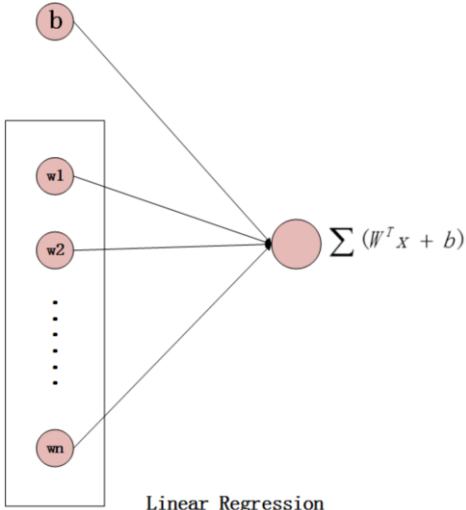


## 计算机科学与技术学院 认知科学与类脑计算实验 课程实验报告

实验题目：实验二 感知器模型实现及分类实验		学号：201600150109
日期：2019.5.17	班级：2016 级人工智能	姓名：沈棋韬
Email: qitaoshen@gmail.com		
实验目的： 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		
实验软件和硬件环境： Windows10 Anaconda3.4 Python3.7		
实验原理和方法：  <p>Linear Regression model <a href="http://log.csdn.net/LoseInVain">http://log.csdn.net/LoseInVain</a></p>		

## 感知器模型

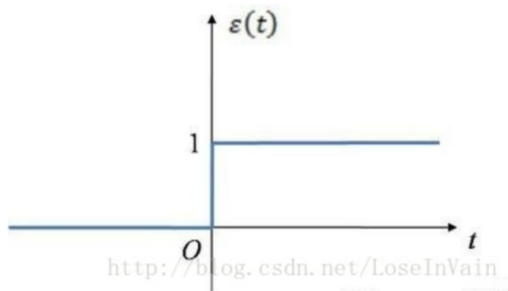
在经典的感知器模型(Perceptron)中，在输出端添加了一个**阶跃函数 (Step Function)**，这样就将输出离散到了0或者1，表达式如：

$$\phi(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

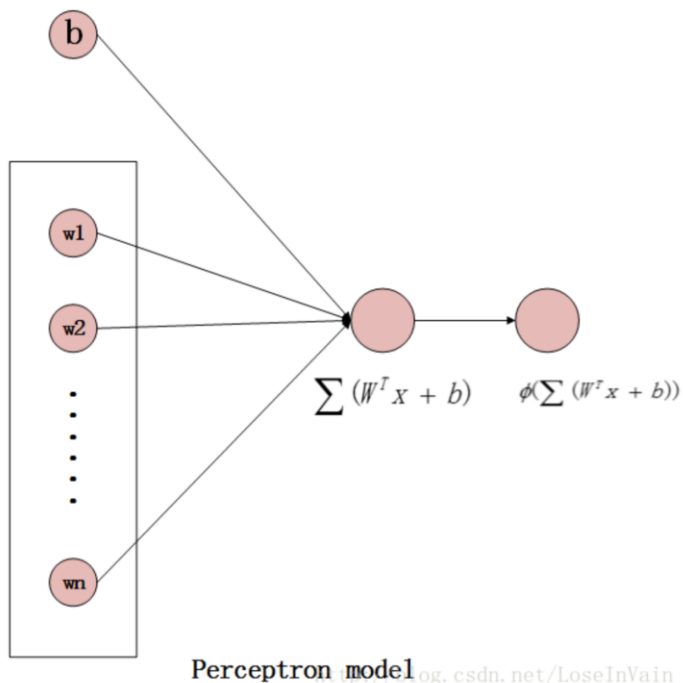
添加了激活函数之后，整个感知器模型的表达式就变为：

$$\phi\left(\sum_{i=1}^m W^T x + b\right)$$

激活函数图像如：



经过改造后的模型示意图如：



感知器采用的训练策略和线性回归，BP 反向传播算法等是不同的，感知器采用了激活函数，而且这个激活函数  $\phi(x)$  是不可导的，因此误差函数的梯度将会没有办法传播到输入层的权值中，因此不能采用梯度反向传播的策略去学习参数，我们在感知器中，采用的是误差驱动更新的策略，也就是将误分类的样本用来更新参数，将正确分类的样本忽略，当所有样本都是正确分类时，就训练完成。而在更新参数的过程中，采用的策略是

```
w:=w+η*(t-yi)*xi  
b:=b+η*(t-yi)
```

实验步骤：（不要求罗列完整源代码）

激活函数负责把到达阈值的神经元激活。

```
def activation(x):  
    if x > 0:  
        return 1  
    else:  
        return 0
```

因为使用的是有 3 个值的真值表，因此把初始 weight 随机化成 3 维向量。初始 bias 为 0.

```
def __init__(self):  
    self.weights = np.random.random(3)  
    self.bias = 0
```

训练过程如之前原理所示，迭代次数为 1000 次，学习率设置为 0.001。需要注意的是向量运算时要注意维数，不能出错。

```
def train(self, feature, label, iter, lr):  
    for i in range(iter):  
        print('iteration: ', i)  
        for i in range(len(feature)):  
            output = 0  
            output = np.sum(feature[i] * self.weights)  
            output += self.bias  
            print(output)  
            output = activation(output)  
            self.weights = self.weights + lr * (label[i] - output) * np.array(feature[i])  
            self.bias = self.bias + lr * (label[i] - output)
```

预测时只需要一个 sample 的 feature，把 feature 与训练好 weights 相乘，再加上 bias，再激活，得到的就是结果。

```
def predict(self, feature):
    output = 0
    for i in range(len(self.weights)):
        output += self.weights[i] * feature[i]
    output += self.bias
    print(output)
    output = activation(output)
    return output
```

训练样本和 label 如图所示， 实现与门：

```
data_feature = np.array([
    [0,0,0],
    [0,0,1],
    [0,1,0],
    [0,1,1],
    [1,0,0],
    [1,0,1],
    [1,1,0],
    [1,1,1]
])

data_label = np.array([
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    1
])
```

训练过程如图所示：

```
classifier = perception()
classifier.train(data_feature, data_label, 1000, 0.001)
```

结论分析与体会：

结果展示：

```
In [34]: runfile('D:/§  
Input data: [0 0 0]  
reslut: 0
```

```
In [35]: runfile('D:/§  
Input data: [0 1 0]  
reslut: 0
```

```
In [36]: runfile('D:/§  
Input data: [1 1 1]  
reslut: 1
```

```
In [37]: runfile('D:/§  
Input data: [0 0 0]  
reslut: 0
```

```
In [38]: runfile('D:/§  
Input data: [0 0 1]  
reslut: 0
```

```
In [39]: runfile('D:/§  
Input data: [0 1 0]  
reslut: 0
```

```
In [40]: runfile('D:/§  
Input data: [0 1 1]  
reslut: 0
```

```
In [41]: runfile('D:/§  
Input data: [1 0 0]  
reslut: 0
```

```
In [42]: runfile('D:/§  
Input data: [1 0 1]  
reslut: 0
```

```
In [43]: runfile('D:/§  
Input data: [1 1 0]  
reslut: 0
```

```
In [44]: runfile('D:/§  
Input data: [1 1 1]  
reslut: 1
```

可以看到分类全部正确。因为这是线性可分的。

就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1—3 道问答题：

问题 1：在训练时 output 本应该是一个数，但是在过程中变成了一个向量，导致无法激活。

解决方法：找到问题原因，在过程中误将 output 与一个向量进行加法操作，系统自动将 output 转化成了一个向量。