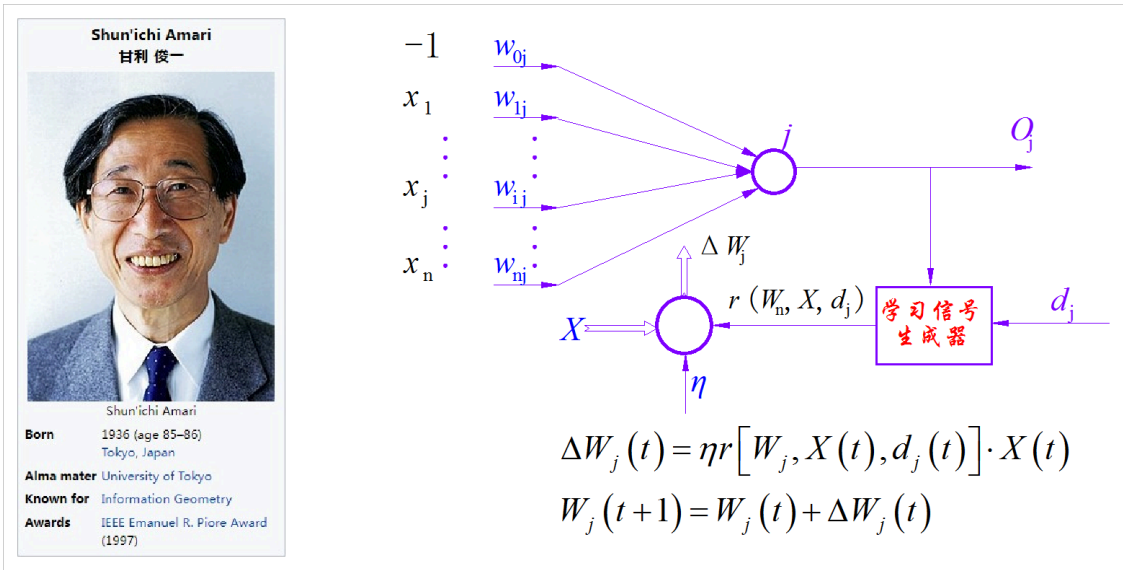
◎ 说明： **(1)** 作业可以使用你所熟悉的编程语言和平台，比如 C，C++、MATLAB、Python等。作业链接； **(2)** 作业中的样本数据可以在作业文档CSDN链接网页中拷贝下来；

01 神经元学习

一、作业内容

1、神经元统一学习算法

在第一章介绍了日本学者 “**甘利俊一**” 提出的统一公式，把对神经元输入连接权系数的修正 ΔW 分成了三个独立成分的乘积：学习速率 η ， 学习信号 $r\left(W,x,d\right)$ 以及输入向量 X 。



▲ 图1.1 神经网络中神经元学习的统一公式

当学习信号 $r\left(W, x, d\right)$ 取不同形式，可以得到神经元的三大类不同修正方式（无监督、有监督、死记忆）：

【表1-1 不同的神经元学习算法】

学习规则	权值调整	学习信号	初始值	学习方式	转移函数
Hebbian	$\Delta W = \eta f\left(W^T X\right) X$	$r = f\left(W^T X\right)$	随机	无监督	任意
Percetron	$\Delta W = \eta\left[d - \operatorname{sgn}\left(W^T X\right)\right] X$	$r = d - f\left(W^T X\right)$	任意	有监督	二值函数
Delta	$\Delta W = \eta\left[d - f\left(W^T X\right)\right] f'\left(W^T X\right) X$	$r = \left[d - f\left(W^T X\right)\right] f'\left(W^T X\right)$	任意	监督	连续可导
Widrow-Hoff LMS	$\Delta W = \eta\left(d - W^T X\right) X$	$r = d - W^T X$	任意	监督	连续
Correlation 相关,外积	$\Delta W = \eta d X$	$r = d$	0	监督死	任意

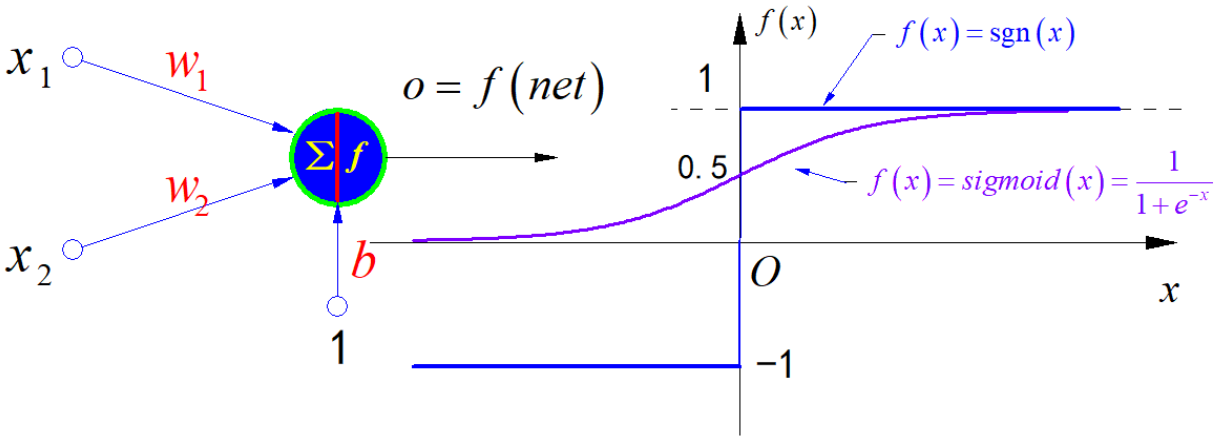
学习规则	权值调整	学习信号	初始值	学习方式	转移函数
				记忆	

下面给出神经元模型和训练样本数据，请通过编程实现上述表格中的五种算法并给出计算结果。通过这个作业练习，帮助大家熟悉神经元的各种学习算法。

2、神经元模型

下面给出神经元模型，根据不同的算法要求：

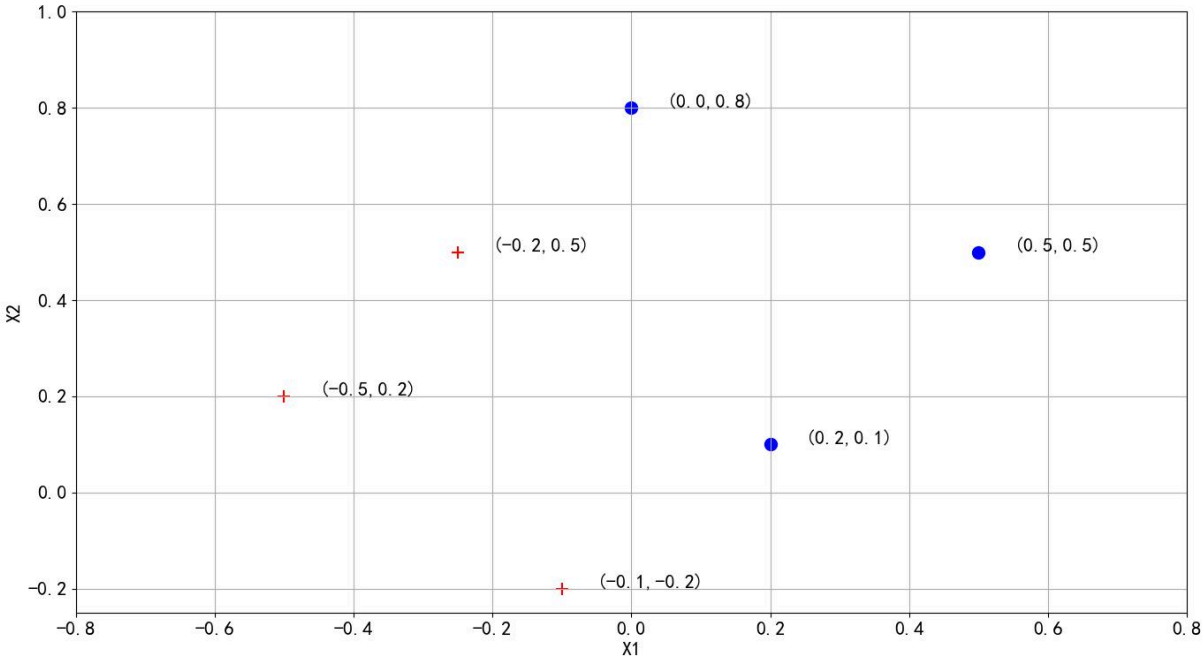
- 选择相应的传递函数种类（离散二值函数、双曲正切函数、ReLU函数）：除了Perceptron算法选择二值函数外，其它都选择双曲正切函数，
- 神经元权系数（ w_1, w_2, b ）都初始化成 0。



▲ 图1.1.2 神经元及其传递函数

3、样本数据

训练样本包括 6 个数据，它们的分布如下图所示：



▲ 图1.1.3 神经元训练数据集合

【表1-2 样本数据】

序号	X1	X2	类别
1	-0.1	-0.2	-1
2	0.5	0.5	1
3	-0.5	0.2	-1
4	-0.2	0.5	-1
5	0.2	0.1	1
6	0.0	0.8	1

```
from headm import *

import sys,os,math,time
import matplotlib.pyplot as plt
from numpy import *

xdim = [(-0.1,-0.2), (0.5,0.5), (-0.5,0.2),(-0.25,0.5),(0.2,0.1),(0,0.8)]
ldim = [-1,1,-1,-1,1,1]

printff("序列", "X1", "X2", "类别")

count = 0
for x,l in zip(xdim, ldim):
```

```
        count += 1
        printf("%d %3.1f %3.1f %d"%(count, x[0], x[1], l))

        if l > 0:
            marker = 'o'
            color = 'blue'
        else:
            marker = '+'
            color = 'red'

        plt.scatter(x[0], x[1], marker=marker, c=color, s=100)
        plt.text(x[0]+0.05,x[1],'(%3.1f,%3.1f)'%(x[0],x[1]))

plt.axis([-0.8, 0.8,-0.25, 1])
plt.xlabel("X1")
plt.ylabel("X2")
plt.grid(True)
plt.tight_layout()
plt.show()
```

二、作业要求

1、必做内容

1. 给出每个学习算法核心代码；
2. 给出经过**两轮**轮样本学习之后神经元的权系数数值结果（w1,w2,b）；

* 权系数初始化为 0；
* 学习速率 $\eta = 0.5$；
* 训练样本按照 表格1-2 的顺序对神经元进行训练；

2、选做内容

1. 在坐标系中绘制出经过两轮训练之后，权系数（w1,w2）所在的空间位置；
2. 简单讨论一下不同算法对于神经元权系数的影响；

```
import sys,os,math,time
import matplotlib.pyplot as plt
from numpy import *

xdim = [(-0.1,0.3), (0.5,0.7), (-0.5,0.2),(-0.7,0.3),(0.7,0.1),(0,0.5)]
ddim = [1,-1,1,1,-1,1]

def sigmoid(x):
    return 1/(1+exp(-x))
```

```
def hebbian(w,x,d):
    x1 = [1,x[0],x[1]]
    net = sum([ww*xx for ww,xx in zip(w, x1)])
    o = sigmoid(net)
    w1 = [ww+o*xx for ww,xx in zip(w,x1)]
    return w1

def perceptron(w,x,d):
    x1 = [1,x[0],x[1]]
    net = sum([ww*xx for ww,xx in zip(w, x1)])
    o = 1 if net >= 0 else -1
    w1 = [ww+(d-o)*xx for ww,xx in zip(w,x1)]
    return w1

def delta(w,x,d):
    x1 = [1,x[0],x[1]]
    net = sum([ww*xx for ww,xx in zip(w, x1)])
    o = sigmoid(net)
    o1 = o*(1-o)
    w1 = [ww+(d-o)*o1*xx for ww,xx in zip(w,x1)]
    return w1

def widrawhoff(w,x,d):
    x1 = [1,x[0],x[1]]
    net = sum([ww*xx for ww,xx in zip(w, x1)])
    o = sigmoid(net)
    w1 = [ww+(d-o)*xx for ww,xx in zip(w,x1)]
    return w1

def correlation(w,x,d):
    x1 = [1,x[0],x[1]]
    w1 = [ww+d*xx for ww,xx in zip(w,x1)]
    return w1

wb = [0,0,0]                          # [b, w1, w2]
for x,d in zip(xdim, ddim):
    wb = correlation(wb,x,d)
    print(wb)
```

---

02 感知机

一、两类问题

1、样本数据

利用单个神经元（具有三个参数【两个权系数和一个偏移量】： w_1, w_2, b ）， 使用感知机算法求解样本分类问题。 样本数据就采用第一道大题中的六个样本数据。 见**【表1-2 样本数据】**。

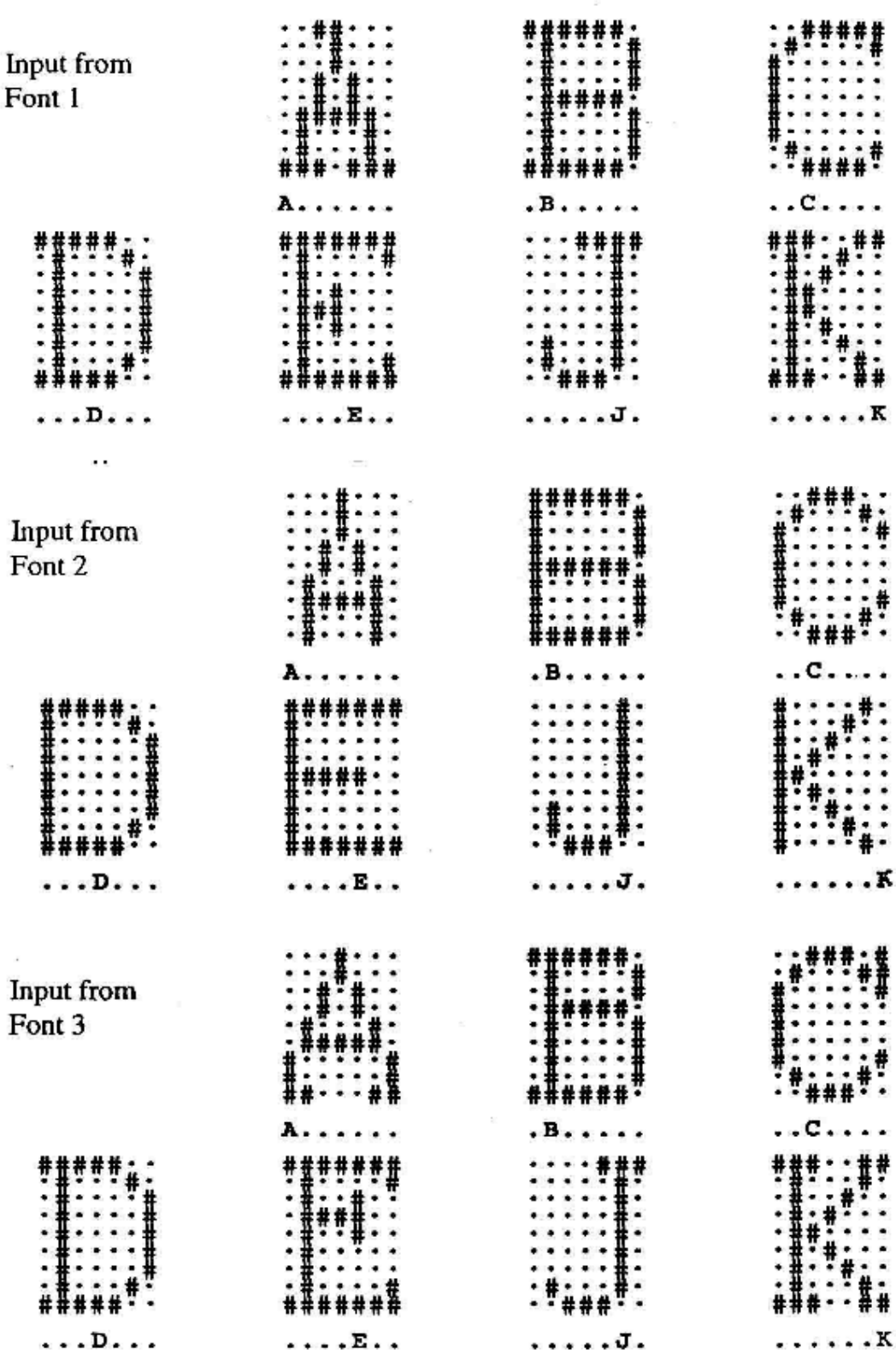
2、作业要求

1. 绘制出网络结构图，并给出算法核心代码；
2. 使用训练结束之后神经元的三个参数所决定的线性分类边界： $f(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$ 。
3. 对比不同学习速率对于训练收敛的影响；比如取学习速率 $\eta = 0.5$ 以及 $\eta = 1.0$ 。

二、多类问题

1、样本数据

如下是三个字母 **A,B,C,D,E,J,K** 的 7×9 的点阵图， 共有三种字体。将它们转换成由（-1,1） 组成的63维向量。



▲ 图2.2.1 A,B,C,D,E,J,K三种字体点阵

```
A = [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
B = [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
C = [0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
D = [1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
E = [1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1,
F = [1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1,
G = [0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
```

```
H = [1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
I = [0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
J = [0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
```

下面每一行给出了21个字符的编码以及期望输出。

> [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
> 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1]
> [1, 0, 0, 0, 0, 0, 0]

> [1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
> 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0]
> [0, 1, 0, 0, 0, 0, 0]

> [0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0]
> [0, 0, 1, 0, 0, 0, 0]

> [1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
> 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1]
> [0, 0, 0, 0, 0, 0, 1]

> [0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
> 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0]
> [0, 0, 0, 0, 0, 1, 0]

> [1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
> 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
> [0, 0, 0, 0, 1, 0, 0]

> [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1,
> 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0]
> [0, 0, 0, 1, 0, 0, 0]

> [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1,
> 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0]

[1, 0, 0, 0, 0, 0, 0]

[0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0]
[0, 1, 0, 0, 0, 0, 0]

[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0]

[1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 1]

[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0]
[0, 0, 0, 0, 0, 1, 0]

[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 1, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1]
[1, 0, 0, 0, 0, 0, 0]

[1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0]
[0, 1, 0, 0, 0, 0, 0]

```
[0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0]
```

```
[1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1]
[0, 0, 0, 0, 0, 0, 1]
```

```
[0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0]
[0, 0, 0, 0, 0, 1, 0]
```

```
[1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 1, 0, 0]
```

```
[1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1,
0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0]
```

2、作业基本要求

1. 建立由七个神经元组成的简单感知机网络，完成上述七个字母的识别训练；
2. 测试训练之后的网络在带有一个噪声点的数据集合上的识别效果。 给图片增加一个噪声点就是随机在样本中选取一个像素， 将其数值进行改变（从-1改变成1， 或者从1改变成-1）。

3、选做内容

1. 测试上述感知机网络在两个噪声点的数据集合上的识别效果；
2. 对比以下两种情况训练的感知机的性能。
    - 第一种情况：只使用没有噪声的七个字母进行训练；
    - 第二种情况：使用没有噪声和有一个噪声点的样本进行训练；
3. 对比不同的学习速率对于训练过程的影响。

三、感知机算法收敛特性

☐ **这是选做题目**

请证明对于线性可分的两类数据集合， 使用感知机算法进行分类。 感知机算法收敛步骤数量的上限与学习速率无关。

03 Adaline网络

一、两类分类问题

1、样本数据

利用单个神经元（具有权系数 w_1, w_2 以及偏移量 b ）， 使用**ADALINE**算法（LMS）求解样本分类问题。 样本数据就采用第一道大题中的六个样本数据。 见 **【表1-2 样本数据】**。

2、作业要求

1. 绘制出网络结构图，并给出算法核心代码：**最小二乘法（LMS）**；
2. 绘制出训练结束之后， 神经元参数对应的线性分类界面函数： $f(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$ 。 对比 ADALINE与感知计算法在分类结果方面的**优劣**。

3、问题讨论

☐ **这部分讨论问题是选做内容**

（1） 请使用数据矩阵**伪逆**的方法，重新计算神经网络的权系数。 对比最小二乘法和伪逆求取的方法所得到的结果。

（2） 讨论改变LMS 算法中的学习速率 η 对于ADALINE训练过程的影响。 观察 η 过大或者过小对于LMS算法收敛速度的影响。

二、鸟类分类（选做题）

1、背景介绍

自适应线性神经元 ADALINE（Adatpive Linear Neuron）是由 Bernard Widrow 与 Ted Hoff 在 1959年提出的算法。关于他们提出算法前后的故事，大家可以参照网文：

The ADALINE - Theory and Implementation of the First Neural Network Trained With Gradient Descent 进行了解。

下面也是根据上述网文中所介绍的两种鸟类（猫头鹰与信天翁）数据集合，产生相应的分类数据集合。大家使用 ADALINE 算法完成它们的分类器算法。

▲ 图3.1.1 猫头鹰与信天翁

2、样本数据

（1）数据参数

根据Wikipedia 中关于 信天翁 **Wandering albatross** 和 猫头鹰（ **Great horned owl** ）的相关数据，这两种鸟类的题中和翼展长度如下表所示。

【表1-3 两种鸟类的体型数据】

种类	体重(kg)	翼展(m)
信天翁	9	3
猫头鹰	1.2	1.2

使用计算机产生两个鸟类体型随机数据数据，下表给出了每一类数据产生的参数：

【表1-4 两类鸟类数据产生参数】

鸟类	体重平均值	体重方差	翼展平均值	翼展方差	个数	分类
信天翁	9000	800	300	20	100	1
猫头鹰	1000	200	100	15	100	-1

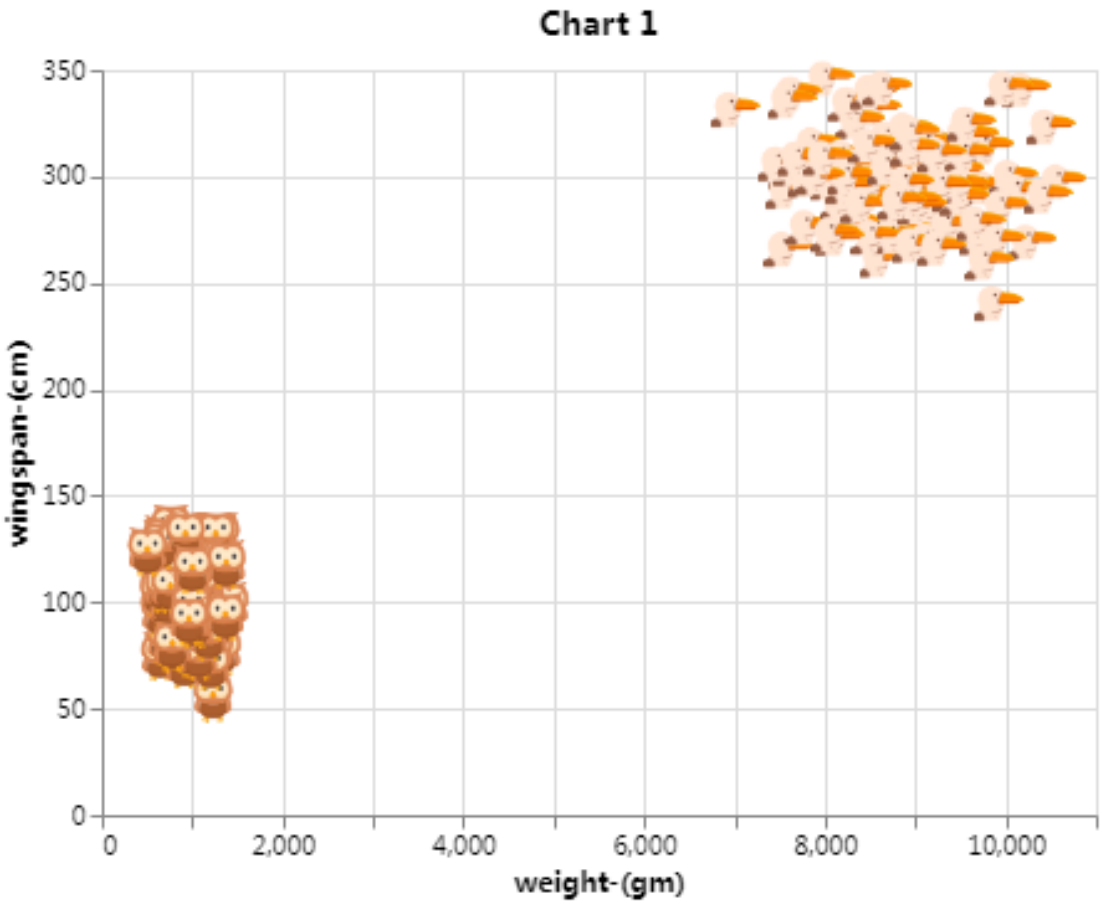
（2）Python示例代码

下面给出了产生随机样本数据的 Python 示例代码。大家可以参照这些代码，使用自己熟悉的 编程语言来实现。

```
def species_generator(mu1, sigma1, mu2, sigma2, n_samples, target, seed):
    '''creates [n_samples, 2] array

    Parameters
    ----------
    mu1, sigma1: int, shape = [n_samples, 2]
        mean feature-1, standar-dev feature-1
    mu2, sigma2: int, shape = [n_samples, 2]
        mean feature-2, standar-dev feature-2
    n_samples: int, shape= [n_samples, 1]
        number of sample cases
    target: int, shape = [1]
        target value
    seed: int
        random seed for reproducibility

    Return
    ------
    X: ndim-array, shape = [n_samples, 2]
        matrix of feature vectors
    y: 1d-vector, shape = [n_samples, 1]
        target vector
    ------
    X'''
    rand = np.random.RandomState(seed)
    f1 = rand.normal(mu1, sigma1, n_samples)
    f2 = rand.normal(mu2, sigma2, n_samples)
    X = np.array([f1, f2])
    X = X.transpose()
    y = np.full((n_samples), target)
    return X, y
```



▲ 图3.1.2 产生两类数据的分布

三、作业要求

1. 构造一个 ADALINE 神经元，完成上述两类鸟类的分类。 由于需要进行分类，在对 ADALINE 的输出在经过一个符号函数（sgn）便可以完成结果的分类；

2. 利用上述数据对 ADALINE 进行训练。观察记录训练误差变化的曲线。

3. 讨论不同的学习速率对于训练结果的影响，看是否存在一个数值，当学习速率超过这个数值之后，神经元训练过程不再收敛。

■ 相关文献链接：

- The ADALINE - Theory and Implementation of the First Neural Network Trained With Gradient Descent
- Wandering albatross
- Great horned owl

● 相关图表链接：

- 图1.1 神经网络中神经元学习的统一公式