## 机器学习之神经网路概论

@Coding迪斯尼  
日期: 2017-07-07

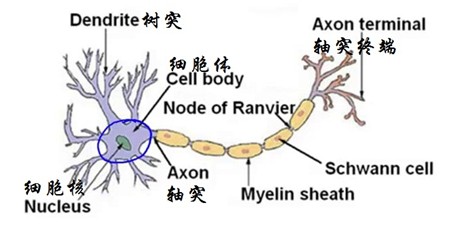
## 要点

ANN的要点就是反复更新权重向量w

感知器 (perceptron)

# 基本神经元 perceptron

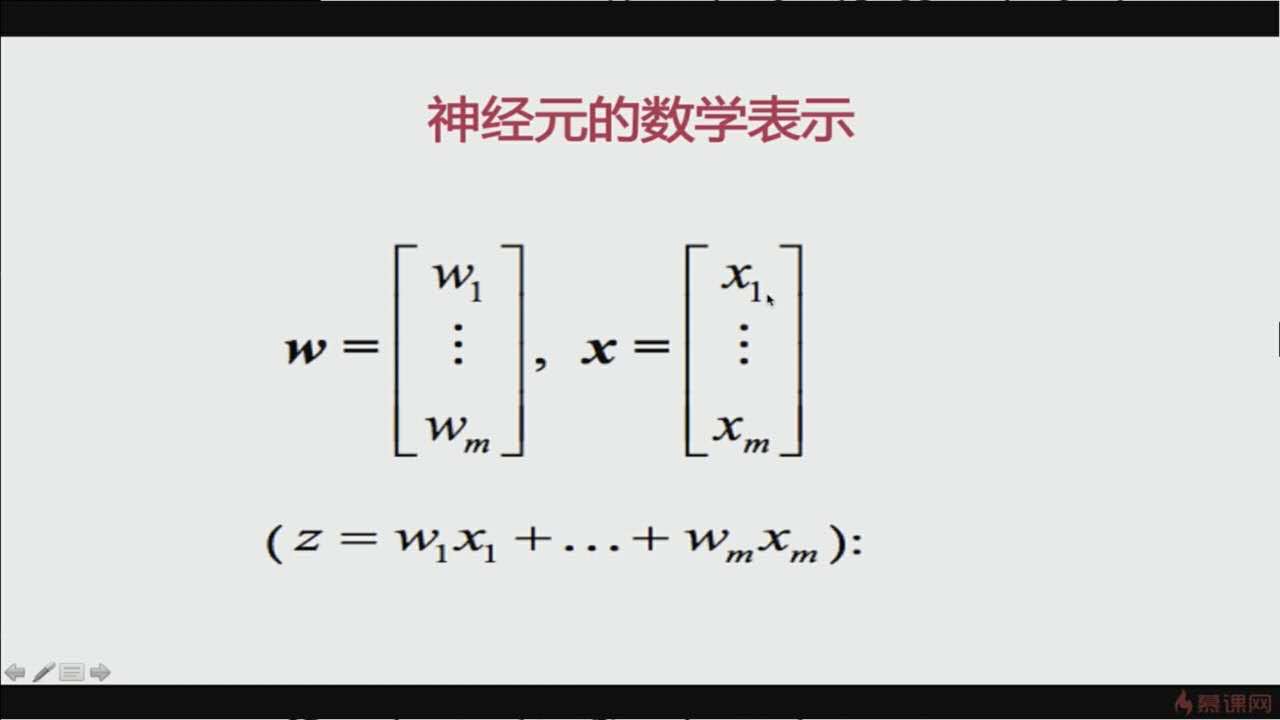
## 模拟神经元



电信号 -> 不同树突进入（弱化） -> 细胞核（统计运算）-> 唯一的电信号 –> 轴突传输到后面 -> 再拆分通过树突传输出去

## 神经元的数学表示

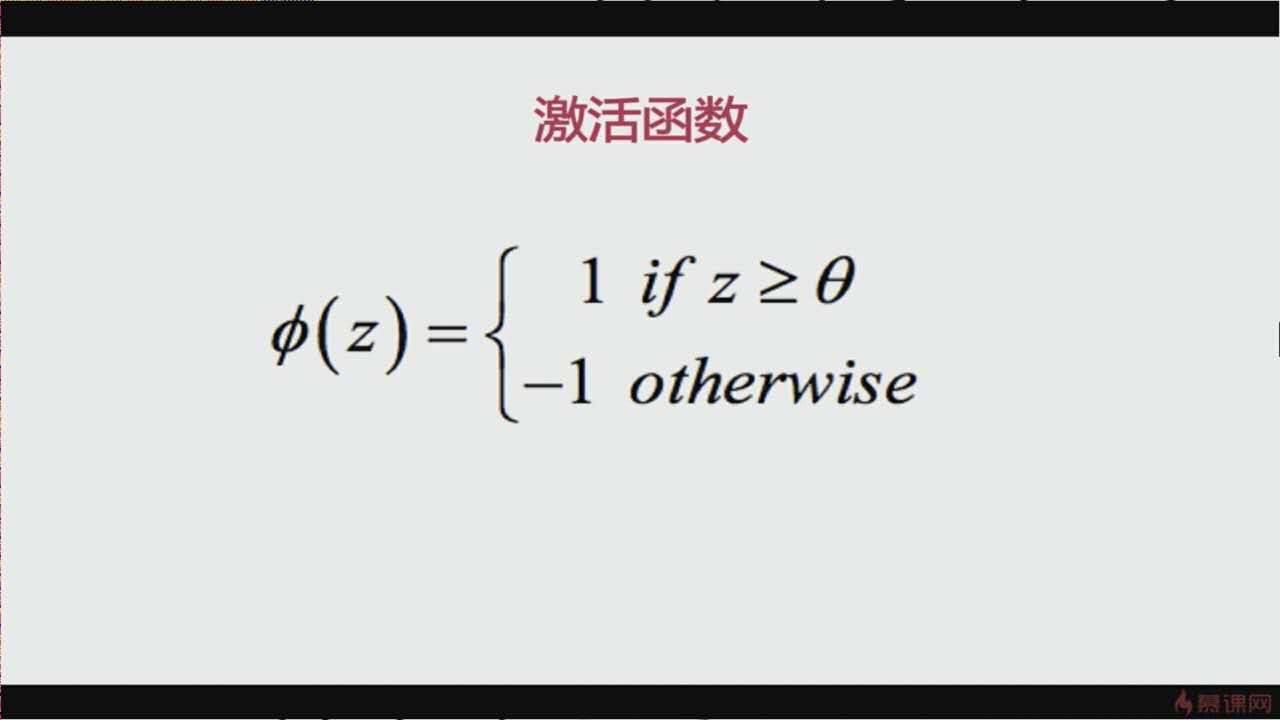
X: 不同树突进来的电信号就是X [X1 … Xm];



W: 弱化处理就是 W [W1…Wm];

Z: 统一处理就是z;

## 激活函数



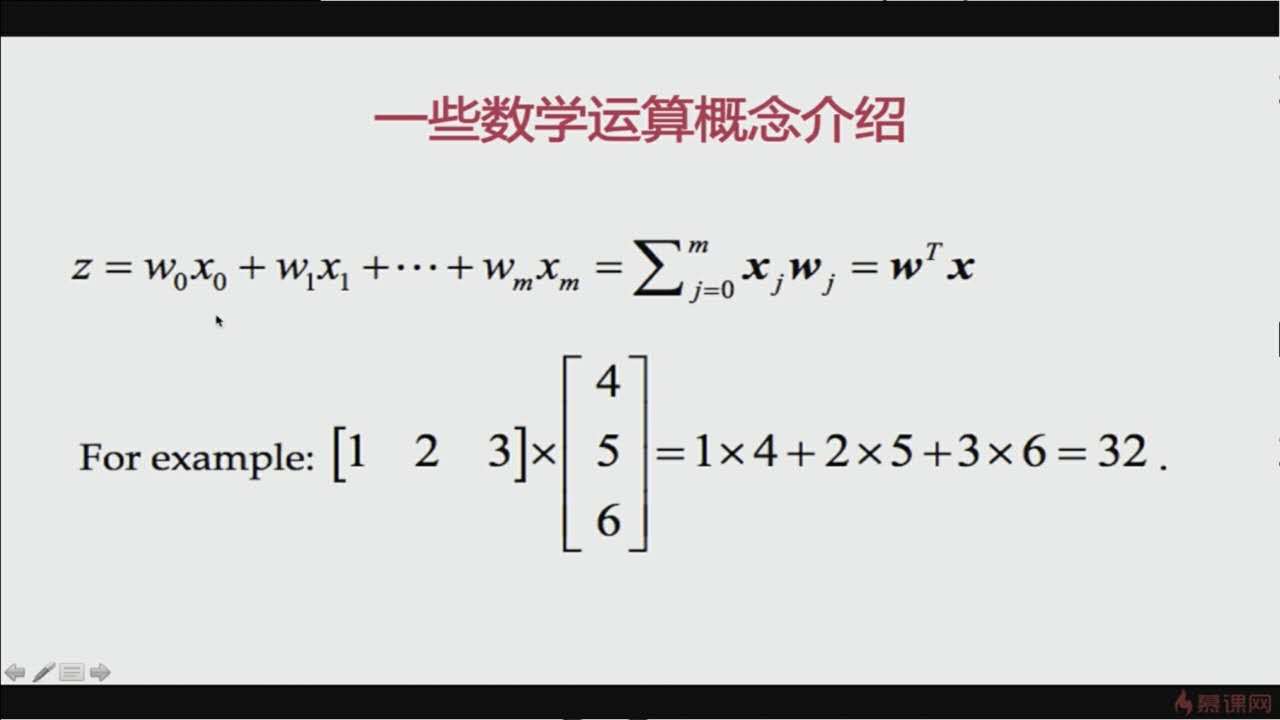
统一处理后的z 就会经过激活函数的处理；

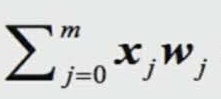
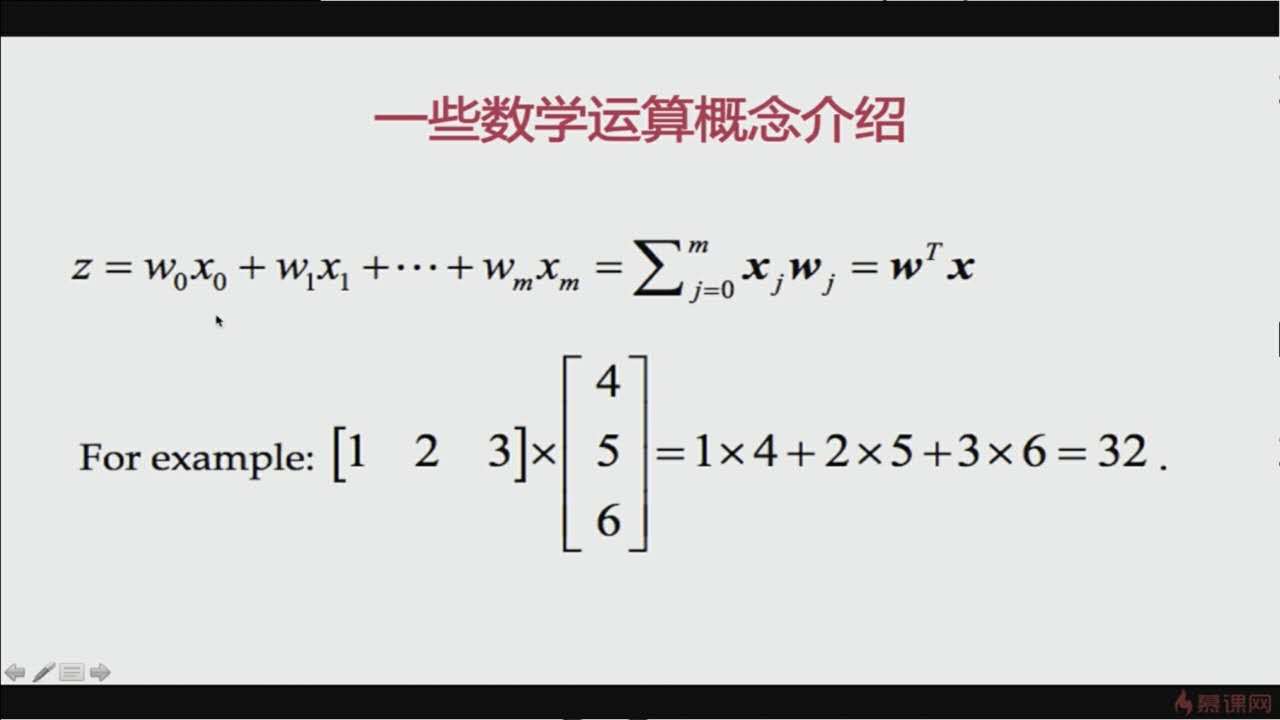
但z 大于或等于某个阈值的时候就会输出 1， 否则输出 -1；

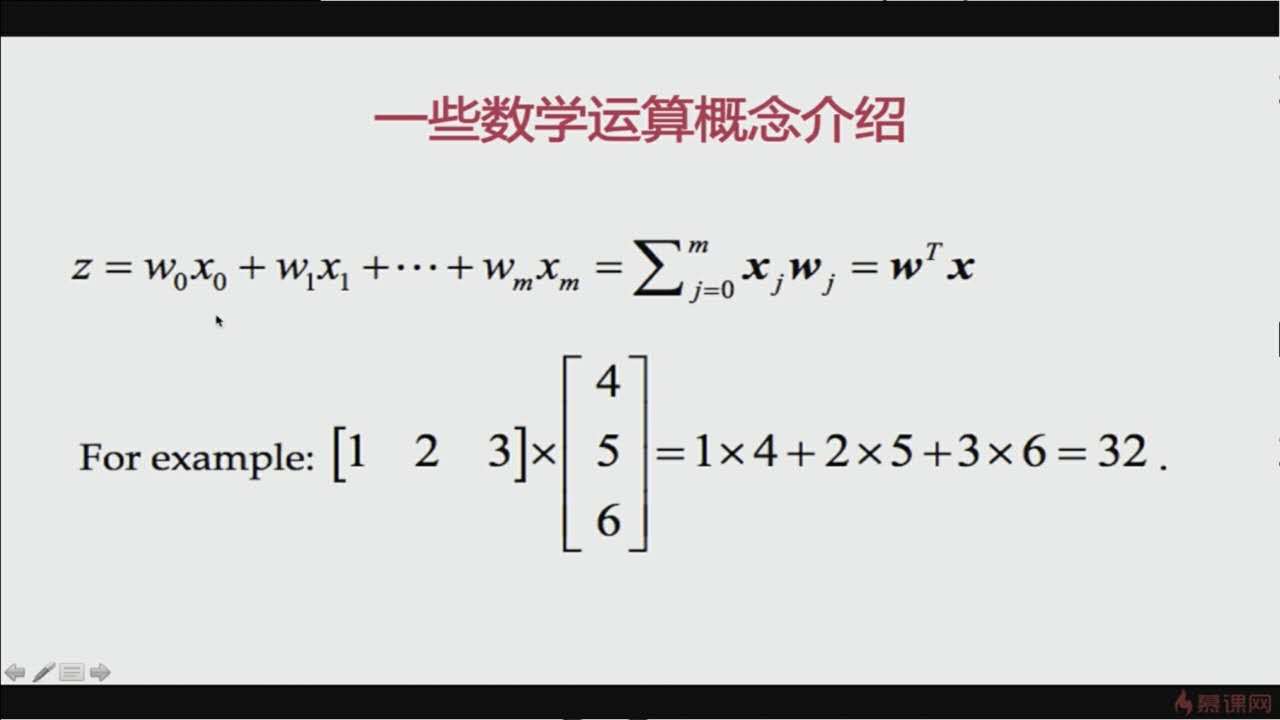
那么输出为1的就是为一类，输出为-1的为另一类；

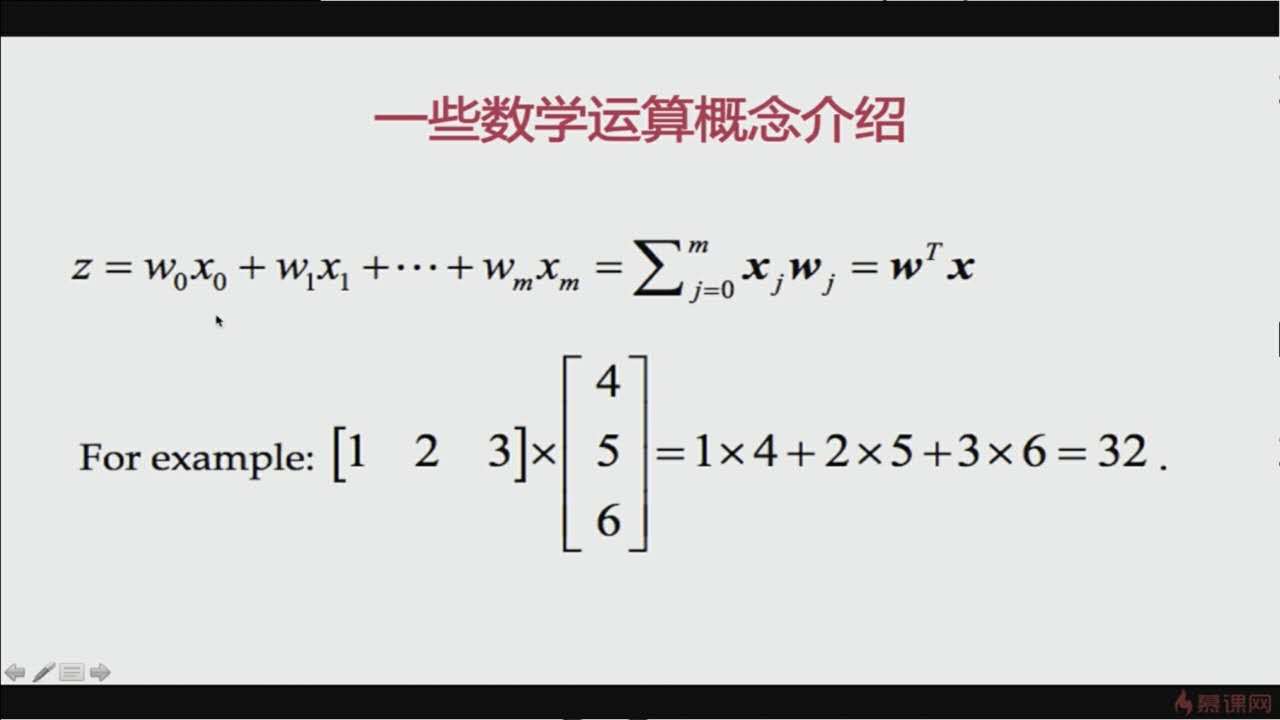
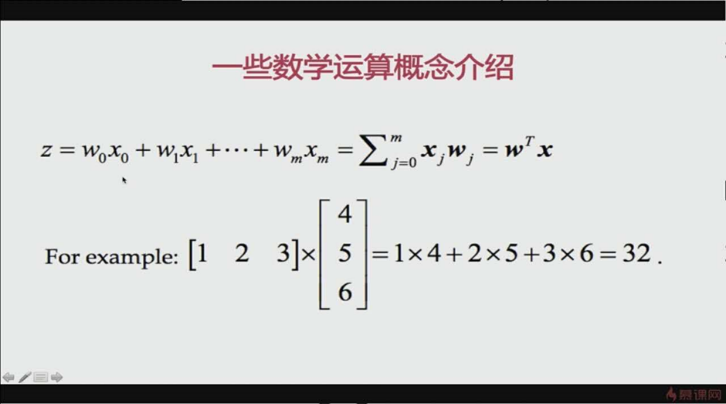
这个激活函数叫做单元步调函数；

## 向量的点积

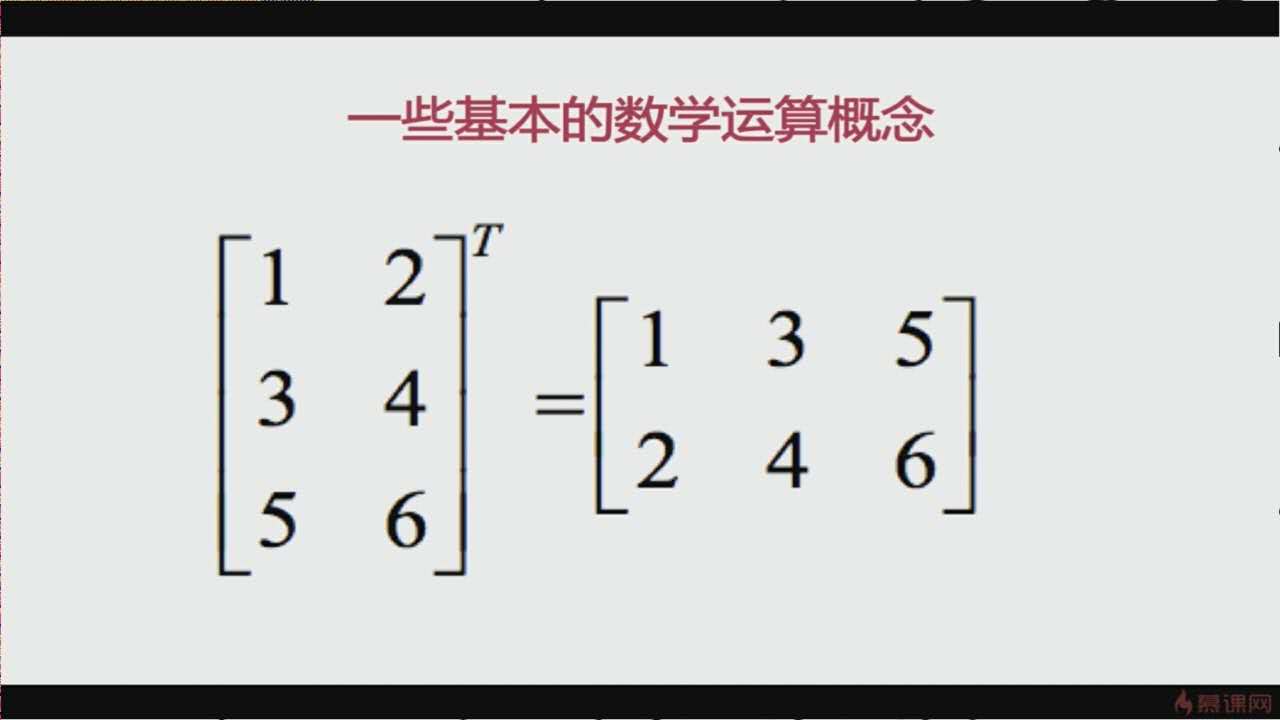


这个表示x 和 w 的乘积再汇总求和，相当于

 T表示向量的转置，w原本是竖着的，然后就变成横着的；

 是 w，  是x；

## 矩阵的转置



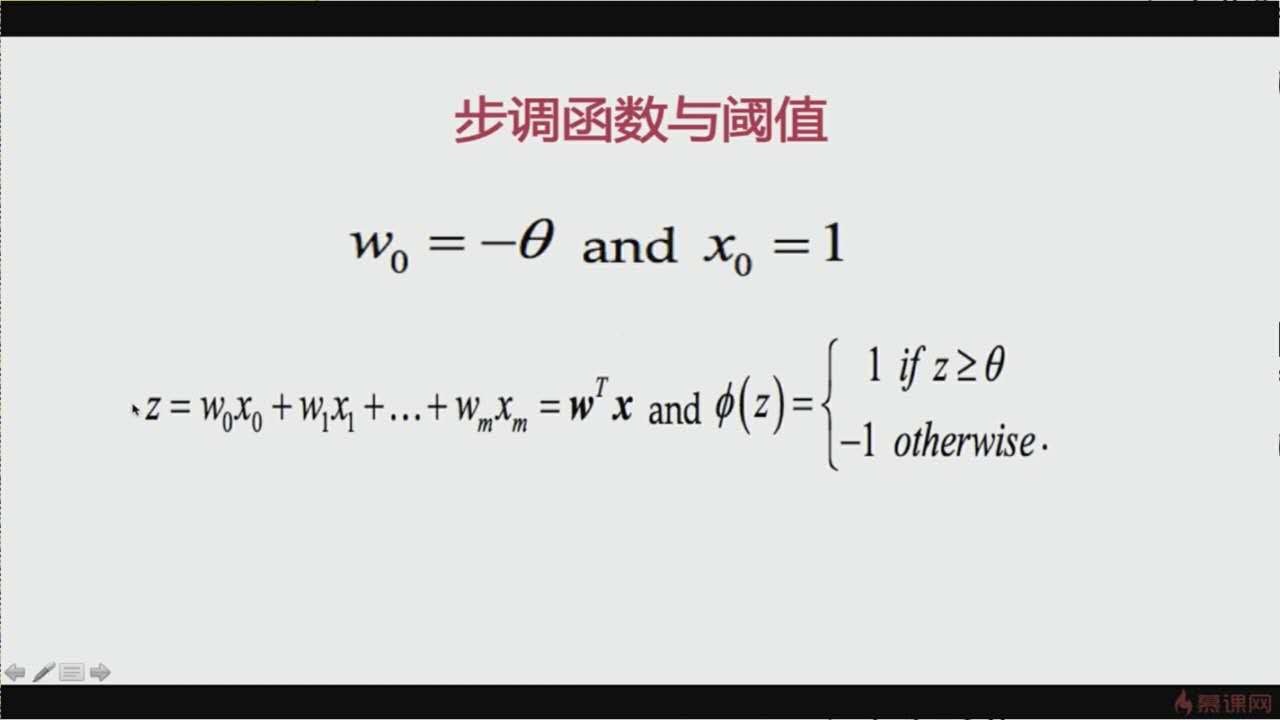
把矩阵的行变成列

## 感知器数据分类算法步骤

权重向量W，训练样本X

1. 把权重向量初始化为0，或把每个分量初始化为[0,1]间任意小数
2. 把训练样本输入感知器，得到分类结果（-1或1）
3. 根据分类结果更新权重向量

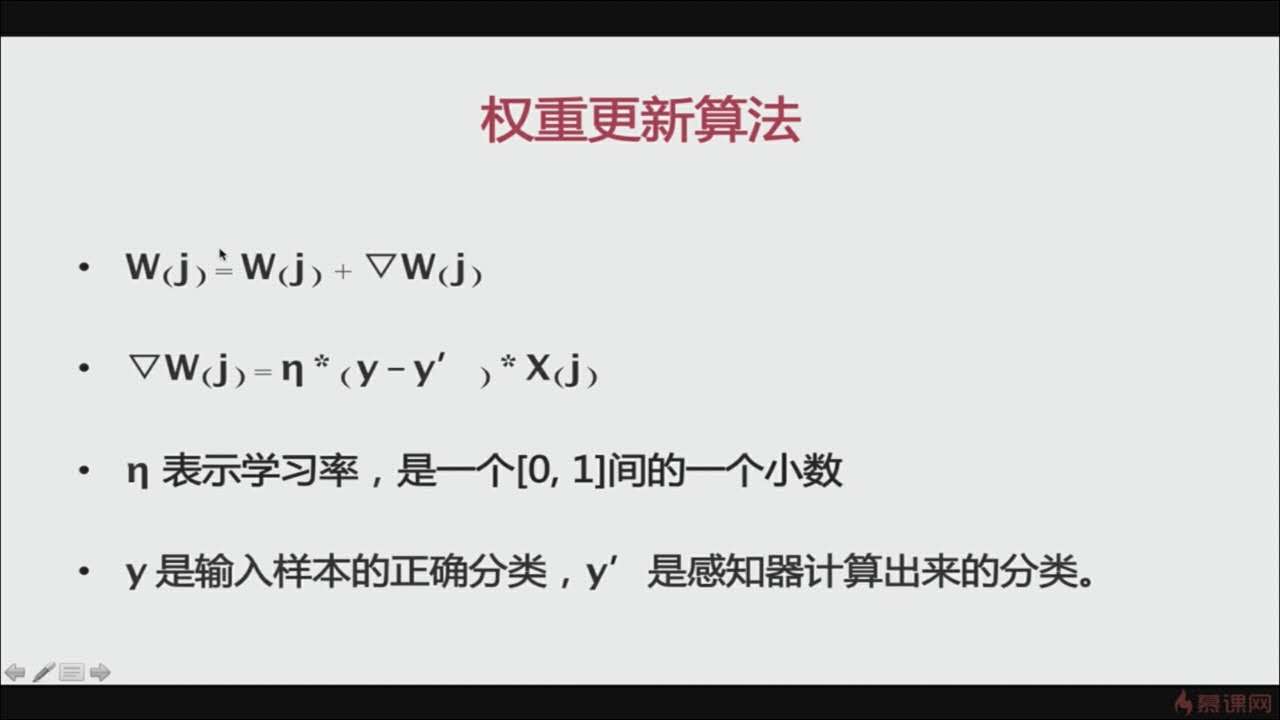
## 步调函数和阈值

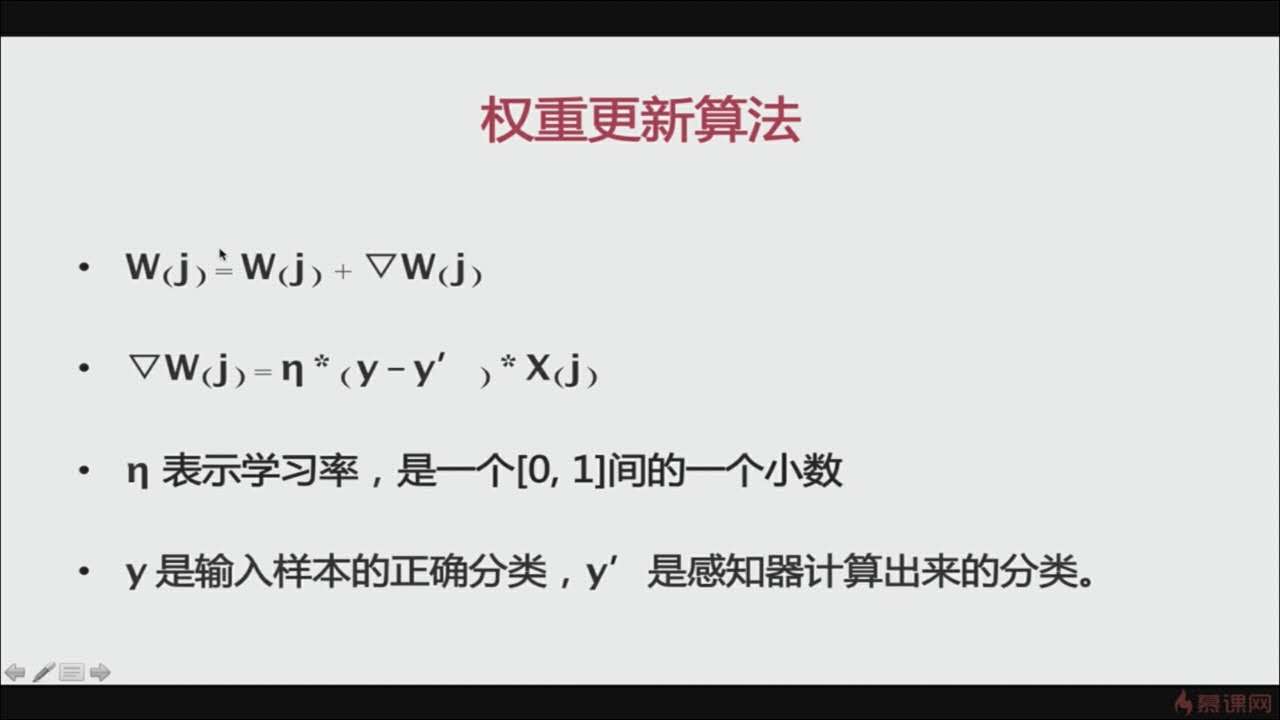


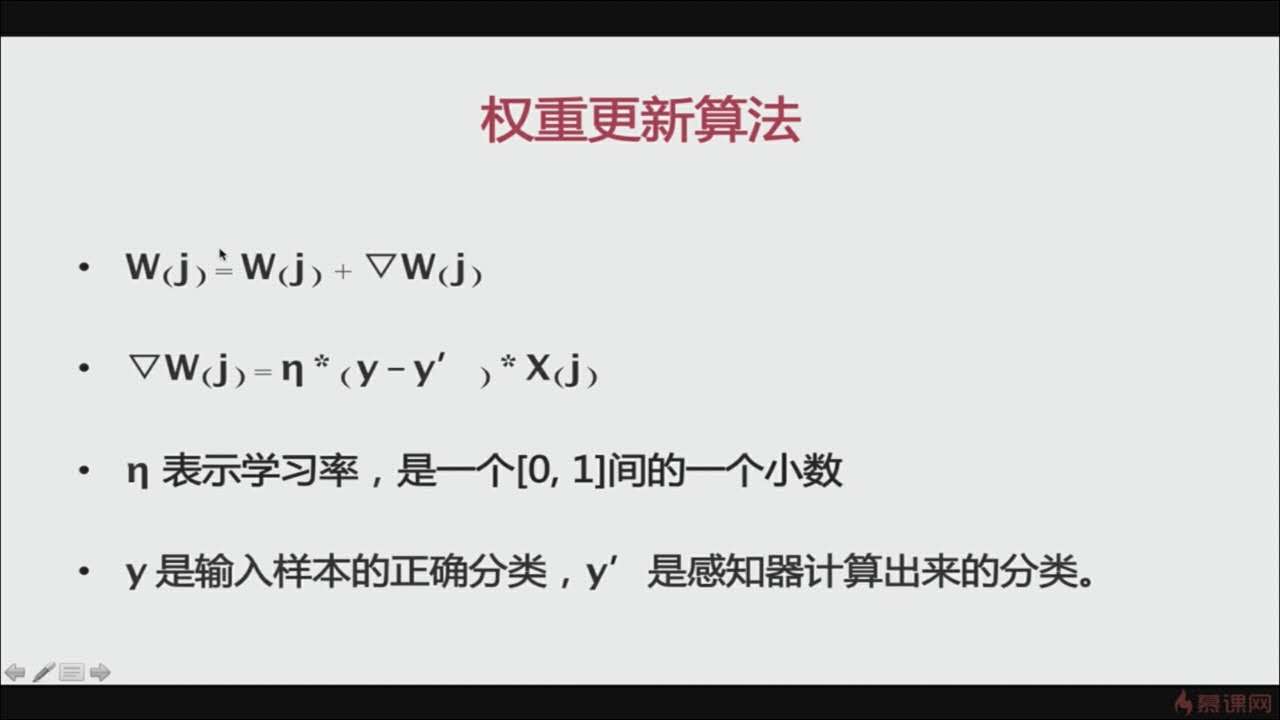
汇总算出z，然后判断z是否大于阈值，如果大于就输出1，否则输出-1；

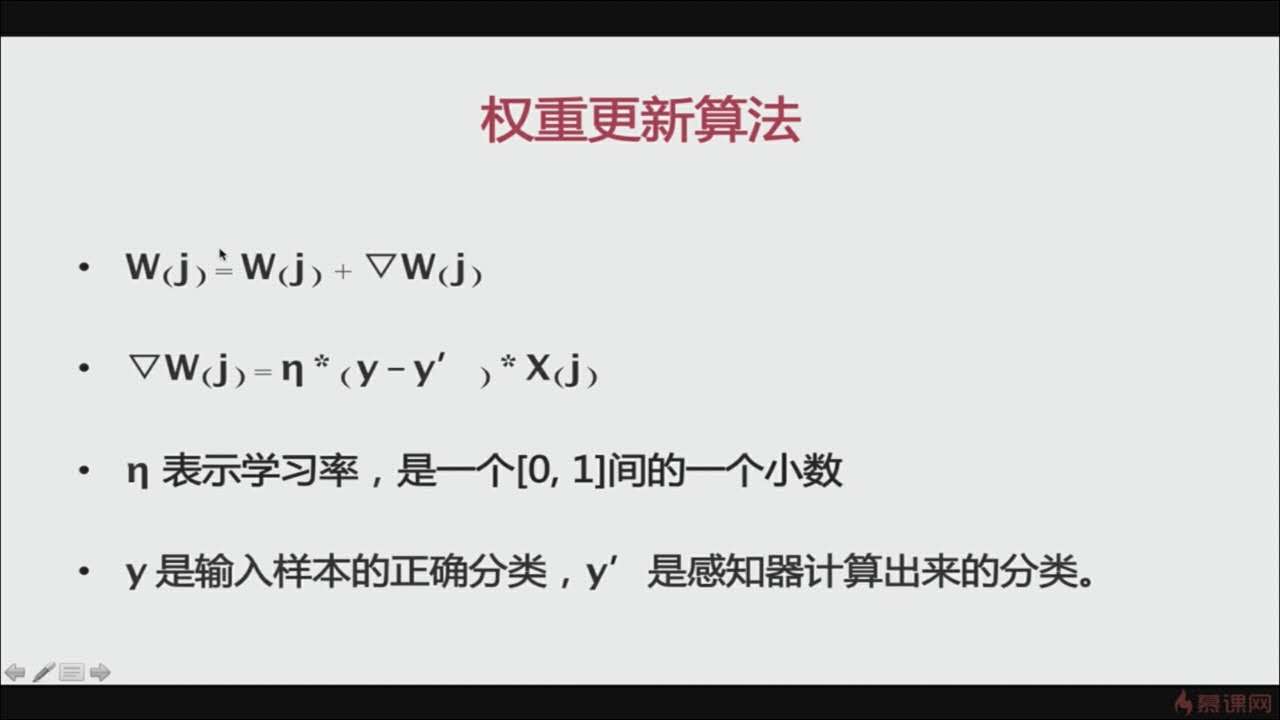
为了后面计算方便，把阈值设为负的w0， 同时把x0 固定为1 => 把w0x0 增加到前面；

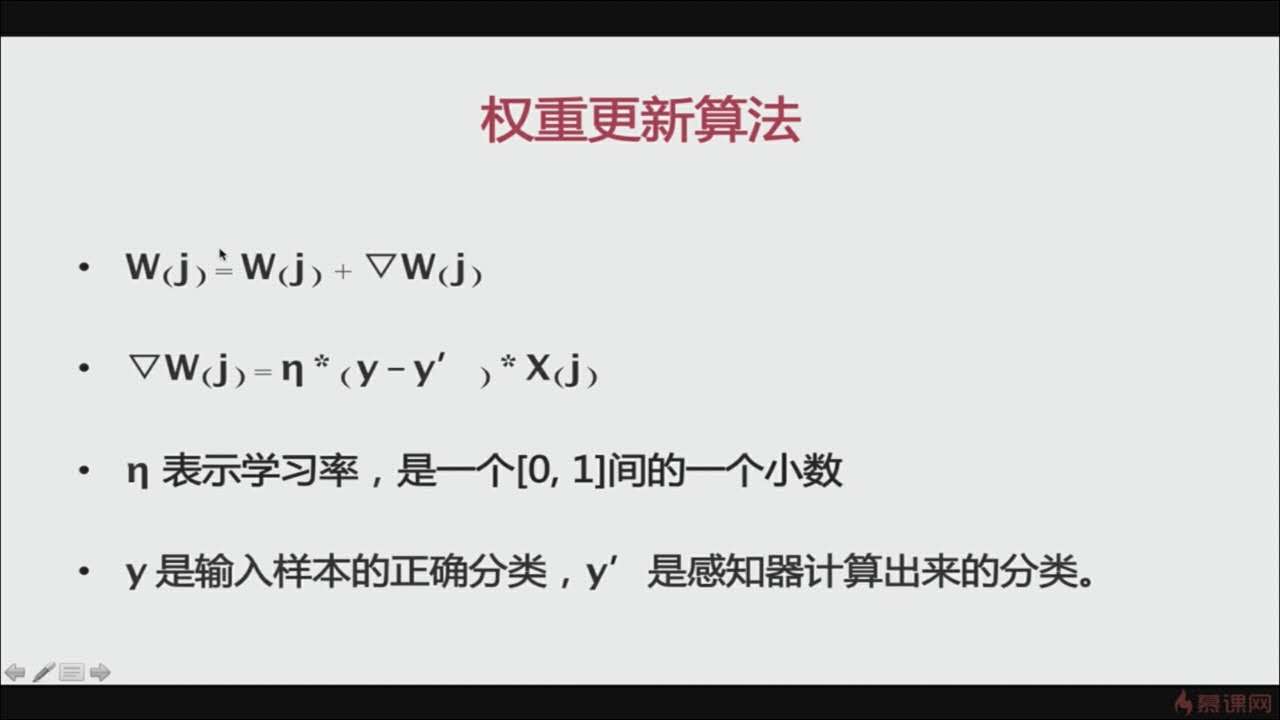
## 权重更新算法



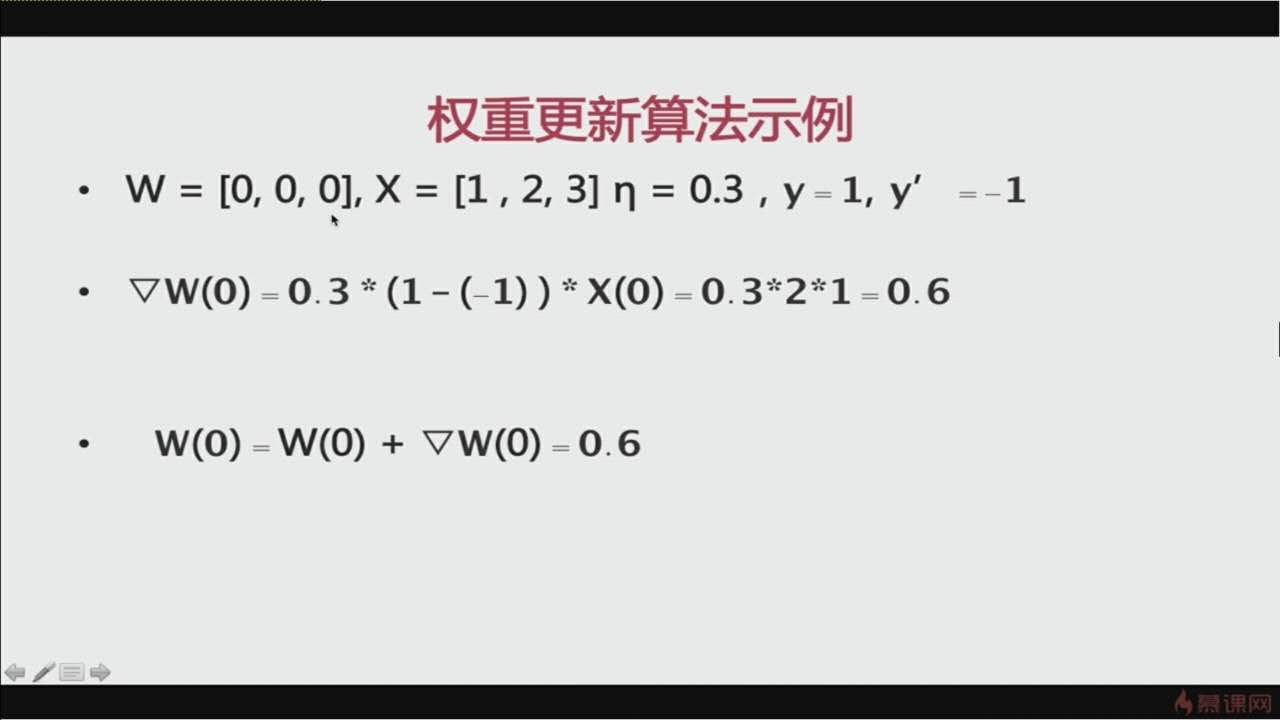
 这里求的 权重=原来的权重 + 变化的权重

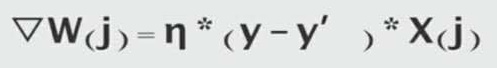
 变化的权重的 = 数值 \* (正确分类 – 感知器算出的分类) \* 输入的相关的电信号

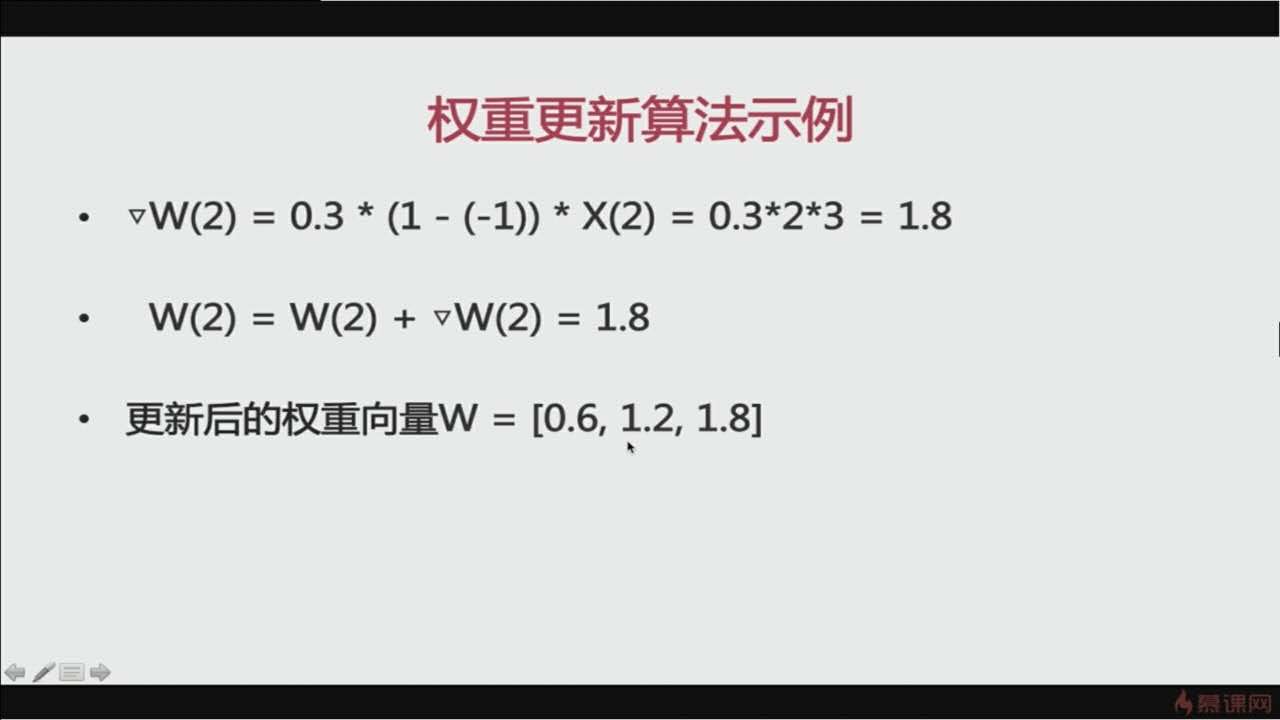
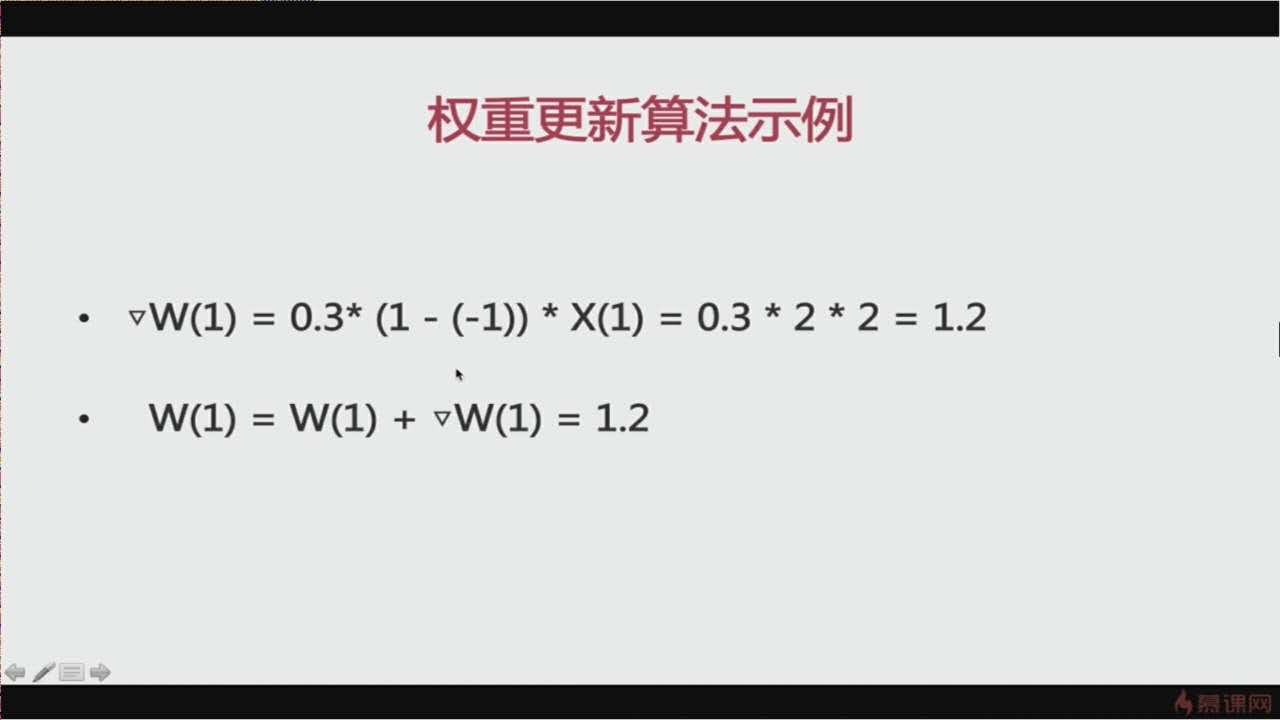
等到正确的分类，相当于 等于 0， 就不需要更新权重；

 是使用者自己设置的，人为根据经验进行调整；

## 权重更新算法示例

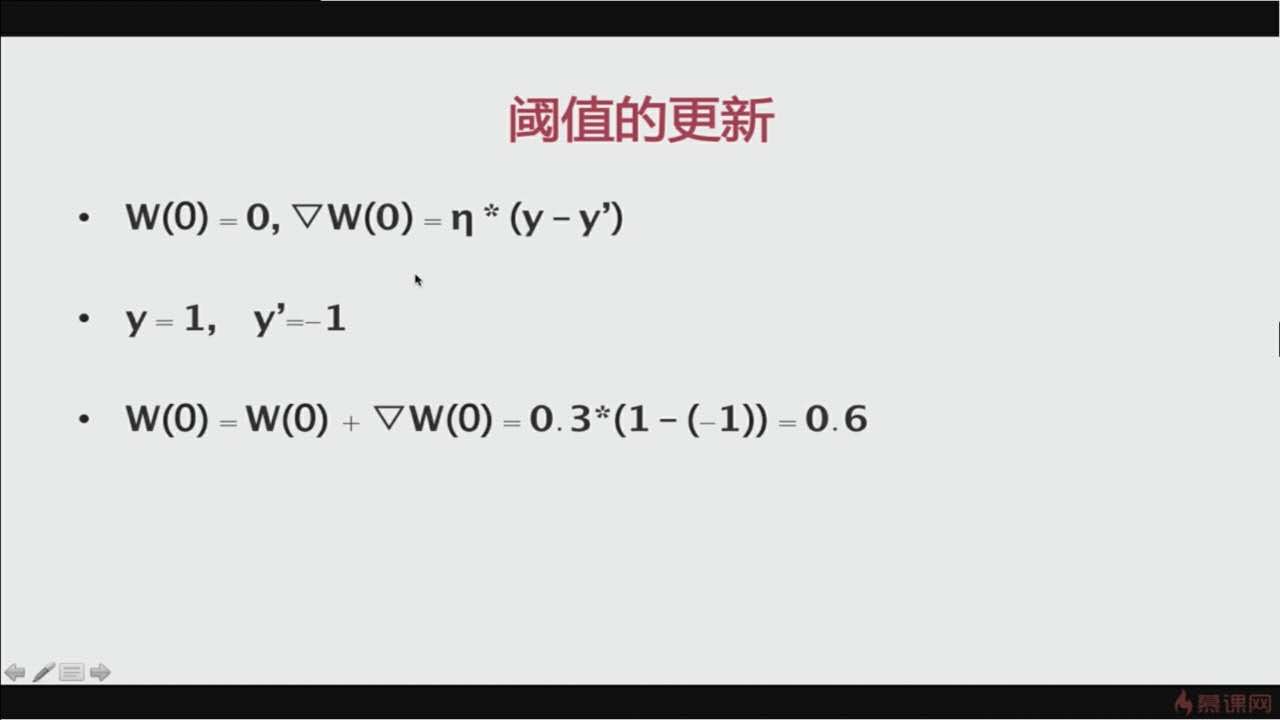


假定一开始的权重向量为 W = [0, 0, 0] ; 训练样本 X = [1, 2, 3] ; 学习率为 0.3； 正确分类为1，感知器分类为 -1； 把数值代入公式得到结果为 0.6 ； 那么权重就是更新为 0.6 ；



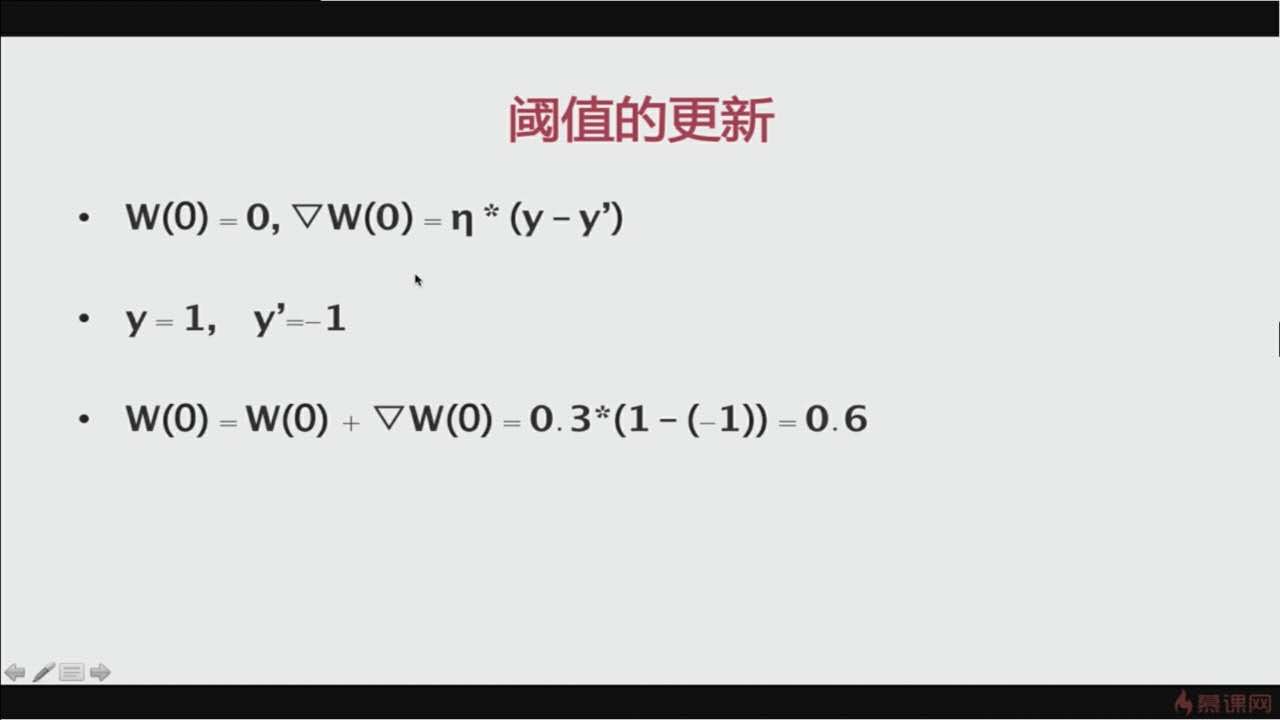
X(\*) 就是代入不同的x值；

## 阈值的更新



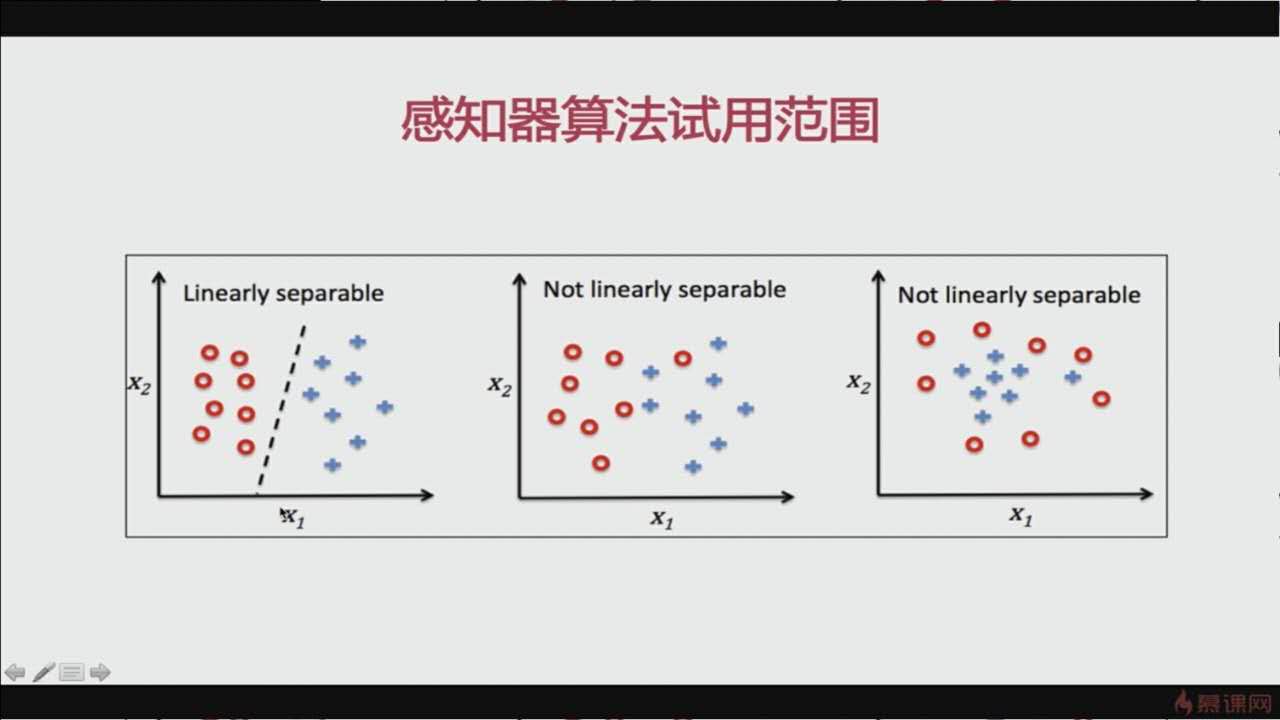
阈值是不断更新的；

一开始假设阈值为w(0) =0；

 后面还应\*x(0) 但因为假设为1，所以省略了；

最后的结果就是新的阈值变成了 0.6， 新的运算就要根据这个阈值来判断了；

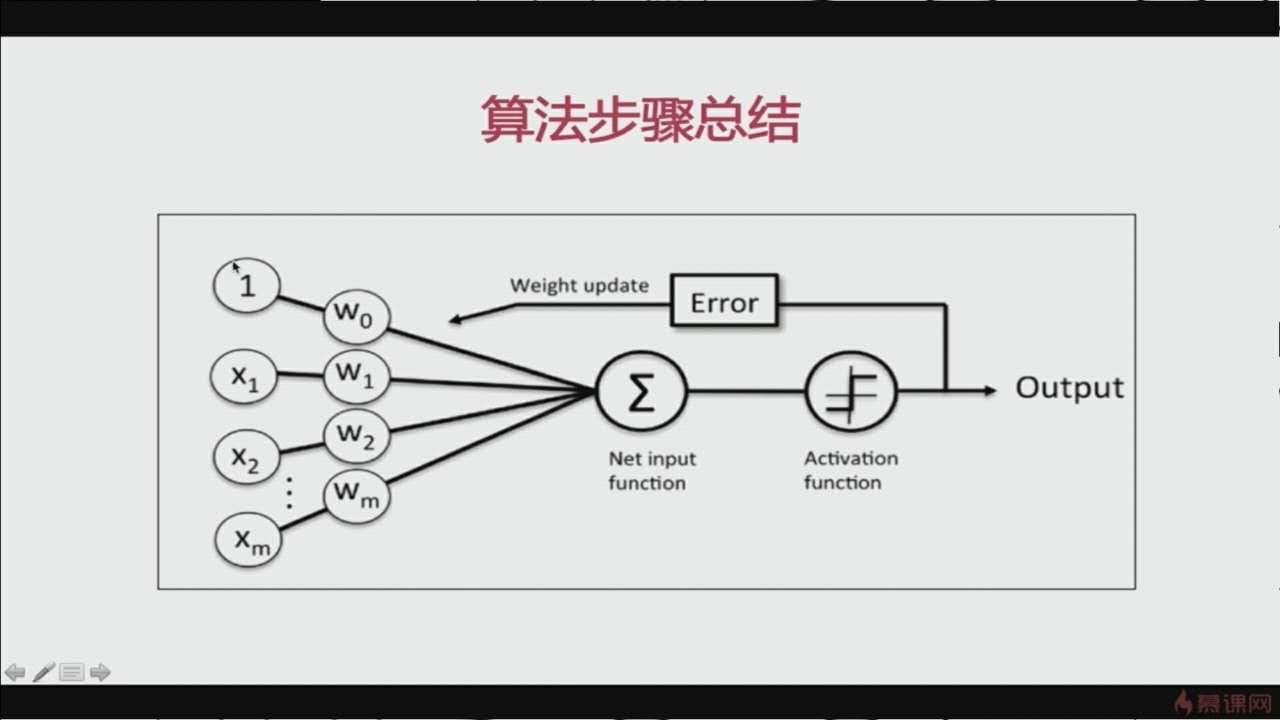
## 感知器算法适用范围



需要可线性分割 <> 线性不可分割；

目标是找到中间的虚线

## 感知器算法总结



1. 初始化权重向量 w
2. 训练样本x输入感知器
3. Net input function：点积求和
4. 结果输入步调函数/激活函数
5. 如果得到结果正确则输出
6. 如果得到结果错误就更新权重重新运算

## Python 代码

代码: https://github.com/FFGF/ML

软件: Anaconda Navigator –> environment –> root –> 右键：open with Jupyter Notebook –> new –> python

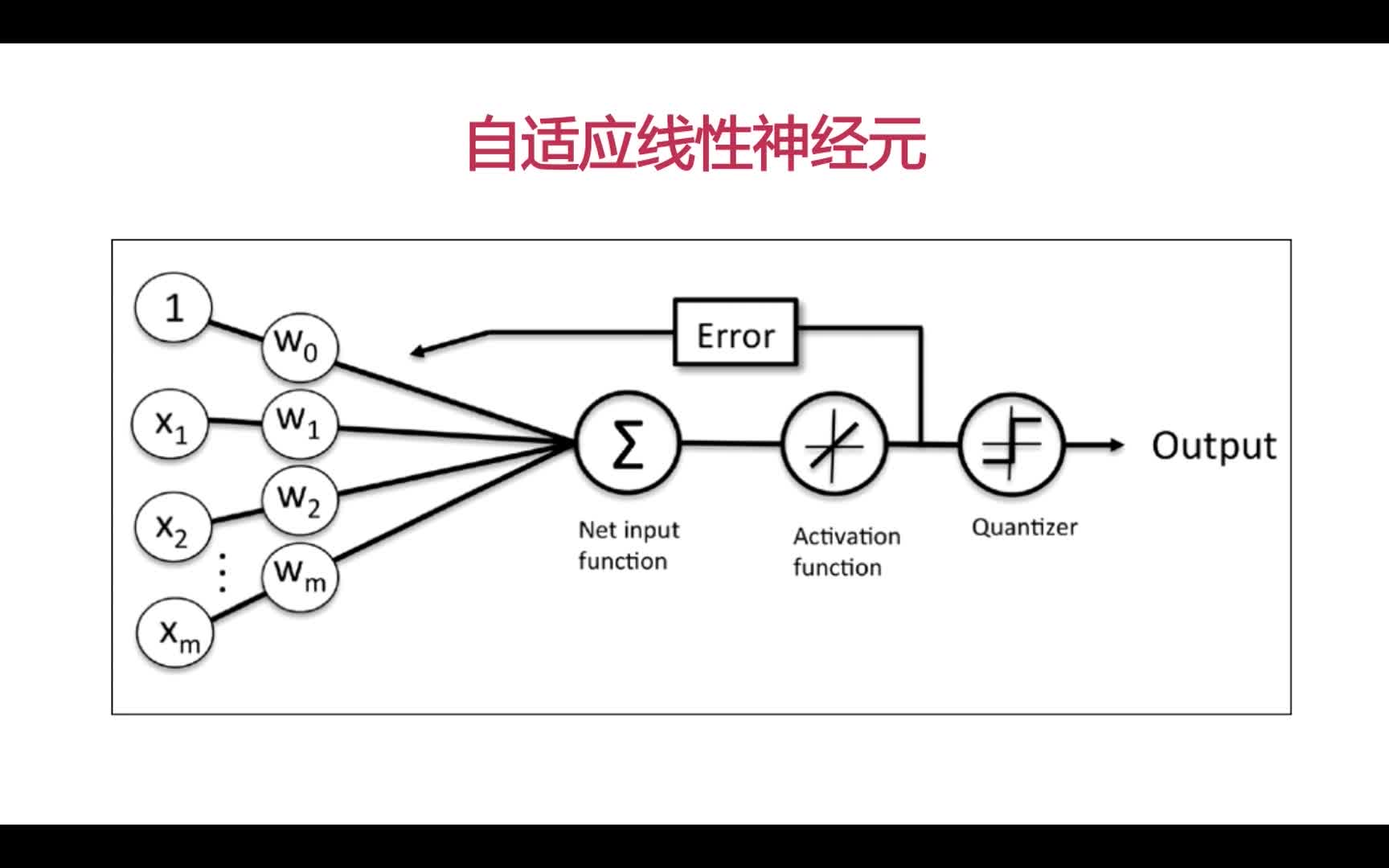
## 学习率

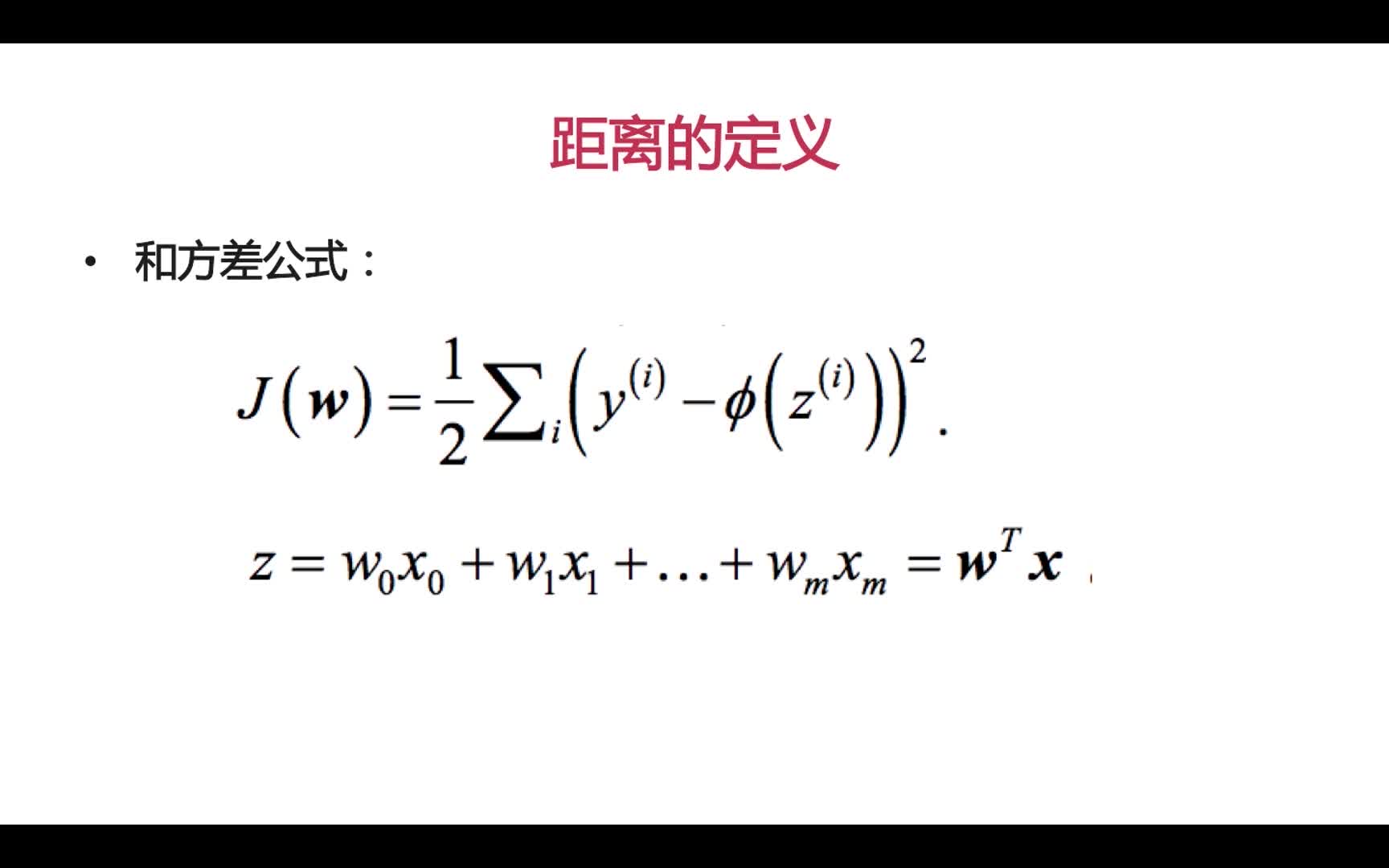
学习率大,如5,变动就越大,就会在数据线的两边大范围跳动

学习率小,如0.1,变动就越小,就慢慢地蔓延到数据线上,但效率低和可能过度拟合

# 自适应线性神经元

## 自适应性神经元





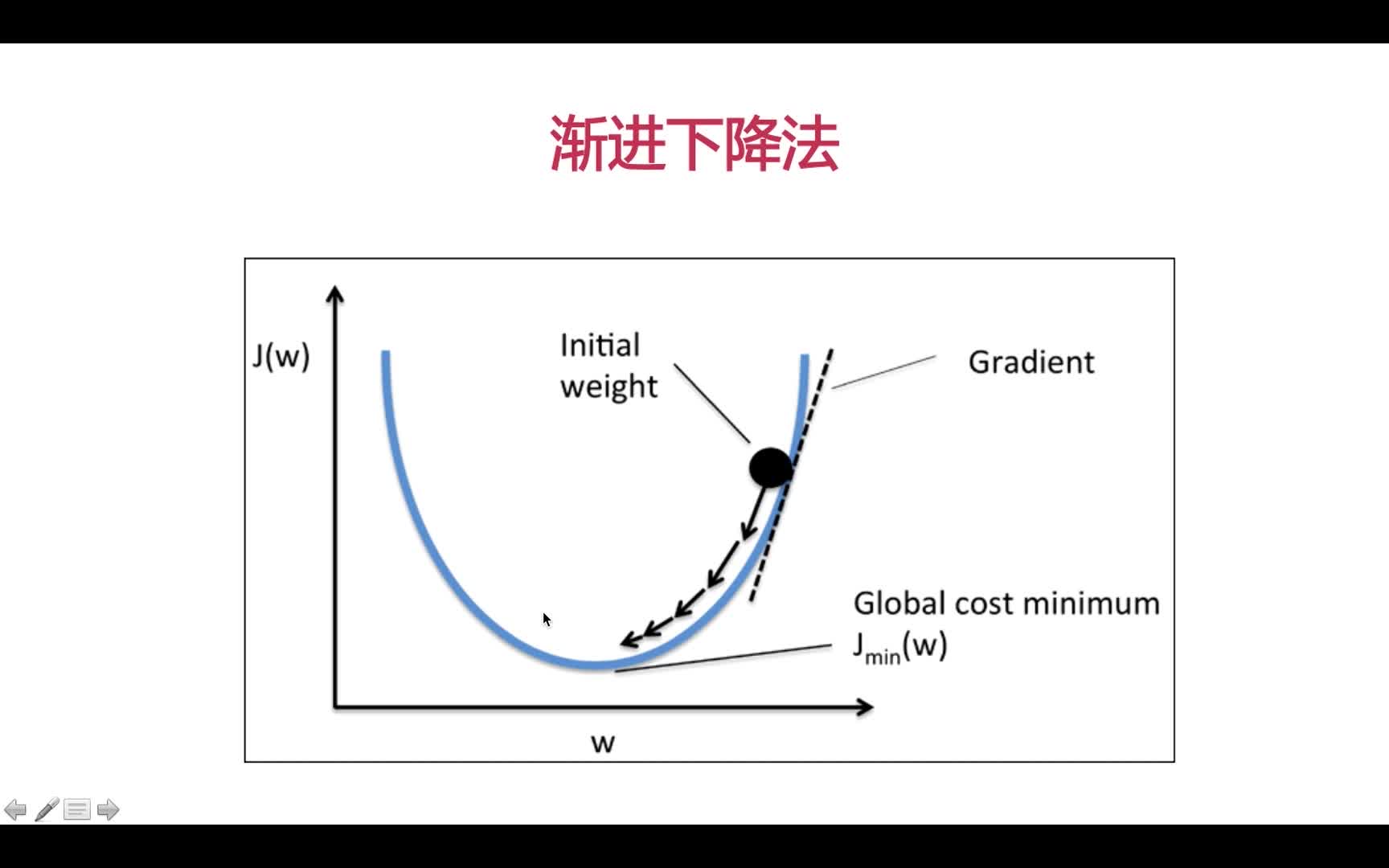
y^i 是i组数据对应的正确结果

z^i 时 i组数据神经网络算出的结果

φ() 是激活函数：判定 两个向量的点积z 而输出0或1

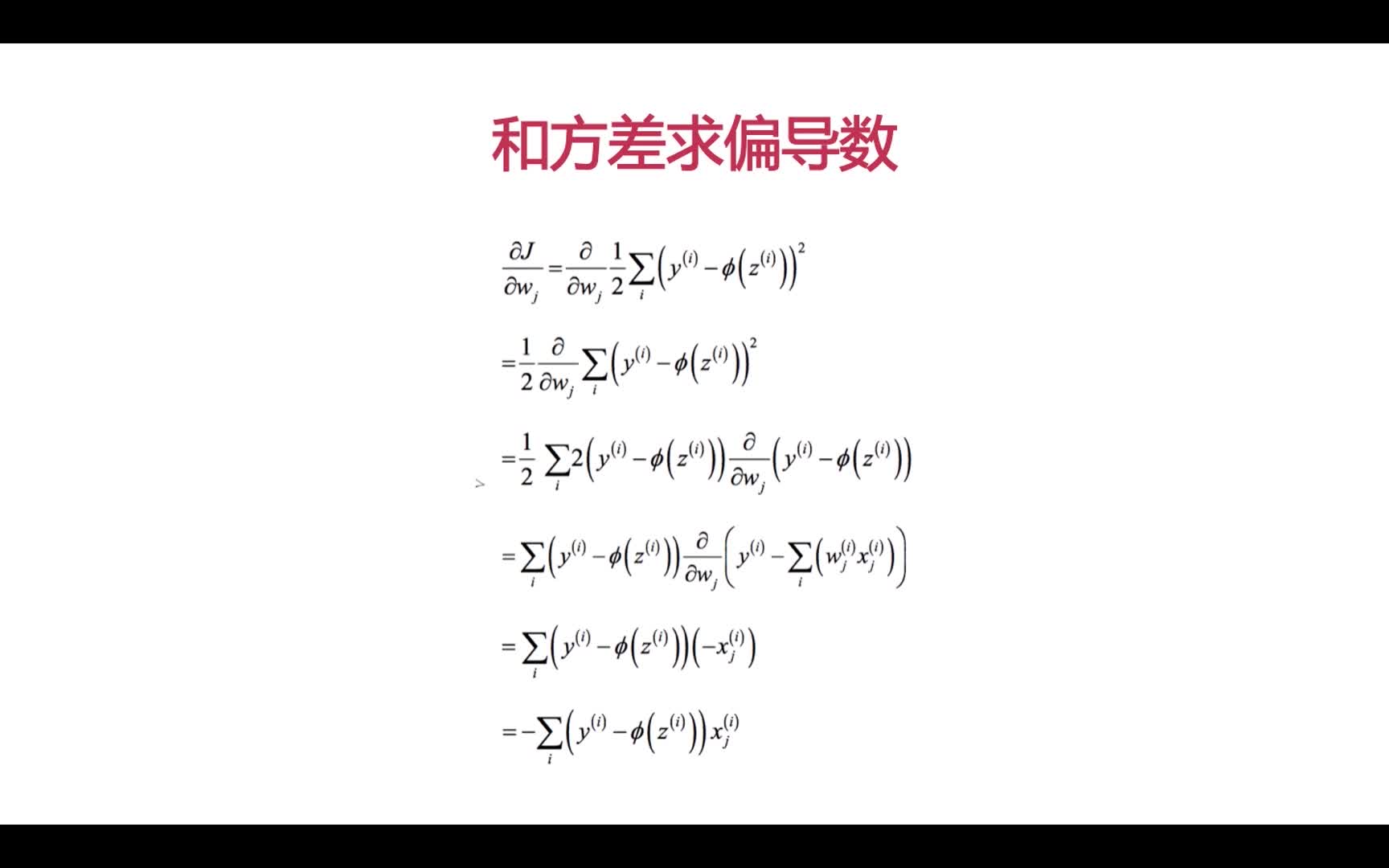
j(w) = ½ \* 误差的平方求和

调整 不同的w得到j(w)结果最小的时候，效果最好

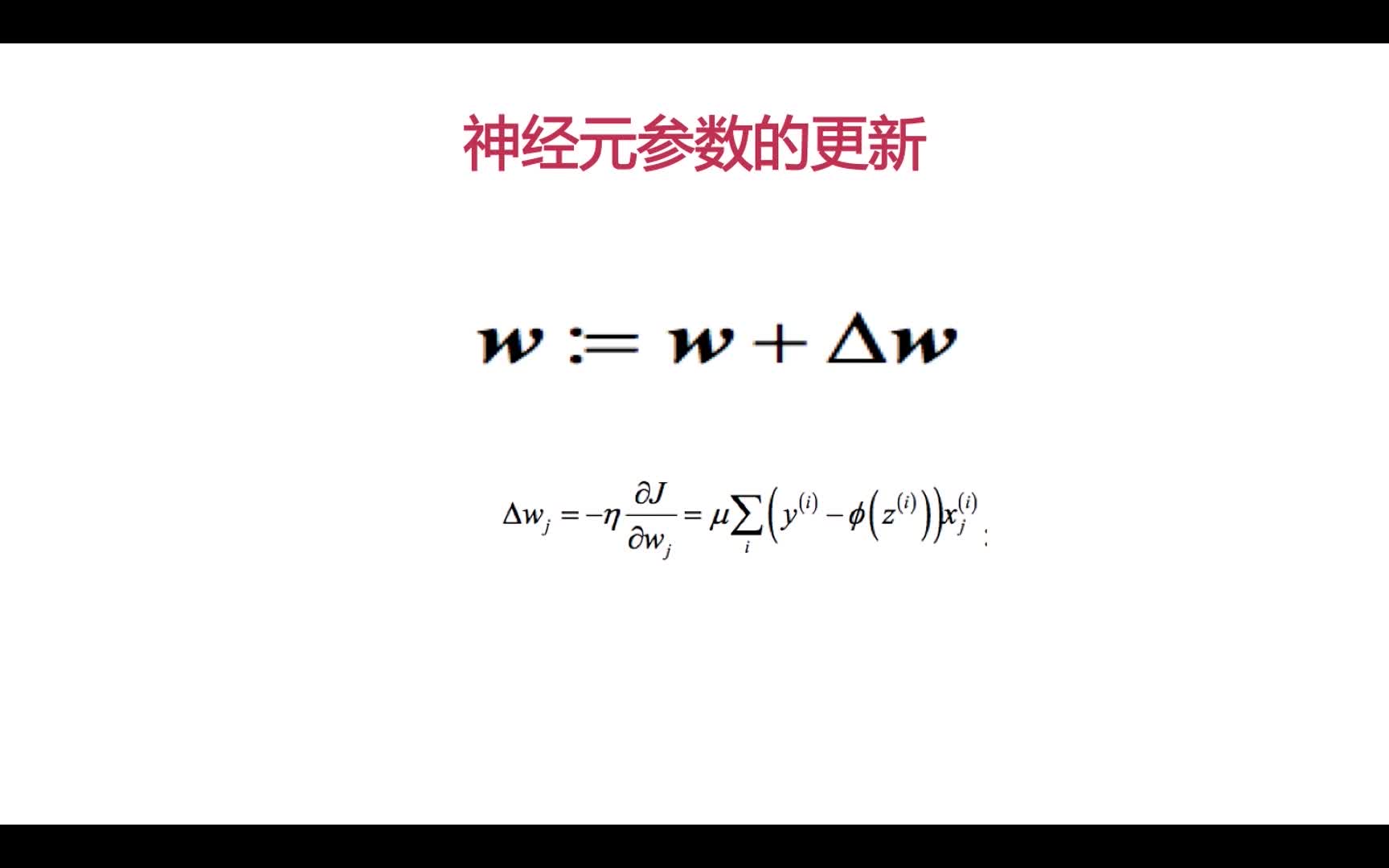


调整算法

根据每个神经元的参数w求偏导数-》看切线的斜率-》如果斜率时大于零（右边）就是减少该神经元的比重w-》如果切线的斜率小于零就增大该神经元的比值



1. 对整个求和公式求偏导
2. 把第一行的最后的平方2提到最前面来
3. 把激活函数转换为基本形式
4. 再对里面的部分求偏导数 –》所有跟J(w) 无关的都变成0，所以只剩下 –x^i\_j
5. 把负号拉到前面就是最后的公式



如果斜率大于零就减少w的值，如果小于零则增加；

η eta是学习率

是斜率

变动越小，神经元的准确率就越高，就越准确