

### Problem 1 .

**Solution:** 记这 10 个点分别为  $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}$

记一开始 3 个聚类分别为  $V_1(6.2, 3.2), V_2(6.6, 3.7), V_3(6.5, 3.0)$

在这中表示下, 通过本地代码给出结果如图

发现: 迭代三次已经完成了 K 聚类任务

```
第1轮
V1包含:x1 x2 x4 x6 x7 x9 x10
V2包含:x5
V3包含:x3 x8
V1中心5.17143 3.17143
V2中心5.5 4.2
V3中心6.45 2.95

第2轮
V1包含:x2 x4 x6 x7
V2包含:x5 x9
V3包含:x1 x3 x8 x10
V1中心4.8 3.05
V2中心5.3 4
V3中心6.2 3.025

第3轮
V1包含:x2 x4 x6 x7
V2包含:x5 x9
V3包含:x1 x3 x8 x10
V1中心4.8 3.05
V2中心5.3 4
V3中心6.2 3.025
```

图 1: solution 1

□

### Problem 2 .

**Solution:** 方法一: 手算:

样本均值恰好为 (0,0), 所以不需要中心化。  $XX^T =$

$$\begin{bmatrix} 2.5 & 1.5 \\ 1.5 & 2.5 \end{bmatrix}$$

所以令  $|\lambda E - XX^T| = \lambda^2 - 5\lambda + 4 = 0$

得  $\lambda_1 = 1, \lambda_2 = 4$ , 而 4 更大, 对应特征向量 (1,1), 标准化为

$W =$

$$\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

计算  $\frac{\text{tr}(W^T XX^T W)}{4}$  得方差是 1。

方法二: 利用 python numpy 在本地实现, 方差为 0.999999... = 1

□

### Problem 3 .

**Solution:** 我认为可以在使得间隔最大化的前提下, 同时追求以下之一的优化目标 (对应各自的做法):

(1) 稳定性、鲁棒性目标

做法: 对输入数据多次进行轻微扰动, 分别观察这几个 margin 相同的超平面分类的正确性, 选取正确性最高的那个

## (2) 维度与效率目标

在 margin 相同的情况下, 我们更喜欢维度比较低的超平面, 这样计算效率高

做法: 筛选维度低的超平面 (法向量有较多分量为 0 的超平面)

## (3) 软间隔情况下追求正确率目标

在不存在一个严格地分离两类数据的情况下, 也可通过软间隔地方法计算 margin (分类错误的点取负值), 我们可以同时追求样本中分类错误的点尽量少的目标

做法: 加入正则化项, 正则化项表示分离错误的点的个数, 我们将其加到目标函数中, 使得它也要比较小, 以此筛选更优的超平面  $\square$

### Problem 4 .

**Solution:** (1) 对梯度消失的理解:

本质: 神经网络层数非常多的情况下, 反向传播中梯度的累乘影响越来越大, 会发生梯度算出来很小很小仿佛消失一样, 导致神经网络参数变化微弱, 使得学习效果很差

具体的例子:

设激活函数为 sigmoid 函数, 则其导数为  $\frac{e^{-x}}{(1+e^{-x})^2}$ , 它严格小于 1

而在层数很多时候, 计算时每经过一层便要乘一次这个小于 1 的导数, 使得梯度算下来非常小

## (2) 缓解方法:

(i) 换新型的激活函数, 函数满足梯度在范围内保持较大, 缓解梯度消失

### • 新型激活函数

$$\begin{aligned}\text{ReLU}(x) &= \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \\ &= \max(0, x), \\ \text{LeakyReLU}(x) &= \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases} \\ &= \max(0, x) + \gamma \min(0, x) \\ \text{ELU}(x) &= \begin{cases} x & \text{if } x > 0 \\ \gamma(\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \\ &= \max(0, x) + \min(0, \gamma(\exp(x) - 1)) \\ \text{softplus}(x) &= \log(1 + \exp(x))\end{aligned}$$

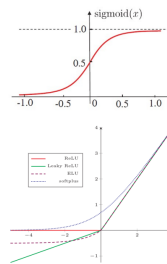


图 2: solution 5

(ii) 进行规范化, 对每层输入进行处理, 使得每层的输入落在激活函数比较敏感 (梯度较大) 的区域

(iii) 可以考虑逐层训练或者少层少层地训练, 这样保证每次独立训练地过程中梯度较大可控

(iv) 梯度裁剪, 在训练的过程中, 严格限制梯度的大小

(v) 权重初始化的时候采用更恰当的方法, 避免初始化为过大或者过小的值  $\square$

### Problem 5 .

**Solution:** (1) 策略迭代

在本地通过代码计算价值函数

```
#include <iostream>
#include <vector>
#include <stack>
#include <string>
#include <list>
#include <cmath>
```

```
#include <math.h>
using namespace std;

int main() {
    double reward[3][3] = { 10,-1,-1,-1,-1,-1,-1,-1,-10 };
    double value[3][3] = { 0 };
    double savevalue[3][3] = { 0 };
    double gamma = 0.9;
    cout << 0 << endl;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << value[i][j] << "␣";
        }
        cout << endl;
    }
    for (int k = 1; k <= 5; k++)
    {
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                savevalue[i][j] = value[i][j];
            }
        }
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                double new_value = 0.0;
                if (i == 0) new_value += (gamma * savevalue[i][j] + reward[i][j]);
                else new_value += (gamma * savevalue[i - 1][j] + reward[i - 1][j]);
                if (i == 2) new_value += (gamma * savevalue[i][j] + reward[i][j]);
                else new_value += (gamma * savevalue[i + 1][j] + reward[i + 1][j]);
                if (j == 0) new_value += (gamma * savevalue[i][j] + reward[i][j]);
                else new_value += (gamma * savevalue[i][j - 1] + reward[i][j - 1]);
                if (j == 2) new_value += (gamma * savevalue[i][j] + reward[i][j]);
                else new_value += (gamma * savevalue[i][j + 1] + reward[i][j + 1]);
                new_value += (gamma * savevalue[i][j] + reward[i][j]);
                new_value /= 5;
                value[i][j] = new_value;
            }
        }
    }
}
```

```
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            value[i][j] = round(value[i][j] * 10) / 10;
        }
    }

    cout << k << endl;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << value[i][j] << " ";
        }
        cout << endl;
    }
}

return 0;
}
```

给出答案如下:

价值函数和策略分别如下:(K 为 0 的时候价值均为 0)(计算过程保留了一位小数 (四舍五入))(○ 表示不动)

```
0
0 0 0
0 0 0
0 0 0
1
5.6 1.2 -1
1.2 -1 -2.8
-1 -2.8 -6.4
2
9.1 2.3 -1.8
2.3 -1.8 -5.3
-1.8 -5.3 -10.9
3
11.3 3 -2.5
3 -2.4 -7.3
-2.5 -7.3 -14.2
4
12.8 3.4 -3.1
3.4 -3 -8.9
-3.1 -8.9 -16.7
5
13.7 3.6 -3.7
3.6 -3.5 -10.1
-3.7 -10.1 -18.6
```

图 3: solution 5-1

策略如下图所示

$\leftrightarrow\updownarrow\bigcirc$	$\leftrightarrow\updownarrow\bigcirc$	$\leftrightarrow\updownarrow\bigcirc$
$\leftrightarrow\updownarrow\bigcirc$	$\leftrightarrow\updownarrow\bigcirc$	$\leftrightarrow\updownarrow\bigcirc$
$\leftrightarrow\updownarrow\bigcirc$	$\leftrightarrow\updownarrow\bigcirc$	$\leftrightarrow\updownarrow\bigcirc$

表 1: 上表  $K=0$ 

$\bigcirc$	$\leftarrow$	$\leftarrow$
$\uparrow$	$\leftarrow\uparrow$	$\leftarrow\uparrow$
$\uparrow$	$\leftarrow\uparrow$	$\leftarrow\uparrow$

表 2: 上表  $K=1$ 

$\bigcirc$	$\leftarrow$	$\leftarrow$
$\uparrow$	$\leftarrow\uparrow$	$\leftarrow\uparrow$
$\uparrow$	$\leftarrow\uparrow$	$\leftarrow\uparrow$

表 3: 上表  $K=2$ 

$\bigcirc$	$\leftarrow$	$\leftarrow$
$\uparrow$	$\leftarrow\uparrow$	$\leftarrow$
$\uparrow$	$\uparrow$	$\leftarrow\uparrow$

表 4: 上表  $K=3$ 

$\bigcirc$	$\leftarrow$	$\leftarrow$
$\uparrow$	$\leftarrow\uparrow$	$\leftarrow$
$\uparrow$	$\uparrow$	$\leftarrow\uparrow$

表 5: 上表  $K=4$ 

$\bigcirc$	$\leftarrow$	$\leftarrow$
$\uparrow$	$\leftarrow\uparrow$	$\leftarrow$
$\uparrow$	$\uparrow$	$\leftarrow\uparrow$

表 6: 上表  $K=5$ 

## (2) 价值迭代

将策略迭代中每个方格价值更新的代码改为  
取五个动作带来的最大值即可

价值和策略分别如下:

```
0
0 0 0
0 0 0
0 0 0
1
10 10 -1
10 -1 -1
-1 -1 -1
2
19 19 8
19 8 -1.9
8 -1.9 -1.9
3
27.1 27.1 16.1
27.1 16.1 6.2
16.1 6.2 -2.7
4
34.4 34.4 23.4
34.4 23.4 13.5
23.4 13.5 4.6
5
41 41 30
41 30 20.1
30 20.1 11.2
```

图 4: solution 5-2

$\leftrightarrow\updownarrow\bigcirc$	$\leftrightarrow\updownarrow\bigcirc$	$\leftrightarrow\updownarrow\bigcirc$
$\leftrightarrow\updownarrow\bigcirc$	$\leftrightarrow\updownarrow\bigcirc$	$\leftrightarrow\updownarrow\bigcirc$
$\leftrightarrow\updownarrow\bigcirc$	$\leftrightarrow\updownarrow\bigcirc$	$\leftrightarrow\updownarrow\bigcirc$

表 7: 上表 K=0

$\bigcirc$	$\leftarrow$	$\leftarrow$
$\uparrow$	$\leftarrow\uparrow$	$\leftrightarrow\updownarrow$
$\uparrow$	$\leftrightarrow\updownarrow$	$\leftrightarrow\updownarrow$

表 8: 上表 K=1

$\bigcirc$	$\leftarrow$	$\leftarrow$
$\uparrow$	$\leftarrow\uparrow$	$\leftarrow\uparrow$
$\uparrow$	$\leftarrow\uparrow$	$\leftrightarrow\updownarrow$

表 9: 上表 K=2

○	←	←
↑	←↑	←↑
↑	←↑	←↑

表 10: 上表 K=3

○	←	←
↑	←↑	←↑
↑	←↑	←↑

表 11: 上表 K=4

○	←	←
↑	←↑	←↑
↑	←↑	←↑

表 12: 上表 K=5

□

此题解完

Problem 6 .

**Solution:** 例如我们小组的 ai 导论期末项目应用的就是强化学习，我们做的是一个东方格斗 ai,MDP 可以如下描述:  
state space: 我们制作的 ai 的血量、蓝量、技能点和位置以及东方游戏自带 ai(电脑对手) 的这些参数等等;  
action space: 保持不动; 进行攻击操作 (包括各种招式和技能); 进行防守; 进行位移 (左右、下蹲和跳跃)  
transition function: 多种多样，可以设置为随机化 (这样不好)，也可以设置为最大化该步获得的奖励，或者最大化全局失败或者胜利的奖励，而奖励的定义也很多样  
reward: 奖励函数和状态下采取行动造成的己方和对方血量变化有关，如果 ai 自己血量减少，给予一个负的奖励；如果对方血量减少，给予一个正的奖励。除此之外，还要考虑获得胜利所需战斗时间、出招耗的蓝量等等方面。 □