详解反向传播算法(上)

目录：

声明：本文内容来自 Calculus on Computational Graphs: Backpropagation 。算是翻译加上自己理解。水平有限，理解错误欢迎指正。

反向传播算法(Backpropagation)已经是神经网络模型进行学习的标配。但是有很多问题值得思考一下：

反向传播算法的作用是什么？ 神经网络模型的学习算法一般是SGD。SGD需要用到损失函数C关于各个权重参数

的偏导数

。一个模型的参数w,b是非常多的，故而需要反向传播算法快速计算

。也就是说反向传播算法是一种计算偏导数的方法。

为什么要提出反向传播算法？ 在反向传播算法提出之前人们应该想到了使用SGD学习模型，也想到了一些办法求解网

络模型的偏导数，但这些算法求解效率比较低，所以提出反向传播算法来更高效的计算偏导数。（那时的网络模型还比较浅只有2-3层，参数少。估计即便不适用反向传播这种高效的算法也能很好的学习。一旦有人想使用更深的网络自然会遇到这个偏导数无法高效计算的问题，提出反向传播也就势在必行了）

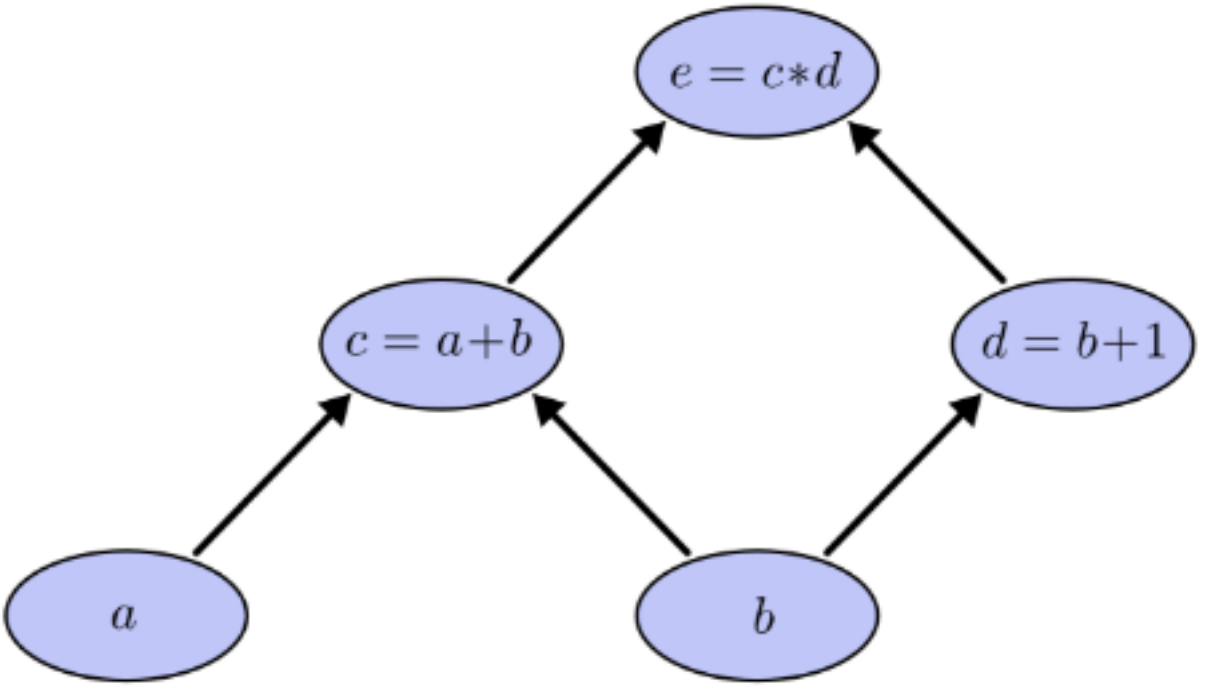
反向传播怎么样实现高效计算偏导数的？ 请先回顾一下当初我们学习微积分时是如何计算偏导数的？ （链式法则，具体看下面）

1 用计算图来解释几种求导方法：

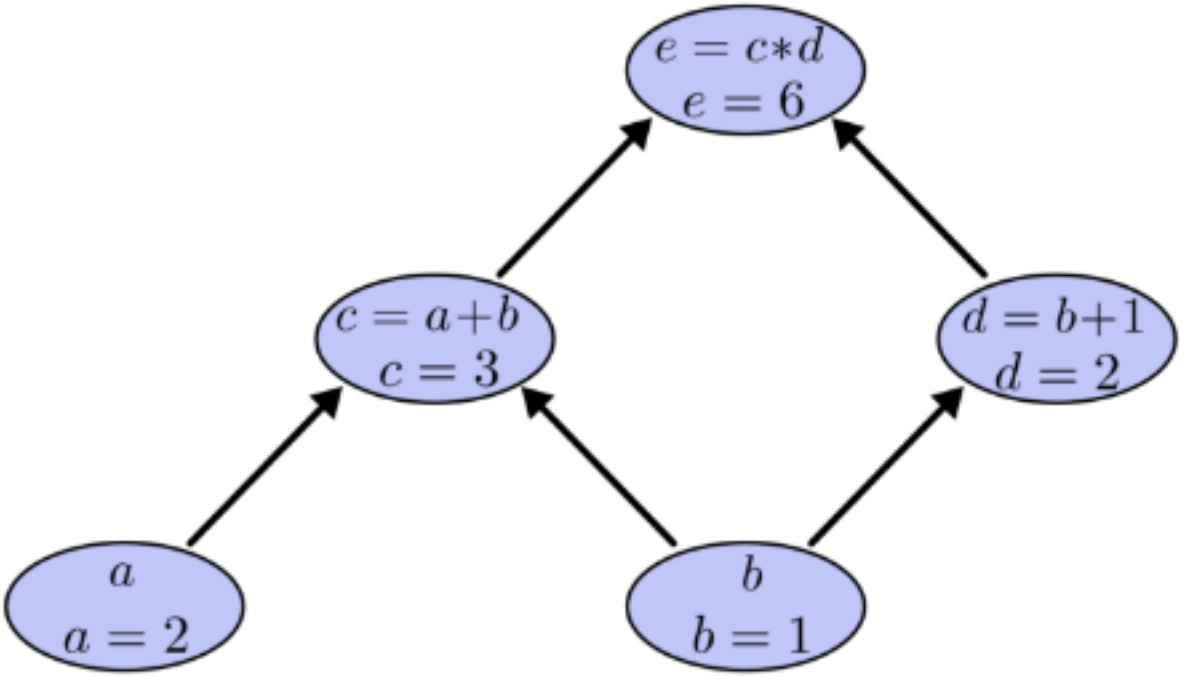
1.1 计算图

式子

可以用如下计算图表达：



令a=2,b=1则有：



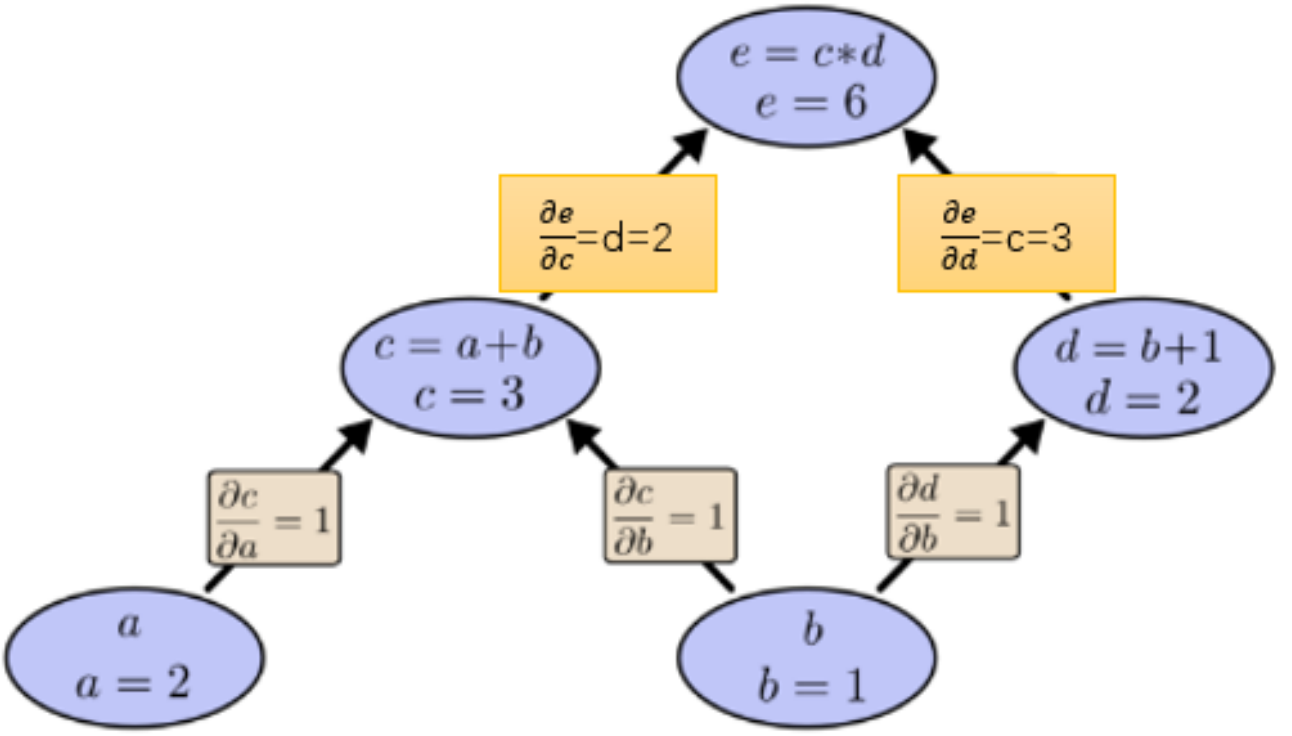
如何在计算图上表达“求导”呢？ 导数的含义是 因变量随自变量的变化率，例如

表示当x变化1个单位，y会变化3个单位。 微积分中已经学过：加法求导法则是

乘法求导法则是

。 我们在计算图的边上表示导数或偏导数：

如下图



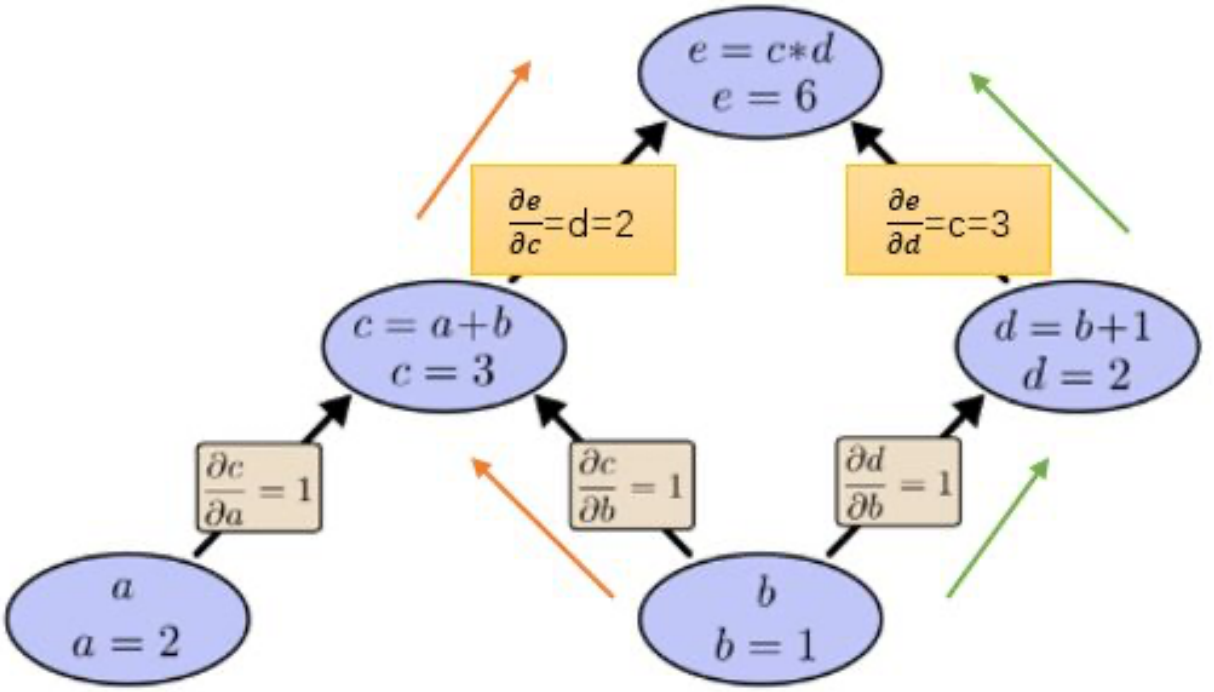
那么

如何求呢?

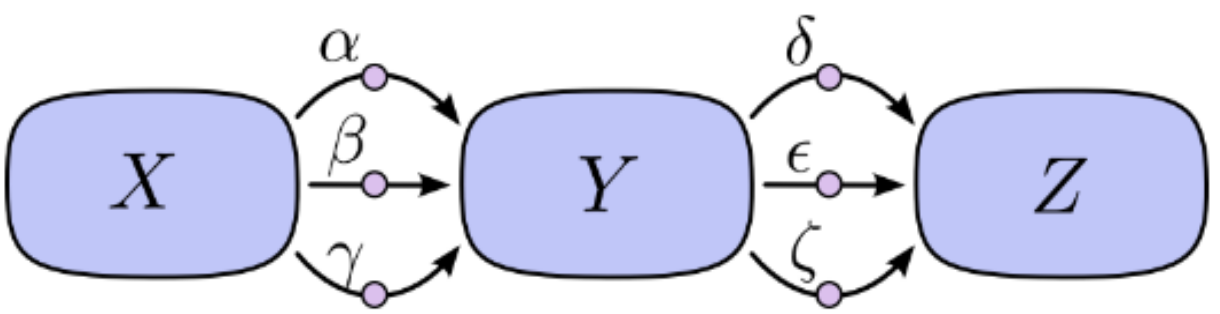
告诉我们1个单位的b变化会引起1个单位的c变换,

告诉我们 1 个单位的c变化会引起2个单位的e变化。所以

吗? 答案必然是错误。因为这样做只考虑到了下图橙色的路径,所有的路径都要考虑:



所以上面的求导方法总结为一句话就是： 路径上所有边相乘，所有路径相加。不过这里需要补充一条很有用的合并策略：

> 例如：下面的计算图若要计算
>
> 就会有9条路径：
>
> 
>
>
> 如果计算图再复杂一些，层数再多一些，路径数量就会呈指数爆炸性增长。但是如果采用合并策略：
>
> 就不会出现这种问题。这种策略不是 对每一条路径都求

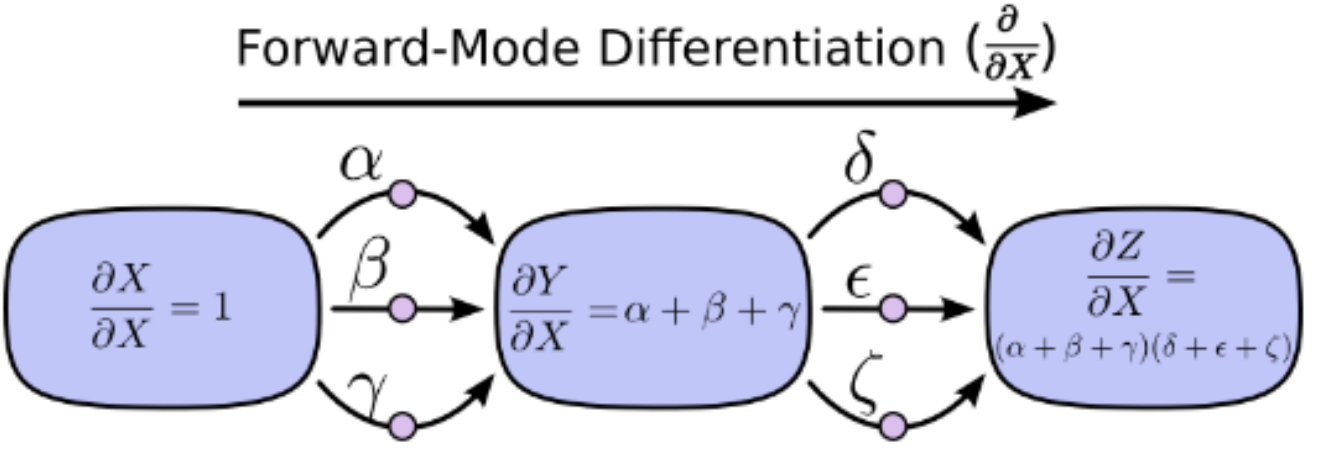
> 和，而是“合并同类路径”，“分阶段求解”。先求X对Y的总影响
>
> 再求Y对Z的总影响
>
> 最后综合在一起。

1.2 两种求导模式：前向模式求导(forward-mode differentiation) 反向模式求导(reverse-mode differentiation)

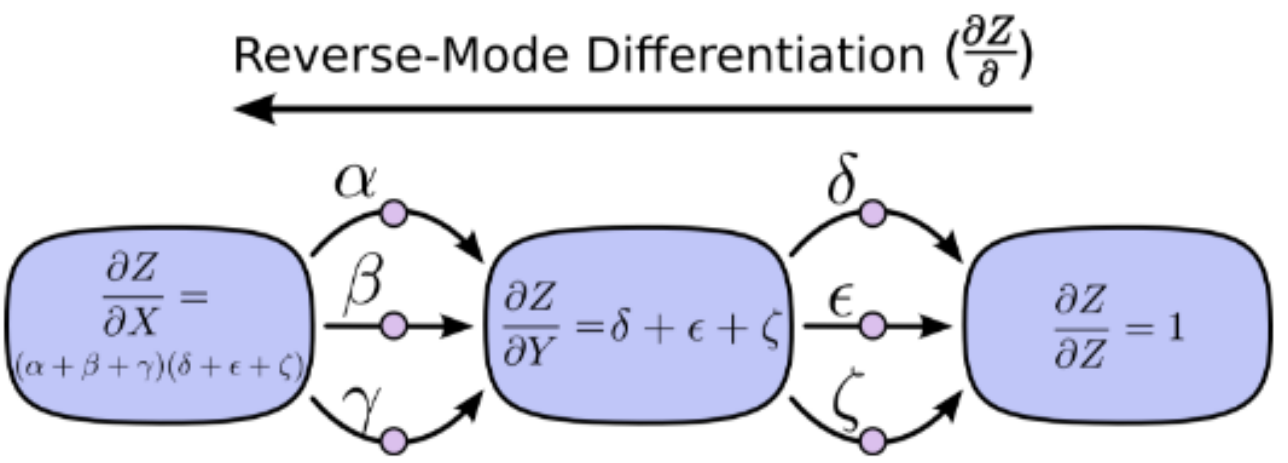
上面提到的求导方法都是前向模式求导(forward-mode differentiation)：从前向后。先求X对Y的总影响

再乘以Y对Z的总影响

。



另一种，反向模式求导(reverse-mode differentiation) 则是从后向前。先求Y对Z的影响再乘以X对Y的影响。



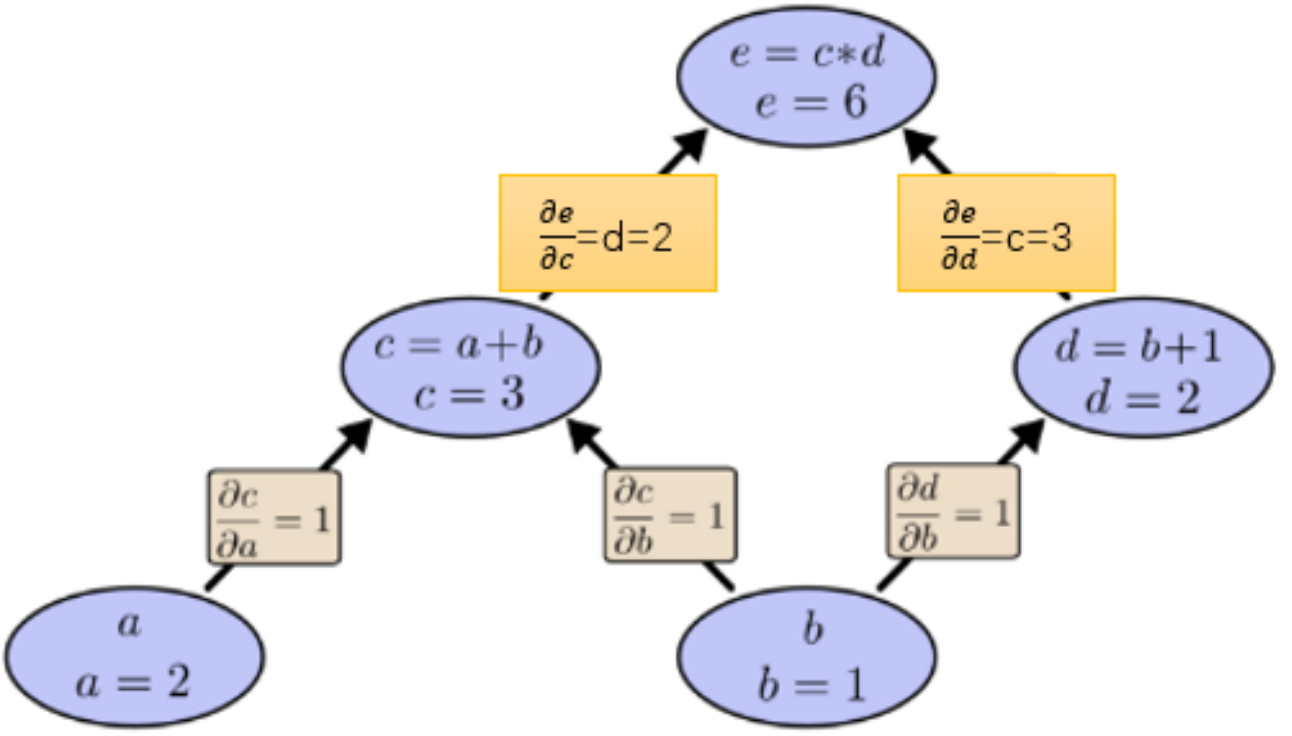
前向求导模式追踪一个输入如何影响每一个节点（对每一个节点进行

操作）反向求导模式追踪每一个节点如何影响一个输出（对每一个节点进行

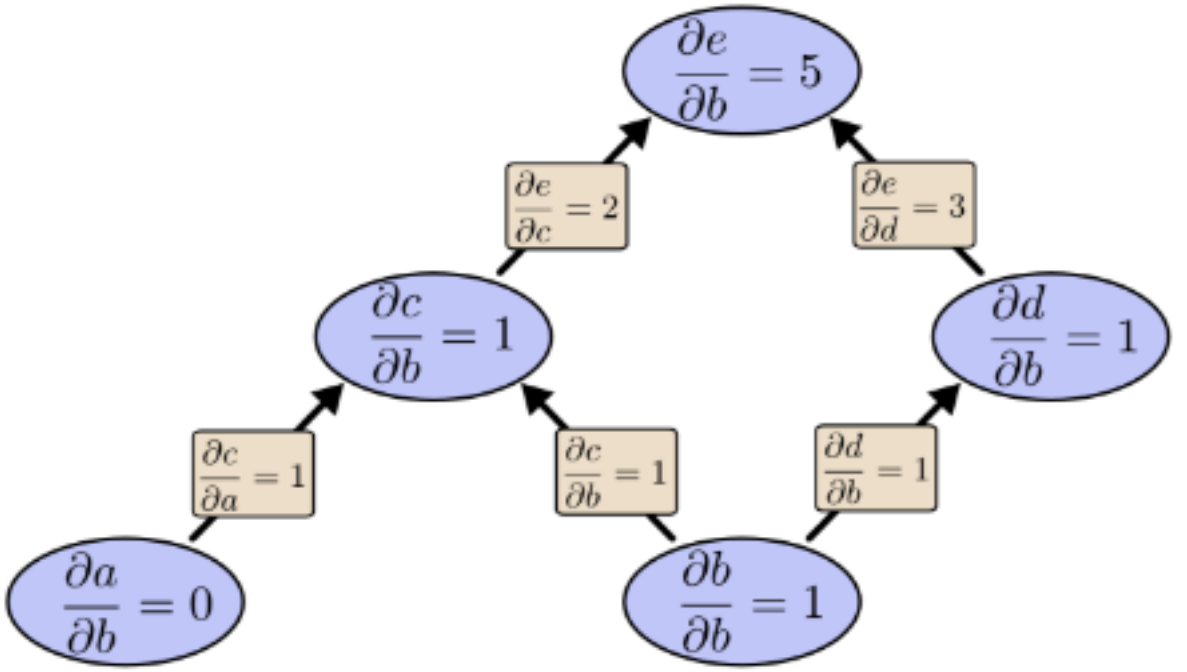
操作）。

1.3 反向求导模式（反向传播算法）的重要性：

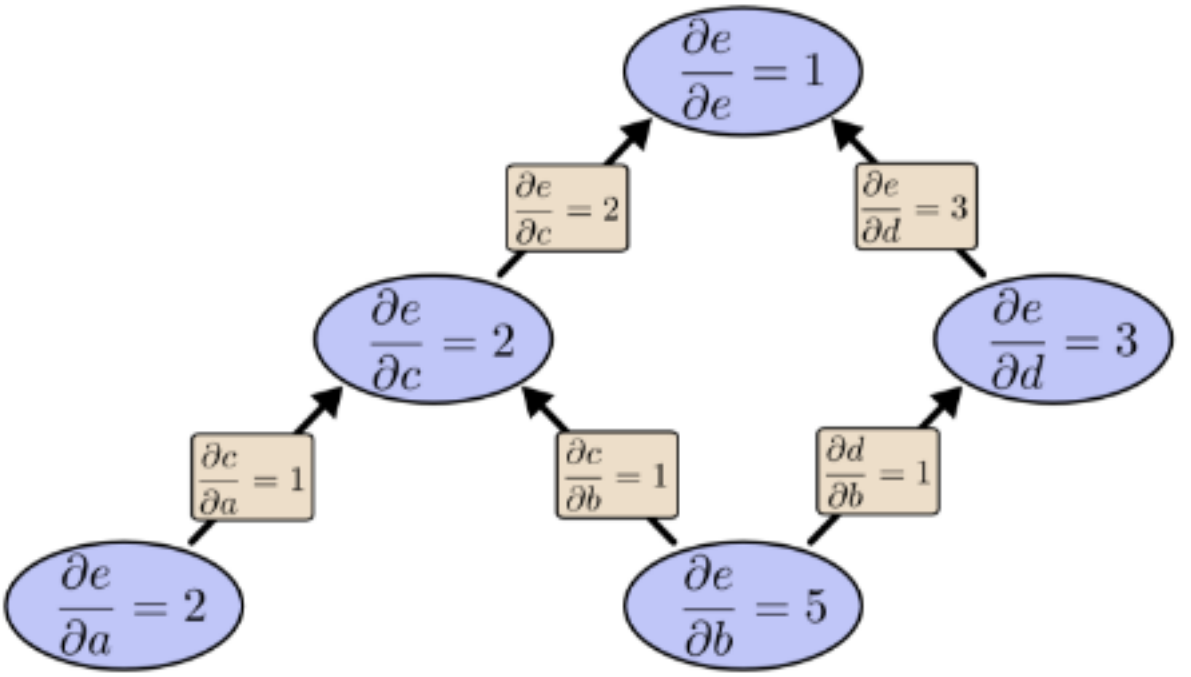
让我们再次考虑前面的例子：



如果用前向求导模式：关于b向前求导一次



如果用反向求导模式：向后求导



前向求导模式只得到了关于输入b的偏导

，还需要再次求解关于输入a的偏导

（运算2遍）。而反向求导一次运算就得到了e对两个输入a,b的偏导

（运算1遍）。上面的比较只看到了2倍的加速。但如果有1亿个输入1个输出，意味着前向求导需要操作1亿遍才得到所有关于输入的偏导，而反向求导则只需一次运算，1亿倍的加速。

当我们训练神经网络时，把“损失“看作”权重参数“的函数，需要计算”损失“关于每一个”权重参数“的偏导数（然后用梯度下降法学习）。神经网络的权重参数可以是百万甚至过亿级别。因此反向求导模式（反向传播算法）可以极大的加速学习。

参考：

- Calculus on Computational Graphs: Backpropagation

- Neural networks and deep learning