在学习神经网路之前，我们先来了解一下神经元是如何工作的：

生物上，神经元有着多样的形式，但是所有的神经元都是将电信号从一端传输到另一端，沿着轴突，将电信号从树突传到树突。继而，这些信号从一个神经元传到另外一个神经元。

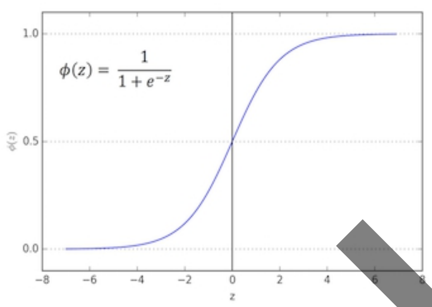
现在我们已经了解了，神经元是如何去工作的---->它接收了一个电输入，输出为另外一个电信号。映射到机器学习上就是，接收一个输入--->中间处理--->弹出一个输出。

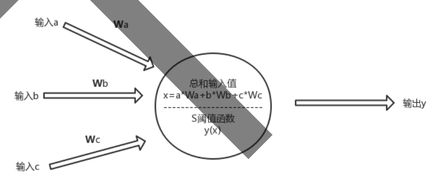
同样的，在生物上，神经元不会立即反应，而是会抑制输出，直到输入增强，强大到可以触发输出。我们可以认为在产生输出之前，输入必须达到一个阈值。就比如往水杯里面倒水---水满了就会溢出。换而言之，我们可以想象一下，用打火机去烧手指，如果你的火在手指下停留的时间短，你几乎不会有疼痛，但火稍微久一点，你就会感到被烧到了，

这就相当于是达到了阈值，你的神经元将输入的信号转换为疼痛感了。

了解了阈值这个概念以后，我们就需要定义神经网络里面阈值的激活函数了，
在数学上，有许多激活函数可以达到这样的效果，如简单的阶跃函数，以及 S 函数（sigmoid function)--->y=1/1+e^-x

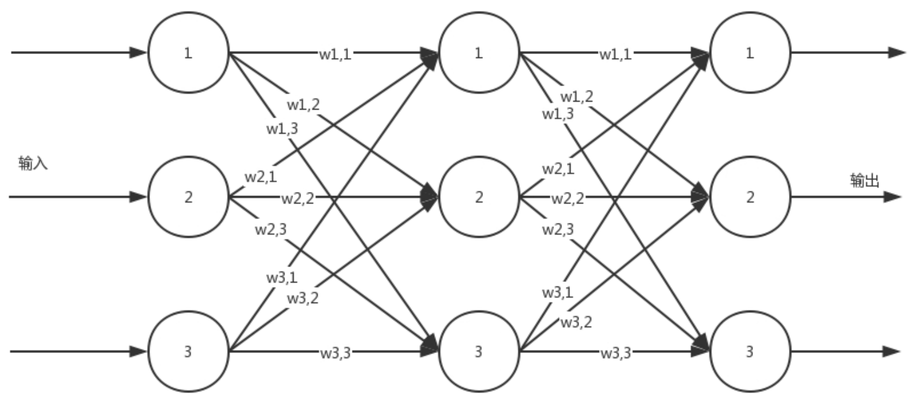
如图我们可以这样理解，当输入较小的时候，输出为 0，一旦输入达到阈值，输出就一跃而起。就如神经元被激发了。





现在我们知道了一个神经元的输入输出了。

但是，实际中，每个神经元接受来自其之前多个神经元的输入，并当神经元被激发了，它也同时提供信号给更多的神经元。将这种理解映射到神经网络上，也就是将这种自然形态映射到我们人造的模型上：就是构建多层神经元，每一层中的神经元都在与其前后层的神经元互相连接.

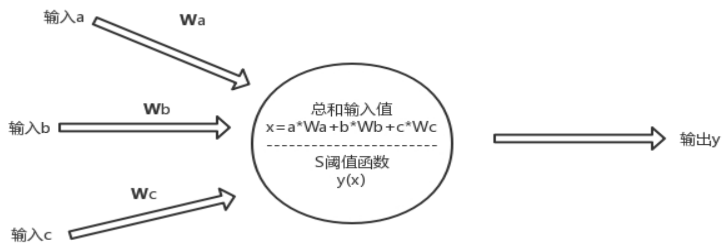


三层神经网络-自制

这里，我们需要解释 w 是什么？

w 就是我们常说的权重，较大的权重将放大信号，较小的权重将弱化信号。

加上权重的神经网络图：



这是我思考的一个问题是：对于这种简单的模型我们可以手写计算，但是对于更复杂的模型我们应该怎么办呢？

这就可以联系到我们学过的线代的知识，利用矩阵计算，而这也是更容易让计算机去表达的一种计算形式。

如：你有三个输入记为 in1, in2, in3, 你的权重就会有 3*3=9 个。

要算出你的总输入值：

$$X=\begin{pmatrix} w1,1 & w2,1 & w3,1 \\ w1,2 & w2,2 & w3,2 \\ w1,3 & w2,3 & w3,3 \end{pmatrix} * \begin{pmatrix} in1 \\ in2 \\ in3 \end{pmatrix}$$

联系上面的三层神经网络图这是很清晰的，利用这种形式，我们将这模型推广可以写为 $X=W^T * \mathrm{I}$,W 为权重矩阵，I 为输入矩阵

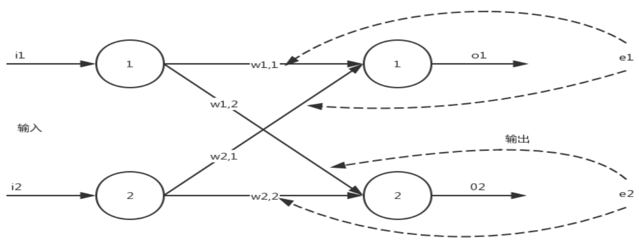
最后输出则为：$\mathrm{O}=\mathrm{sigmoid}(\mathrm{X})$,O 为输出矩阵。

上面我们介绍了输入和输出两层。但隐藏在输入输出中还有若干的隐藏层我们也称其为中间层。

如定义:$\mathrm{I}=\begin{pmatrix} 0.5 \\ 0.6 \\ 0.1 \end{pmatrix}$,$W_{input\_hidden}^T=\begin{pmatrix} 0.1 & 0.7 & 0.8 \\ 0.3 & 0.6 & 0.9 \\ 0.6 & 0.5 & 0.1 \end{pmatrix}$,$W_{in_hidden}^T$为输入层与隐藏层之间的权重矩阵，,$X_{hidden}=W_{in\_hidden}^T * \mathrm{I}$,$X_{hidden}$就是我们所求的中间层。$O_{hidden}=sigmoid(X_{hidden})$,$O_{hidden}$为中间层的输出。

假设只有一层中间层则，$X_{output}=W_{output\_hidden}^T * O_{hidden}$

看到这里，似乎这一切都完美的结束了。其实不然，我们还需要考虑学习来自多个节点的权重及更新权重等问题。



利用反向传播误差的方法，观察上图：

将第一个输出节点的误差记为 e1，这个值由训练数据的所期望的输出值 t1 与实际输出值 o1 之间的差，即 e1=t1-o1

从图中，按照所连链接的比例，也就是权重 $w_{1,1}$ 与 $w_{2,1}$，对误差 e1 进行了分割，即使用 e1 的一部分更新 $w_{1,1}$ 为$\frac{w1,1}{w1,1+w2,1}$，另外一部分更新 $w_{2,1}$ 为 $\frac{w2,1}{w1,1+w2,1}$

当有多个层的时候，我们需要考虑隐藏层。

$$error_{hidden} = W^{T}_{output\_hidden} {}^{*} error_{output}$$

如何更新权重？这是我们需要思考的一个问题？

不想了，直接肝。

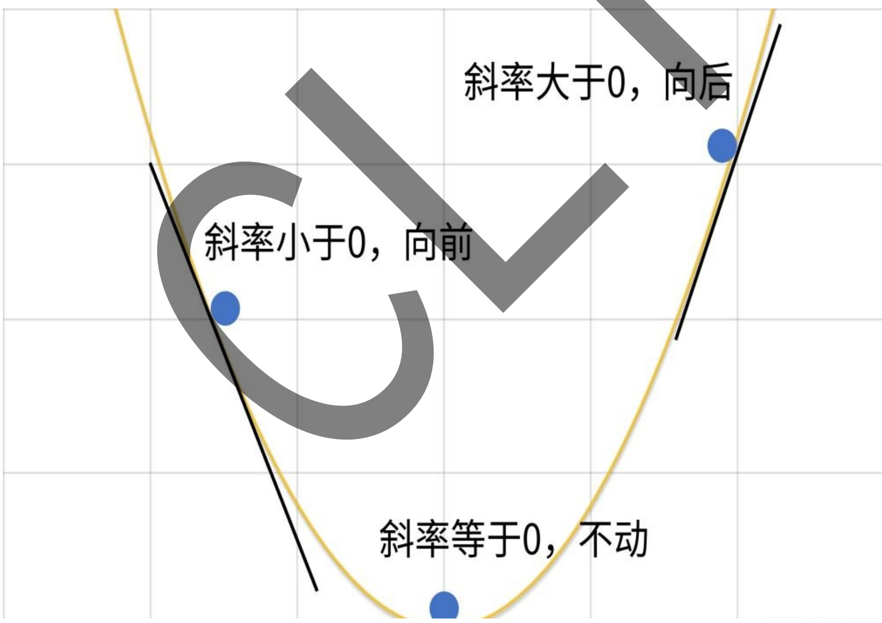
对于反向传播就是为了最小化 loss 求梯度的过程了。

梯度概念这里不做解释了。

假如有一个函数，z=f(x, y)，那么 z 的梯度为：

∂z/∂x i + ∂z/∂y j

z 函数梯度的方向是 z 增加最快的方向。我们在深度学习里，需要降低 loss，因此我们是选择 loss 函数梯度的反方向



推荐一篇经典论文供大家研读：

[1] Hochreiter S , Schmidhuber J . Long Short-Term Memory[J]. Neural Computation, 1997, 9(8):1735-1780.

CLT

下面我们具体讲一些神经网络的模型：

一、感知器神经网络(Perceptron neural network)

其算法框架为：

Perceptron Algorithm

1. 随机取 w,b
2. 取一个训练样本（x,y）

    （1）若w^T+b>0 且 y=-1 则：

    w=w-x,b=b-1

    （2）若w^T+b<0 且 y=1 则：

    w=w+x,b=b+1

3. 再取另一条件(x,y) 回到 2
4. 终止条件，直到所有输入输出对都不满足 2 中（1）、（2）之一，退出循环

可以想一下：为什么w^Tx+b>0，且 y=-1 则：w=w-x,b=b-1

意思是当w^Tx+b 为一个正数的时候，y 是一个复数，这表明着 w、b 对 x 分类错误。

所以给出 w=w-x,b=b-1,那为什么是 w=w-x,b=b-1？

我给出一个简单的推导：

令$\mathrm{w}_{新} = \mathrm{w}-\mathrm{x}, \mathrm{b}_{新} = \mathrm{b}-1$，带入$w^T x$+b，有$\mathrm{w}_{新}^{\mathrm{T}}\mathrm{x}+\mathrm{b}_{新} = (\mathrm{w}-\mathrm{x})^{\mathrm{T}} * \mathrm{x} + (\mathrm{b}-1)$

化简得到$\mathrm{w}_{新}^{\mathrm{T}}\mathrm{x}+\mathrm{b}_{新} = w^T\mathrm{x}+\mathrm{b}-(||\mathrm{x}||^2+1)$,这样就浅显易懂了。

因为 x 模的平方是一个正数再加上 1 就为一个恒大于 1 的数，结合算法就是迭代一次就向负方向走之上 1 的数，不断迭代将分类错误的拉到正确的这边就结束迭代。

同理（2）+b<0 且 y=1 则：w=w+x,b=b+1 的道理一样。

感知器是由两层神经网络构成，所以感知器模型公式可以写成：

$$\mathrm{y} = \mathrm{f}\left(\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{w}_{\mathrm{i}}\mathrm{x}_{\mathrm{i}} - \theta\right) = \mathrm{f}(\mathrm{w}^{\mathrm{T}}\mathrm{x}-\theta)$$

其中，x 为样本的特征向量，是感知器模型的输入；w, θ 是感知器模型的参数，w 为权重，θ为阈值。假定 f 为阶跃函数，那么感知机模型的公式可进一步表示为：$\mathrm{y} = \mathrm{sgn}(\mathrm{w}^{\mathrm{T}}\mathrm{x}-\theta) = \begin{cases} 1, \mathrm{w}^{\mathrm{T}}\mathrm{x}-\theta \geq 0 \\ 0, \mathrm{w}^{\mathrm{T}}\mathrm{x}-\theta < 0 \end{cases}$，sgn 代表符号函数。

在 n 维超平面中，感知器模型将 n 维空间划分为,$\mathrm{w}^{\mathrm{T}}\mathrm{x}-\theta \geq 0$ 和$\mathrm{w}^{\mathrm{T}}\mathrm{x}-\theta < 0$ 两个子空间，并输出相应的值 1、0。

CLT

二、误差逆传播（BP)算法

适用：多层前馈神经网络、递归神经网络等

BP 算法是什么？（下面具体理论来源于周志华老师的西瓜书内容）

下面我们来看看 BP 算法究竟是什么样. 给定训练集 $D = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \ldots, (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$, $\boldsymbol{x}_i \in \mathbb{R}^d$, $\boldsymbol{y}_i \in \mathbb{R}^l$, 即输入示例由 d 个属性描述, 输出 l 维实值向量. 为便于讨论, 图 5.7 给出了一个拥有 d 个输入神经元、l 个输出神经元、q 个隐层神经元的多层前馈网络结构, 其中输出层第 j 个神经元的阈值用 θ_j 表示, 隐层第 h 个神经元的阈值用 γ_h 表示. 输入层第 i 个神经元与隐层第 h 个神经元之间的连接权为 v_{ih}, 隐层第 h 个神经元与输出层第 j 个神经元之间的连接权为 w_{hj}. 记隐层第 h 个神经元接收到的输入为 $\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$, 输出层第 j 个神经元接收到的输入为 $\beta_j = \sum_{h=1}^{q} w_{hj} b_h$, 其中 b_h 为隐层第 h 个神经元的输出. 假设隐层和输出层神经元都使用图 5.2(b) 中的 Sigmoid 函数.

对训练例 $(\boldsymbol{x}_k, \boldsymbol{y}_k)$, 假定神经网络的输出为 $\hat{\boldsymbol{y}}_k = (\hat{y}_1^k, \hat{y}_2^k, \ldots, \hat{y}_l^k)$, 即

$$\hat{y}_j^k = f(\beta_j - \theta_j) , \tag{5.3}$$

则网络在 $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ 上的均方误差为

$$E_k = \frac{1}{2} \sum_{j=1}^{l} (\hat{y}_j^k - y_j^k)^2 . \tag{5.4}$$

CLT

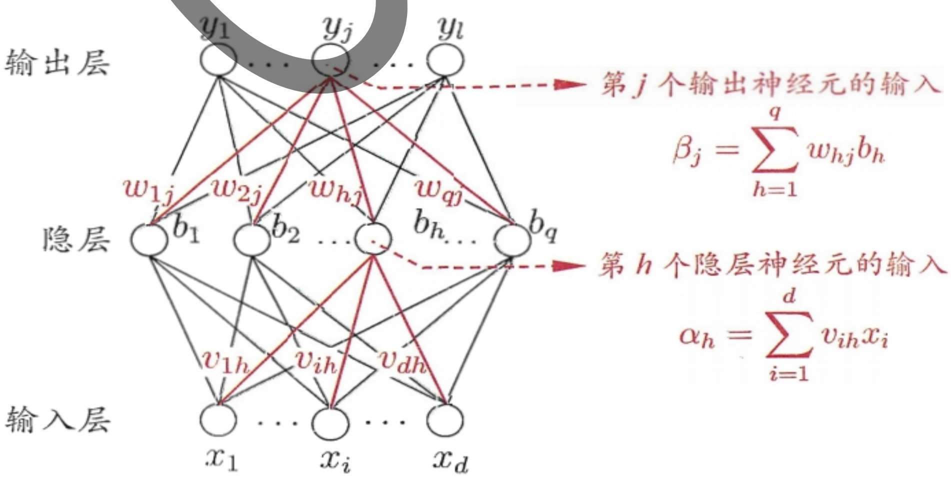


图 5.7 BP 网络及算法中的变量符号

下为对输出层权重、阈值变化量公式的推导：

No. Date.

均方误差：

$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j - y_j)^2 \xrightarrow[\text{对}\hat{y}]{\text{求导}} \boxed{\frac{\partial E}{\partial \hat{y}_j} = \hat{y}_j - y_j}$

第j个输出层神经元的输入：

$\beta_j = \sum_{h=1}^{q} w_{hj} b_h \xrightarrow[\text{求导}]{\text{对}w_{hj}} \frac{\partial \beta_j}{\partial w_{hj}} = b_h$

又BP算法基于(梯度下降)策略，以目标的负梯度方向对参数进行调整，对误差E，给定学习率η，有：

$$\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}}$$

又根据链式求导：其中 $\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}}$

由第j个输出层神经元的输出为：

$\hat{y}_j = f(\beta_j - \theta_j) \xrightarrow[\text{求导}]{\text{对}\beta_j} \frac{\partial \hat{y}_j}{\partial \beta_j} = f'(\beta_j - \theta_j)$

由此处f函数是用的sgn为激活函数：有.

$f'(x) = f(x)(1-f(x))$

$\therefore \frac{\partial \hat{y}_j}{\partial \beta_j} = f'(\beta_j - \theta_j) = f(\beta_j - \theta_j)(1 - f(\beta_j - \theta_j))$

$= \hat{y}_j(1-\hat{y}_j)$

从而 $\boxed{\Delta w_{hj} = -\eta \cdot \hat{y}_j(1-\hat{y}_j) \cdot b_h}$

输出层权数变化量.

输出层阈值变化量：

由 $\frac{\partial E}{\partial \theta_j}=\frac{\partial E}{\partial \hat{y}_j}*\frac{\partial \hat{y}_j}{\partial \theta_j}=\frac{\partial E}{\partial \hat{y}_j}*\frac{\partial[f(\beta_j-\theta_j)]}{\partial \theta_j}$

$=\frac{\partial E}{\partial \hat{y}_j}*f'(\beta_j-\theta_j)*(-1)$

$=\frac{\partial E}{\partial \hat{y}_j}*f(\beta_j-\theta_j)*[1-f(\beta_j-\theta_j)]*(-1)$

$=\frac{\partial E}{\partial \hat{y}_j}*\hat{y}_j(1-\hat{y}_j)*(-1)$

$=\frac{\partial\left[\frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j-y_j)^2\right]}{\partial \hat{y}_j}*\hat{y}_j(1-\hat{y}_j)*(-1)$

$=\frac{1}{2}\times 2*(\hat{y}_j-y_j)*1*\hat{y}_j(1-\hat{y}_j)*(-1)$

$=(y_j-\hat{y}_j)\hat{y}_j(1-\hat{y}_j)$

$=g_j$

$\therefore\ \Delta\theta_j=-\eta\frac{\partial E}{\partial \theta_j}=-\eta g_j$

$=-\eta\hat{y}_j(1-\hat{y}_j)(y_j-\hat{y}_j)$

同理：通过对输出层公式的推导我们可以推出隐藏层的权重与阈值的变化量：$\Delta v_{ih}=\eta b_h(1-b_h)\sum_{j=1}^{l}w_{hj}g_jx_i$

$\Delta\gamma_h=-\eta b_h(1-b_h)\sum_{j=1}^{l}w_{hj}g_j$

介绍到这里我们可以使用 python 对其进行实现