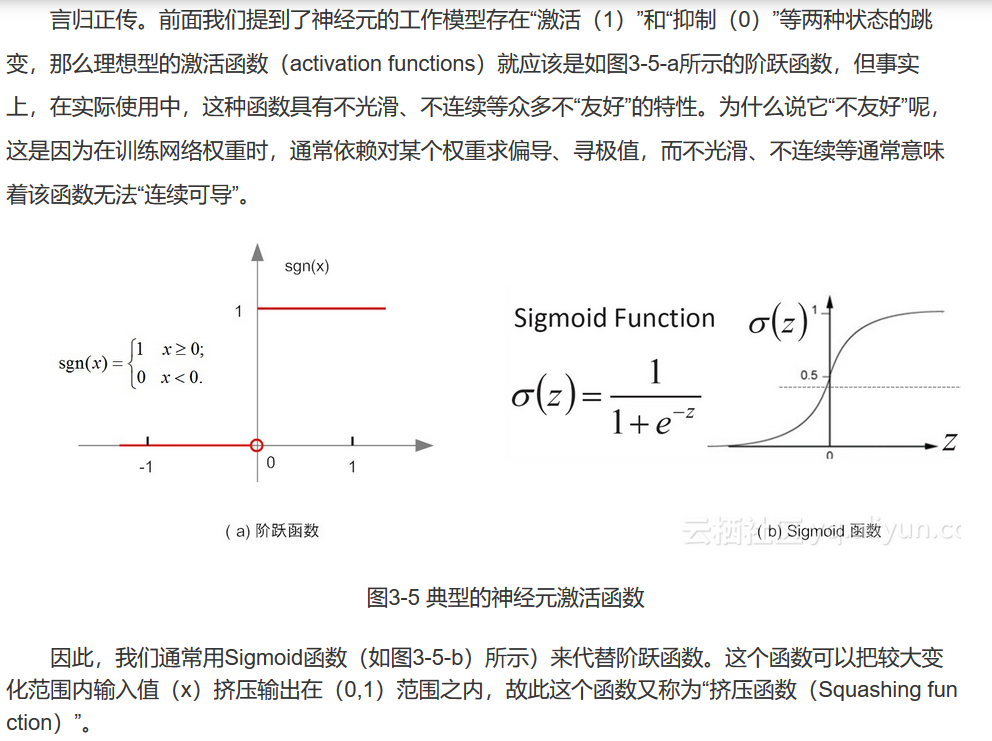
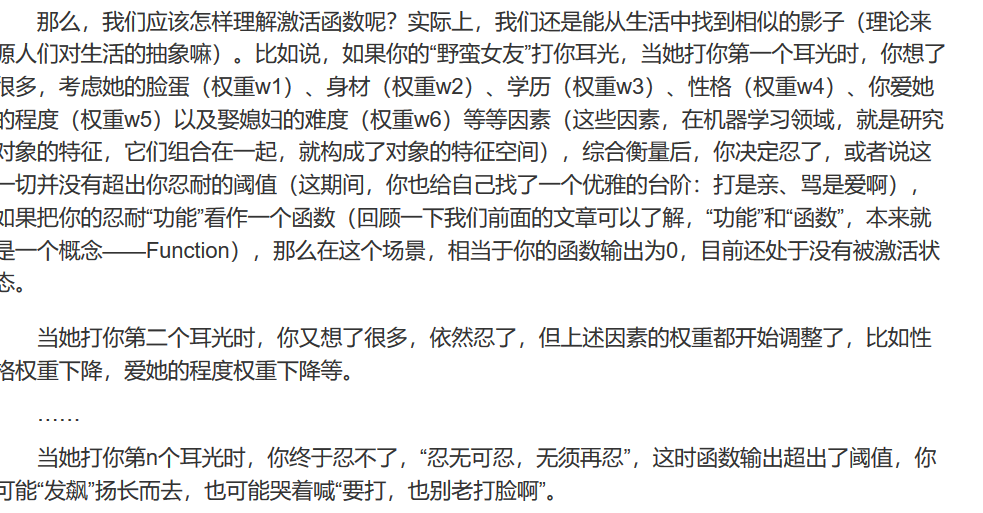
1 激活函数



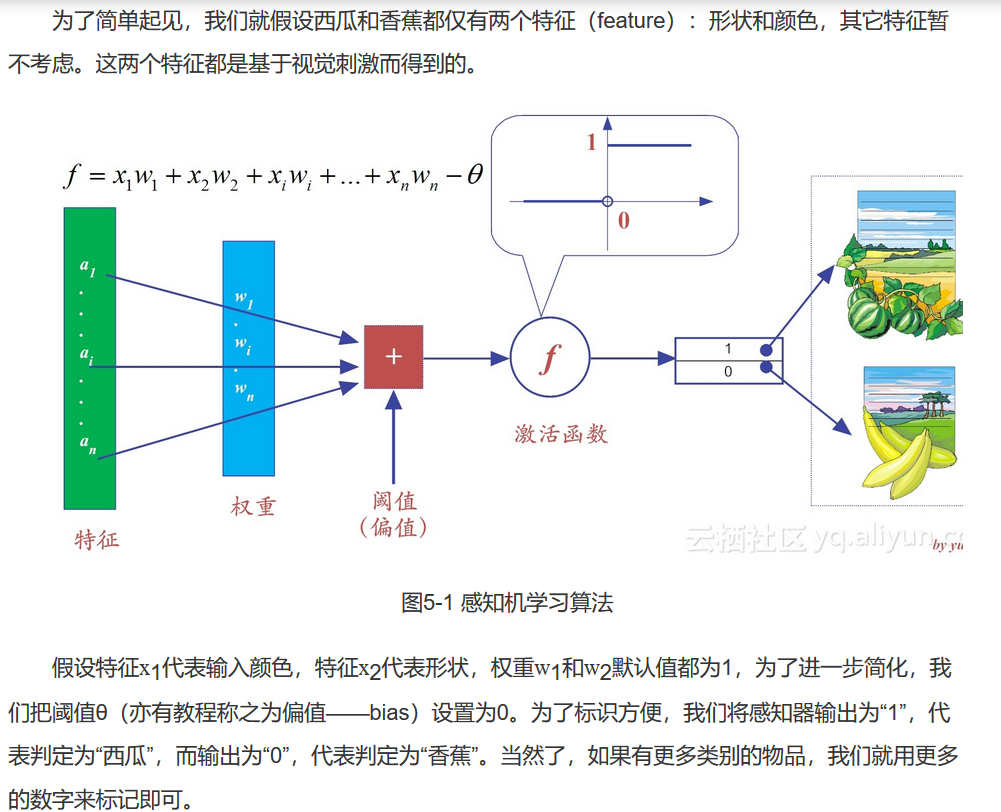
2 什么是激活函数

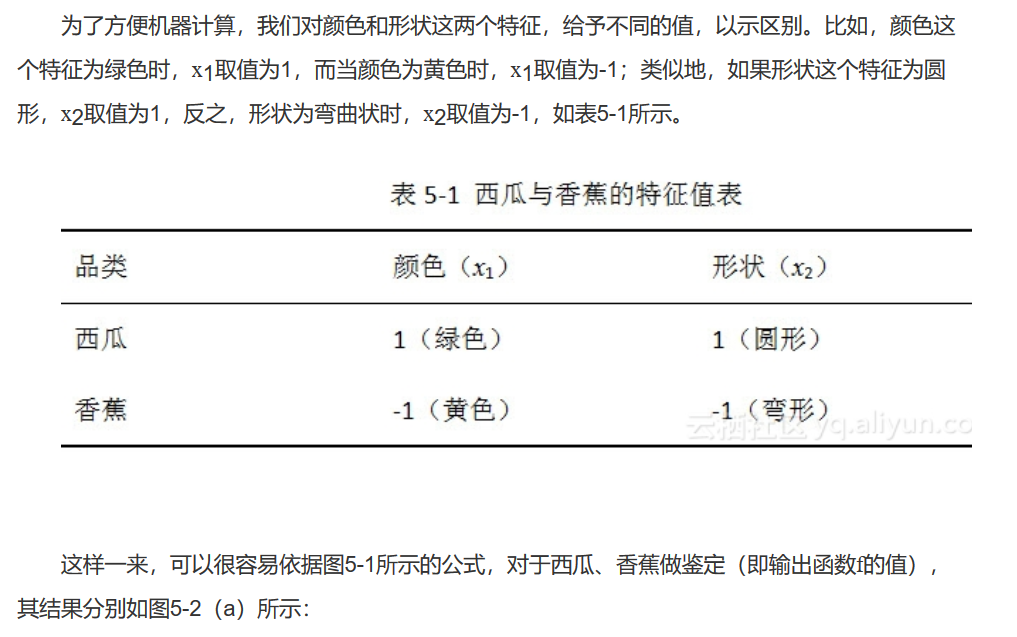


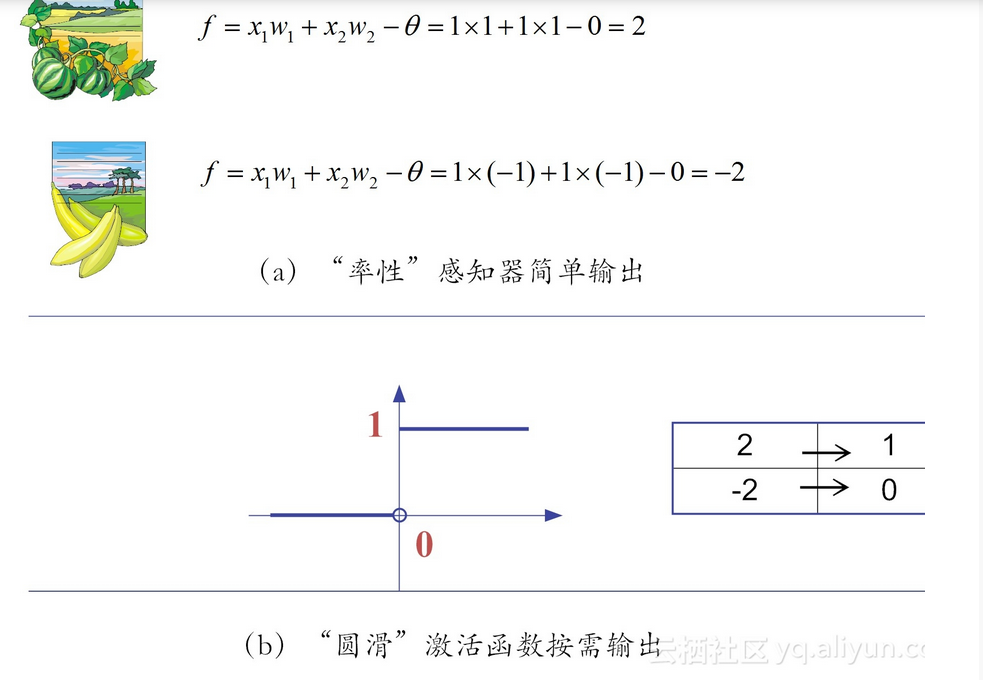
激活函数存在于隐藏层和输出层，对当前神经元聚合之后的结果进行决策，决定输出状态，激活还是抑制，1,0

3感知机

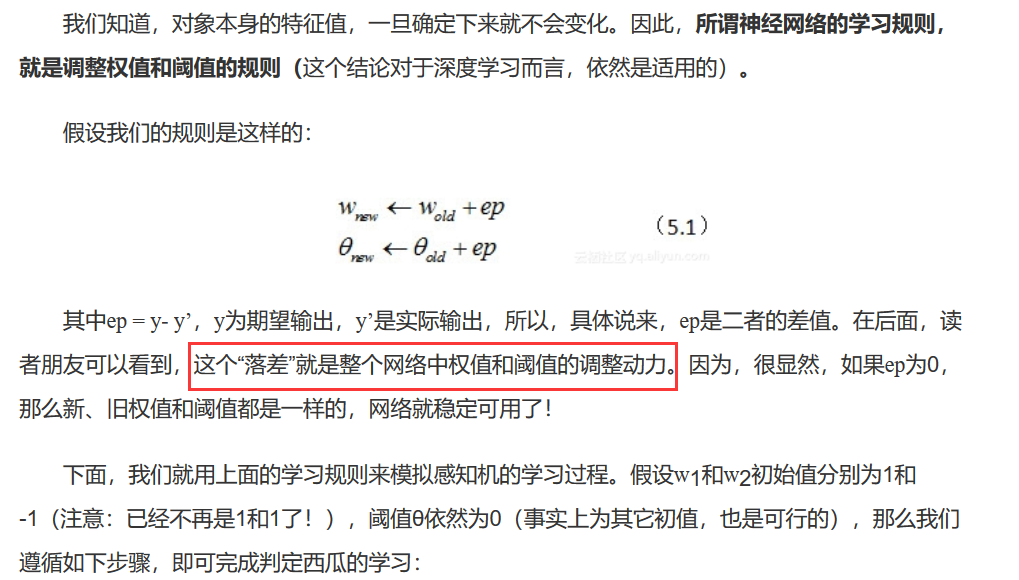
我们已经提到，所谓的感知机，其实就是一个由两层神经元构成的网络结构，它在输入层接收外界的输入，通过激活函数（含阈值）的变换，把信号传送至输出层，因此它也称之为“阈值逻辑单元（threshold logic unit）”。

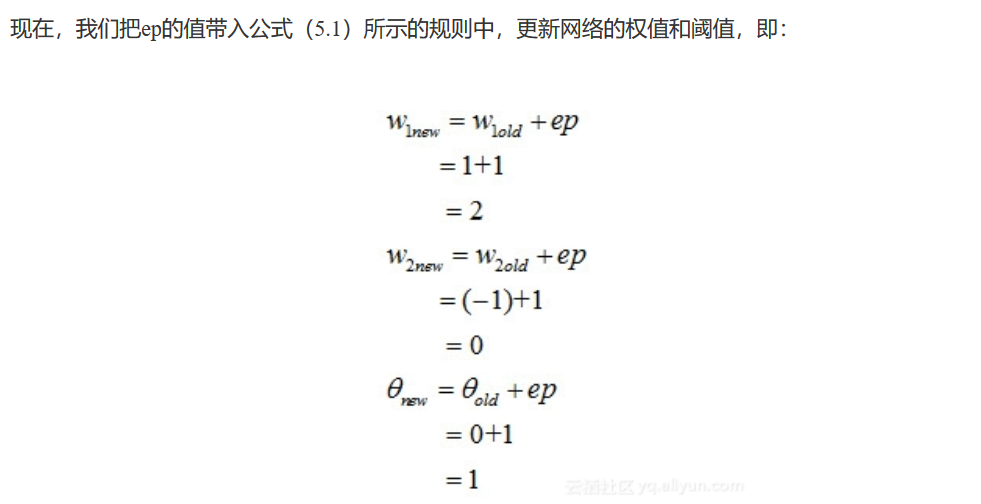






4 感知机如何学习

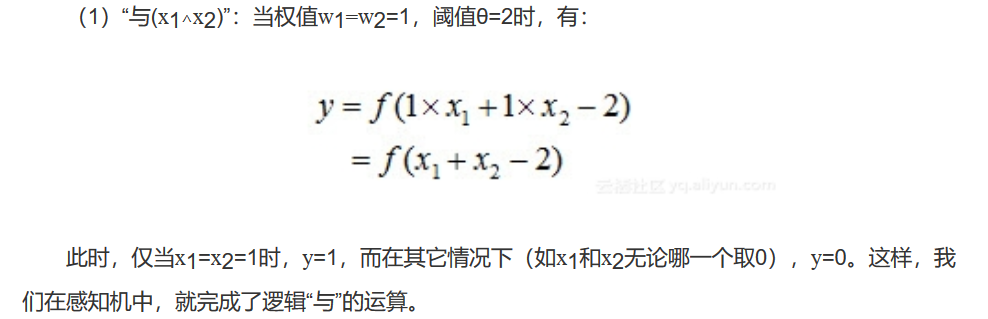




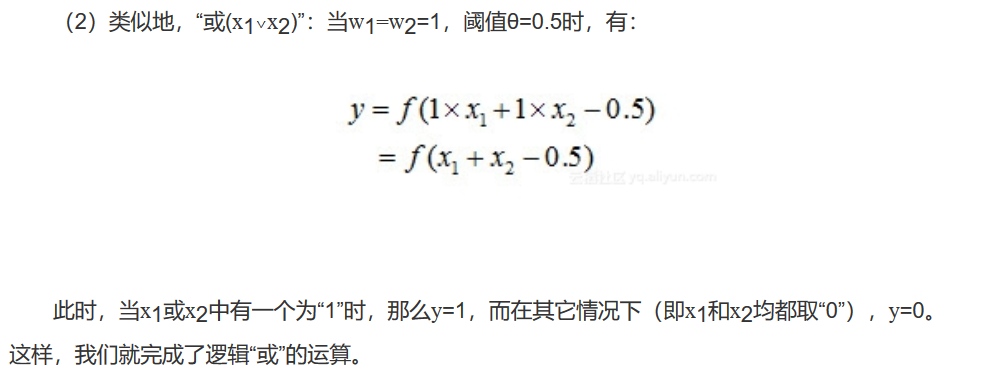
5 感知机更一般的训练法则

通过前面的分析，我们可以看到，感知机是很容易实现逻辑上的“与（AND）”、“或（OR）”、“非（NOT）”等原子布尔函数（Primitive Boolean function），如图5-3所示（睿智如你，你肯定发现了，这里的确没有“异或”，就是相同为1，不同为0）[2]。

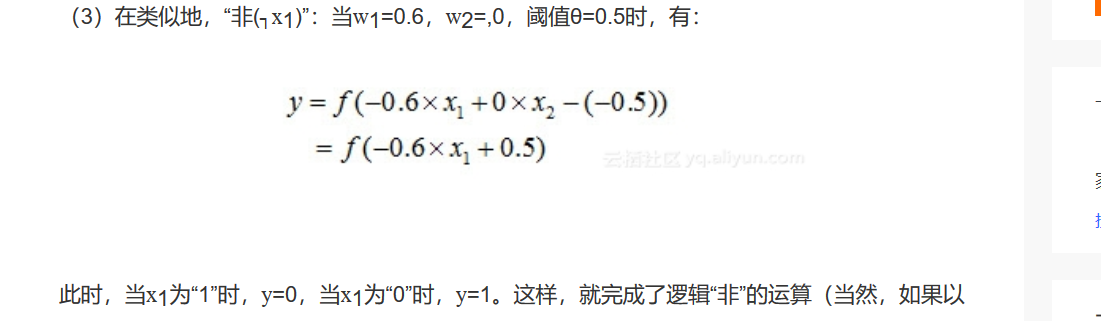
与操作



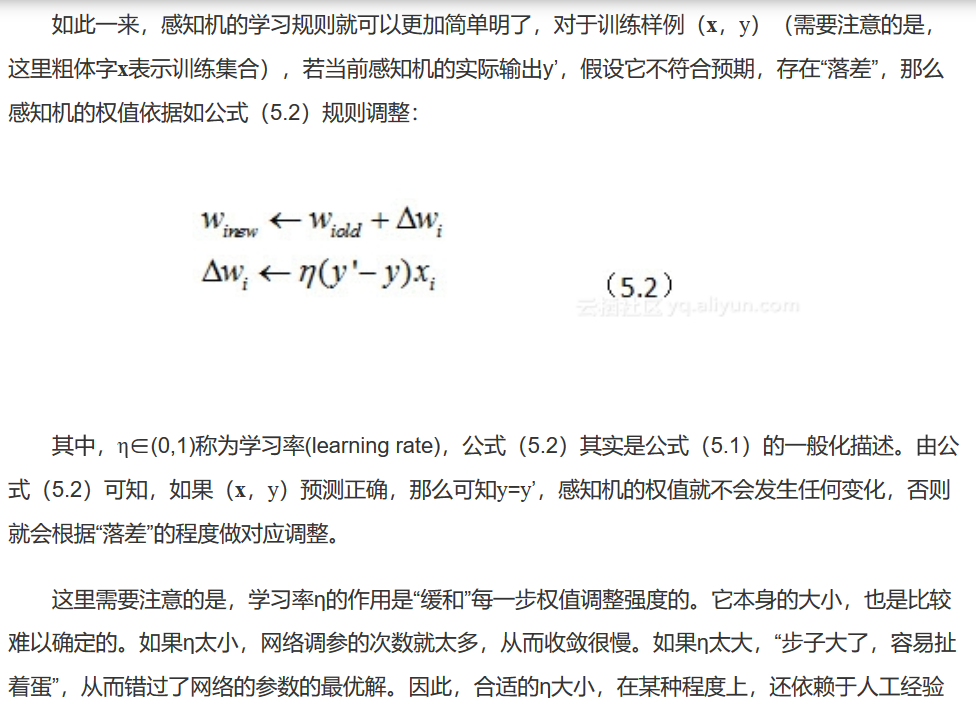
或操作



非操作

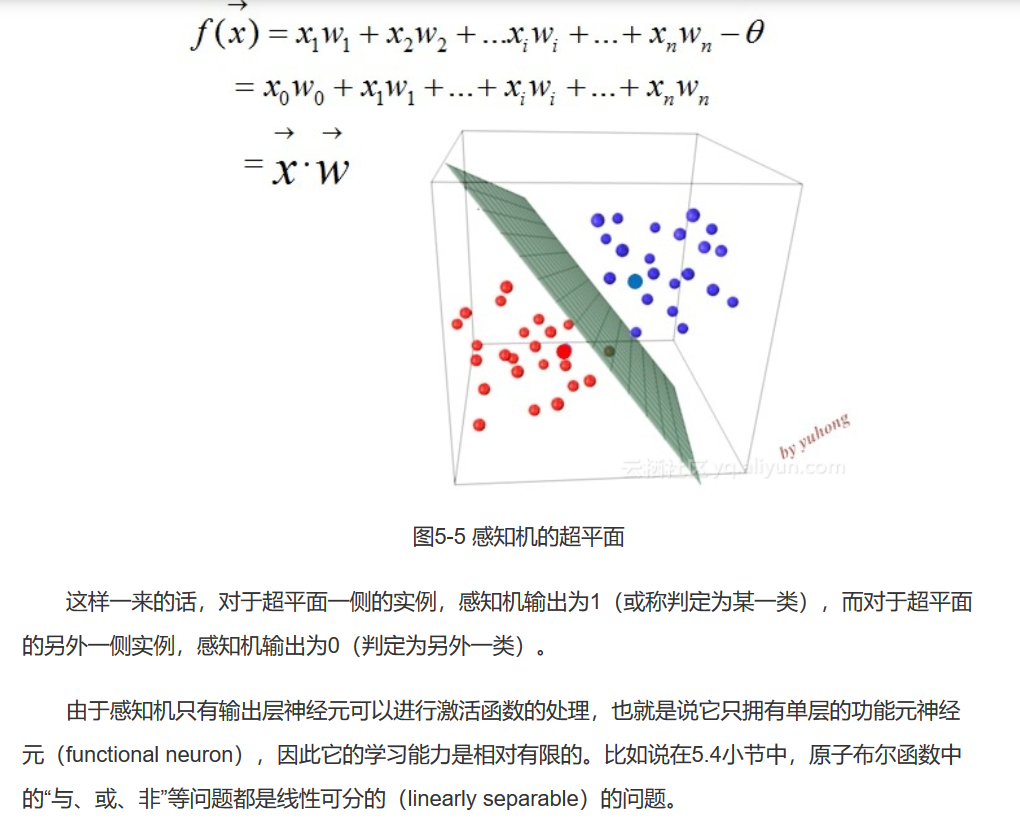


更一般地，当我们给定训练数据，神经网络中的参数（权值wi和阈值θ）都可以通过不断地“纠偏”学习得到。为了方便起见，我们把阈值θ视为w0，而其权值设为固定值“-1”，那么阈值θ就可视为一个“哑节点（dummy node）”。这样一来，权重和阈值的学习可以“一统天下”称为“权重”的学习。



5.5 感知机的表征能力

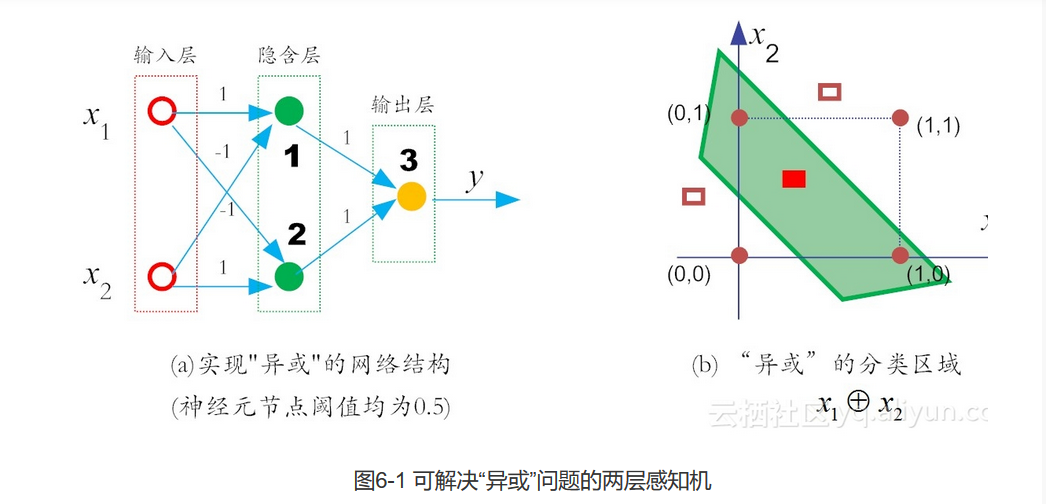
如果识别对象x有n个特征，那么感知机可以看做，在n维实例空间（即点空间）中的超平面决策面，以向量的模式写出来就是如图5-5所示。



由于感知机只有输出层神经元可以进行激活函数的处理，也就是说它只拥有单层的功能元神经元（functional neuron），因此它的学习能力是相对有限的。比如说在5.4小节中，原子布尔函数中的“与、或、非”等问题都是线性可分的（linearly separable）的问题。

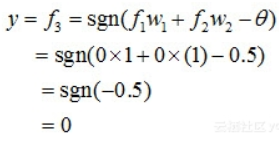
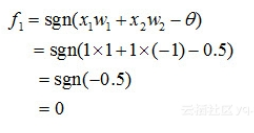
6多层神经网络

想解决“异或”问题，就需要让网络复杂起来。这是因为，复杂的网络，表征能力就比较强[1]。按照这个思路，我们在输入层和输出层之间，添加一层神经元，将其称之为隐含层（hidden layer，亦有简称为“隐层”）。这样一来，隐含层和输出层中的神经元都拥有激活函数。假设各个神经元的阈值均为0.5，权值如图6-1所示，就实现了“异或”功能。



**那么，对于x1和x2相同（即均为1）时，对于在隐含层的神经元1有：**

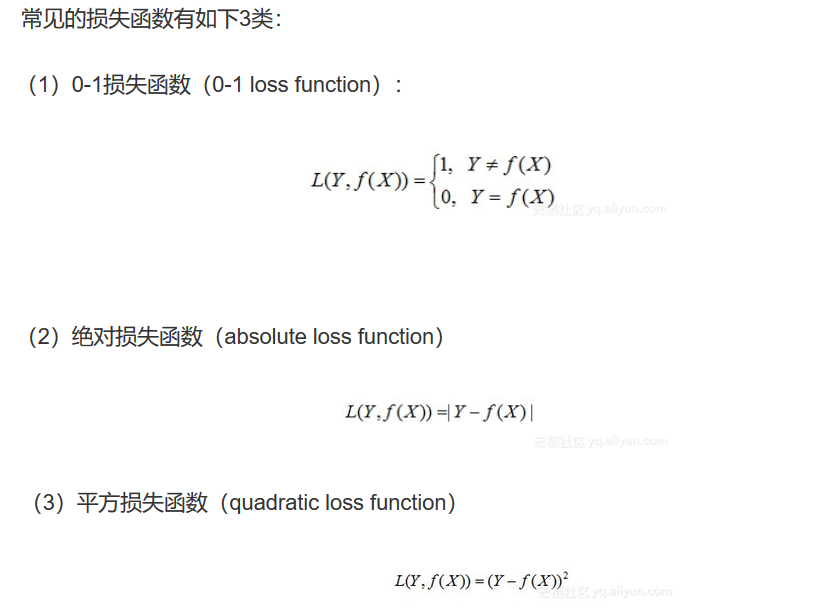
**H1 H2 output**

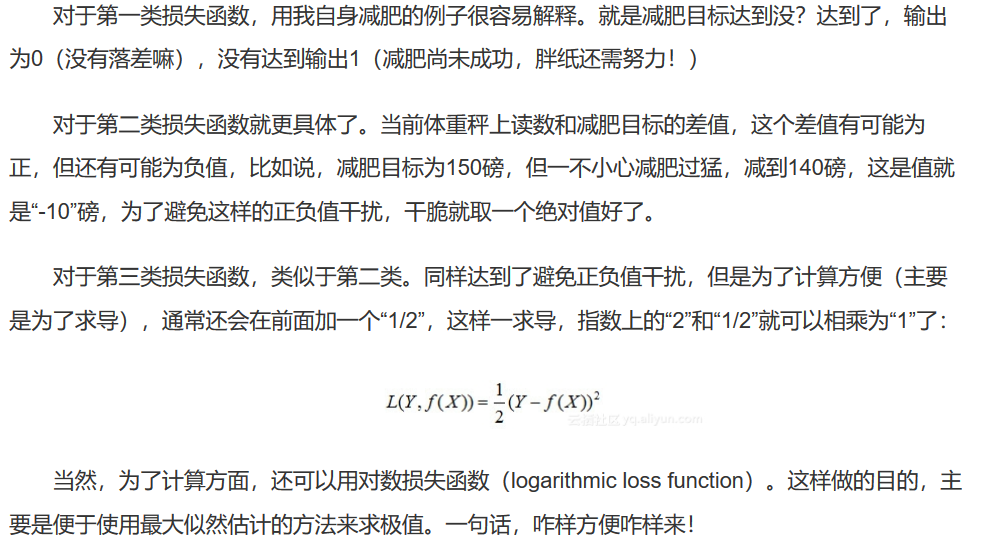


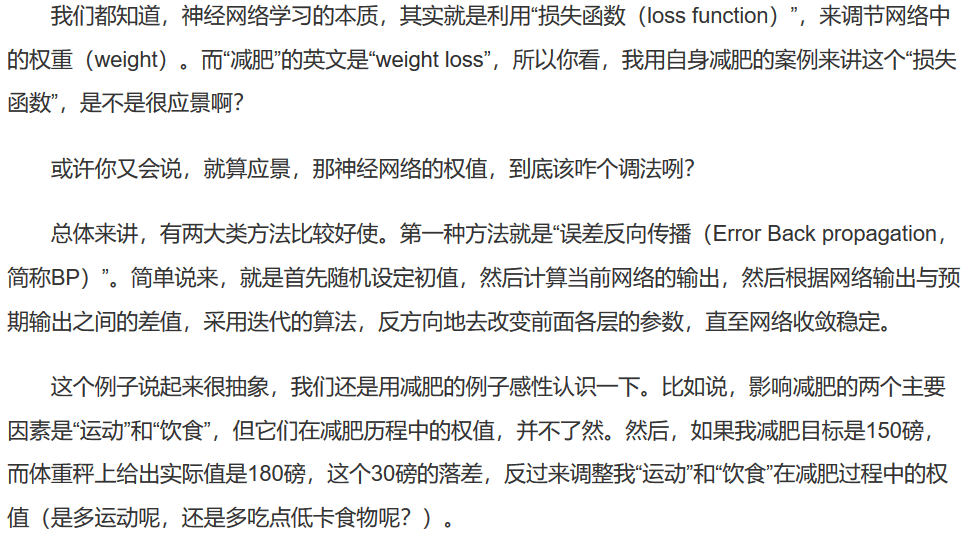
**隐藏层两个节点的输出都是0 ，接下来进入输出层，H1 H2是他的输入**

**，x1和x2同为1时，输出为0。如果x1和x2不同时，输出为1，从而证明了多层神经网络可以实现异或操作**

**7 损失函数**



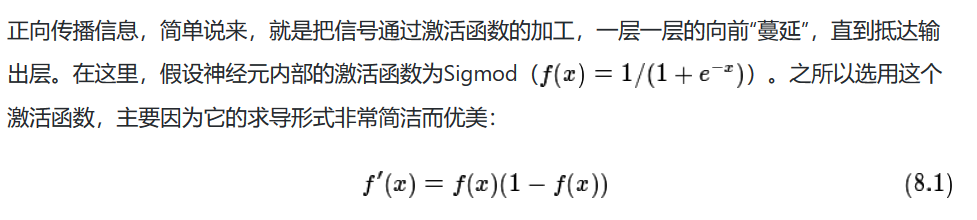




**8 反向传播算法**

**BP算法，我们通常强调的是反向传播，但其实呢，它是一个典型的双向算法。更确切来说，它的工作流程是分两大步走：（1）正向传播输入信号，输出分类信息（对于有监督学习而言，基本上都可归属于分类算法）；（2）反向传播误差信息，调整全网权值（通过微调网络参数，让下一轮的输出更加准确）。**

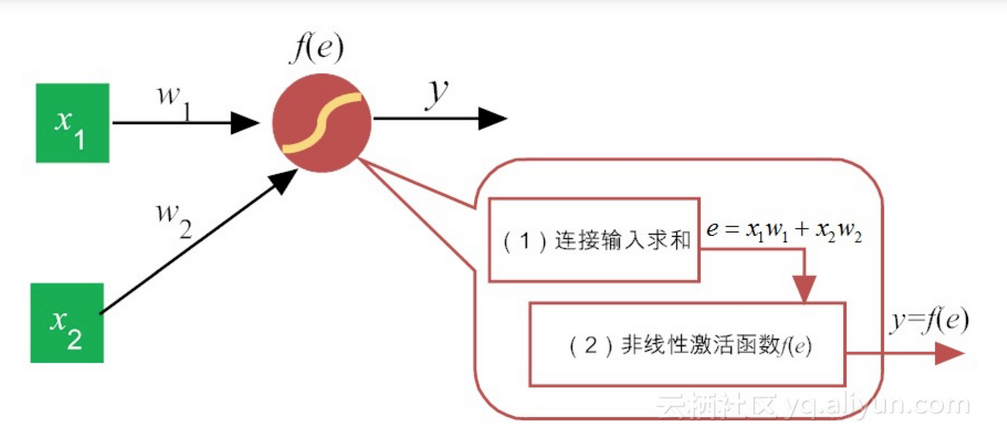
1. **正向传播信息**

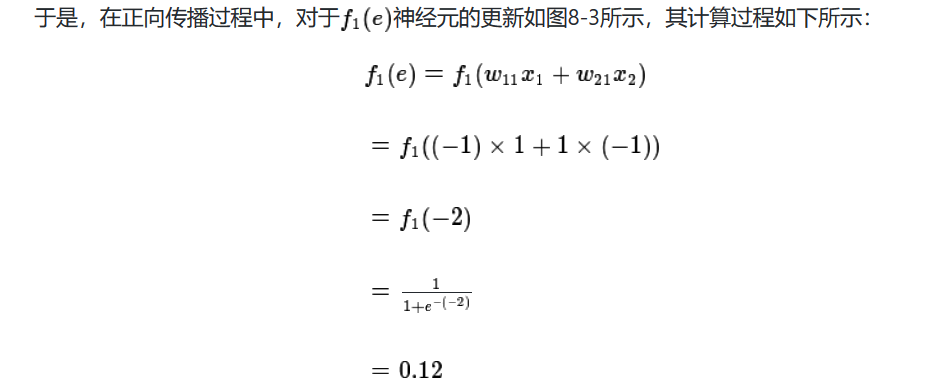


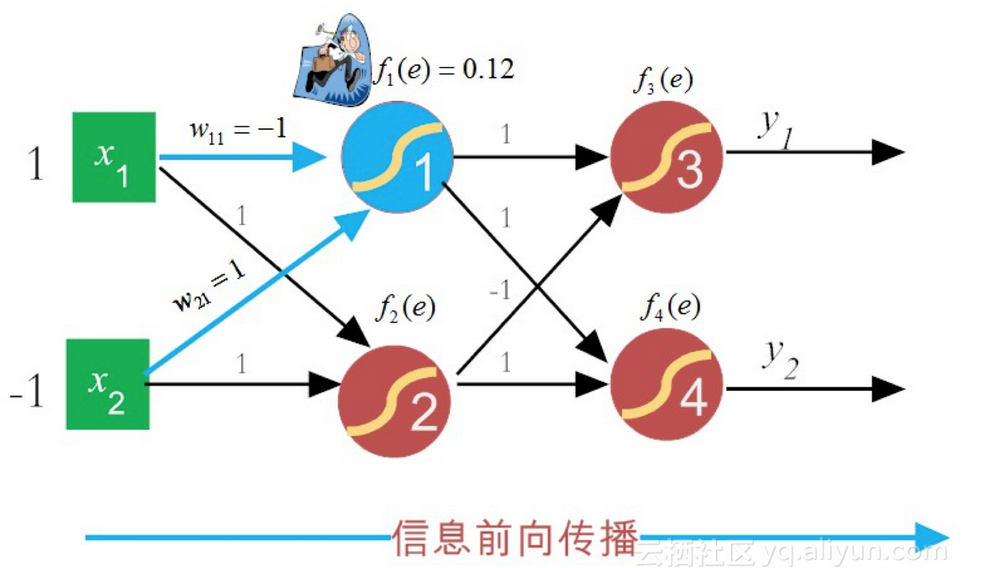
**事实上，类似于感知机，每一个神经元的功能都可细分两大部分：**

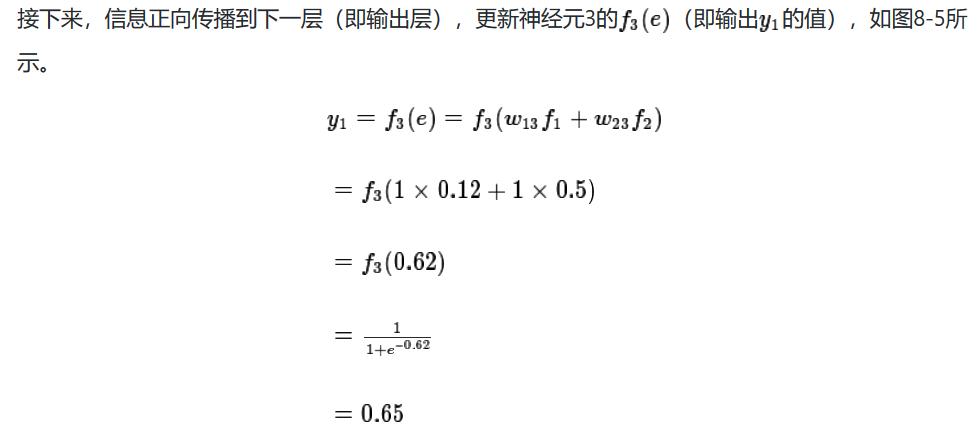
**（1）汇集各路链接带来的加权信息；**

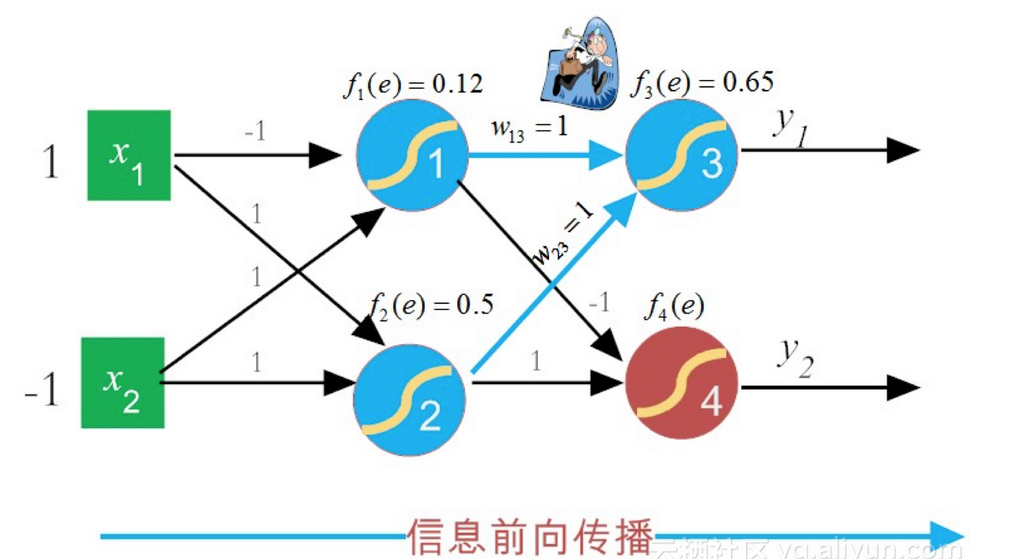
**（2）加权信息在激活函数的“加工”下，神经元给出相应的输出**





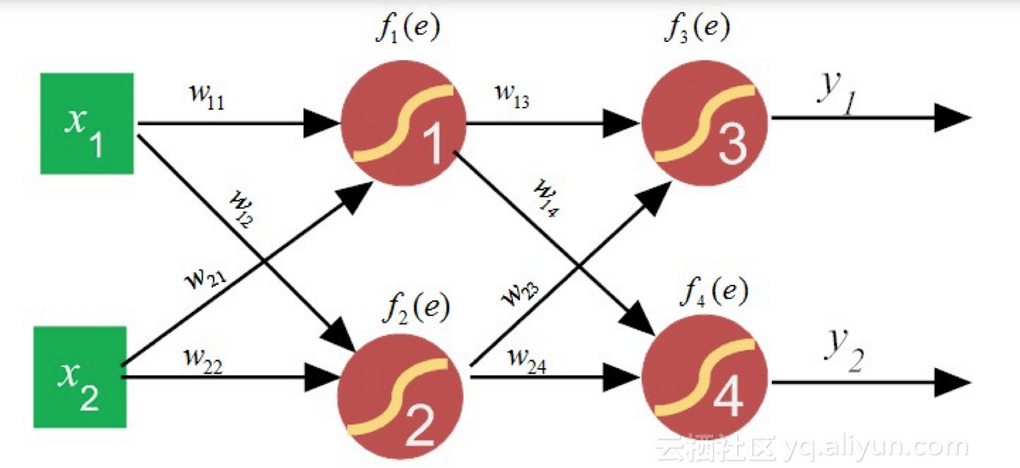




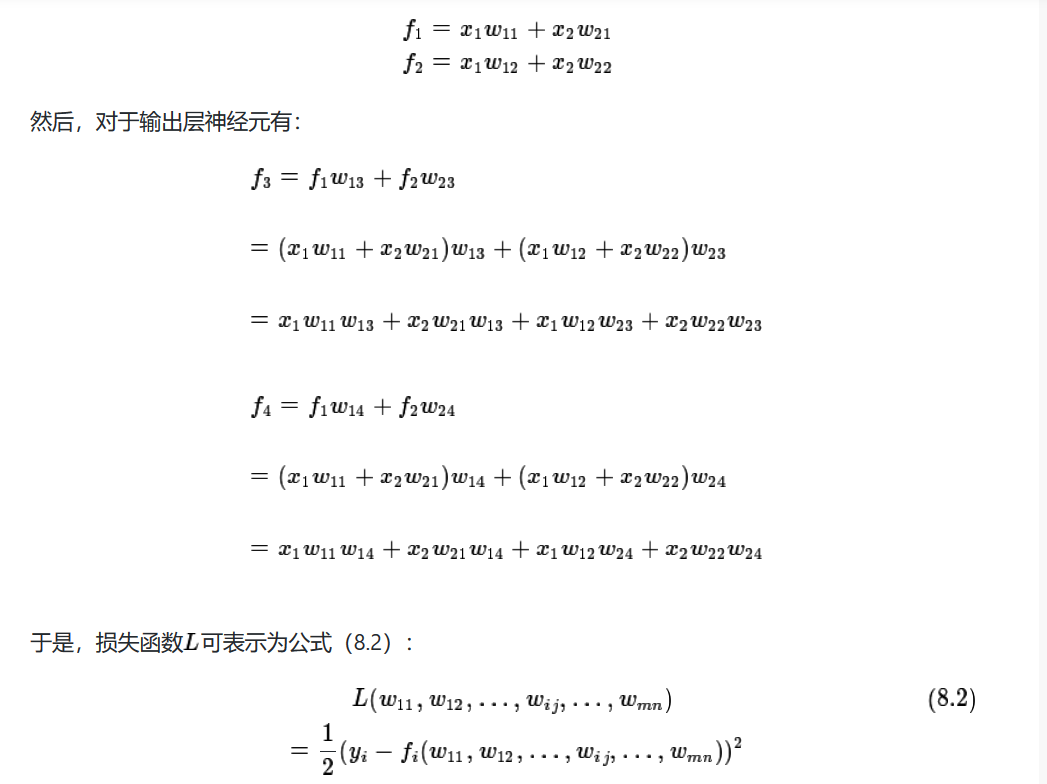


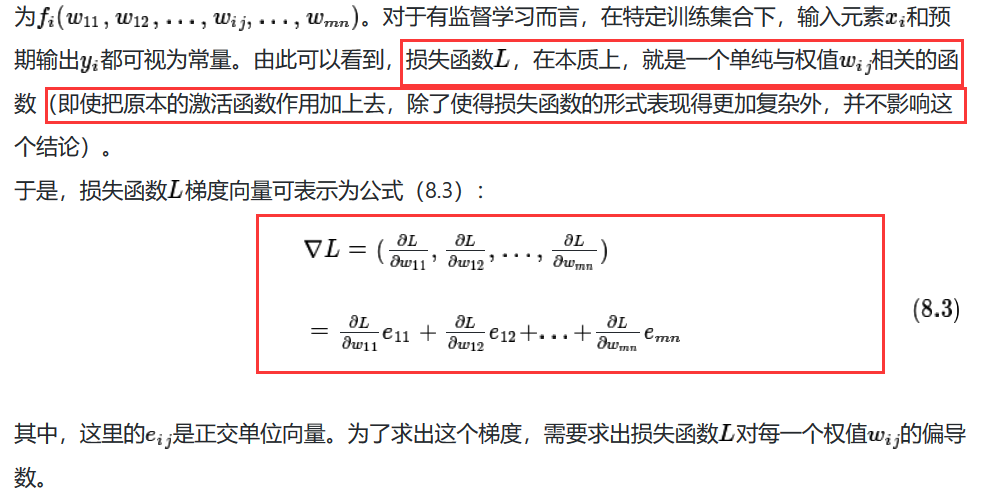
到此，在第一轮信号前向传播中，实际输出向量已计算得到 ，但我们预期输出的向量（即教师信号）是 ，这二者之间是存在“误差”的。于是，重点来了，下面我们就用“误差”信息反向传播，来逐层调整网络参数。为了提高权值更新效率，这里就要用到下文即将提到的“反向模式微分法则（chain rule）”。

8.2 反向传播算法 最重点



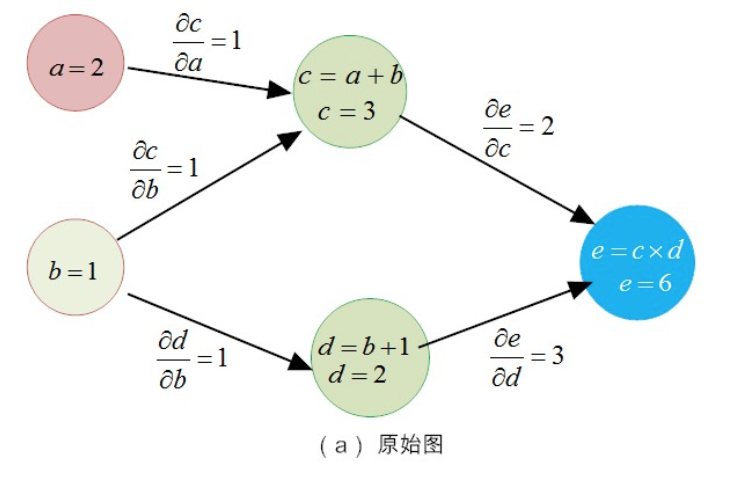
这里为了简化理解，我们暂时假设神经元没有激活函数（或称激活函数为y=x ），于是对于隐含层神经元，它的输出可分别表示为

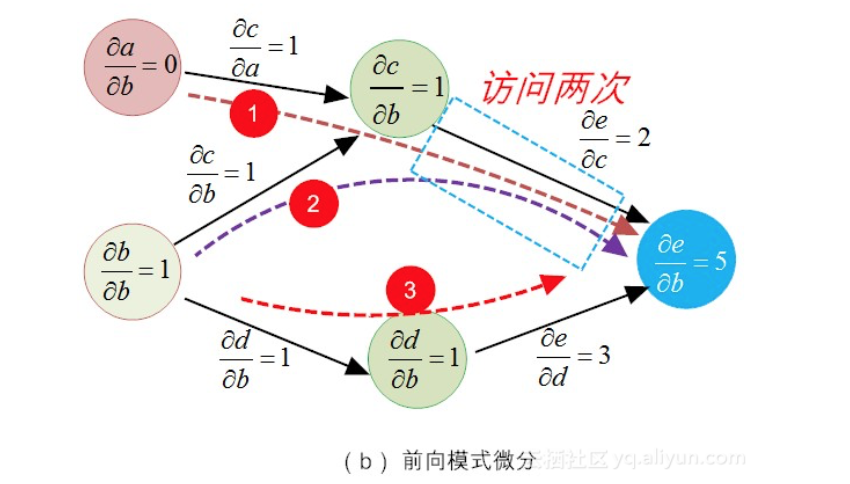




求偏导数方法，有两种

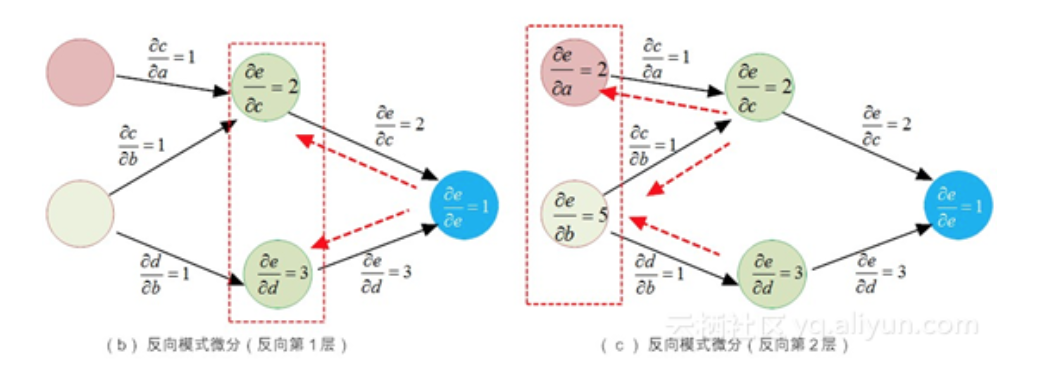
“前向模式微分（forward-mode differentiation）”

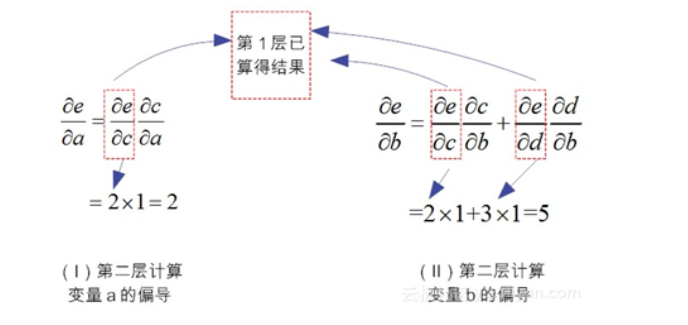




前向微分模式，会重复一些链路的求导过程，增加了计算复杂度

“反向模式微分（reverse-mode differentiation）

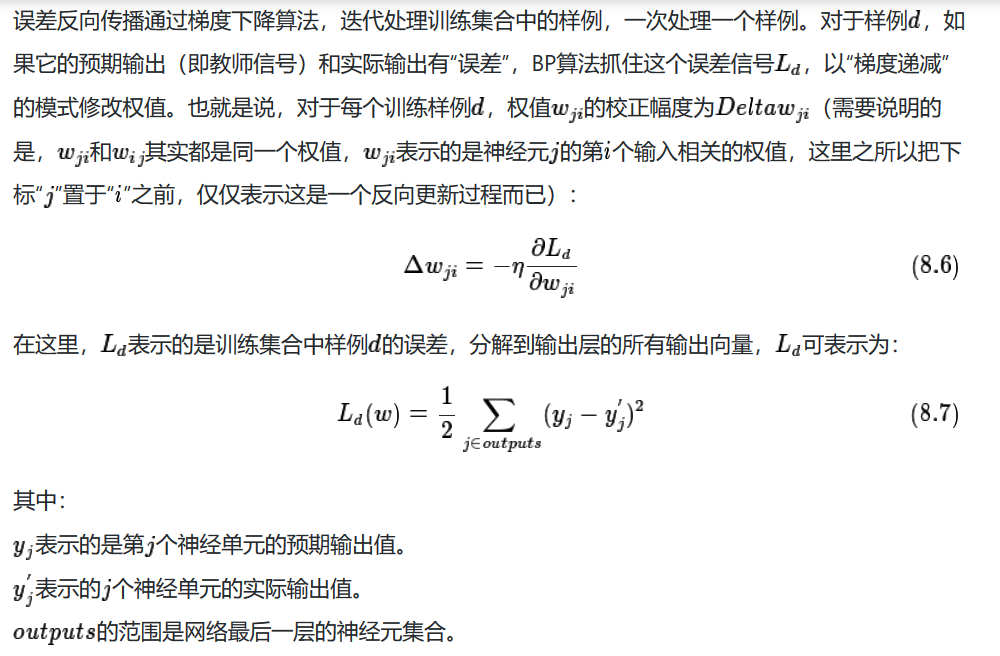




反向模式微分方法，每个路径仅仅遍历一次，就可以求得所有输出（如 节点）对输入（如 或 节点）的偏导，干净利落，没有任何冗余！

在第七章中，我们曾提到，“BP算法把网络权值纠错的运算量,从原来的与神经元数目的平方成正比，下降到只和神经元数目本身成正比。”其功劳，正是得益于这个反向模式微分方法节省的计算冗余

8.3误差反向传播



Softmax函数

https://blog.csdn.net/bitcarmanlee/article/details/82320853

机器学习尤其是深度学习中，softmax是个非常常用而且比较重要的函数，尤其在多分类的场景中使用广泛。他把一些输入映射为0-1之间的实数，并且归一化保证和为1，因此多分类的概率之和也刚好为1。

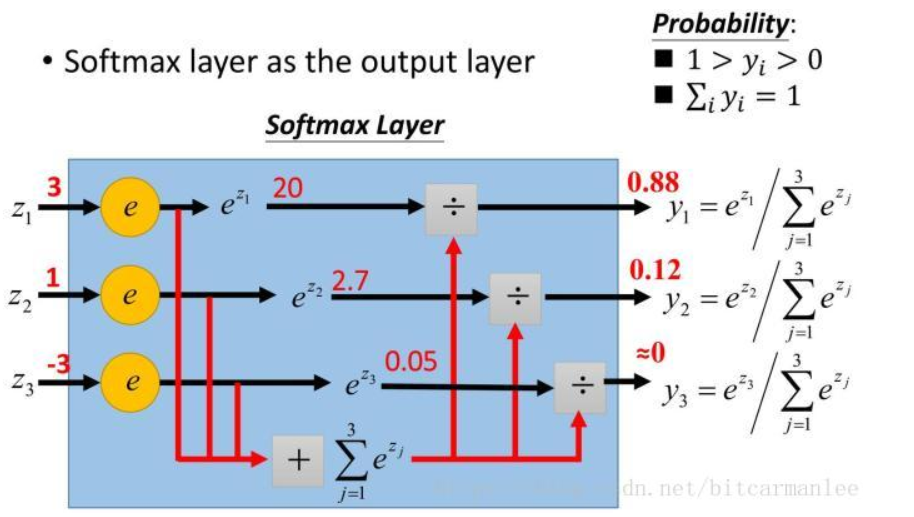
首先我们简单来看看softmax是什么意思。顾名思义，softmax由两个单词组成，

其中一个是max。对于max我们都很熟悉，比如有两个变量a,b。如果a>b，则max为a，反之为b。用伪码简单描述一下就是 if a > b return a; else b。

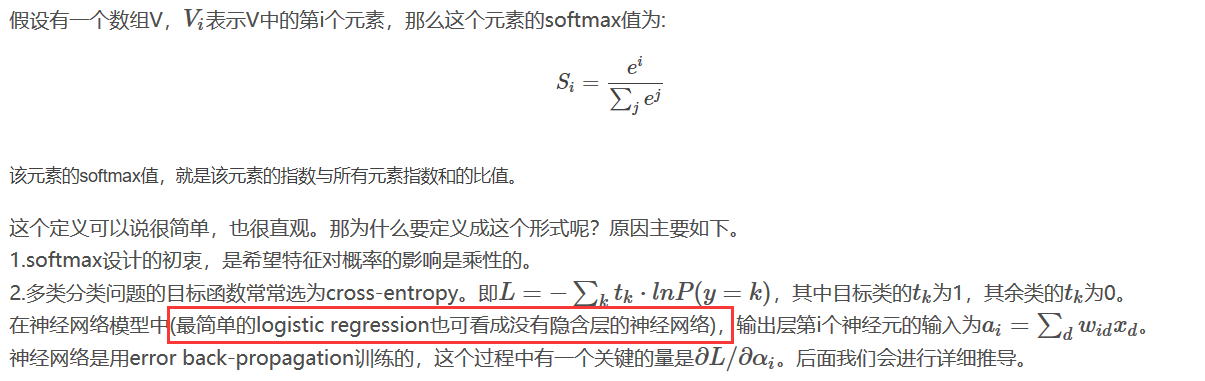
另外一个单词为soft。max存在的一个问题是什么呢？如果将max看成一个分类问题，就是非黑即白，最后的输出是一个确定的变量。更多的时候，我们希望输出的是取到某个分类的概率，或者说，我们希望分值大的那一项被经常取到，而分值较小的那一项也有一定的概率偶尔被取到，所以我们就应用到了soft的概念，即最后的输出是每个分类被取到的概率。

softmax的定义

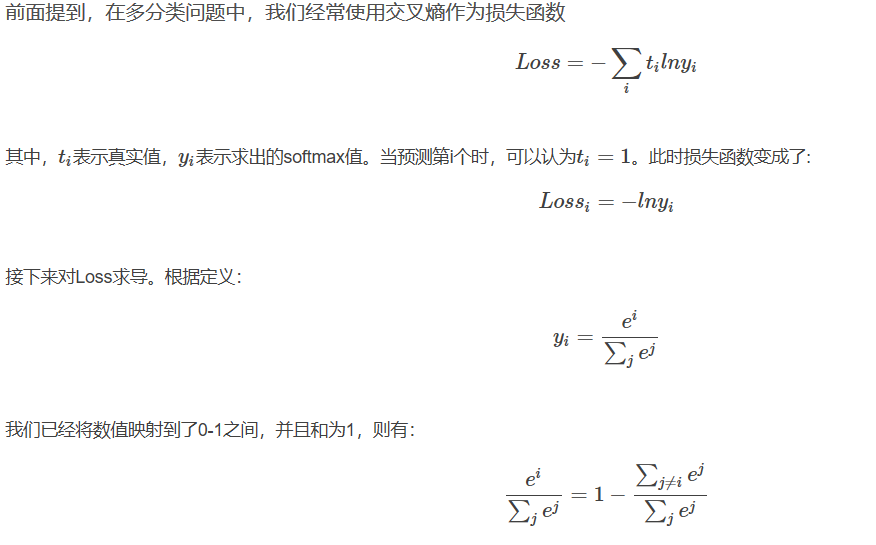
首先给一个图，这个图比较清晰地告诉大家softmax是怎么计算的。

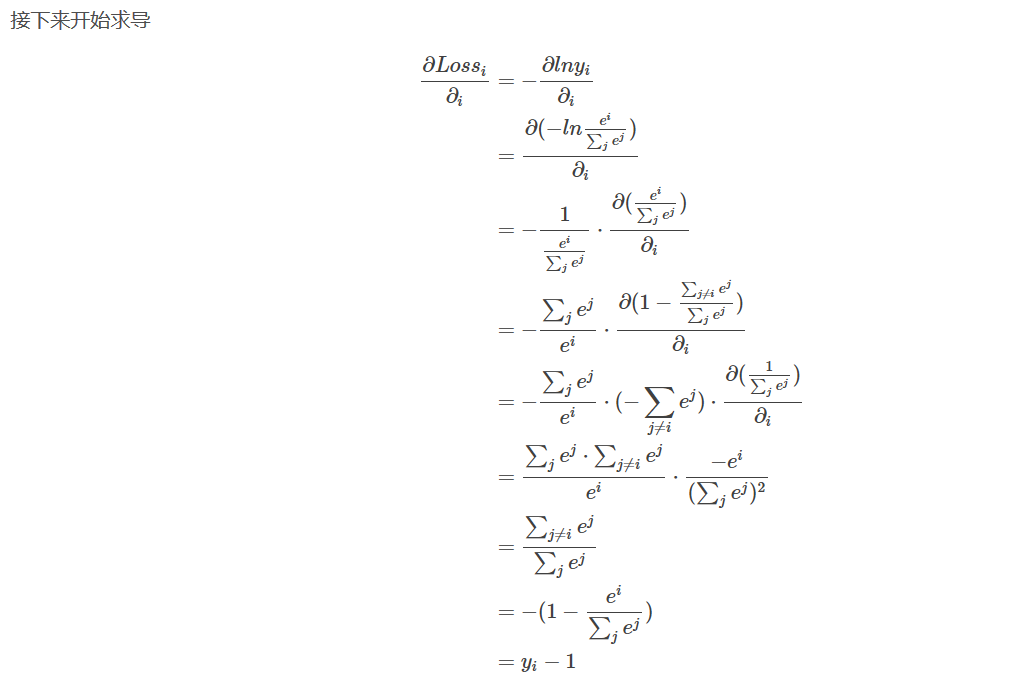


主要就是两步，第一步通过指数函数转换，第二步归一化



**softmax求导**





上面的结果表示，我们只需要正想求出*yi*，将结果减1就是反向更新的梯度，导数的计算是不是非常简单！