王小草【深度学习】笔记第一弹--神经网络

发表于2016/8/16 12:33:48 1138人阅读

分类： 王小草深度学习笔记

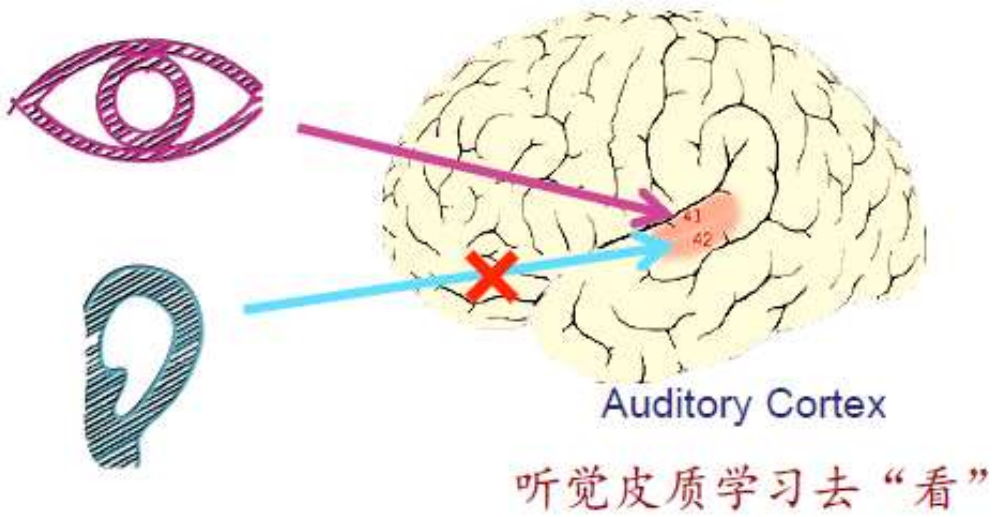
王小草深度学习笔记

课程来自：寒小阳
笔记整理者：王小草
时间：2016/08/15
欢迎交流：qq:1057042131

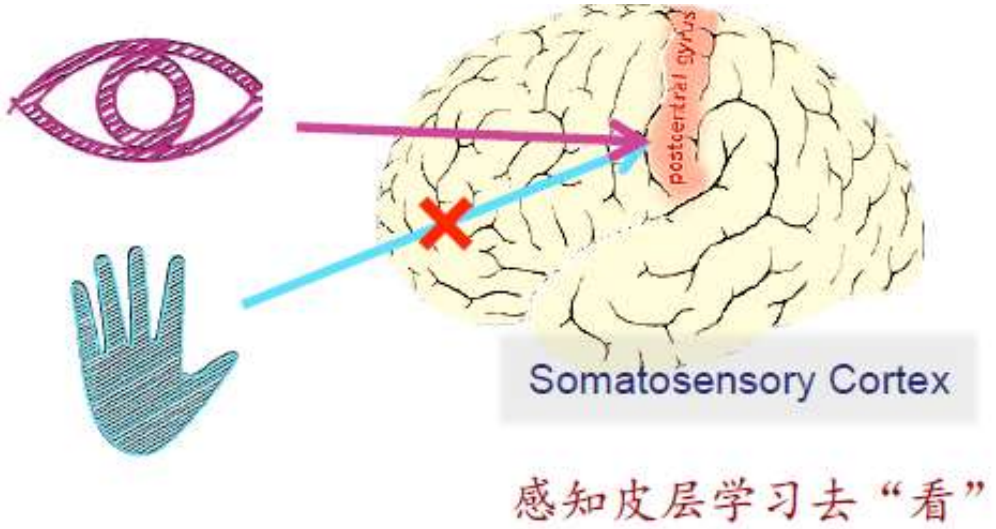
1. 神经网络前言

1.1 背景

在进入神经网络之前，先讲述两个略带血腥的实验。
第一个实验是科学家将耳朵到大脑听觉区的神经给切断了，然后将眼睛到大脑听觉区的神经接起来，之后发现大脑听觉皮质也会慢慢去学习视觉传入的信息，即学会去“看”。

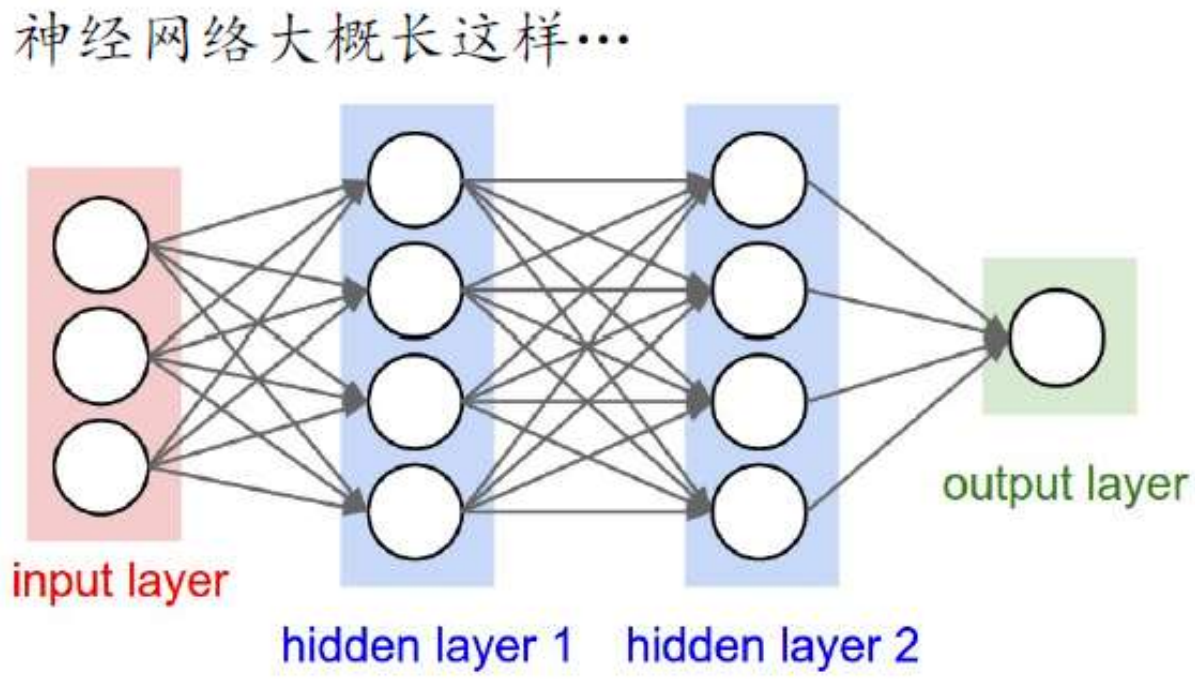


另一个实验也雷同，将触觉神经割断，然后将眼睛与感知皮层连上神经，发现感知皮层也能学会视觉神经传入的信息，也学会了“看”。



1.2 神经网的基本结构

神经网络的基本结构如下，由三部分组成，最左边的是输入层，最右边的是输出层，中间是隐藏层



1.2.1 从逻辑回归到神经元(感知机)

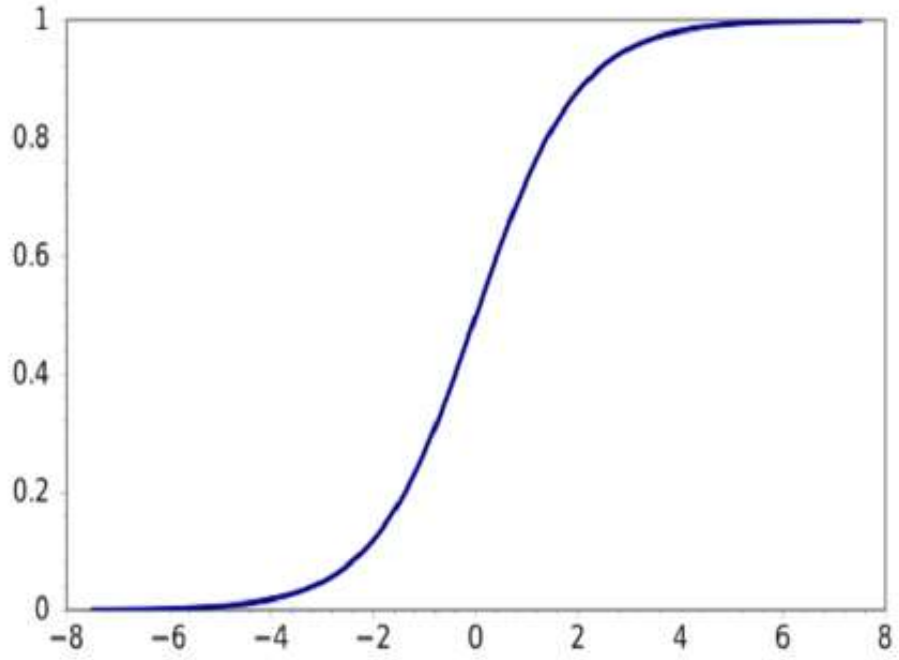
我们回忆一下逻辑回归的原理
首先，将特征输入到一个一元或多元的线性函数z中。

$$z = \theta_0 + \theta_1 X_1 + \theta_2 X_2$$

然后将这个线性函数z作为一个输入，输入到函数g(z)中

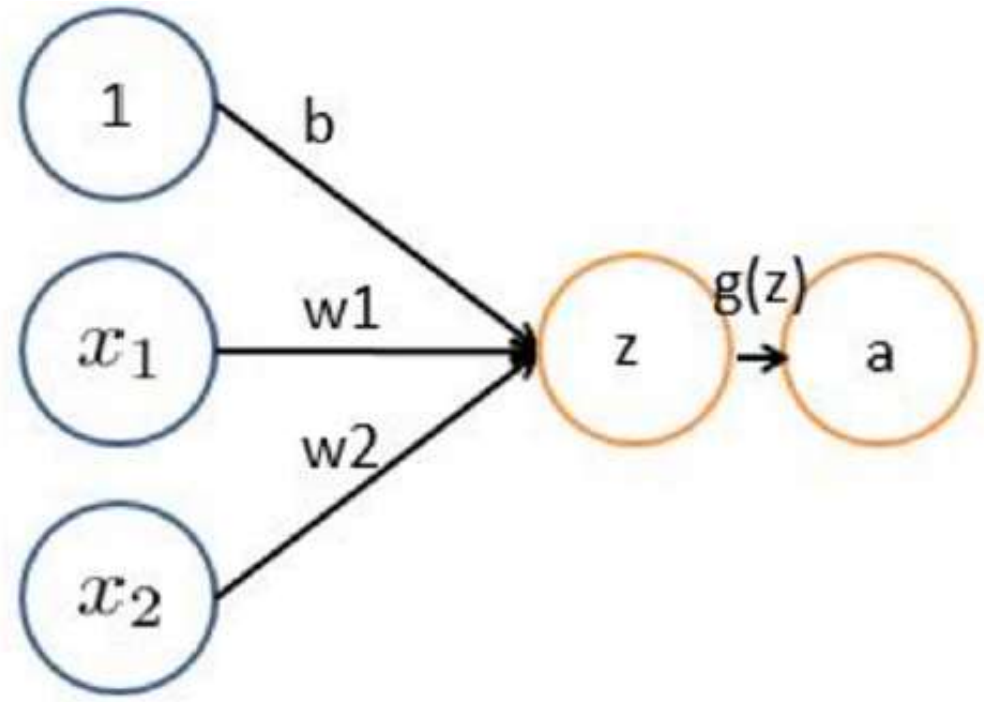
$$a = g(z) = \frac{1}{1+e^{-z}}$$

从g(z)公式可以看到，当z越大的时候，g(z)会越来越向上接近1，当z越小的时候，分母就趋于无限大，g(z)就越来越向下趋近0，当z为0时，g(z)等于0.5。我们将g(z)函数画在坐标轴上，就是如下形状的。



根据图形，我们可以更清晰地看到，当z小于0，则g(z)小于0.5，当z大于0，则g(z)大于0.5。因此它可以作为一个二元的分类器，大于0.5的分为正类，小于0.5的分为负类。

之所以在这里提及逻辑回归，因为逻辑回归就可以当成神经元中的感知器。下图中，最左侧就是输入z函数的变量（或者叫特征或因子），1为常量，x1,x2为两个特征。从这三个变量到z有三条边，分别是权重，也就是逻辑回归z函数里的系数，将变量与对应的系数相乘并线性相加的过程就是线性函数z的求解过程，通过这一步，我们求出了z。从中间的小圆z到右边的小圆a，就是逻辑回归的第二步了，即将z作为输入变量代入g(z)中，求解出g(z)或a。

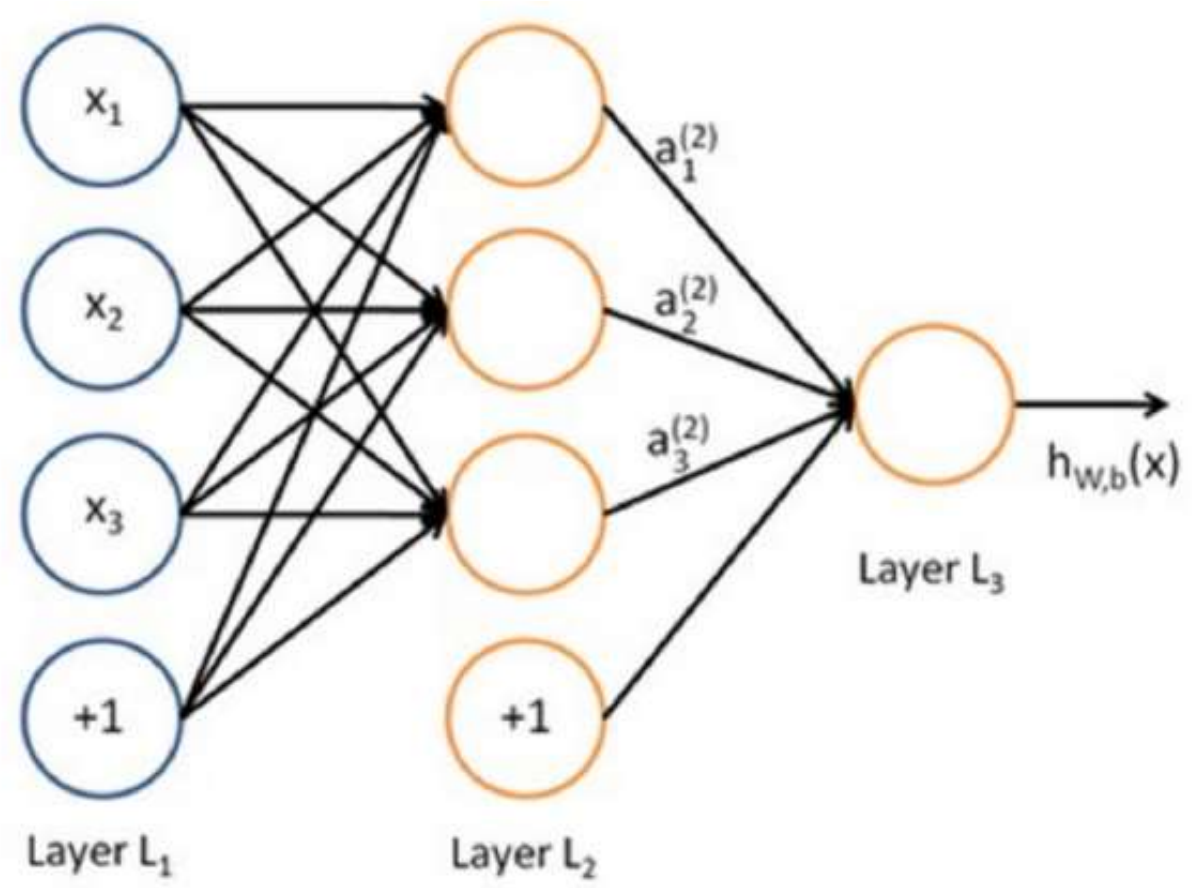


以上整个过程，就是神经元的感知器的原理。

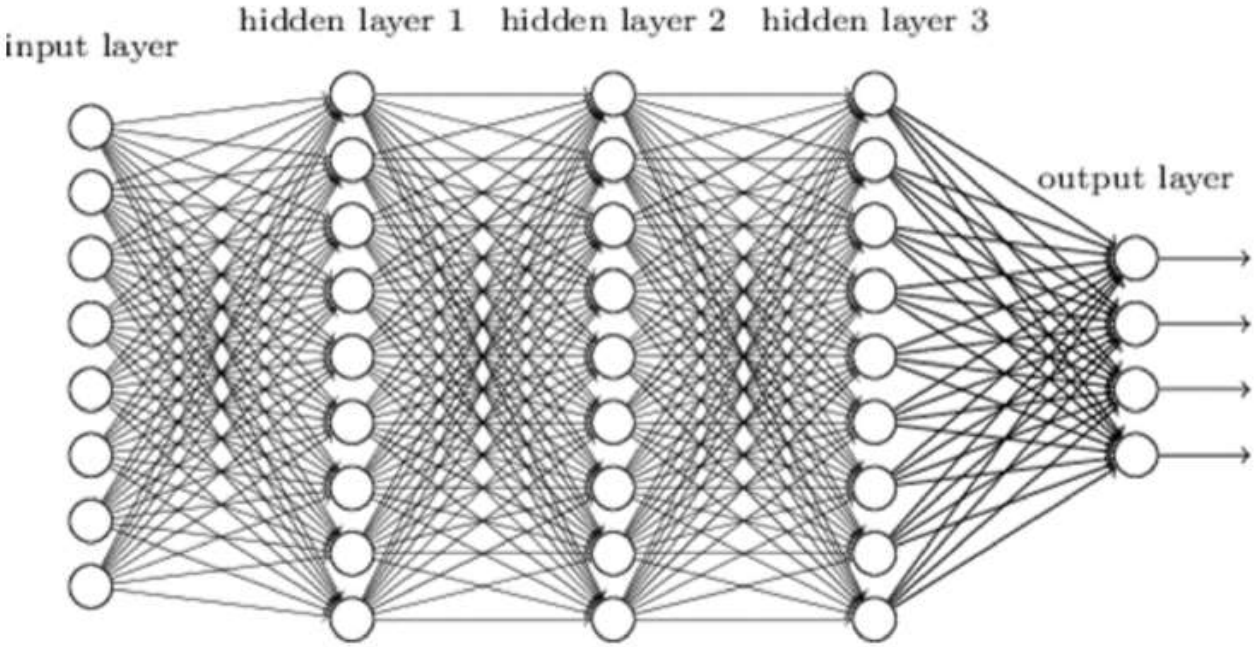
1.2.2 浅层神经网络与深层神经网

现在我们添加少量的隐藏层，简单的感知器就变成了一个浅层神经网了（SNN），从上一层到下一层就是

一个感知器



如果添加更多的隐藏层，就形成了一个深层神经网了（DNN），包含了多个感知器。

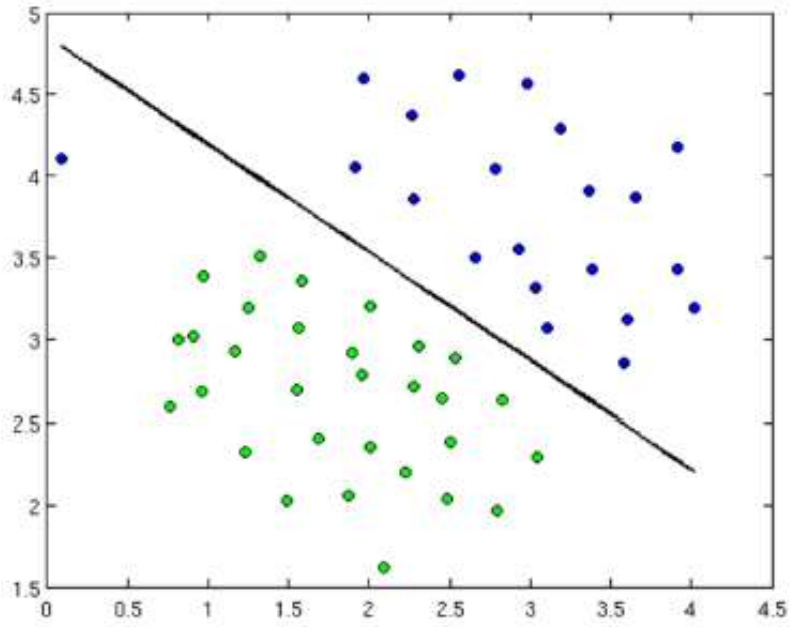


2. 神经网的能力与原理

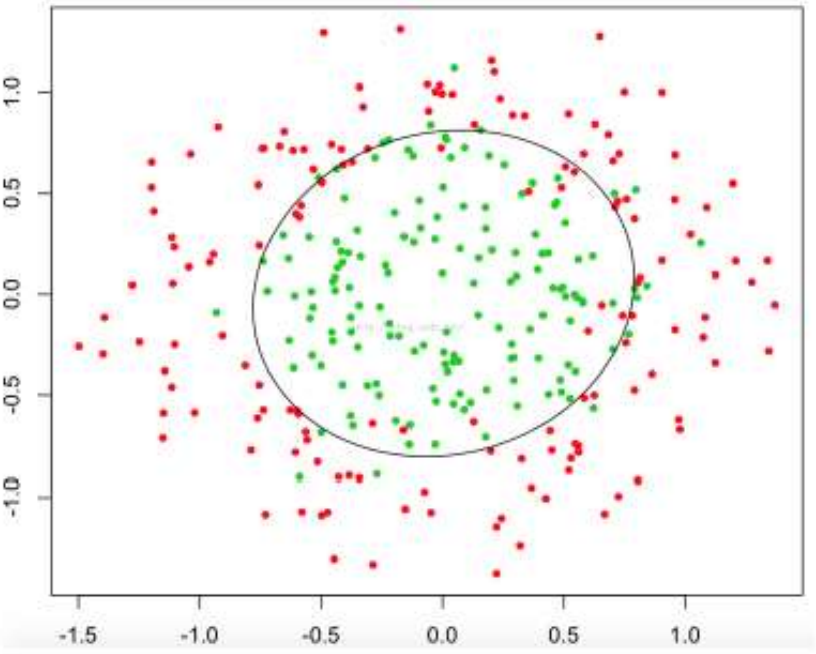
2.1 非线性的切分能力

逻辑回归（LR），支持向量机（SVM）是最常见的线性分类器，比如分布在二维坐标轴上的点，LR与SVM

试图去寻找一条直线，可以将不同类型的点分在线的两边。



但是在实际需求中，点的分布往往不是可以用直线完美分割的，比如说像下图，需要用一个圈来分割圈内与圈外的点。

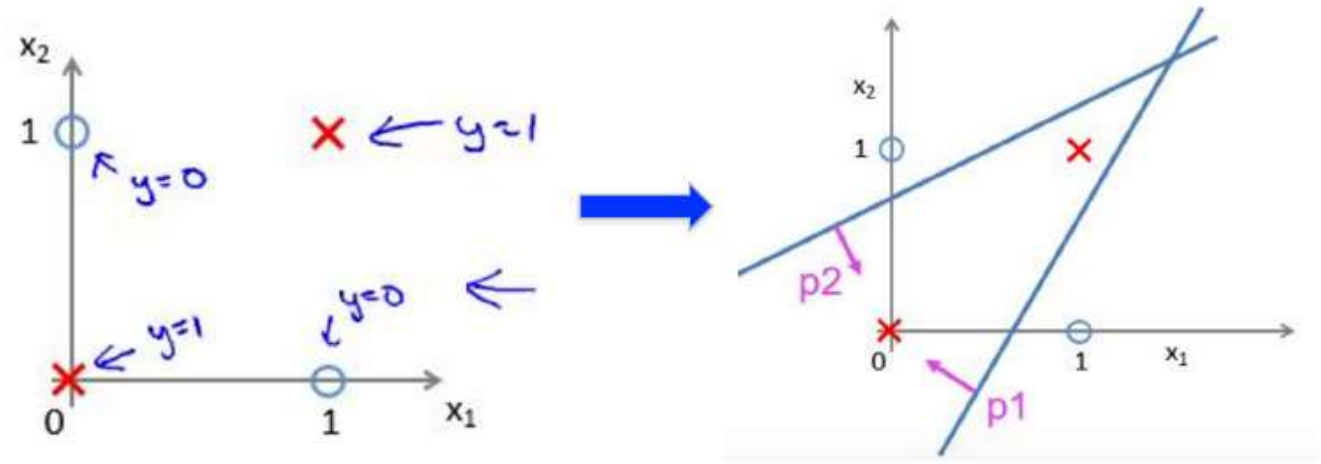


当然，LR,SVM这些线性分类器也可以转变成非线性分类器，主要有两种方式，第一，增加更多的变量（X3,X4,X5…）；第二，将现有的变量（X1,X2）转换成（X1^2,X2^2)的平方形式，就可以去预测分类上图的分布数据。

但是的但是！这样的做法非常的复杂与不实际，因为对于一堆未知的样本数据，我们不知道到底是用平方，还是三次方，还是n次方去转换原变量才得到最符合的非线性模型。许多时候对于模型的选取，也只能通过肉眼探索数据点的分布图，然后一个一个去尝试，如此，既耗时又费力又低效。

于是的于是！就轮到神经网络的闪亮登场了！

先请看左下图，在二维的坐标轴上有四个点，圆和叉分别是两个类别。想用一条直线来线性地区分这两个类，无论如何调整斜率和结局都是是如何都做不到的。
然后请看右下图，如果我们使用两条直线就能有效地进行分类了，位于p1之上，并且p2之下的点为一类，位于p1之下或者p2之上的点为一类。这样多条线去做分类可以用神经网络来实现。



2.2 神经网的逻辑门

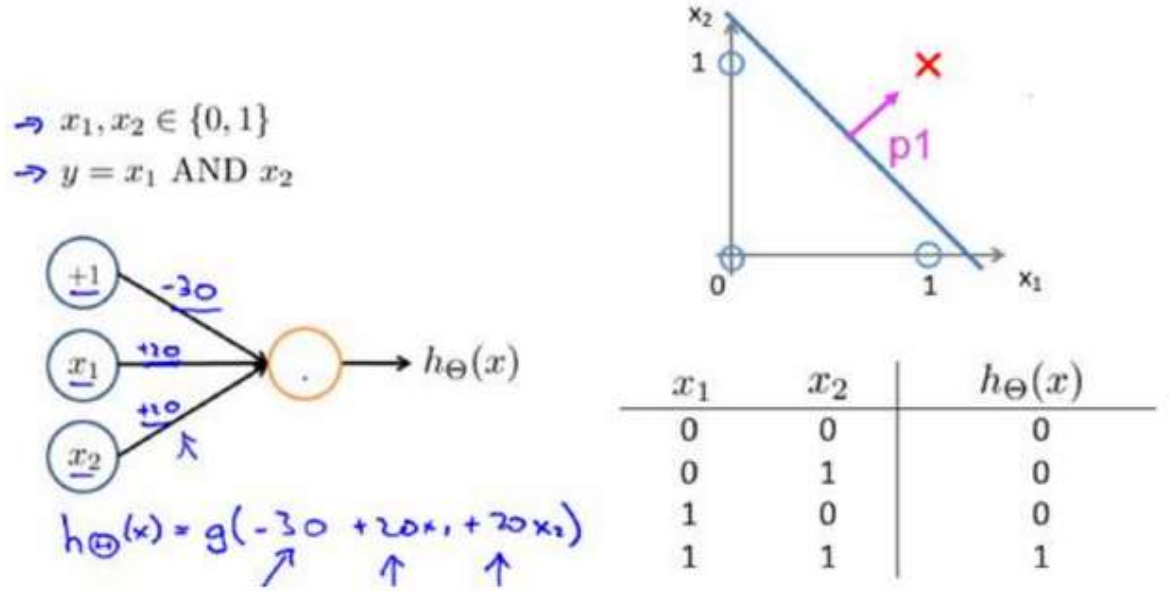
2.2.1 神经元完成“逻辑与”

假设我们现在有两个输入的变量x1,x2,这两个变量的取值是0或者1。
另外，输入常量仍然是1。

如下图左边所画，假设每条边的权重（系数）分别是-30，20，20。（权重的确定后续会讲，这里先假设我们已经知道了权重）

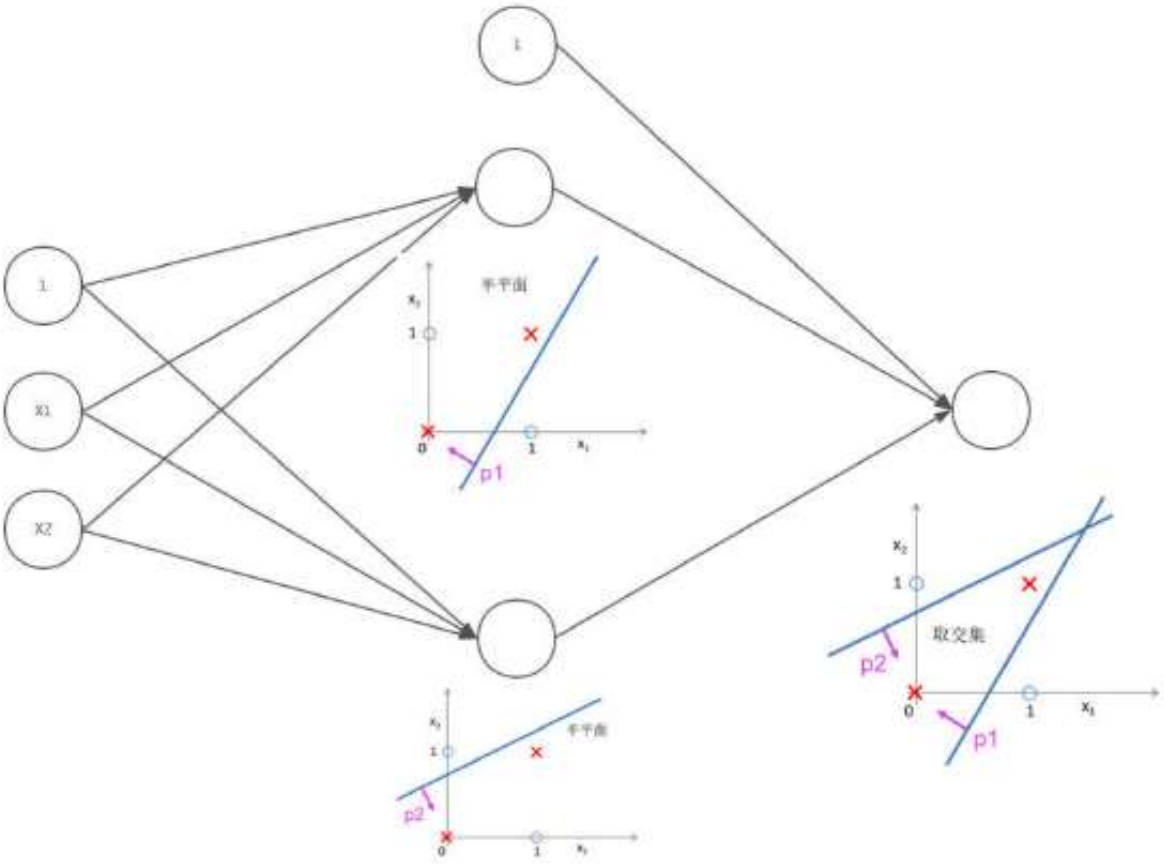
根据上文所描述的感知器，我们将输入数据做一个逻辑回归计算，然后输出结果h(x)为分类的所属类别（1或0）
根据公式h(x) = g(-30+20*x1+20*x2)，我们去求x1,x2两两配对的所有可能的值。
当x1=0,x2=0时，z=-30,回忆一下逻辑回归函数g(z)的图，当z < 0时，g(z)<0.5, 故分到0类
同理，我们把四种可能的配对都求一遍，结果如下图中有下方的表格所示，只有x1=1 并且x2=1 时，才满足被分到1类中。

这就实现了逻辑“与”，只有当输入的都是正类的时候，最后输出的才是正类，否则为负类。



x_1	x_2	$h_\Theta(x)$
0	0	0
0	1	0
1	0	0
1	1	1

那么现在我们再继续把2.1中的问题放进来，假设叉叉为正类1，要正确的分出正类，必须要满足两个线性分类器（p1,p2)都分到了正类（p1下方为正类，p2上方为正类）,这是便是一个“逻辑与”问题了。先来看看下图：



图中最左边的一层为输入层，即待分类的点的两个特征（二维坐标的横纵坐标）与常数1.

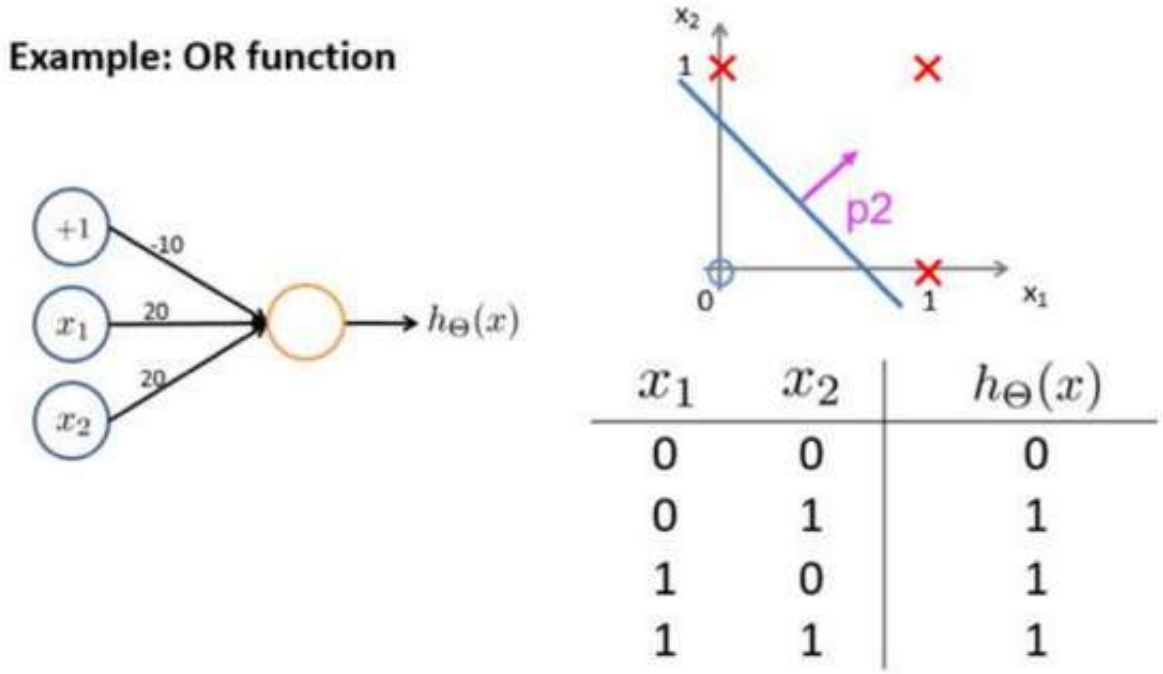
由输入层的点伸出向上的三条边到中间层的某个点（边的权重我们也先假设好），这个过程是一个感知器，也即一个p1线性分类器,中间层的这个点会有一个分类的输出（1或0）。
同理输入层又向下指向了中间层的另一个点，这有是另一个线性分类器p2,也会计算一个分类结果（1或0）。

再看由中间层的两个点一起指向最右边的输出层，这是另一个感知器，中间层其实就是这个感知器的输入了，上文中已经解释了，只有当中间层的两个点都为1的时候（满足“逻辑与”），最后的输出层才会是1。否则就为0类。

有没有发现，经过以上过程，我们已经完美地实现了用两个分类器去实现非线性分类！圆和叉现在已经有效地被区分了！

2.2.2 神经元完成“逻辑或”

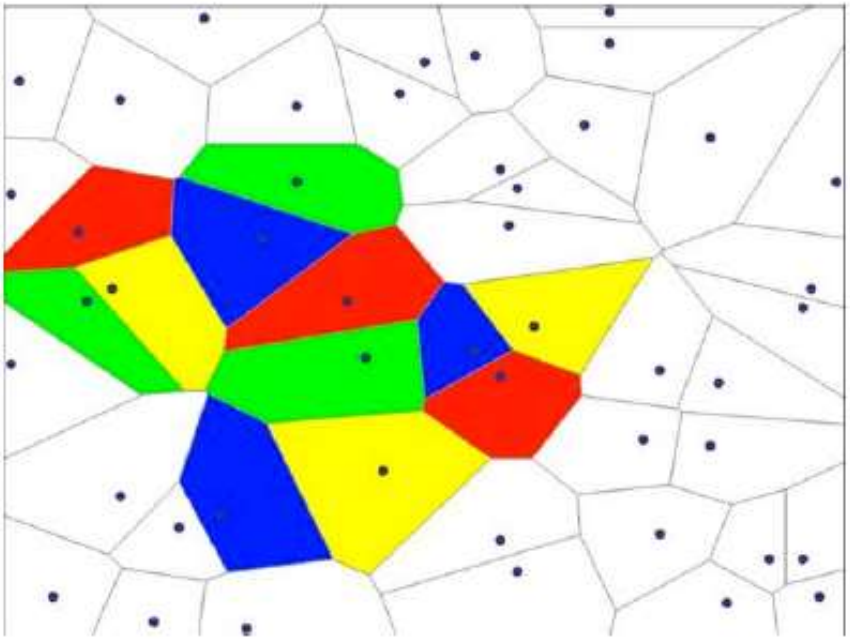
同样的原理，我们来看逻辑或，假设系数分别为-10,20,20,右下表可见，只要满足其中一个输入为1，那么最终类别也为1。



x_1	x_2	$h_\Theta(x)$
0	0	0
0	1	1
1	0	1
1	1	1

2.2.3 线性分类器“与”和“或”的组合

请看下图，假设我们要区分出不同颜色的类别，比如绿色的类别分别散落在三个不想连的区域，那么使用传统的svm,LR都将纷繁复杂或是终不可实现，而使用神经网的逻辑“或”和“与”就可以轻易实现有效分类。



第一步：假设我们现在要先找出最上方那块绿色的区域，它有7条边构成，就可以看成7个线性分类器，只有同时满足7个分类器都为1，最后的输出才为1.
同理，左边的绿色区域由4条边组成，对应了4个线性分类器，同时满足4个类别都为1，最终的输出才为1.
有下方的绿色区域也同理。
也就是说，为了找出分散的三个区域，我们做了3次”逻辑与”

第二步：只要这个点落在任一一个绿色区域中，他的类别就是1，这就是“逻辑或”，只要满足其中一个条件就可以。

总结：于是我们发现，要分出绿色区域，其实就是先“逻辑与”再“逻辑或”的过程！

2.3 网络表达力与过拟合

来看一看神经网的几个主要结构。

首先是无隐层，只有输入层和输出层，可用于将一个超平面分成两个。

第二是单隐层，即只有一个隐藏层，适用于分出开凸区域和闭凸区域，下表中第2行第4列的图中，A坐在的区域就是一个开凸区域，B所在的就是闭凸区域

第三是双隐层，由两个隐藏层，可以分类出任意形状。

结构	决策区域类型	区域形状	异或问题
无隐层	由一超平面分成两个		
单隐层	开凸区域或闭凸区域		
双隐层	任意形状（其复杂度由单元数目确定）		

但是！是不是双隐层一定比单隐层更加好呢？答案肯定是否。机器学习中我们已经对这样的问题不陌生了，对训练样本越复杂精确的模型往往会存在过拟合的问题，在这里也一样！

对于已知的A和B四个点，上图中的单隐层和双隐层都已经正确地分出了两个类别。但是如果忽然又来了一个新的点A落在了原有的两个A之间，这样单隐层还是分类正确，但是双隐层却错误了。这就是神经网的过拟合问题。

关于神经网的层数与效果，有如下几点经验：

- 理论上说单隐层神经网络可以逼近任何连续函数（只要隐层的神经元个数足够多）。
- 虽然从数学上看表达能力一致，但是多隐藏层的神经网络比单隐藏层的神经网络工程效果好很多。
- 对于一些分类数据（比如CTR预估里），3层神经网络效果优于2层神经网络，但是如果把层数再不断增加(4,5,6层)，对最后结果的帮助就没有那么大的跳变了。
- 图像数据比较特殊，是一种深层(多层次)的结构化数据，深层次的卷积神经网络，能够更充分和准确地把这些层级信息表达出来。

可能乍一看没有那么明白，下面对每一条都做一下解释：

1. 单隐层神经网络可以逼近任何连续函数。连续函数你可以看成是一条不断的线，可以是一条任何弯曲的线，线的两侧是被区分的两个类别。之所以说可以是任一的线，是因为再怎么扭曲的线都是可以用无限小的直线连接而成，既然是直线，那就可以使用线性分类器，所以，一条曲线就可以用许多线性分类器组成，当同时满足线性分类器都为1时，那么整个曲线的神经网就被满足为1了。其实这个就是上文我们详述的“逻辑与”，而“逻辑与”我们只需要使用“单层神经网”。如果在这个隐层中的神经元的个

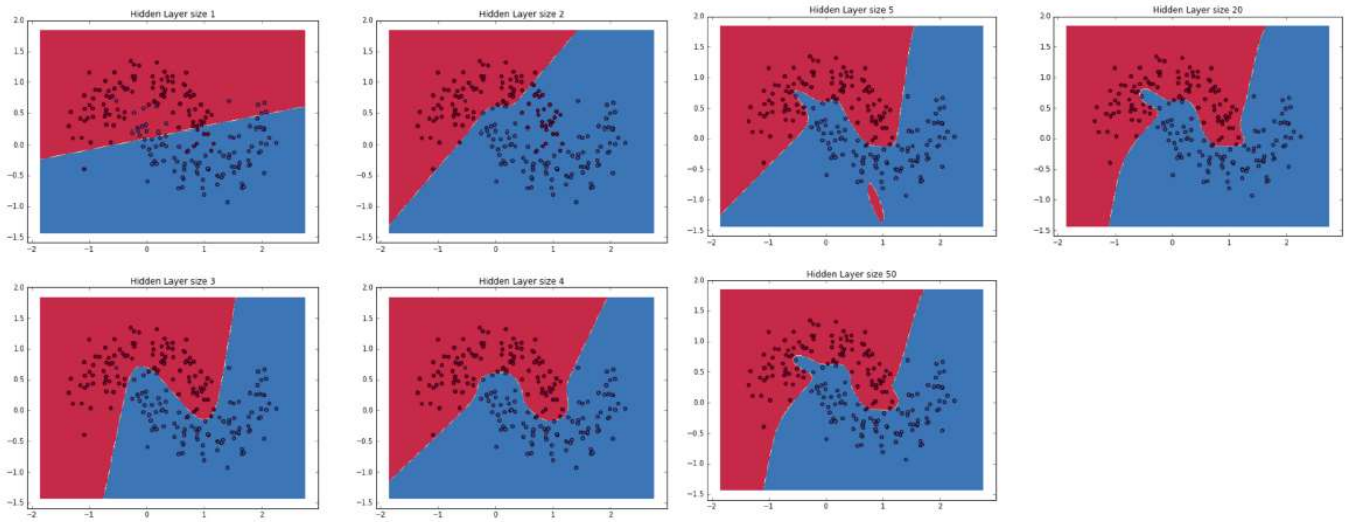
数越多，就越逼近。（一个神经元就是一个线性分类器，分类器越多，就越你和曲线）

2. 多隐藏层比单隐藏层效果好。举个例子，我们使用单隐层，里面有1000个神经元，和我们使用一个多隐层，里面有50个神经元，也许达到的效果差不多，但是工业上我们仍然选择后者会有更好的效率。

3. 对于分类，3层神经网（1个隐层）比2层神经网（无隐层）的效果好，因为前者可以区分非线性。但是！层数再往上增多，则效果的上升就不那么明显了。所以考虑“使用最简单的模型达到最好的效果”这个原则，我们更愿意选择3层神经网。

4. 图像处理之所以需要深层次的神经网，是因为将图像转换成结构化的数据需要有非常多的维度，也就是说需要非常多的特征才能来描述图片。

随着神经元个数的增加，我们来看下神经网的分类效果是如何变化的：



上图中，神经元越多，则空间表达能力越强。但是过多的隐藏层和神经元自然也会带来过拟合的问题。但是的但是，如果要解决过拟合的问题，绝对不能通过降低神经元个数和隐藏层个数这些参数来调节，而是应该使用正则化或者dropout(让一部分神经元休眠），关于这个，后续会详细讲解。

3. 神经网的参数估计 - BP算法与SGD

终于到了重头戏 - 参数估计。

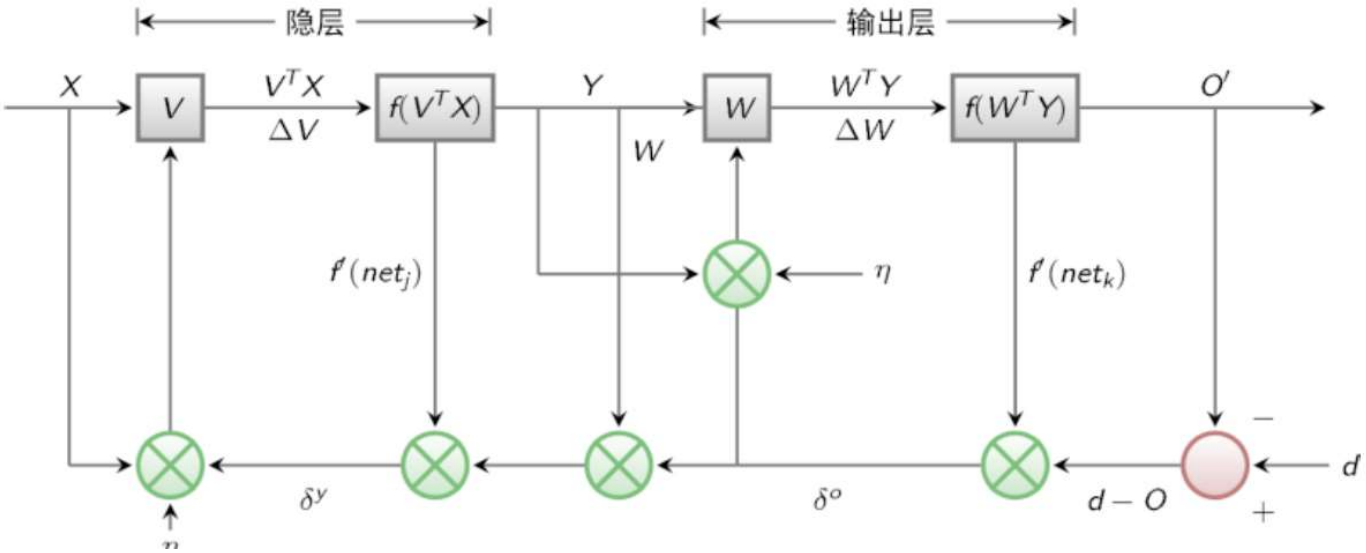
前面我们看到了神经网强大的分类能力和原理。还记不记得，我们在逻辑门的章节我们事先对神经元上的参数做了预估，那么这个参数是怎么来的呢？参数到底设为多少，才能实现我们的分类目的和效果呢？这里我们使用BP算法去求解最优的参数。

3.1 BP算法

BP算法叫“前向计算”,它的步骤如下：

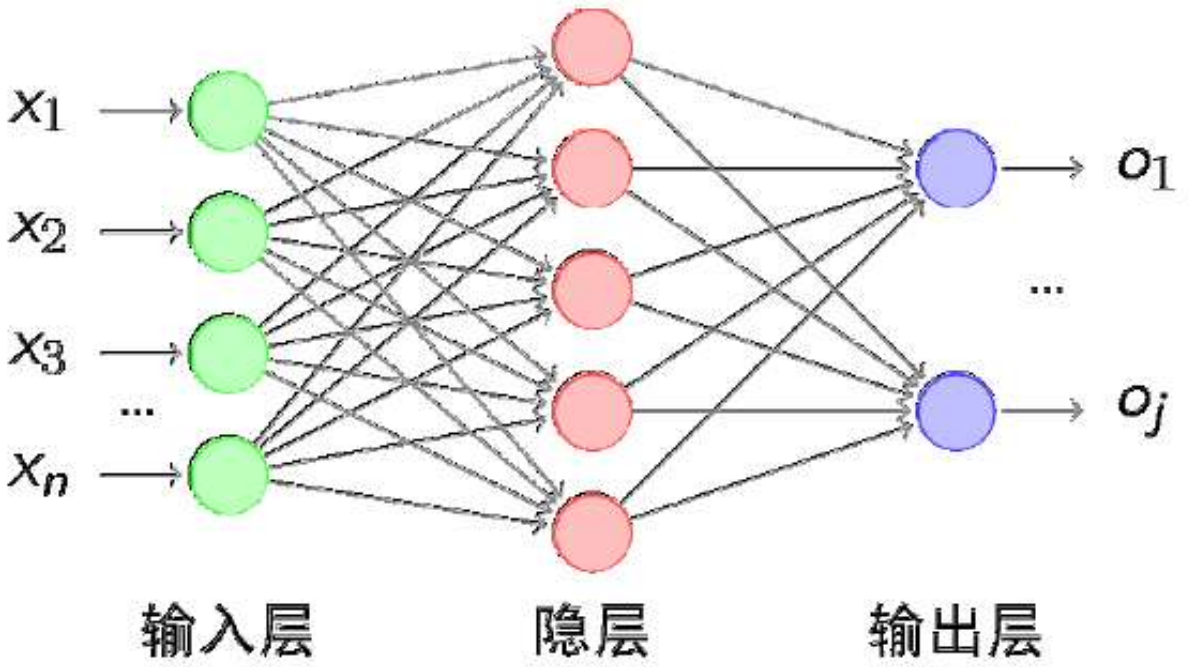
1. 先人为设置参数，然后从输入层开始一层一层根据上面所说的原理计算下去，最后计算出输出层的值。
2. 将实际的值与输出层的值做对比，求出损失
3. 以损失最小为目标，“反向传播”损失，得到最优的参数。

总体过程如下图，V和W分别是两个隐层中的参数，灰色的过程为“正向传播”计算损失，下面绿色的过程为“反向传播”回传误差，在这个过程中求得最优的V和W，使得再次正向传播的时候损失达到了最小。



我们来看一看具体的推导和公式。

假设我们有以下这样的神经网络结构（即单隐层）。N个输入节点和N个输出节点。



1.通过计算我们会得到输出层的结果，将输出的结果与实际的结果做比较可以得到误差或损失。输出层的误差我们可以用以下公式表示，dk为实际值，Ok为输出值，两者相减为损失，因为有多个输入与输出，所有用Σ求和所有的损失。

$$E = \frac{1}{2}(d-O)^2 \qquad = \frac{1}{2}\sum_{k=1}^{\ell}(d_k - o_k)^2$$

2.从输出层倒推回隐层，Ok是通过前一级的函数f(x)得到的（就是之前的g(z)),假设z的值为netk,那么Ok = f(netk)。而g(z)中的z其实是通过一个输入*权重的线性函数得到的，即netk = w*y。故形成了以下公式。

$$E=\frac{1}{2}\sum_{\kappa=1}^{\ell}[d_\kappa-f(net_\kappa)]^2 \qquad =\frac{1}{2}\sum_{\kappa=1}^{\ell}[d_\kappa-f(\sum_{j=0}^{m}\omega_{j\kappa}y_j)]^2$$

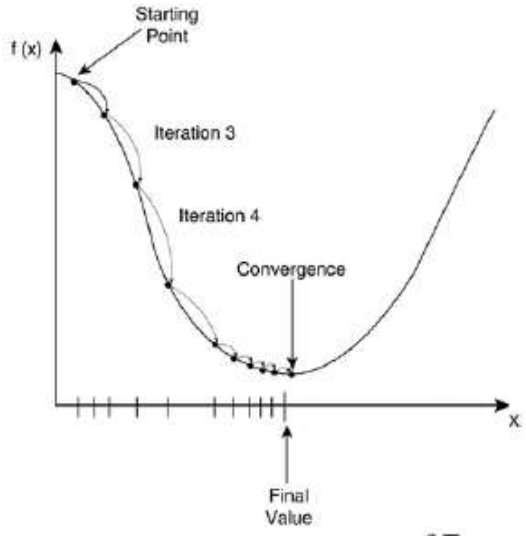
3.上面公式中的y是隐藏层的值，这个值其实是由前面的输入层*权重的线性函数，再经过一个g(x)的变换得到的，所以我们将替换掉，最终形式如下：

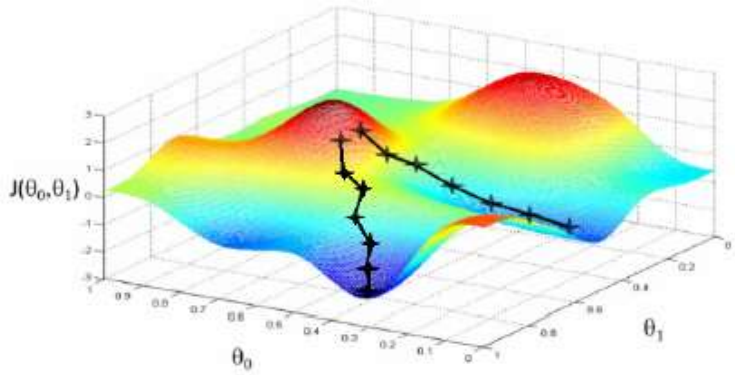
$$E=\frac{1}{2}\sum_{\kappa=1}^{\ell}d_\kappa-f[\sum_{j=0}^{m}\omega_{j\kappa}f(net_j)]^2 \qquad =\frac{1}{2}\sum_{\kappa=1}^{\ell}d_\kappa-f[\sum_{j=0}^{m}\omega_{j\kappa}f(\sum_{j=0}^{n}v_{ij}\chi_i)]^2$$

上面这个公式，左边是损失，右边的输入值x和实际值dk是已知的，未知的只有W和V这两个参数。所以！是一个损失函数，我们要去求得当损失最小的时候，参数为多少，即求函数的最小值！

3.2 随机梯度下降SGD

如果是一个凸函数，我们可以用拿所有数据使用梯度下降法去求解函数最小值（如下左图）。但是在神经网络中，远远没有凸函数那么简单。我们遇到的很有可能是像下右图中的情景，它有很多局部的最小值，在用梯度下降法的时候很有可能会一不小心把局部最优当成全局最优了。



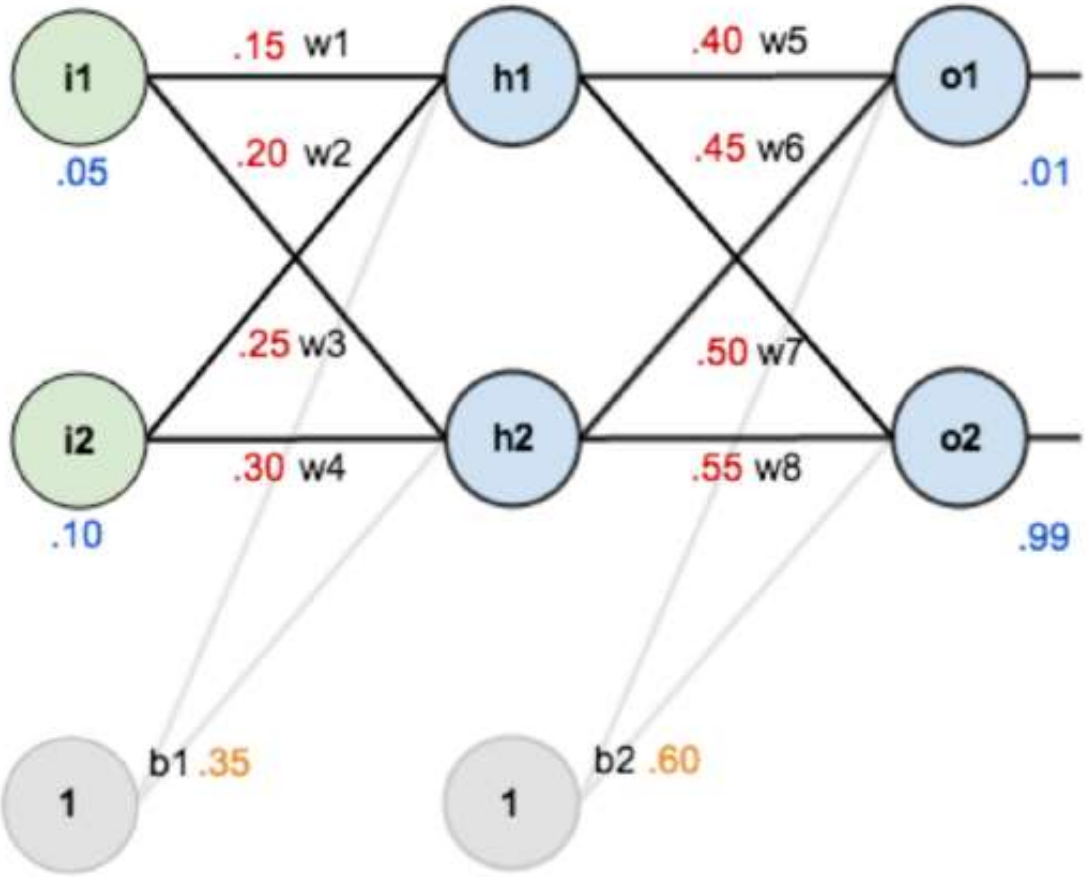


为了尽量避免以上问题，BP算法中使用随机梯度下降法（SGD)求损失函数的最小值。SGD会随机从数据中抽样出一部分的样本，然后再去做梯度下降。要注意的是这并不能完全保找到了全局最优的点，而是增大了找到全局最优的概率。
（梯度下降法和随机梯度下降法会另起章节详述）

3.3 案例

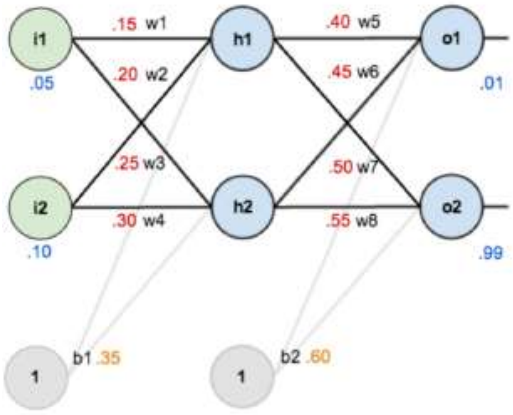
至此，我们已经学习了神经网络的整个推理过程以及参数的计算，下面看一个案例，一起来走一遍。

下面给出了一个3层的神经网络，输入是2个变量（0.5,0.1）+1个常量，输出是2个变量，中间有有个隐藏层（2个神经元）。第一次输出的结果已经给出是0.1,0.99。初始的权重（参数）已经给出。接下来要做的是根据随时函数，求解误差最小时的参数值。



我将整个计算过程的截图贴上，有兴趣的可以自己跟着算一遍：

前向运算：



$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

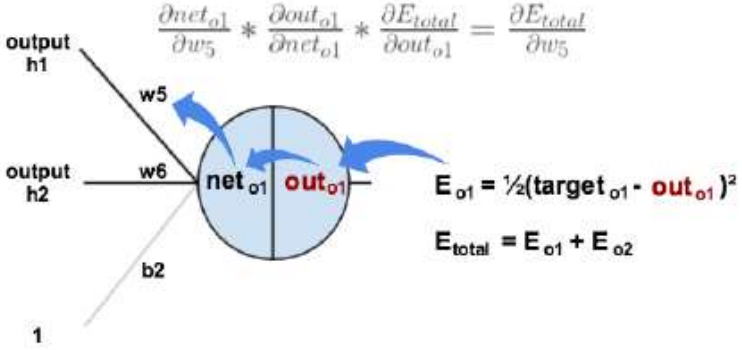
$$out_{o2} = 0.772928465$$

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

反向传播：

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$