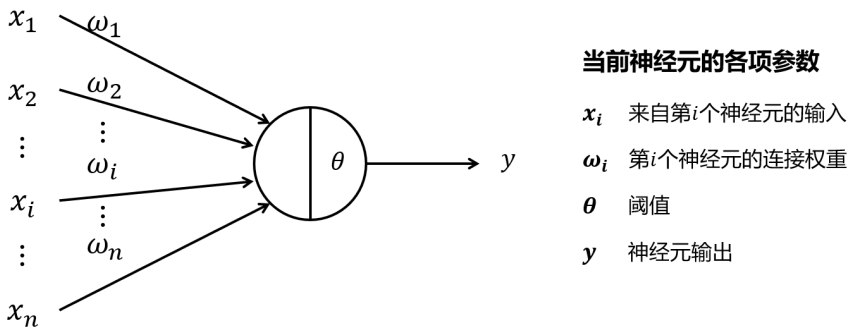
Back Propagation Neural Networks

神经元模型

M-P 神经元模型

基本神经元模型中，神经元接收到来自n个其他神经元传递过来的输入信号，这些输入信号通过带权重的连接(connection)进行传递
神经元接收到的总输入值将与神经元的阈值进行比较，然后通过 **激活函数（activation function）** 处理以产生神经元的输出

- 神经元模型如下



x_i 是来自第 i 个神经元的输入，每个神经元会接收到 n 个神经元的输入。每个输入值会占据一定的权重

$$\omega_i x_i$$

所有输入 (n 个输入) 加权后进行累加

$$\sum_{i=1}^{n} \omega_i x_i = \omega_1 x_1 + \omega_2 x_2 + ... + \omega_n x_n$$

当该累加值超过一定的阈值 θ 之后，该神经元就会被“激活”

$$f(\sum_{i=1}^{n} \omega_i x_i - \theta)$$

该神经元“激活”之后，就会向其他神经元发送信息，这个信息就是该神经元的输出 y

$$y = f(\sum_{i=1}^{n} \omega_i x_i - \theta)$$

- 为什么要加权 ω_i?

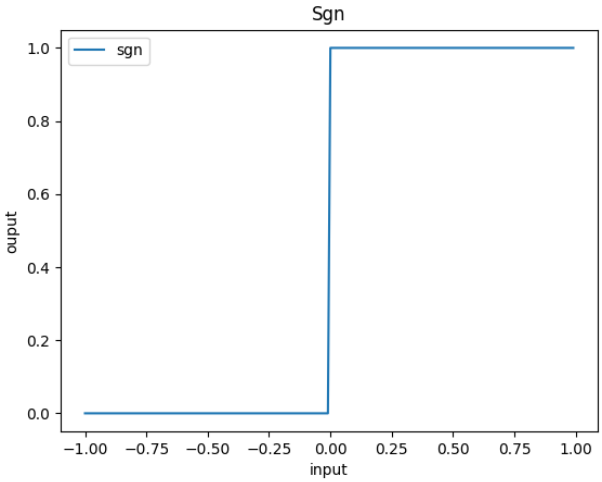
> 每个输入信息对于结果会有不同的影响，而我们需要衡量每个输入对于结果产生的影响的程度就需要这样一个权重。
> 翻译成人话就是，你对象和你舍友都在你生日送礼物，但是你收到两个礼物之后的开心程度(应该？)是不一样的。

- **激活**是含义?

> 不好理解的是，为什么超过一定阈值之后，神经元会被激活
> 其实和和生物学上是类似的，神经元的电位超过一定**阈值**之后就会被激活。当神经网络中的神经元接收到的值超过**阈值**θ 时，才会将信息继续传递，或者说，才会产生输出。
> 翻译成人话就是，你在生日收到足够多的礼物之后，才会开心。
> 因此，产生一个词叫**激活函数**，激活函数所做的事情就是去判定该**阈值**、衡量**多少礼物才叫足够多**这个问题。

- 激活函数

> **理想**的**激活函数**是一个**阶跃函数**，如下



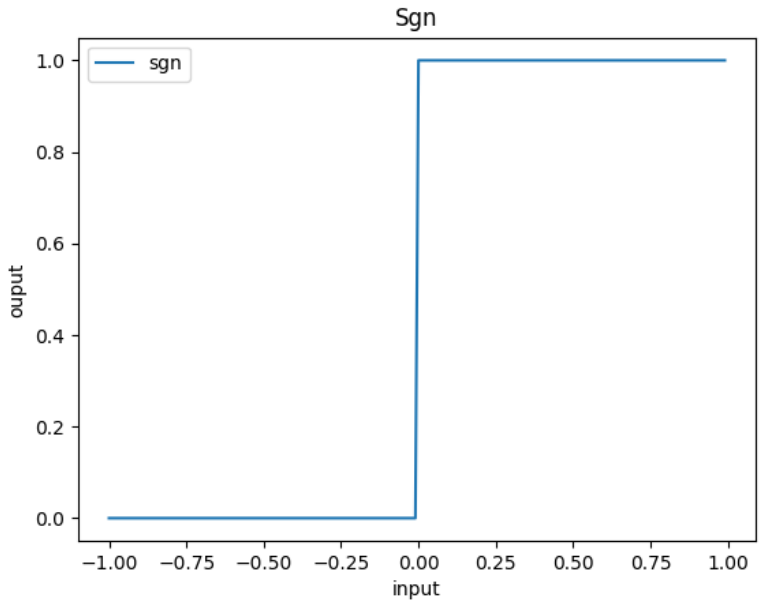
可以看到当输入的值大于0的时候，会输出1，这时候该神经元表现为**激活状态**，输出的1是对下一个神经元有贡献的，也就是说该神经元的输出是可以传递给下一个神经元；
当输入的值小于0的时候，会输出0，0对于下一个神经元没有贡献，这时候的神经元是**未激活**的。
这里的**贡献**是指，下一个神经元接收到的 $x_i = 0$ ，那也就不会对下一个神经元产生任何影响；下一个神经元接收到的 $x_i \neq 0$时，那么会对下一个神经元产生影响
减去阈值 θ 再激活，也就是把判断是否**激活**的条件统一成 $x \geq 0$ (便于计算)，如下

$$sgn(x) = \begin{cases} 1, & x \geq \theta \\ 0, & x < \theta \end{cases} \quad \Rightarrow \quad sgn(x-\theta) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

激活函数

- 理想激活函数

理想激活函数是一个阶跃函数

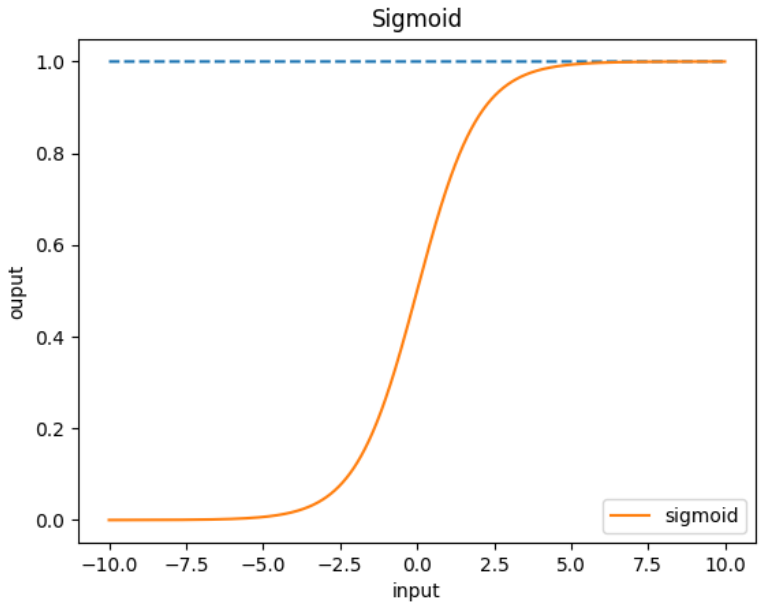


$$sgn(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

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

阶跃函数具有不连续、不光滑等缺点，因此常使用 Sigmoid 函数作为激活函数

- Sigmoid 函数



$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

```
def sigmoid(x):
    return 1./(1.+np.exp(-x))
```

感知机与多层网络

感知机 (Perceptron) 由两层神经元组成

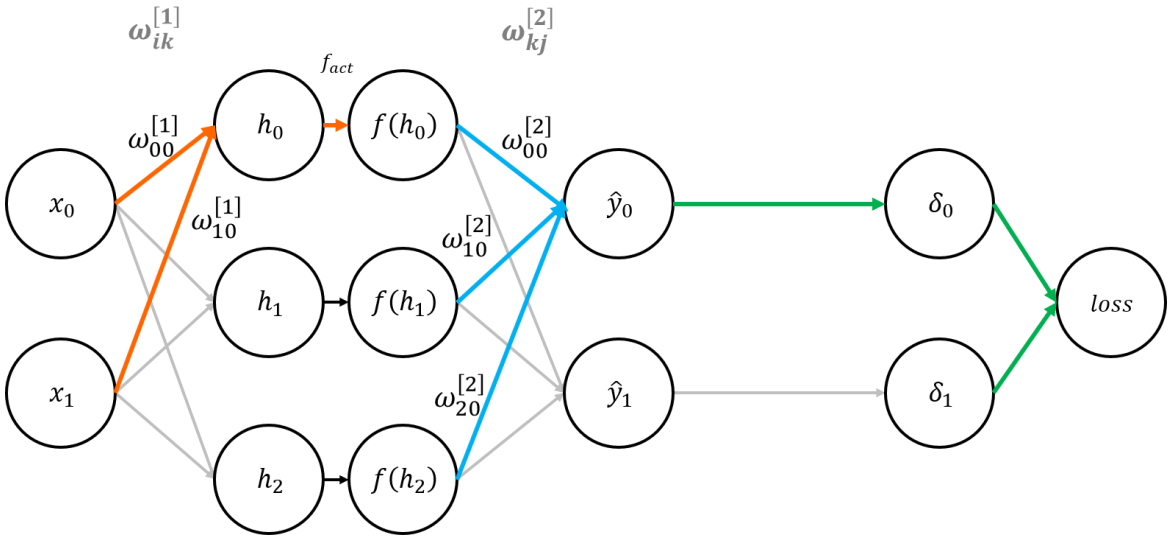
BP神经网络原理

推导思路和过程

网络的前向传播过程的推导

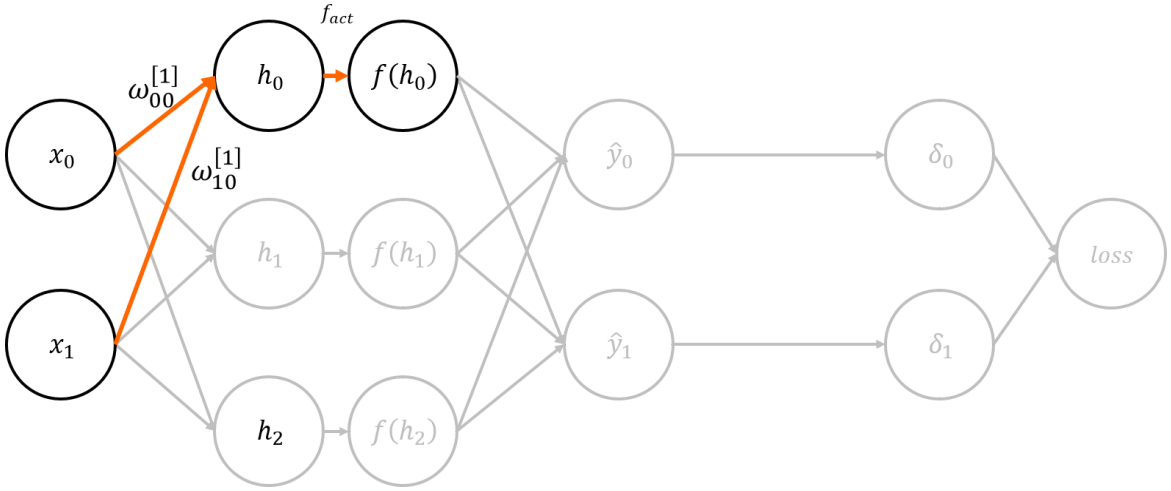
> 网络前向传播的过程也叫**推理** (inference)

这里我们做最简单的网络模型



- 其中，输入层有 n 个神经元结点 x_i，　$i = 0, 1, ..., n$
- 其中，隐藏层有 t 个神经元结点 h_k，　$k = 0, 1, ..., t$
- 其中，输出层有 m 个神经元结点 $\hat{y}_j$，　$j = 0, 1, ..., m$

先推理第一层网络



第一层网络的传播过程可以总结为以下公式

$$h_k = \sum_{i=0}^{n-1} \omega_{ik}^{[1]} x_i + b$$

算上激活之后就是这一层网络的最终输出

> 这里神经元应该只画一个就好，但是为了便于理解和后面反向传播的过程，我这里画两个圆◯

$$f(h_k) = f_{act}(\sum_{i=0}^{n-1} \omega_{ik}^{[1]} x_i + b_0^{[1]})$$

其中，有 n 个输入 $x_i, i = 0, ..., n-1$

> 这里是为了迎合计算机编程数组下标从 0 开始的习惯

下面两个参数都是训练神经网络的时候**最重要的参数**，也是需要训练的参数，神经网络的训练也就是为了调整下面两个参数，使得网络的推理性能达到最佳

- 参数 $\omega_{ik}^{[l]}$ ，表示在第 l 层网络中，第 i 个输入和第 k 个隐藏层神经元连接的权重
- 参数 $b_k^{[l]}$ $(bias)$ ，表示第 l 层网络、第 k 个结点的偏置项

$f_{act}(x)$ 是激活函数 (activation)

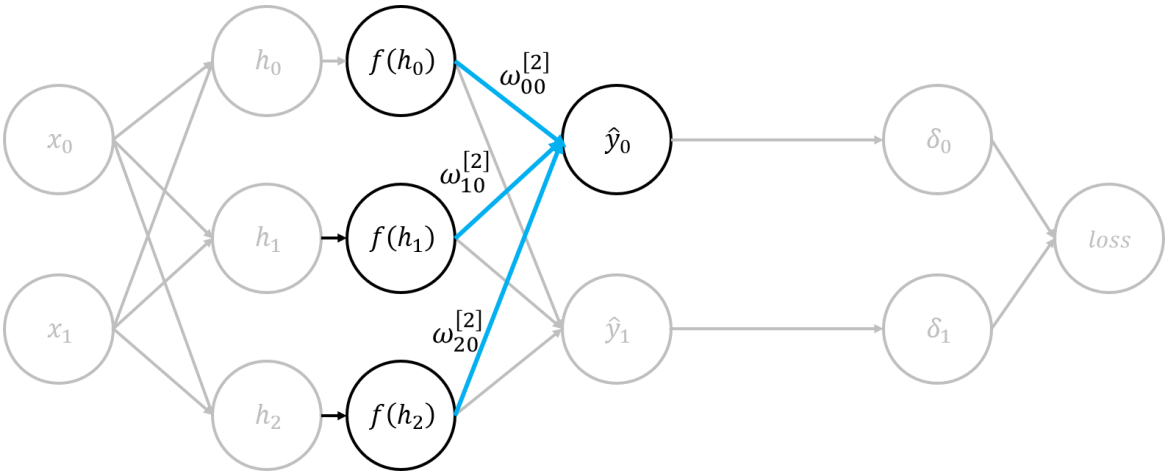
那么对于隐藏层的第一个结点 $f(h_0)$

$$\begin{aligned} h_0 &= \sum_{i=0}^{1} \omega_{i0}^{[1]} x_i + b_0^{[1]} \\ &= \omega_{00}^{[1]} x_i + \omega_{10}^{[1]} x_i + b_0^{[1]} \end{aligned}$$

激活

$$\begin{aligned} f(h_0) &= f_{act}(\sum_{i=0}^{1} \omega_{i0}^{[1]} x_i + b_0^{[1]}) \\ &= f_{act}(\omega_{00}^{[1]} x_i + \omega_{10}^{[1]} x_i + b_0^{[1]}) \end{aligned}$$

> 这里的激活函数选择哪一个函数对对最后得出的结论没有影响，并且会使得推理过程比较混乱，暂不实例化该函数
> 如果选择 `Sigmoid` 函数作为激活函数，那么输出如下：
>
> $$\begin{aligned} f(h_0) &= f_{act}(\sum_{i=0}^{1} \omega_{i0}^{[1]} x_i + b_0^{[1]}) \\ &= f_{act}(\omega_{00}^{[1]} x_i + \omega_{10}^{[1]} x_i + b_0^{[1]}) \\ &= \frac{1}{1 + e^{-(\omega_{00}^{[1]} x_i + \omega_{10}^{[1]} x_i + b_0^{[1]})}} \end{aligned}$$



第二层网络，也就是输出层网络的传播过程可以总结为以下公式

$$\hat{y}_j = \sum_{k=0}^{m-1} \omega_{kj}^{[2]} h_k + b_0^{[2]} \qquad k = 0, ..., t$$

其中，有 t 个隐藏层输入 h_k，这里输出值 $\hat{y}_j$ 也是预测值

对于输出层的第0个输出结点，有

$$\hat{y}_0 = \sum_{k=0}^{m-1} \omega_{k0}^{[2]} h_k + b_0^{[2]}$$

至此，推理过程全部结束 😇

推理是指在网络的单向传播过程，为的是得到网络的输出，这个输出就是我们需要的，我们可以根据这个输出进行判断

但是，为了进行参数调整，我们必须计算损失函数

网络的反向传播过程的推导

我们得到了网络的输出，也就是一个网络给出的预测值，我们就可以用这个预测值来判断结果了，那这个预测值和实际值是不是一致的呢？这是我们不知道的。

> 比如我们一个模型的功能是用来识别一张图片中是不是一个人，那么我们向网络送入一张图片的时候，网络就会推理得到一个预测值，这个预测值实际上是一个概率，理想情况下，我们给一个人的照片，输出概率为1，但是实际上这个概率不大可能是1的，但是我们希望做到这个概率尽可能解决1以使得网络的识别性能达到最优。

为了使得输出概率最大，我们调参的时候就需要修正网络中的不同参数。

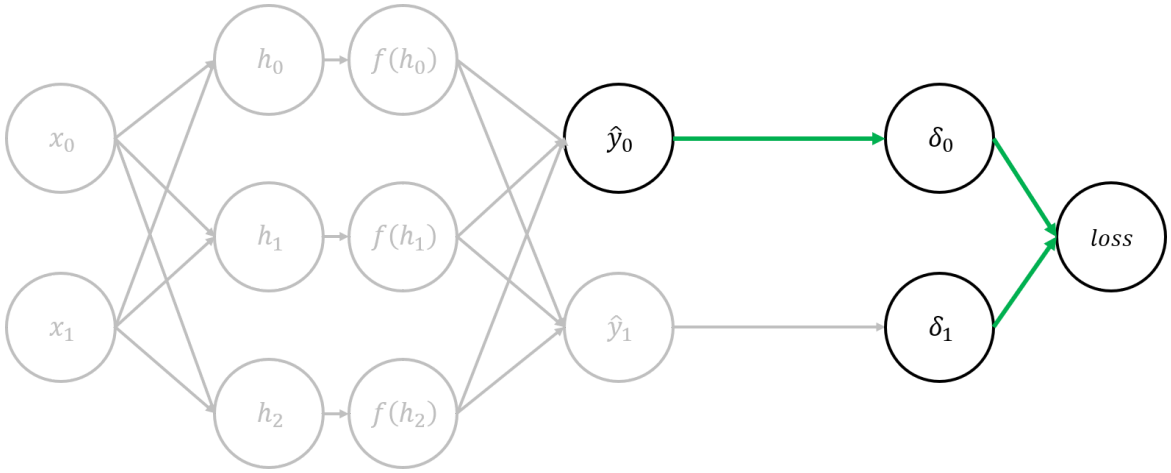
但是问题来了，当我们不只有一个输出的时候（如下图），我们如何去界定每一个输出达到最优呢？调整参数的时候也会对各个输出有影响的。

> 比如在二输出网络中，有 $\hat{y}_0$ 和 $\hat{y}_1$。当我们好不容易调整好网络参数使得每次输出的 $\hat{y}_0$ 都可以达到最优（最优是指，预测值和实际值的误差尽可能小），但是 $\hat{y}_1$ 却很大了，这就破坏了网络。

因此，我们想到一个思路，就是计算每个输出的误差，然后对误差进行累加，只要总误差最小，那也就是使得各个误差 尽可能 达到最小，那么这个网络也会最优。

因此需要引入一个**误差函数**来**量化**预测值和输出值的差异

> 管你听没听懂，看就完事了 😊



得到了输出的预测值 $\hat{y_0}$ ，和我们真实值之间的误差为 $|\hat{y_0} - y_0|$
为了消除绝对值的影响、便于计算，我们用我们小学二年级就学过的 **MSE（均方方差）**，这也是常用的误差函数

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2$$

> 当然，根据不同的需求选择不同的损失函数，损失函数可以大致分为两类：分类损失（Classification Loss）和回归损失（Regression Loss）。

不过在这里我们还是用 $\delta_j = Loss(\hat{y}_j, y_j)$ 来表示损失函数，也是便于计算而已。

得到了 δ_j 之后，我们就可以计算总误差

$$loss = \sum_{j=0}^{m-1}\delta_j = \sum_{j=0}^{m-1} Loss(\hat{y}_j, y_j)$$

得到了以上的通用公式之后，我们就可以计算出最后**总**损失函数了

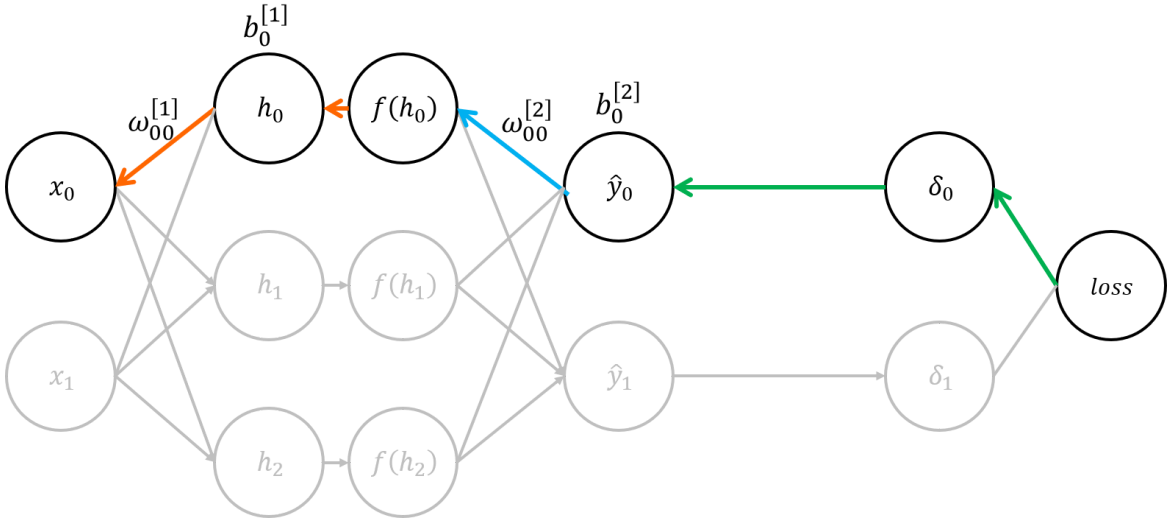
$$
\begin{aligned}
loss &= \sum_{j=0}^{m-1} \delta_j = \sum_{j=0}^{1} Loss(\hat{y}_j, y_j) \\
&= Loss(\hat{y}_0, y_0) + Loss(\hat{y}_1, y_1)
\end{aligned}
$$

在上面的全部计算中，我们只计算了一条线，是经过 x_0、h_0、$f(h_0)$、$\hat{y}_0$、δ_0、$loss$，但是该网络还有其他的参数$\omega_{ik}^{[l]}$、$b_k^{[l]}$，为每一层的权重和偏置。

$loss$ 是仅与权重 `weight` 和偏置 `bias` 有关，而与各个神经元的输出无关的函数，实际上是这样的，所以在之后对 $loss$ 求导的时候都是求偏导

$$
loss = loss(\omega_{00}^{[1]}, \omega_{01}^{[1]}, ..., \omega_{ik}^{[l]}; b_0^{[1]}, b_1^{[1]}, ...b_k^{[l]})
$$

损失函数有了，那我们现在开始正式的反向传播 😃



我们反向传播的目的是**通过损失函数去调整网络参数，从而使得此后的推理过程中，求得的损失函数最小，这时候得到的输出是最好的**

而**梯度下降**（这应该是神经网络中最重要的公式吧）

$$
param_{new} = param - lr \times \frac{\partial loss}{\partial param}
$$

> lr 是学习率
> 先把这个提到前面，暂时不需要理解，我们先知道有这么一个公式，而我们需要知道的是最后的求偏导的部分

那我们开始吧

先从 $\omega_{00}^{[2]}$ 开始，利用我们小学二年级就学过的**链式求导法则**

$$\frac{\partial loss}{\partial \omega_{00}^{[2]}} = \frac{\partial loss}{\partial \delta_0} \times \frac{\partial \delta_0}{\partial \hat{y}_0} \times \frac{\partial \hat{y}_0}{\partial \omega_{00}^{[2]}}$$

$$loss = \sum_{j=0}^{m-1} \delta_j = \delta_0 + \delta_1 \qquad \therefore \frac{\partial loss}{\partial \delta_0} = 1$$

$$\delta_0 = Loss(\hat{y}_0, y_0) \qquad \therefore \frac{\partial Loss(\hat{y}_0, y_0)}{\partial \hat{y}_0}$$

$$\hat{y}_0 = \sum_{k=0}^{m-1} \omega_{k0}^{[2]} h_k + b_0^{[2]} \qquad \therefore \frac{\partial \hat{y}_0}{\partial \omega_{00}^{[2]}} = h_0$$

综上

$$\frac{\partial loss}{\partial \omega_{00}^{[2]}} = 1 \times \frac{\partial Loss(\hat{y}_0, y_0)}{\partial \hat{y}_0} \times h_0$$

如果选损失函数为 均方误差

$$MSE = \frac{1}{2} \sum_{i=0}^{1} (y_i - \hat{y}_i)^2 = (y_0 - \hat{y}_0)^2 + (y_1 - \hat{y}_1)^2$$

那么可以进一步

$$\begin{aligned} \frac{\partial loss}{\partial \omega_{00}^{[2]}} &= 1 \times \frac{\partial Loss(\hat{y}_0, y_0)}{\partial \hat{y}_0} \times h_0 = 1 \times \frac{\partial MSE(\hat{y}_0, y_0)}{\partial \hat{y}_0} \times h_0 \\ &= 1 \times (-2\hat{y}_0) \times h_0 = -2\hat{y}_0 h_0 \end{aligned}$$

> 所以其实损失函数选择哪一个对公式的推导是没有影响的，为了简化推导流程，我们在上面都不会计算具体的损失函数，激活函数也是如此的，因此推导过程都没有实例化激活函数和损失函数
>
> 奇怪的是，MSE如果定义成 $MSE = \frac{1}{n} \sum_{i=0}^{n-1} (\hat{y}_i - y_i)^2$ ，那么得到的结果就是 $2\hat{y}_0 h_0$ 那就和上面的结果不一样了?
> 其实是一样的，梯度下降的过程是可以从左边趋向于极小值点，也可以从右边趋向于极小值点，这就是上面取正还是取负的区别，但最后都会逼近到极小值点。
>
> 这里为什么说是极小值点，而不是最小值点呢?
> 因为在 MSE 中极小值点就是最小值点，但是有其他的一些损失函数不一定只有一个极小值点，这里只是比较严谨一点。

> 也可以看出，为了方便寻找损失函数的最小值，一般选择的损失函数只存在一个极小值点。这也是好的损失函数必须具备的。在自定义损失函数的时候就必须考虑到这一点。

那么修正后的参数

$$\begin{aligned}\omega_{00}^{[2]}{}_{new} &= \omega_{00}^{[2]} - lr \times \frac{\partial loss}{\partial \omega_{00}^{[2]}} \\ &= \omega_{00}^{[2]} - lr \times \left(-2\hat{y}_0 h_0\right)\end{aligned}$$

下一次推理就用这个参数进行推理，得到的损失函数（理论上）就会更小

> 请自行推导 $\frac{\partial loss}{\partial \omega_{00}^{[1]}}$

练习题

已知一个 2 层的BP神经网络可实现**逻辑同或**运算，输入节点数为2，隐层节点数为3，输出节点数为1，具体构造如图所示。

编程实现该神经网络的网络结构并且训练该神经网络，其中学习率和初始权值需随机产生，激活函数为 `Sigmoid` 函数，即 $f(x) = \frac{1}{1+e^{-x}}$

要求程序能显示每次迭代的权值和神经网络输出结果，并显示整个训练的迭代次数和运行时间