

Backpropagation


Gradient Descent

gradient descent 的使用方法，跟前面讲到的 linear Regression 或者 Logistic regression 是一模一样的，唯一的区别就在于当它用在 neural network 的时候，network parameters $\theta = w_1, w_2, \dots, b_1, b_2, \dots$ 里面可能会有将近 million 个参数

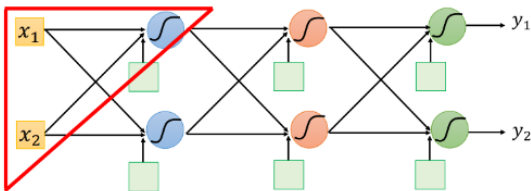
所以现在最大的困难是，如何有效地把这个近百万维的 vector 给计算出来，这就是 Backpropagation 要做的事情，所以 Backpropagation 并不是一个和 gradient descent 不同的 training 的方法，它就是 gradient descent，它只是一个比较有效率的算法，让你在计算这个 gradient 的 vector 的时候更有效率

Chain Rule

Backpropagation 里面并没有什么高深的数学，你唯一需要记得的就是 Chain Rule(链式法则)，对整个 neural network，我们定义了一个 loss function: $L(\theta) = \sum_{n=1}^N l^n(\theta)$ ，它等于所有 training data 的 loss 之和

Backpropagation 

$$L(\theta) = \sum_{n=1}^N l^n(\theta) \quad \Rightarrow \quad \frac{\partial L(\theta)}{\partial w} = \sum_{n=1}^N \frac{\partial l^n(\theta)}{\partial w}$$



我们把 training data 里任意一个样本点 x^n 代到 neural network 里面，它会 output 一个 y^n ，我们把这个 output 跟样本点本身的 label 标注的 target \hat{y}^n 作 cross entropy，这个交叉熵定义了 output \hat{y}^n 和 target \hat{y}^n 之间距离 $l^n(\theta)$ ，如果 cross entropy 比较大的话，说明 output 和 target 之间的距离很远，这个 network 的 parameter 的 loss 是比较大的，反之则说明这组 parameter 是比较好

然后 summation over 所有 training data 的 cross entropy $l^n(\theta)$ ，得到 total loss $L(\theta)$ ，这就是我们的 loss function，用这个 $L(\theta)$ 对某一个参数 w 做偏分，表达式如下：

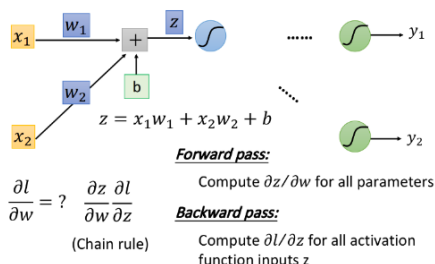
$$\frac{\partial L(\theta)}{\partial w} = \sum_{n=1}^N \frac{\partial l^n(\theta)}{\partial w}$$

这个表达式告诉我们，只需要考虑如何计算对某一笔 data 的 $\frac{\partial l^n(\theta)}{\partial w}$ ，再将所有 training data 的 cross entropy 对参数 w 的偏微分累计求和，就可以把 total loss 对某一个参数 w 的偏微分给计算出来，我们先考虑某一个 neuron，先拿出上图中被红色三角形圈住的 neuron，假设只有两个 input x_1, x_2 ，通过这个 neuron，我们先得到 $z = b + w_1x_1 + w_2x_2$ ，然后经过 activation function 从这个 neuron 中 output 出来，作为后续 neuron 的 input，再经过了非常非常多的事情以后，会得到最终的 output y_1, y_2 。现在的问题是：这样 $\frac{\partial l}{\partial w}$ 该怎么算？按照 chain rule，可以

把它拆分成两项， $\frac{\partial l}{\partial w} = \frac{\partial z}{\partial w} \cdot \frac{\partial l}{\partial z}$ ，这两项分别去把它计算出来。计算 $\frac{\partial z}{\partial w}$ 是比较简单的，而 $\frac{\partial l}{\partial z}$ 是比较复杂的

计算 $\frac{\partial z}{\partial w}$ 的 process，称之为 Forward pass，而计算后面这项 $\frac{\partial l}{\partial z}$ 的 process，我们称之为 Backward pass

Backpropagation



Forward pass

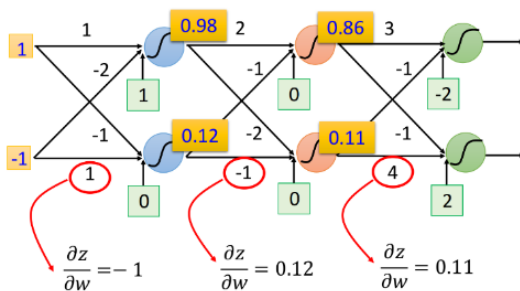
先考虑 $\frac{\partial z}{\partial w}$ 这一项，完全可以秒算出来， $\frac{\partial z}{\partial w_1} = x_1, \frac{\partial z}{\partial w_2} = x_2$

它的规律是这样的：求 $\frac{\partial z}{\partial w}$ ，就是看 w 前面连接的 input 是什么，那微分后的 $\frac{\partial z}{\partial w}$ 值就是什么，因此只要计算出 neuron network 里面每一个 neuron 的 output 就可以知道任意的 z 对 w 的偏微分

- 比如 input layer 作为 neuron 的输入时， w_1 前面连接的是 x_1 ，所以微分值就是 x_1 ； w_2 前面连接的是 x_2 ，所以微分值就是 x_2
- 比如 hidden layer 作为 neuron 的输入时，那该 neuron 的 input 就是前一层 neuron 的 output，于是 $\frac{\partial z}{\partial w}$ 的值就是前一层的 z 经过 activation function 之后输出的值(下图中的数据是假定 activation function 为 sigmoid function 得到的)

Backpropagation – Forward pass

Compute $\partial z / \partial w$ for all parameters



Backward pass

再考虑 $\frac{\partial l}{\partial z}$ 这一项，它是比较复杂的，这里我们依旧假设 activation function 是 sigmoid function

公式推导

我们的 z 通过 activation function 得到 a ，这个 neuron 的 output 是 $a = \sigma(z)$ ，接下来这个 a 会乘上某一个 weight w_3 ，再加上其它一大堆的 value 得到 z' ，它是下一个 neuron activation function 的 input，然后 a 又会乘上另一个 weight w_4 ，再加上其它一堆 value 得到 z'' ，后面还会发生很多很多其它事情，不过这里我们就只考虑下一步会发生什么事情：

$$\frac{\partial l}{\partial z} = \frac{\partial a}{\partial z} \frac{\partial l}{\partial a}$$

这里的 $\frac{\partial a}{\partial z}$ 实际上就是 activation function 的微分(在这里就是 sigmoid function 的微分)，接下来的问题是 $\frac{\partial l}{\partial a}$ 应该长什

么样子呢？ a 会影响 z' 和 z'' ，而 z' 和 z'' 会影响 l ，所以通过 chain rule 可以得到

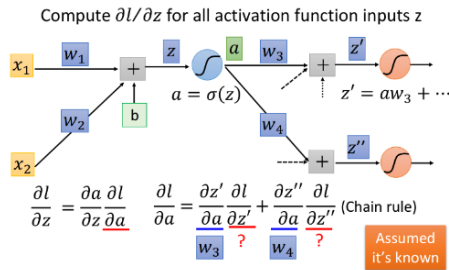
$$\frac{\partial l}{\partial a} = \frac{\partial z'}{\partial a} \frac{\partial l}{\partial z'} + \frac{\partial z''}{\partial a} \frac{\partial l}{\partial z''}$$

这里的 $\frac{\partial z'}{\partial a} = w_3$ ， $\frac{\partial z''}{\partial a} = w_4$ ，那 $\frac{\partial l}{\partial z'}$ 和 $\frac{\partial l}{\partial z''}$ 又该怎么算呢？这里先假设我们已经通过某种方法把 $\frac{\partial l}{\partial z'}$ 和 $\frac{\partial l}{\partial z''}$ 这两项给算出来

了，然后回过头去就可以把 $\frac{\partial l}{\partial z}$ 给轻易地算出来

$$\frac{\partial l}{\partial z} = \frac{\partial a}{\partial z} \frac{\partial l}{\partial a} = \sigma'(z) \left[w_3 \frac{\partial l}{\partial z'} + w_4 \frac{\partial l}{\partial z''} \right]$$

Backpropagation – Backward pass



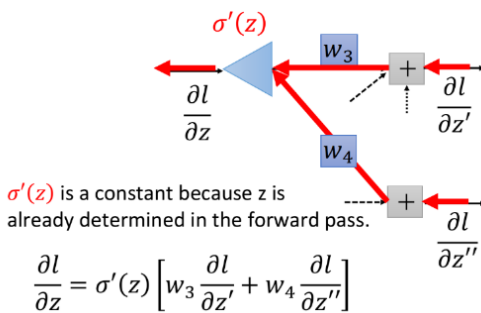
The other idea

这个式子还是蛮简单的，然后，我们可以从另一个观点来看待这个式子

你可以想象说，现在有另一个 neuron，它不在我们原来的 network 里面，在下图中它北华城三角形这个 neuron

的 input 就是 $\frac{\partial l}{\partial z'}$ 和 $\frac{\partial l}{\partial z''}$ ，那 input $\frac{\partial l}{\partial z'}$ 就乘上 w_3 ，input $\frac{\partial l}{\partial z''}$ 就乘上 w_4 ，它们连个相加再乘 activation function 的微分

$\sigma'(z)$ ，就可以得到 output $\frac{\partial l}{\partial z}$



这张图描述了一个新的 “neuron”，它的含义跟图下方的表达式是一模一样的，作这张图的目的是为了方面理解 **值得注意的是**，这里的 $\sigma'(z)$ 是一个 constant 常数，它并不是一个 function，因为 z 其实在计算 forward pass 的时候就已经被决定好了， z 是一个固定的值

所以这个 neuron 其实跟我们之前看到的 sigmoid function 是不一样的，它并不是把 input 通过一个 non-linear 进行转换，而是直接把 input 乘上一个 constant $\sigma'(z)$ ，就得到了 output，因此这个 neuron 被画成三角形，代表它跟我们之前看到的圆形的 neuron 的运作方式是不一样的，它是直接乘上一个 constant (这里的三角形有点像电路里的运算放大器 op-amp，它也是乘上一个 constant)

两种情况

现在我们最后需要解决的问题是，怎么计算 $\frac{\partial l}{\partial z'}$ 和 $\frac{\partial l}{\partial z''}$ 这两项，假设有两种不同的 case:

Case1: Output layer

假设蓝色的这个 neuron 已经是 hidden layer 的最后一层了，也就是说连接在 z' 和 z'' 后的这两个红色的 neuron 已经是 output layer，它的 output 就已经是整个 network 的 output 了，这个时候计算就比较简单

$$\frac{\partial l}{\partial z'} = \frac{\partial y_1}{\partial z'} \frac{\partial l}{\partial y_1}$$

其中 $\frac{\partial y_1}{\partial z'}$ 就是 output layer 的 activation function (softmax) 对 z' 的偏微分

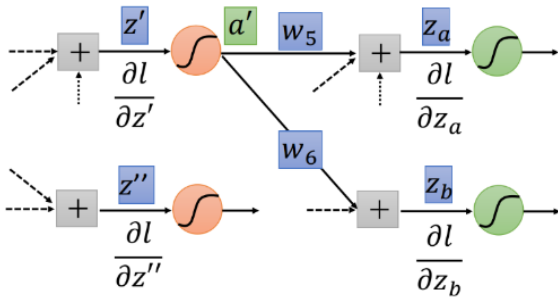
而 $\frac{\partial l}{\partial y_1}$ 就是 loss 对 y_1 的偏微分，它取决于你的 loss function 是怎么定义的，也就是你的 output 和 target 之间是怎么

么 evaluate 的，你可以用 cross entropy，也可以用 mean square error，用不同的定义， $\frac{\partial l}{\partial y_1}$ 的值就不一样

这个时候，你就可以把 l 对 w_1 和 w_2 的偏微分 $\frac{\partial l}{\partial w_1}$ 、 $\frac{\partial l}{\partial w_2}$ 算出来了

Case2: Not Output layer

假设现在红色的 neuron 并不是整个 network 的 output，那 z' 经过红色 neuron 的 activation function 得到 a' ，然后 output a' 和 w_5 、 w_6 相乘并加上一堆其他东西分别得到 z_a 和 z_b ，如下图所示



根据之前的推导证明类比，如果知道 $\frac{\partial l}{\partial z_a}$ 和 $\frac{\partial l}{\partial z_b}$ ，我们就可以计算 $\frac{\partial l}{\partial z'}$ ，如下图所示，借助运算放大器的辅助理解，将

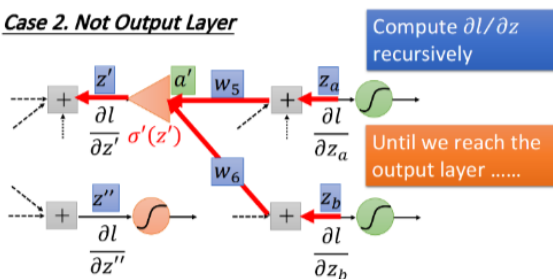
$\frac{\partial l}{\partial z_a}$ 乘上 w_5 和 $\frac{\partial l}{\partial z_b}$ 乘上 w_6 的值加起来再通过 op-amp，乘上放大系数 $\sigma'(z')$ ，就可以得到 output $\frac{\partial l}{\partial z'}$

$$\frac{\partial l}{\partial z'} = \sigma'(z') \left[w_5 \frac{\partial l}{\partial z_a} + w_6 \frac{\partial l}{\partial z_b} \right]$$

Backpropagation – Backward pass

Compute $\partial l / \partial z$ for all activation function inputs z

Case 2. Not Output Layer



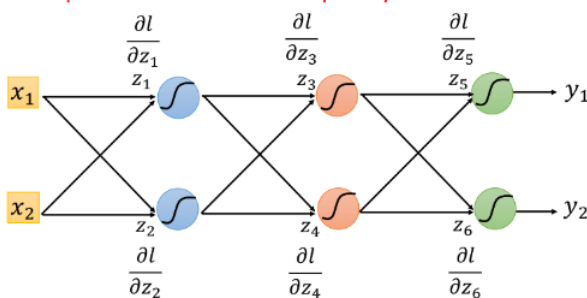
知道 z' 和 z'' 就可以知道 z ，知道 z_a 和 z_b 就可以知道 z' ，.....，现在这个过程就可以反复进行下去，直到找到 output layer，我们可以算出确切的值，然后在一层一层反推回去

Example

假设现在有 6 个 neuron，每一个 neuron 的 activation function 的 input 分别是 $z_1, z_2, z_3, z_4, z_5, z_6$ ，我们要计算 l 对这些 z 的偏微分，按照原来的思路，我们想要知道 z_1 的偏微分，就要去算 z_3 和 z_4 的偏微分，想知道 z_3 和 z_4 的偏微分，就又要计算两遍 z_5 和 z_6 的偏微分，因此如果我们是 z_1, z_2 的偏微分开始算，那就没有效率

Compute $\partial l / \partial z$ for all activation function inputs z

Compute $\partial l / \partial z$ from the output layer



这里每一个 op-amp 的放大系数就是 $\sigma'(z_1), \sigma'(z_2), \sigma'(z_3), \sigma'(z_4)$ ，所以整个流程就是，先快速地计算出 $\frac{\partial l}{\partial z_5}$ 和 $\frac{\partial l}{\partial z_6}$ ，

然后再把这个偏微分的值乘上路径上的 weight 汇集到 neuron 上面，再通过 op-amp 的放大，就可以得到 $\frac{\partial l}{\partial z_3}$ 和 $\frac{\partial l}{\partial z_4}$

这两个偏微分的值，在让它们乘上一些 weight，并且通过一个 op-amp，就得到 $\frac{\partial l}{\partial z_1}$ 和 $\frac{\partial l}{\partial z_2}$ 这两个偏微分的值，这样就

计算完了，这个步骤，就叫做 Backward pass

在做 Backward pass 的时候，实际上的做法就是建另外一个 neural network，本来正向 neural network 里面的 activation function 都是 sigmoid function，而现在计算 Backward pass 的时候，就是建一个反向的 neural network，它的 activation function 就是一个运算放大器 op-amp，每一个反向 neuron 的 input 是 loss l 对后面一层

layer 的 z 偏微分 $\frac{\partial l}{\partial z}$, output 则是 loss l 对这个 neuron 的 z 的偏微分 $\frac{\partial l}{\partial z}$, 做 Backward pass 就是通过这样一个反向

neural network 的运算, 把 loss l 对每一个 neuron 的 z 的偏微分 $\frac{\partial l}{\partial z}$ 都算出来

注:如果是正向做 Backward pass 的话, 实际上每次计算一个 $\frac{\partial l}{\partial z}$, 就需要把该 neuron 后面所有的 $\frac{\partial l}{\partial z}$ 都给计算一

遍, 会造成很多不必要的重复计算, 如果写成 code 的形式, 就相当于调用了很多次重复的函数; 而如果是反向做 Backward pass, 实际上就是把这些调用函数的过程都变成调用“值”的过程, 因此可以直接计算出结果, 而不需要占用过多的堆栈空间

总结

最后, 我们来总结一下 Backpropagation 是怎么做的

Forward pass, 每个 neuron 的 activation function 的 output, 就是他所连接的 weight 的 $\frac{\partial z}{\partial w}$

Backward pass, 建一个与原来方向相反的 neural network, 它的三角形 neuron 的 output 就是 $\frac{\partial l}{\partial z}$

把通过 forward pass 得到的 $\frac{\partial z}{\partial w}$ 和通过 backward pass 得到的 $\frac{\partial l}{\partial z}$ 乘起来就可以得到 l 对 w 的偏微分 $\frac{\partial l}{\partial w}$

$$\frac{\partial l}{\partial w} = \frac{\partial z}{\partial w} \Big|_{\text{forward pass}} \frac{\partial l}{\partial z} \Big|_{\text{backward pass}}$$

Backpropagation – Summary

