反向传播及其在深度学习中的应用

1、简介

反向传播算法是一种可使深度学习模型计算上可行的重要算法。对现代的神经网络来说，**该算法实现的梯度下降训练要比那些幼稚实现快上百万倍**，该算法的通用的名称为**“反向模式求导”**。该算法最主要的功能是能够**快速计算求导**。

2、计算图

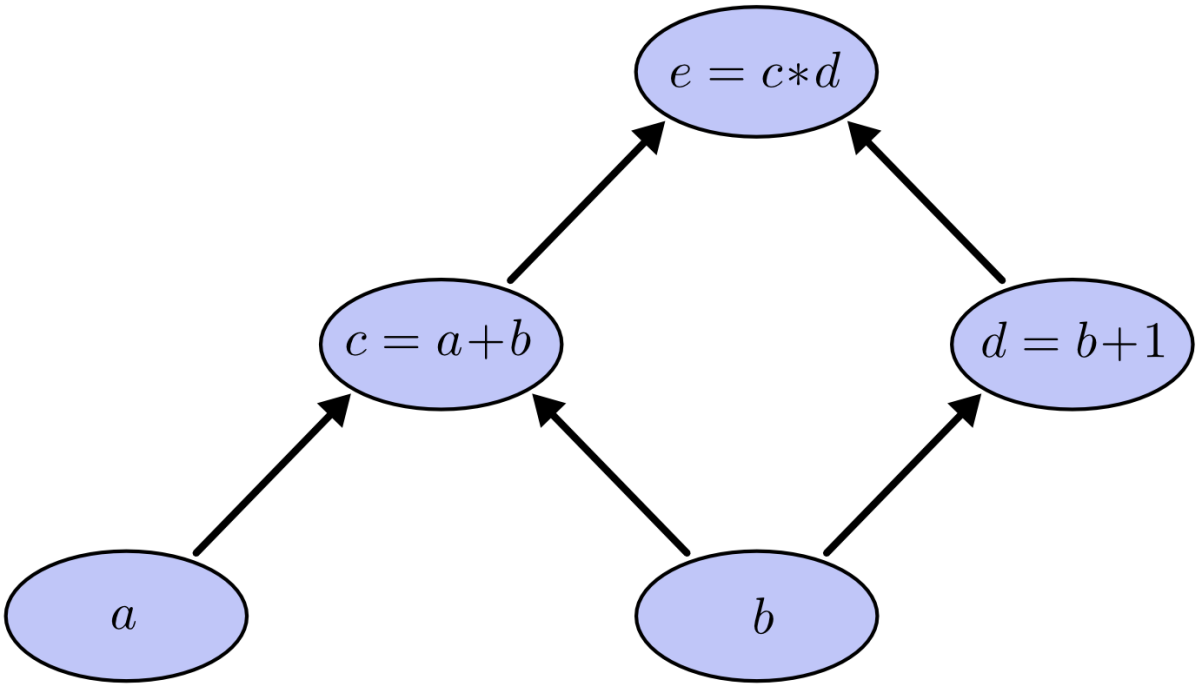
计算图是一种好理解数学表达式的方式，比如，表达式 $e=(a+b)*(b+1)$。 这有三个算式：两个加法一个乘法。为了方便理解，引入两个中间变量 c 和 d，这样每个算式都有输出结果变量。如下：

$$c=a+b$$

$$d=b+1$$

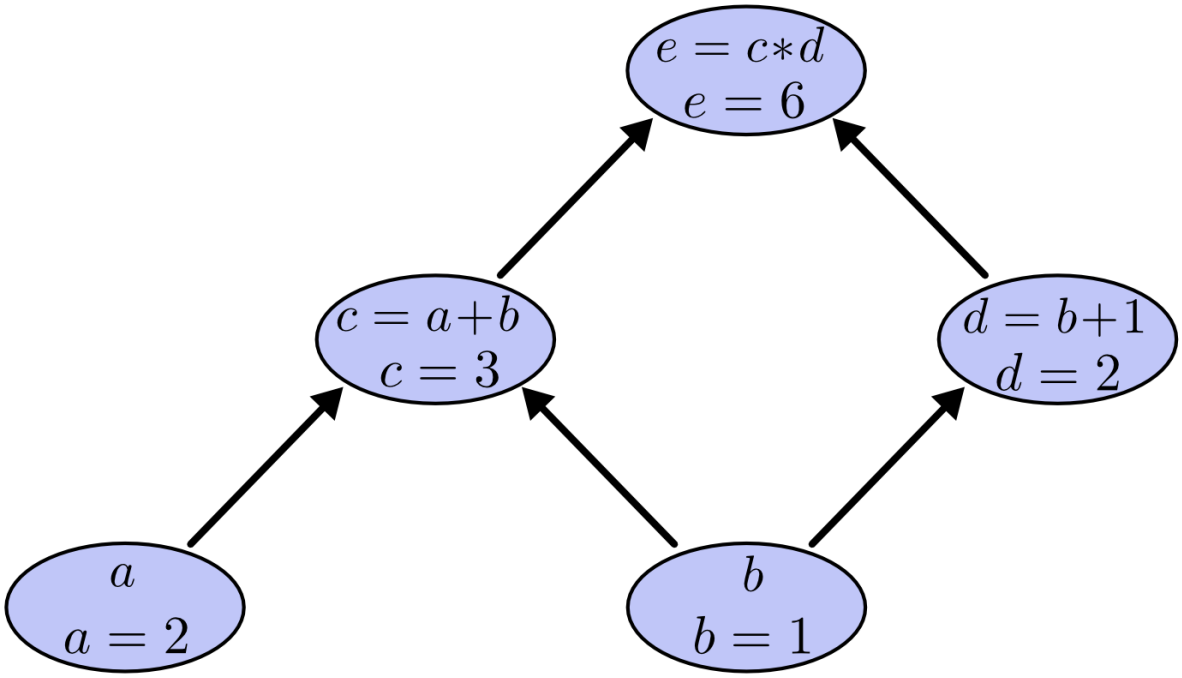
$$e=c*d$$

为了创建计算图，将每个算式和输入变量放于每个节点中，箭头的指向关系为一个节点计算结果指向该节点作为输入结果的节点



这种类型的图经常出现在计算机科学中，特别是在谈及函数编程，这很接近依赖图和调用图的概念，同时也是现流行深入学习框架Tensorflow的核心抽象，我们可以设置初始数值，从图自底向上计算每个算式，例如 $a=2$ 和 $b=1$

generated by haroopad



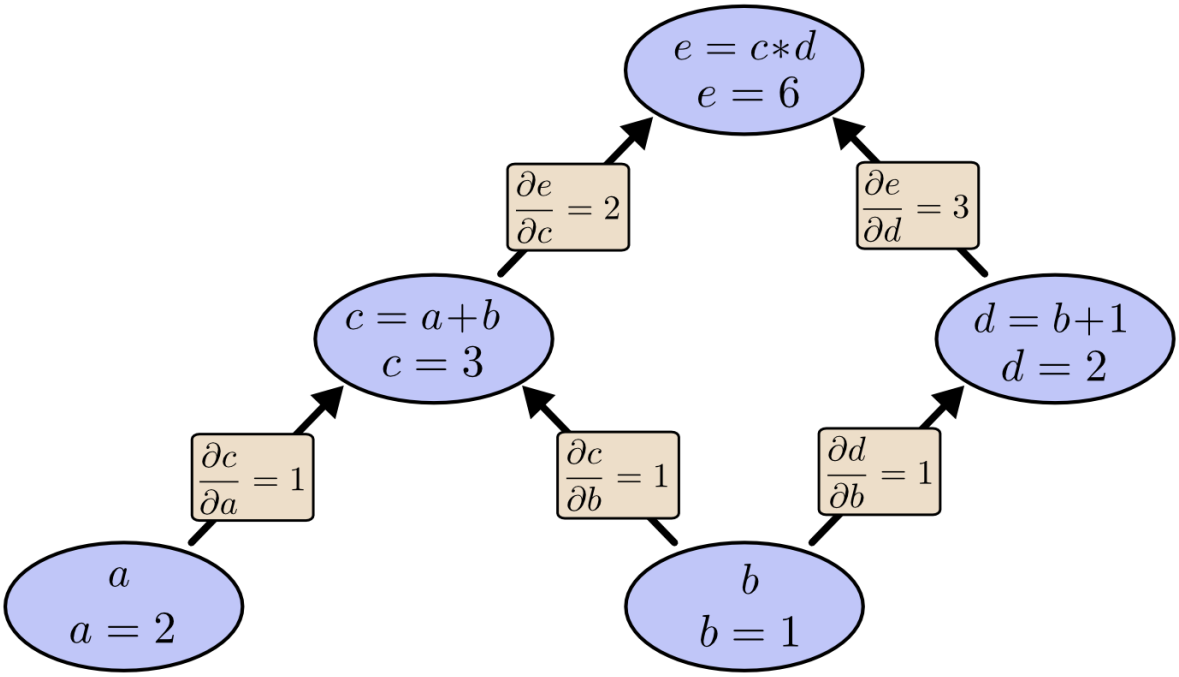
3、计算图的求导

如果想理解计算图的求导，重要的是**理解图边上的求导。**如果 a 直接影响 c，我们想知道 a 怎么影响 c，如果 a 变化一点，c 会怎么改变呢？我们称 c 对 a 的偏导，为了计算该计算图的偏导，我们得需下面的两个规则**（复合函数求偏导）**：

$$\frac{\partial(a+b)}{\partial a} = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} = 1$$

$$\frac{\partial uv}{\partial u} = u\frac{\partial v}{\partial u} + v\frac{\partial u}{\partial u} = v$$

如图，每个偏导都在图边标出



generated by haroopad

如果我们想了解不是直接相连节点间的影响,比如图中的 e 是怎么被 a 影响的。如图可知，我们若以速率 1（即斜率 1）改变 a，c 也会跟随以相同的速率改变。相对应的 c 以速率1改变，那么 e 就以速率2改变。所以 e 相对 a 就以 $1*2$ 的速率改变。

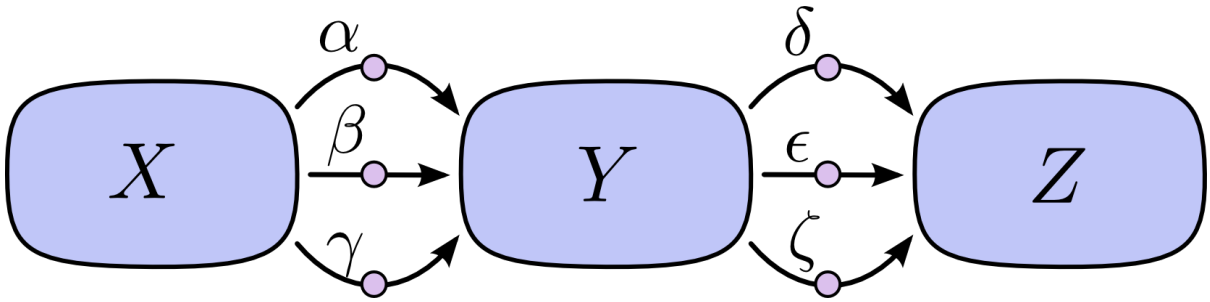
总结来说就是将两节点间**可能边相乘再相加。**例如，求对 e 求 b 的偏导：

$$\frac{\partial e}{\partial b}=\frac{\partial e}{\partial c}\cdot\frac{\partial c}{\partial b}+\frac{\partial e}{\partial d}\cdot\frac{\partial d}{\partial b}=1\times 2+1\times 3=5$$

这表明的是 b 怎么通过影响 c 和 d 来间接引起 e 的变化的。

4、考虑边间

上面的问题可以很容易获得一个各个边的加和展开式



上图中，从 X 到 Y，Y 到 Z 各有三条路径，如果我们想求 $\frac{\partial Z}{\partial X}$ 就要将所有的路径相加，要有九条路径

$$\frac{\partial Z}{\partial X}=\alpha\delta+\alpha\varepsilon+\alpha\zeta+\beta\delta+\beta\varepsilon+\beta\zeta+\gamma\delta+\gamma\varepsilon+\gamma\zeta$$

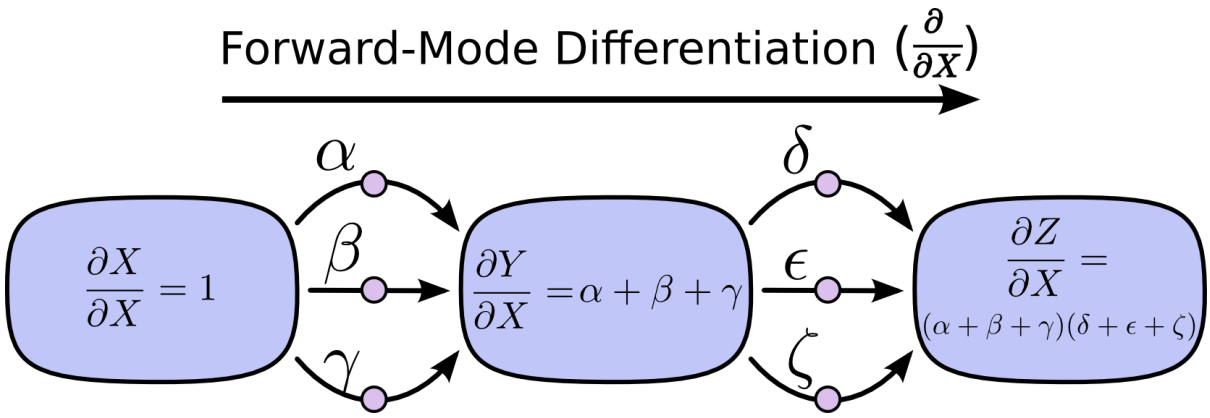
上面的展开式有9路径，当图更复杂的时候，如上的展开式就会暴涨，展开项合并因式，比起简单的相加各边会更好些。

$$\frac{\partial Z}{\partial X}=(\alpha+\beta+\gamma)(\delta+\varepsilon+\zeta)$$

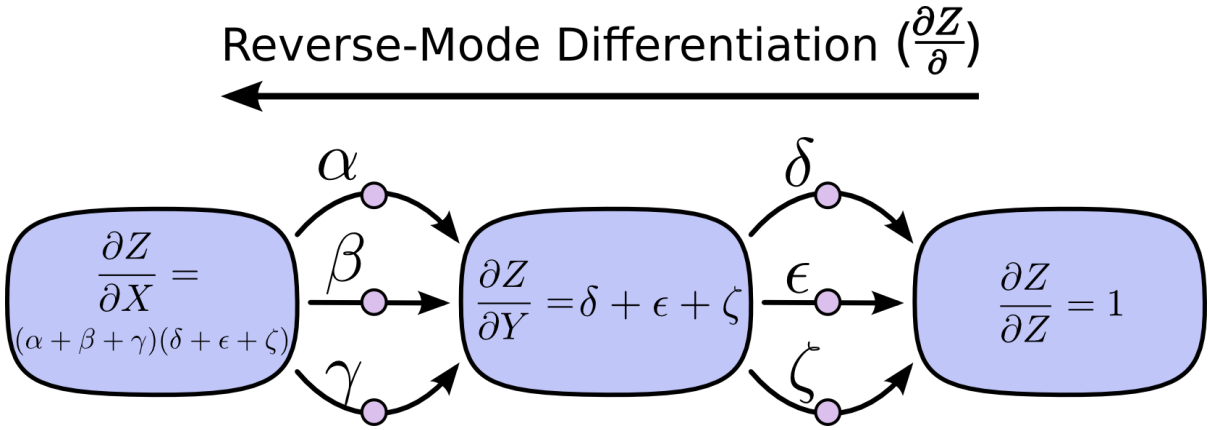
这就是**“正向求导”和“反向求导”**的来源。利用求各个边的因素和来使得计算效率化，不是显式地去求各个路径的和，而是将每个路劲求和反向合并获取每个节点值，并且两个算法都是只用到一次节点的边。

“正向求导”从计算图的输入开始，直到到结束点。**在每个节点，都算一次输入到该节点的边的加和**，每个路径表示一条影响该节点的方式，将其求和演化出每个节点被引起变化的各个方面之和，**每次都是计算** $\frac{\partial}{\partial X}$：

generated by haroopad



另一方面 “反向求导”是从图的输出开始，向前移动。每个节点将其产生的路径合并:



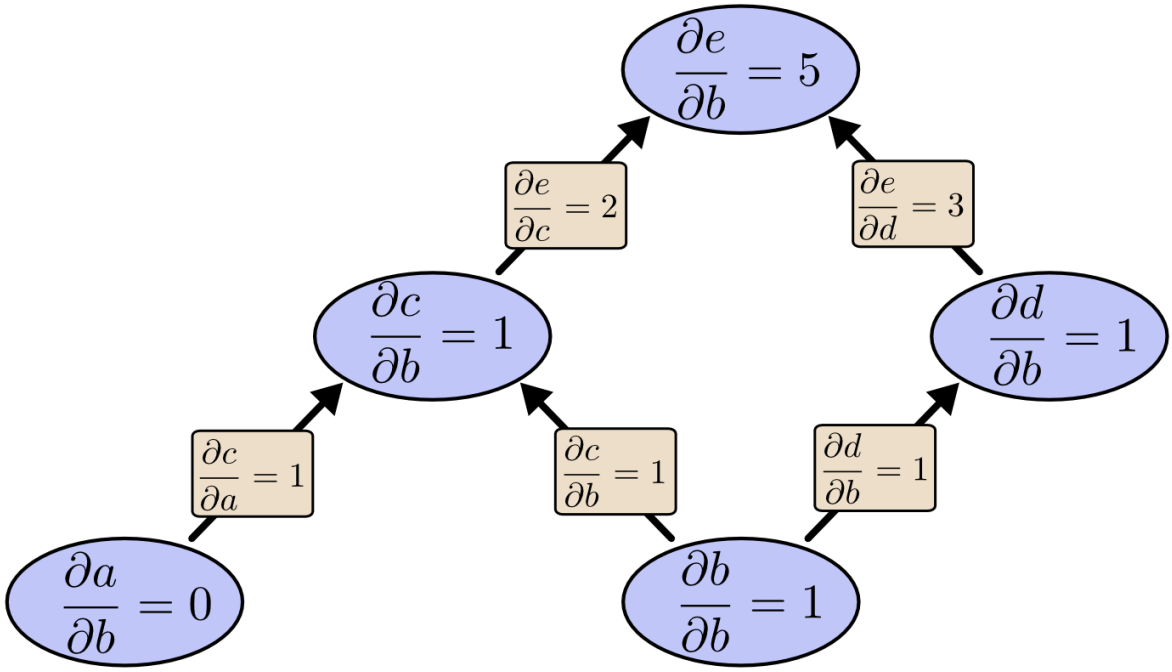
顺序模式求导，记录一个输入怎么影响每个节点。反序模式记录每个节点怎么影响一个输入，也就是说顺序模式对**每个节点** $\frac{\partial}{\partial X}$，而反序模式是对**每个节点求** $\frac{\partial Z}{\partial}$。

5、计算上的优势

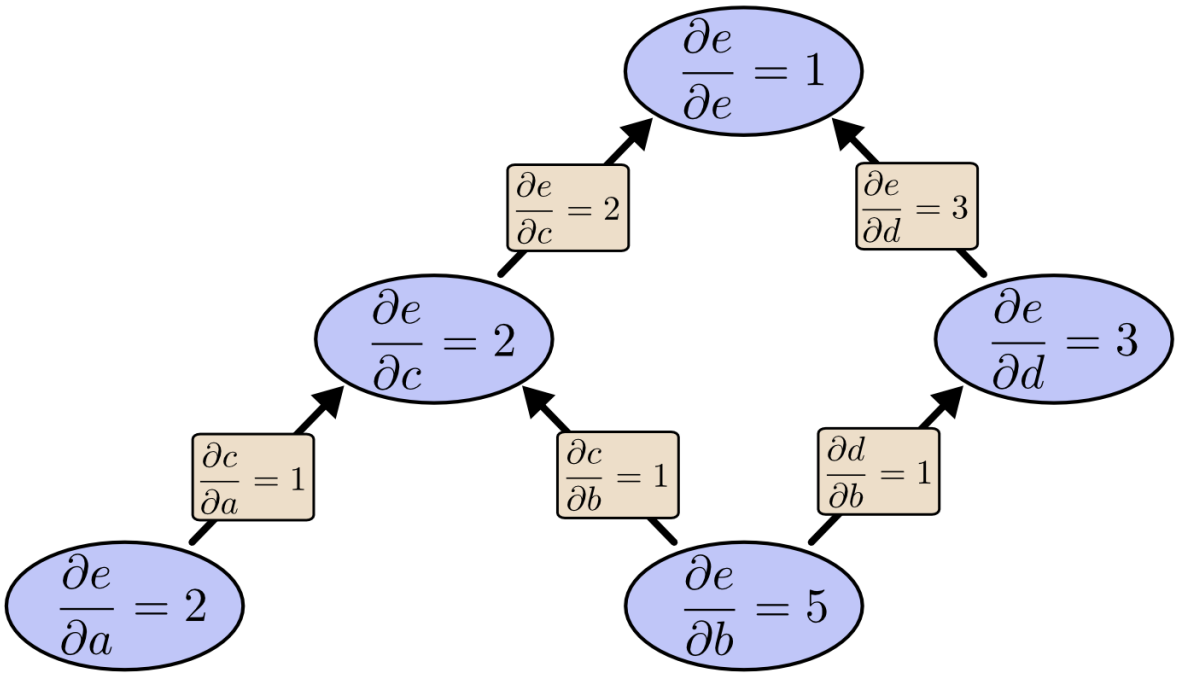
至此，你可能疑惑，反序模式求导做的事情貌似和正序模式是一样的，为什么还有人会对反序模式求导那么上心呢？让我们考虑最开始的例子吧：

我们利用正序求导**自底向上**求解。下图展示了每个节点对 b 的求导：

generated by haroopad



我们已经计算出**输出对输入**的求偏导：$\frac{\partial e}{\partial a}$ ，如果我们从e 往下求反序偏导呢？下图给出了每个 e 对其他每个节点的求导:



当说到反序模式求导列出输入e对每个节点的偏导，**的的确确是指的是每个节点。**我们已经获得了**两个输出对输入的求导**：$\frac{\partial e}{\partial a}$ 和 $\frac{\partial e}{\partial b}$ 。**正序模式只返回一个，反序模式却给出所有的**！

对此文来说，这只是两倍的加速，但是想象下，如果有百万个输入且只有一个的输出的情况下，正序模式求导要进行百万次的遍历计算图来获取输出对每个输入点的偏导，但是反序模式只需要一次就能做到。百万倍的加速还是很可观的吧！！

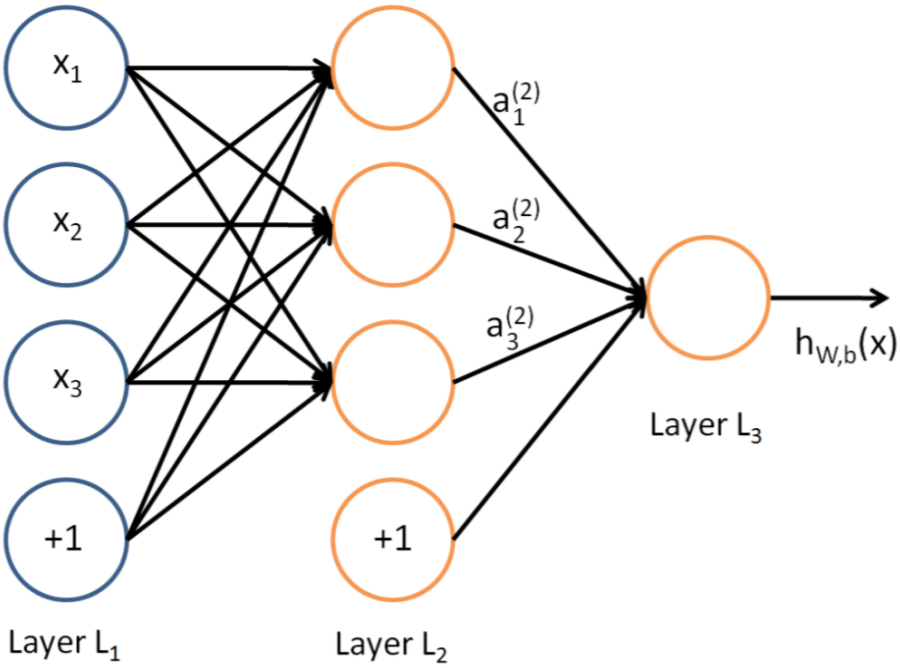
当进行神经网络训练，我们要考虑代价（一描述神经网络差劲的指数）作为一个各个参数的函数（描述该神经网络行为的参数）。为了梯度下降算法，我们要计算代价指数对每个参数的偏导。现在由于每个

神经网络要有数百万甚至上千万的参数，所以反序模式的求导，在神经网络里被称为 **“反向传播算法”就可以使计算得到很大加速。**

（有没有其他的场景使得正序求导算法更适用些呢？有的，由于反序模式求导一次给出一个输出对所有的输入的偏导，而正序模式求导一次给出的其实是所有输入对一个输入的求偏导，所以有一个函数伴随多个输出，正序模式求导会更快速）

6、反向传播在神经网络上的应用

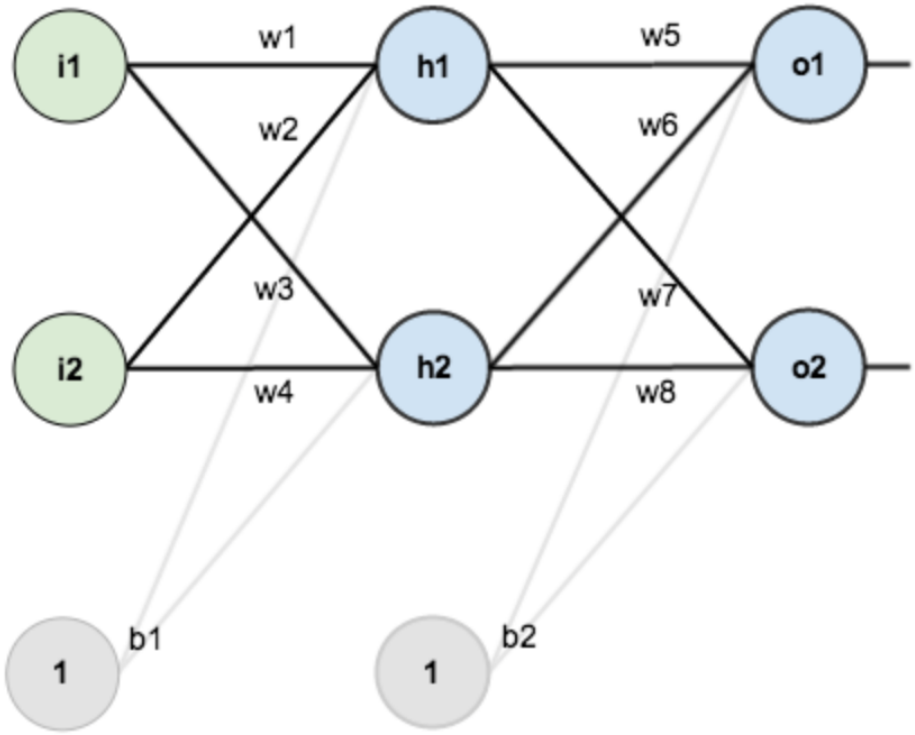
下图是一个简单的神经网络模型：



这是典型的三层神经网络的基本构成，$L1$ 是输入层，$L2$ 是隐含层，$L3$ 是输出层，我们现在手里有一堆数据 $\{x_1, x_2, x_3, \ldots, x_n\}$，输出也是一堆数据 $\{y_1, y_2, y_3, \ldots, y_n\}$，现在要他们在隐含层做某种变换，让你把数据灌进去后得到你期望的输出。如果你希望你的输出和原始输入一样，那么就是最常见的自编码模型（Auto-Encoder），如果你的输出和原始输入不一样，那么就是很常见的人工神经网络了，相当于让原始数据通过一个映射来得到我们想要的输出数据。

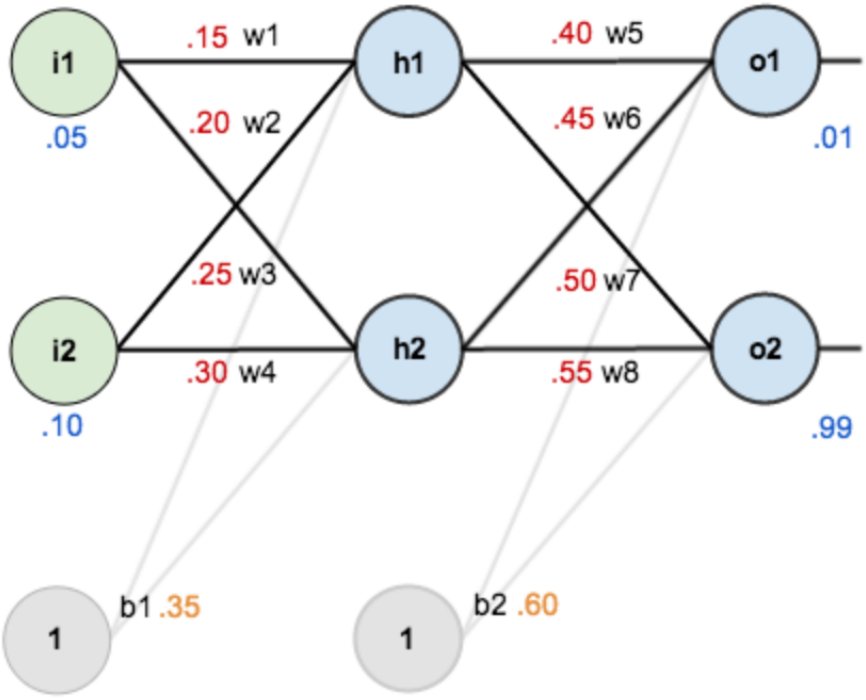
假设有如下的网络层：

generated by haroopad



第一层是输入层，包含两个神经元 i_1，i_2，和截距项 b_1；第二层是隐含层，包含两个神经元 h_1, h_2 和截距项 b_2，第三层是输出 o_1, o_2，每条线上标的 w_i 是层与层之间连接的权重，激活函数我们默认为 $sigmoid$ 函数。

现在对他们赋上初值，如下图：



输入数据: $i_1 = 0.05,\ i_2 = 0.10$
输出数据: $o_1 = 0.01, o_2 = 0.99$
初始权重:

generated by haroopad

$w_1 = 0.15, w_2 = 0.20, w_3 = 0.25, w_4 = 0.30, w_5 = 0.40, w_6 = 0.45, w_7 = 0.50, w_8 = 0.55$
目标：给出输入数据 $i_1, i_2(0.05$ 和 $0.10)$，使输出尽可能与原始输出$o_1, o_2(0.01$和$0.99)$接近。

Step1: 输入层 → 隐含层

$$net_{h_1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1 = 0.3775$$

神经元 h_1 的输出为(使用 $sigmoid$ 函数):

$$out_{h_1} = \frac{1}{1 + e^{-net_{h_1}}} = 0.5932$$

同理可以计算出 h_2 的输出：

$$out_{h_2} = \frac{1}{1 + e^{-net_{h_2}}} = 0.5968$$

Step2: 隐含层 → 输出层
计算输出层神经元 o_1 和 o_2 的值：

$$net_{o_1} = out_{h_1} * w_5 + out_{h_2} * 6 + b_2 * 1 = 1.1059$$

$$out_{o1} = \frac{1}{1 + e^{-net_{o_1}}} = 0.7513$$

同理可以计算 o_2 的输出：

$$out_{o2} = \frac{1}{1 + e^{-net_{o_2}}} = 0.7729$$

这样前向传播的过程就结束了，我们得到输出值为 $[0.7513, 0.7729]$，与实际值 $[0.01, 0.99]$ 相差还很远，现在我们对误差进行反向传播，更新权值，重新计算输出。

Step3: 输出层 → 隐含层
总误差可以表示为：

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

由于包含两个输出，所以需要计算两个输出的误差，并将其加和：

$$E_{o1} = \frac{1}{2} (target_{o_1} - out_{o_1})^2 = \frac{1}{2} (0.01 - 0.7513)^2 = 0.2748$$

$$E_{o2} = \frac{1}{2} (target_{o_2} - out_{o_2})^2 = \frac{1}{2} (0.99 - 0.7729)^2 = 0.0236$$

总误差就是对这两者求和：

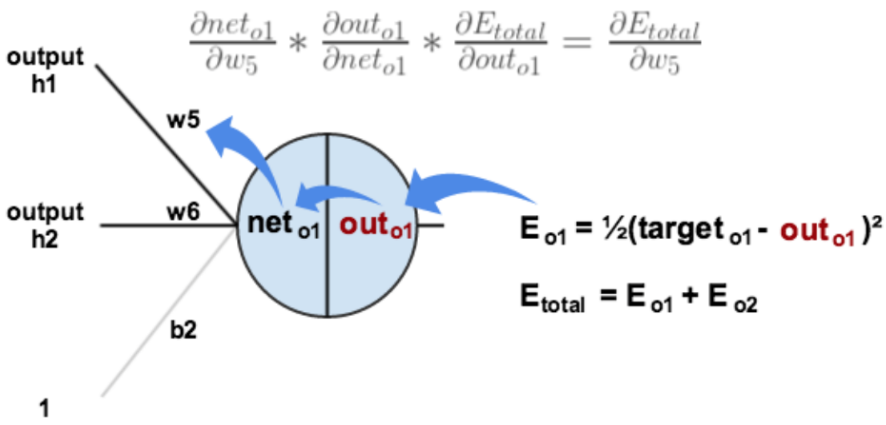
$$E_{total} = E_{o_1} + E_{o_2} = 0.2984$$

以权重参数 w_5 为例，如果我们想知道 w_5 对整体误差产生了多少影响，可以用整体误差对 w_5 求偏导求出：（链式法则）

generated by haroopad

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

下面的图可以更直观的看清楚误差是怎样反向传播的：



现在我们来分别计算每个式子的值：

计算：$\frac{\partial E_{total}}{\partial out_{o1}}$:

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

$$\frac{\partial E_{total}}{\partial out_{o1}} = -2 * \frac{1}{2}\left(target_{o1} - out_{o1}\right) = -\left(target_{o1} - out_{o1}\right) = 0.7413$$

计算：$\frac{\partial out_{o1}}{\partial net_{o1}}$，这里利用了 $sigmoid$ 求导方法:

$$f'(z) = \left(\frac{1}{1+e^{-z}}\right)'$$

$$= \frac{e^{-z}}{\left(1+e^{-z}\right)^2}$$

$$= \frac{1+e^{-z}-1}{\left(1+e^{-z}\right)}$$

$$= \left(\frac{1}{1+e^{-z}}\right)\left(1-\frac{1}{1+e^{-z}}\right)$$

$$= f(z)(1-f(z))$$

上面的推导就是 $sigmoid$ 函数求导过程，下面，我们可以直接使用结论：

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

generated by haroopad

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.1868$$

计算：$\frac{\partial net_{o1}}{\partial w_5}$

$$net_{o1} = out_{h1} * w_5 + out_{h2} * 6 + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = out_{h1} = 0.5933$$

最后三者相乘：

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5} = 0.0822$$

这样我们就计算出整体误差 E_{total} 对 w_5 的偏导值。回过头来再看看上面的公式，我们发现：

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

为了表达方便，我们用 δ_{o1} 来表示输出层 net_{o1} 的误差，只表示刚刚输入的数据的误差：

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

$$\delta_{o1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1})$$

因此，整体误差 E_{total} 对 w_5 的偏导公式可以写成，梯度是带方向的：

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} * out_{h1}$$

最后我们来更新 w_5 的值，往梯度的负方向更新：

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5}$$

Step4: 隐含层 → 输入层

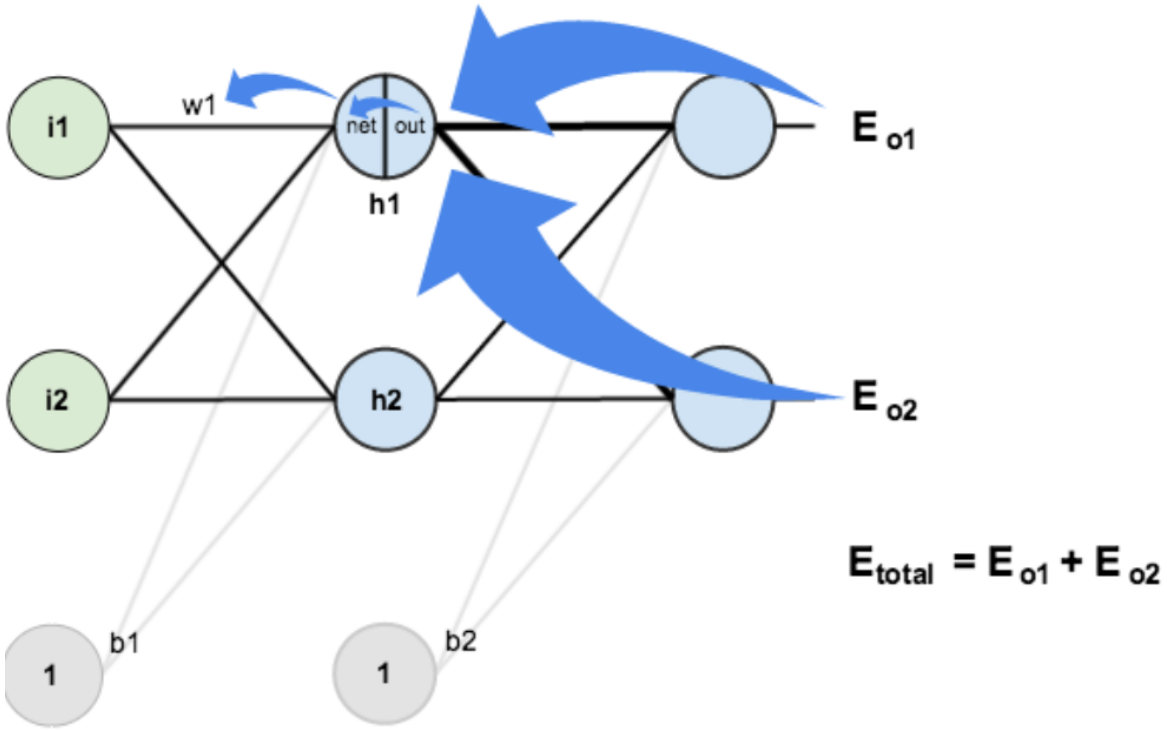
方法其实与上面说的差不多，但是有个地方需要变一下，在上文计算总误差对 w_5 的偏导时，是从 $out_{o1} \rightarrow net_{o1} \rightarrow w_5$,但是在隐含层之间的权值更新时，是$out_{h1} \rightarrow net_{h1} \rightarrow w_1$,而 out_{h1} 会接受 E_{o1}和 E_{o2}两个地方传来的误差，所以这个地方两个都要计算。

generated by haroopad

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



同样使用上面的公式，我们就可以求出每个权重 w，使用反向传播方法，可以有效的减少计算量。

generated by haroopad