自动求导

by Xehau

数值求导

数值微分是用数值方法计算函数$f(x)$的导数。函数$f(x)$的点x的导数定义为：

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

要计算$f(x)$在点x的导数，可以对x加上一个很少的非零扰动，然后通过上述定义来直接计算函数$f(x)$的梯度。

数值微分方法非常容易实现，但是找到一个合适扰动非常难，如果扰动过小会引起数值计算问题，比如舍入误差；如果扰动过大，会增加截断误差，使得导数计算不准确，因此数值微分的实用性比较差，在实际应用中，常利用中心差分格式来减少截断误差：

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x}$$

舍入误差：是指数值计算中由于数字舍入造成的近似值和精确值之间的差异，比如用浮点数来表示实数。
截断误差：是指数学模型的理论解与数值计算问题的精确解之间的误差
数值求导计算量大，求解速度慢，但可以用来验证其他方法的正确性，通常取$e = 10^{-6}$。

符号求导

符号求导将输入的式子解释成符号表达式树（计算图），利用各种求导规则进行求导，输出的也是一个表达式。这种方式其实已经实现了自动求导的功能，但是也存在问题，对于比较复杂的函数，符号微分可以很容易产生指数量级的符号表达式，因此需要很长时间来求值，这种问题称为表达式膨胀（expression swell）；还有就是我们只关心最终的求到结果，保存中间的表达式结果显得无用且浪费。

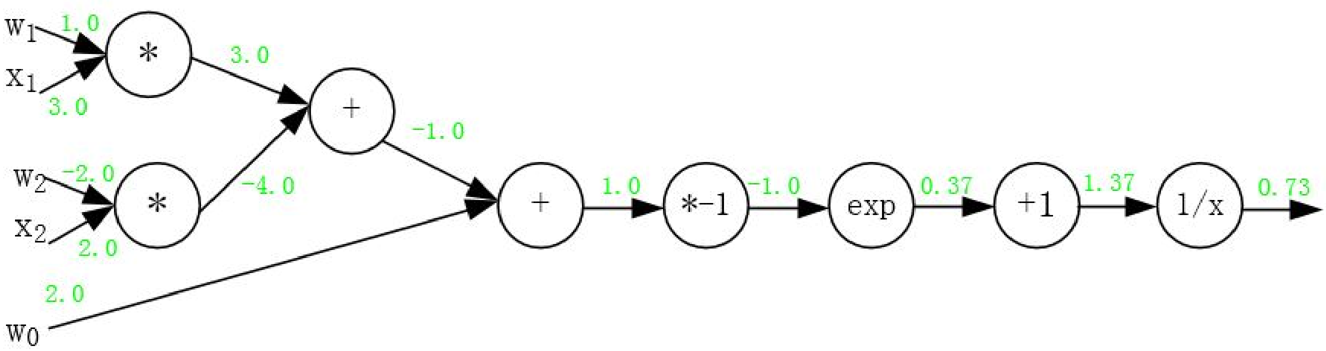
自动求导

当我们只需要得到微分结果一个准确的数值表示，而不关心它具体的代数形式时，理论上可以把中间子表达式的结果存在内存里，这样就可以极大化简计算过程。更进一步地，可以将微分操作和化

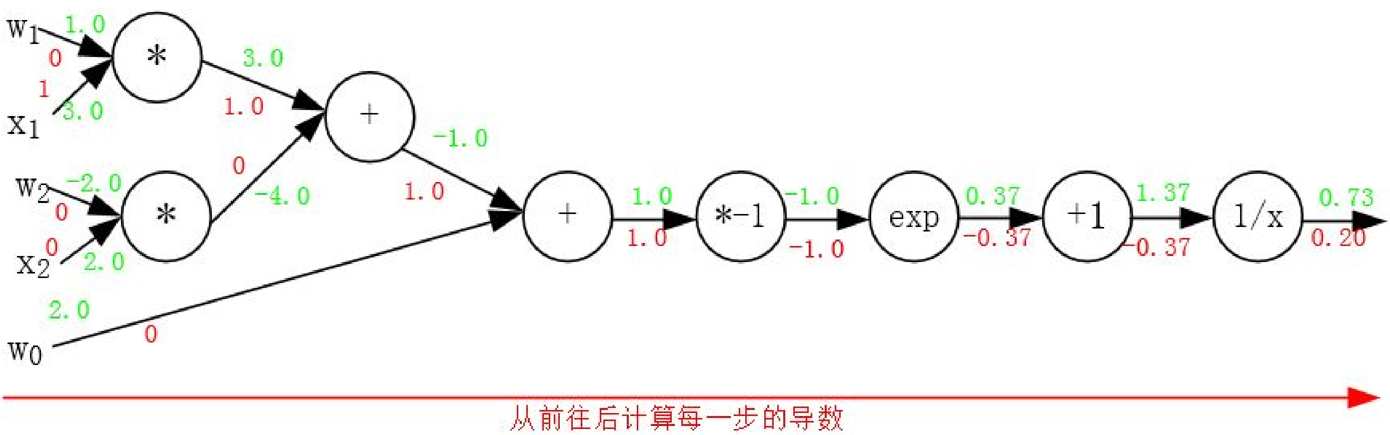
简操作交替进行，这样效率更高。这种交替操作的思想构成了自动微分的基础：在基本操作的层面上使用符号微分，保留中间的数值结果。自动微分可以分为两种实现思路：前向模式和后向模式。

1.前向模式（Forward Mode）

考虑函数$f=\frac{1}{1+e^{-(w_0+w_1x_1+w_2x_2)}}$，可以生成如下的计算图，给定输入，可以向前计算出每一步的值。



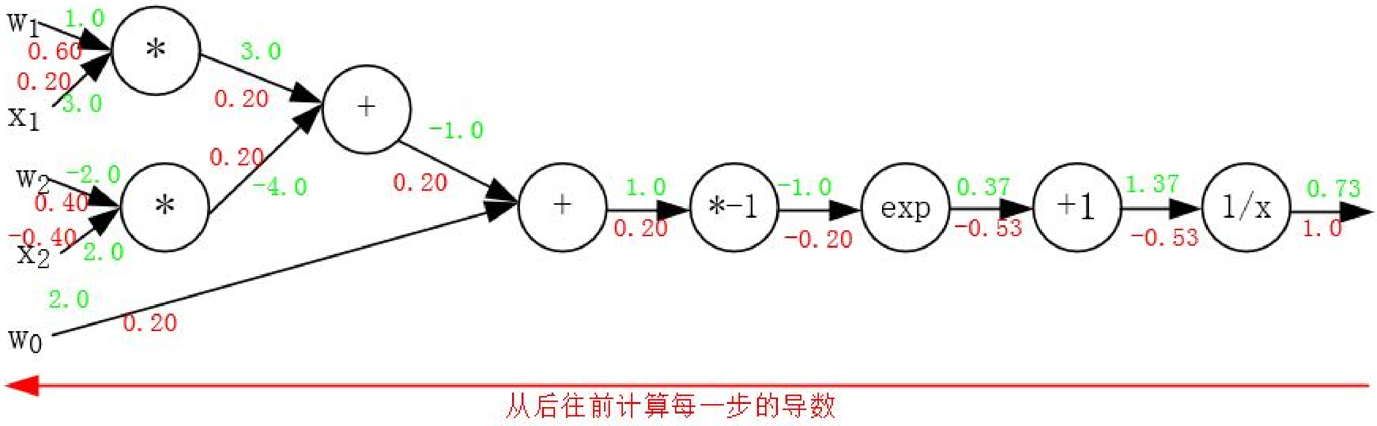
现利用前向模式求解f'_{x1}，前向模很简单，就是从头到尾逐一计算所有中间表达式对x_1的导数值，因此针对上述的表达式，可以计算出每一步的结果。



很显然，这种模式首先计算最开始的输入变量的倒数，然后根据计算图一步步计算中间变量的导数。相比较于符号求导，这种方法就是将中间求导表达式的值直接计算出来保存，从而避免了表达式膨胀。但它的缺点也很明显，当输入向量维度n过大（输入参数很多），求解每一个参数的偏导数的计算就非常的慢，因为要迭代n次分别计算。

2.后向模式（Backward Mode）

后向模式就能够解决前向模式计算慢的缺点，和前向模式相反，后向模式计算导数时采取从后往前逐步计算的思路。具体地，针对上述例子，计算步骤如下：



后向模式的最大优势是一次后向模式就可以得到全部梯度。由于机器学习问题大部分都是要求标量目标函数对高维参数的梯度，因此在这样的场景下后向模式的效率比前向模式的效率要高很多。

对于后向模式，它的问题在于存储空间上，由于反向传播求导过程中会用到前向计算的表达式结果，所以这些中间结果必须要保存起来，所需存储空间与函数操作数的数量成正比，而如何减少存储空间的利用也是学界研究的一个方向。

对于目前主流的深度学习框架，其每个op（op指的是最小的计算单元，caffe里叫layer）都预先定义好了forward和backward（或者叫grad）两个函数，这里的backward也就是求导。也就是说每个op的求导都是预先定义好的，或者说是人手推的。当定义好了一个神经网络，深度学习框架将其解释为一个DAG（有向无环图），DAG里每个节点就是op，从loss function这个节点开始，通过链式法则一步一步从后往前计算每一层神经网络的梯度，整个DAG梯度计算的最小粒度就是op的backward函数。

参考资料

1.http://txshi-mt.com/2018/10/04/NMT-Tutorial-3b-Autodiff/
2.CSE599W: Backpropagation and Automatic Differentiation
3.https://www.zhihu.com/question/66200879