.调试分析：

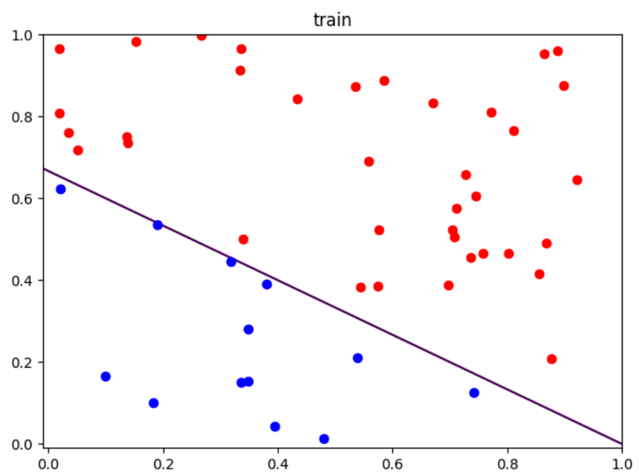
在该模型中，由于样本只有四个，所以解空间是一个可数集，由该算法与 bp 的区别可见，在迭代过程中，只要将样本全部分对，则停止更新，所以最终的结果与初始化与更新的参数关系较大，是一个不稳定的结果。虽然结果的数值不稳定，但是收敛性十分稳定，证明见上。

建立一个新的随机线性可分数据集的数据集，$w_1x_1+w_2x_2+b=0$，然后验证其在数值上的不稳定性。

我们设置参数如下，作为分界线

```
w1 = 0.4
w2 = 0.6
b = -0.4
```

生成 50 个随机数据点，绘制图像如下

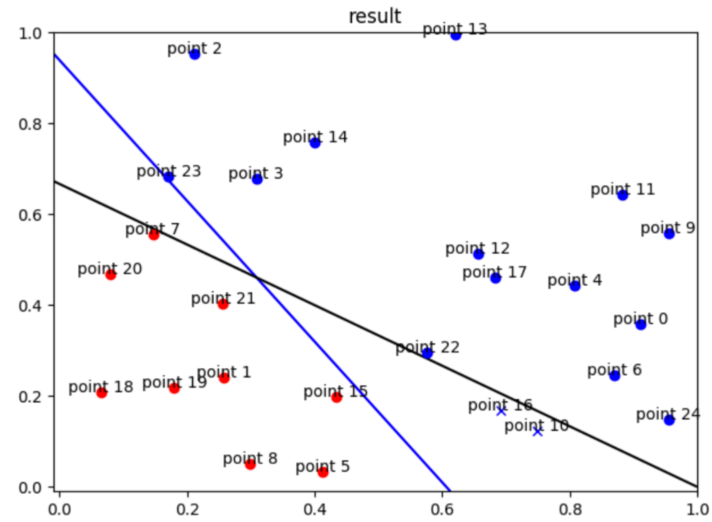


迭代 10 次，学习率为 0.01，最终得到数值如下

w1=0.60995907 w2=0.39397893 b=-0.37

随机生成 25 个数据点测试，绘制图像如下

其中黑色直线为原直线，蓝色直线为训练直线，蓝色圆圈为测试正确的正例，红色圆圈点为测试正确的负例，蓝色 x 为测试错误的负例，红色 x 为测试错误的正例，绘制图像如下

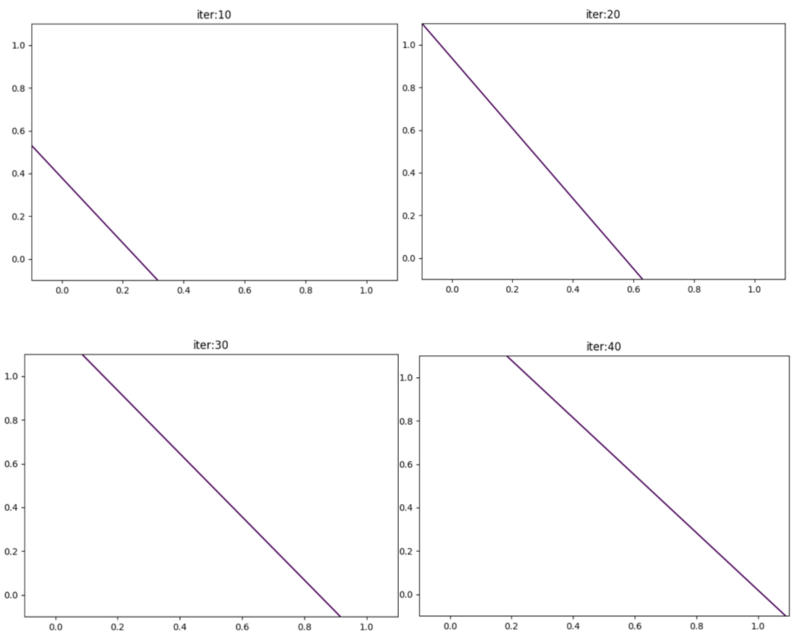


所以，只要测试集分类全部正确，就停止更新，所以只是得到了解空间中取决于初始化和学习率的解，并没有筛选一个最优的解，该算法较 bp 相比存在却缺陷，但其解释性与收敛性是一个优点。

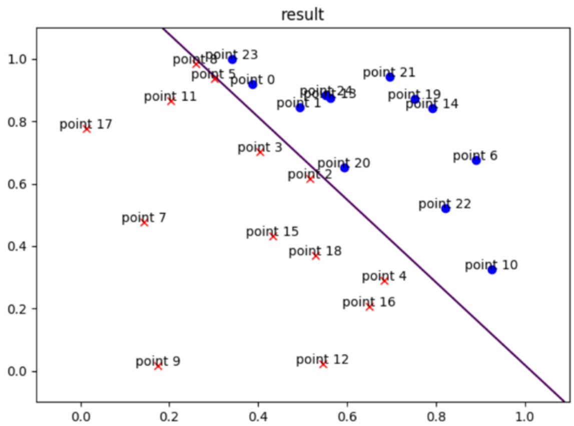
五.测试结果：

与运算：

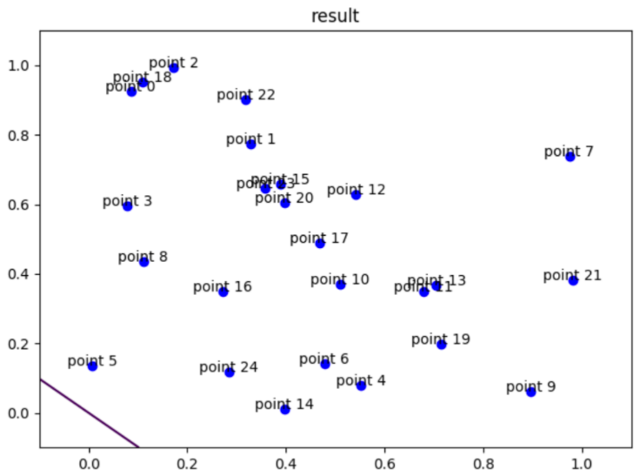
将更新参数设为 0.001，迭代次数为 50



最终迭代完成的结果，我们在 0-1 之间随机采样进行测试（25 点）



我们还使用了或运算数据集进行测试，得到结果如下



六.附录：

附录代码如下 文件为 perception.py

```
import numpy as np
import matplotlib.pyplot as plt

def ac_fun(x):
    return 1 if x>0 else 0

class Perception(object):
    def __init__(self, size):
        self.weight = np.random.random(size)
        self.bias = 0

    def train(self, input_x, label_y, lr):
        for x, y in zip(input_x, label_y):
```

```
            p = ac_fun(sum(np.array(x) * self.weight) + self.bias)
            dl = y - p
            self.bias += lr * dl
            self.weight += lr * dl * np.array(x)
        return self.weight, self.bias

    def predict(self, input_x):
        input_x = np.array(input_x)
        list_pre = []
        for x in input_x:
            pre = ac_fun(sum(np.array(x) * self.weight) + self.bias)
            list_pre.append(pre)

        return list_pre

def one_zero():
    iter = 50
    pic = 1
    loc = int(iter/pic)
    lr = 1e-2
    input_x = [[1,1],[1,0],[0,1],[0,0]]
    label_y = [1,1,1,0]

    for id, x in enumerate(input_x):
        if label_y[id]==0:
            plt.plot(x[0], x[1], 'ro')
        else:
            plt.plot(x[0], x[1], 'bo')
    plt.show()

    n = 25
    test = np.random.random((n,2))

    p = Perception(2)
    for i in range(iter):
        w, b = p.train(input_x, label_y, lr)
        if ((i+1)%loc==0):
            x1 = np.arange(-0.1, 1.1, .01)
            x2 = np.arange(-0.1, 1.1, .01)
            x1, x2 = np.meshgrid(x1, x2)
            f = x1 * w[0] + x2 * w[1] + b
            plt.contour(x1, x2, f, 0)
            plt.title('iter:{}'.format(i+1))
            plt.show()
```

```
    pre = p.predict(test)
    plt.figure()
    plt.contour(x1, x2, f, 0)
    for id, x in enumerate(test):
        if pre[id]==0:
            plt.plot(x[0], x[1], 'rx')
        else:
            plt.plot(x[0], x[1], 'bo')
        plt.text(x[0], x[1], 'point {}'.format(id),
horizontalalignment="center")
    plt.title('result')
    plt.show()
    print(pre)

def wb_con():
    w1 = 0.4
    w2 = 0.6
    b0 = -0.4
    x1 = np.arange(-0.01, 1.01, .01)
    x2 = np.arange(-0.01, 1.01, .01)
    x1, x2 = np.meshgrid(x1, x2)
    f0 = x1 * w1 + x2 * w2 + b0
    m = 50
    input_x = np.random.random((m,2))
    label_y = np.zeros((m,1))
    for id, x in enumerate(input_x):
        if x[0] * w1 + x[1] * w2 + b0 >=0:
            label_y[id] = 1
            plt.plot(x[0],x[1],'ro')
        else:
            label_y[id] = 0
            plt.plot(x[0],x[1],'bo')

    plt.contour(x1, x2, f0, 0)
    plt.title('train')
    plt.show()

    iter = 8
    lr = 1e-2
    p = Perception(2)
    for i in range(iter):
        w, b = p.train(input_x, label_y, lr)
```

```
    n = 25
    test = np.random.random((n,2))
    pre = p.predict(test)
    f = x1 * w[0] + x2 * w[1] + b
    plt.figure()
    plt.contour(x1, x2, f, 0, colors='blue')
    plt.contour(x1, x2, f0,0, colors='black')
    for id, x in enumerate(test):
        if pre[id]==0:
            if (x[0] * w1 + x[1] * w2 + b0 <0):
                plt.plot(x[0], x[1], 'ro')
            else:
                plt.plot(x[0], x[1], 'rx')
        else:
            if (x[0] * w1 + x[1] * w2 + b0 >=0):
                plt.plot(x[0], x[1], 'bo')
            else:
                plt.plot(x[0], x[1], 'bx')
        plt.text(x[0], x[1], 'point {}'.format(id),
horizontalalignment="center")
    plt.title('result')
    plt.show()
    print(pre)
    print(w,b)


one_zero()
#wb_con()
```