反向传播

神经网络之所以发展迅速，就是因为它的优化速度极快。这源于它的神经元拓扑结构。下面我们就讲解一下：

基本原理

我们回顾神经网络的计算（为了方便表达，我们把上文中的$W^\top$直接记为W）：

$$\begin{aligned} Z_1 &= X \cdot W_1 \\ A_1 &= f(Z_1) \\ Z_2 &= A_1 \cdot W_2 \\ A_2 &= f(Z_2) \\ Z_3 &= A_2 \cdot W_3 \\ A_3 &= f(Z_3) \end{aligned}$$

当然，网络的层数可以是任意的，但是我们就以3层为例。也就是说，A_3就是输出层的输出，我们也可以叫它$\hat{Y}$，表示对于Y的估计值。同时，我们也可以叫X为A_0，以达到形式上的统一。

最后假设我们定义loss function为

$$\mathcal{L}(\hat{Y}, Y) = \frac{1}{2}\|\hat{Y} - Y\|_2^2$$

在训练神经网络的过程中，我们的目标就是通过改变W的值来让$\hat{Y}$和Y尽可能接近，也就是让损失函数$\mathcal{L}$尽可能的小。

要通过W减小$\mathcal{L}$，我们就用到梯度下降。也就是说，我们需要计算$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}W}$。

得益于神经网络的拓扑结构（层状结构），我们可以轻而易举的利用链式法则计算梯度，比如

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}W_3} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}A_3}\frac{\mathrm{d}A_3}{\mathrm{d}Z_3}\frac{\mathrm{d}Z_3}{\mathrm{d}W_3}$$

或者比如说

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}W_1} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}A_3}\frac{\mathrm{d}A_3}{\mathrm{d}Z_3}\frac{\mathrm{d}Z_3}{\mathrm{d}A_2}\frac{\mathrm{d}A_2}{\mathrm{d}Z_2}\frac{\mathrm{d}Z_2}{\mathrm{d}A_1}\frac{\mathrm{d}A_1}{\mathrm{d}Z_1}\frac{\mathrm{d}Z_1}{\mathrm{d}W_1}$$

由于每一部分都很简单，我们可以轻而易举的推导出梯度。（注意，为了方便读者理解，这里的数学表达并不严谨。严谨的推导在最后）

简单推导

这里的推导只是展示反向传播的基本意思，数学表达上并不严谨。具体的推导在最后，可以选择忽略。

从上面的公式我们可以看出，这个梯度主要分成3部分：

- 从损失函数传递到输出层：$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}A_L}$
- 层内传播：$\frac{\mathrm{d}A_l}{\mathrm{d}A_{l-1}} = \frac{\mathrm{d}A_l}{\mathrm{d}Z_l}\frac{\mathrm{d}Z_l}{\mathrm{d}A_{l-1}}$
- 从层到权重：$\frac{\mathrm{d}Z_l}{\mathrm{d}W_l}$

当网络定义好了之后，每一部分都很简单。下面我们就推导它们。

损失函数到输出层

当我们定义

$$\mathcal{L}(A_L, Y) = \frac{1}{2}\|A_L - Y\|_2^2$$

之后，我们就可以得到：

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}A_L} = A_L - Y$$

这里的Y显然是已知的，而A_L的值其实在计算过程中也是已已经算出来了的。

当然，只要定义的损失函数可导，我们都可以计算梯度，例如我们现在用cross entropy当作损失函数来对来对分类任务进行训练：

$$\mathcal{L} = -Y \log A_L - (\mathbf{1} - Y)\log(\mathbf{1} - A_L)$$

先不考虑它的具体意义，只考虑数学推导，那么我们能得到

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}A_L} = -\frac{Y}{A_L} + \frac{1-Y}{1-A_L}$$

这里的除法指的是每个元素的计算，而不是"矩阵除法"。

层与层之间的传播

$$\frac{\mathrm{d}A_l}{\mathrm{d}A_{l-1}} = \frac{\mathrm{d}A_l}{\mathrm{d}Z_l}\frac{\mathrm{d}Z_l}{\mathrm{d}A_{l-1}}$$

对于第一部分，当我们定义了激活函数之后，就可以轻易得到，我们以sigmoid为例：

$$A_l = \frac{1}{1 + e^{-Z_l}}$$

那么

$$\begin{aligned} \frac{\mathrm{d}A_l}{\mathrm{d}Z_l} &= -\left(\frac{1}{e^{-Z_l}+1}\right)^2 e^{-Z_l}(-1) \\ &= \frac{e^{-Z_l}}{(e^{-Z_l}+1)^2} \\ &= \frac{e^{-Z_l}+1-1}{(e^{-Z_l}+1)^2} \\ &= \frac{e^{-Z_l}+1}{(e^{-Z_l}+1)^2} - \frac{1}{(e^{-Z_l}+1)^2} \\ &= \frac{1}{e^{-Z_l}+1} - \frac{1}{(e^{-Z_l}+1)^2} \\ &= A_l - A_l^2 \\ &= A_l(\mathbf{1} - A_l) \end{aligned}$$

注意这里的乘法也是元素相乘，而不是矩阵计算。我们一般把它记为

$$\frac{\mathrm{d}A_l}{\mathrm{d}Z_l} = A_l \circ (\mathbf{1} - A_l)$$

对于第二部分，因为

$$Z_l = A_{l-1} \cdot W_l$$

我们可以轻松的得到

$$\frac{\mathrm{d}Z_l}{\mathrm{d}A_{l-1}} = W_l$$

那么从第l层的输出A_l到第$l-1$层的输出A_{l-1}就是

$$\frac{\mathrm{d}A_l}{\mathrm{d}A_{l-1}} = \frac{\mathrm{d}A_l}{\mathrm{d}Z_l}\frac{\mathrm{d}Z_l}{\mathrm{d}A_{l-1}} = A_l \circ (\mathbf{1} - A_l) \circ W_l$$

这里的A_{l-1}也会在正向传播的时候算出来，而W_l则是模型当前的参数值。这两个都是已知的。

注意这里都是元素相乘，详细推导在后面，有兴趣可以看。

从层到权重的传递

由于

$$Z_l = A_{l-1} \cdot W_l$$

所以

$$\frac{\mathrm{d}Z_l}{\mathrm{d}W_l} = A_{l-1}$$

这里的A_{l-1}也会在正向传播的时候算出来，也是已知量。

总结

这一章节我们推导了梯度从损失函数传递到权重的过程。可以看出，当模型一旦确定，这些梯度的推导公式就确定了。唯一要做的就是正向传播一次，计算出每一层的Z_l和A_l，然后再借助目标数据Y和模型当前的参数值W_l，就可以通过简单的矩阵乘法得到W_l的梯度。这是一种非常高效的方法，因此也使得优化上万千参数成为可能。

详细推导

这一部分有兴趣的读者可以看。

下面的推导均建立在sigmoid作为激活函数，cross entropy作为损失函数的网络。但是正如上面所讲的，无论怎么更改函数，只要把更改部分的梯度表达式推导出来，就可以进行局部替换。但是整体的反向传播过程不变。

关于偏差b，它往往和W合并在一起，然后把X扩展一列1出来。在下面的推导中，我们把它分开写。如果想要改成b和W合起来的形式，只需要忽略与b有关的内容即可。

下面我们从1个神经元，1个输入数据，2个分类的最简单情况，逐步扩展到I个神经元，E个数据，K个分类的通用情况。

1神经元，1数据，2分类

这种情况下，所有的值都是单个的实数标量。 正向传播：

$a_1 = x$	$\mathbb{R}$
$z_l = w_l a_{l-1} + b_l$	$\mathbb{R}$
$a_l = \sigma(z_l)$	$\mathbb{R}$
$z_L = w_L a_{L-1} + b_L$	$\mathbb{R}$
$a_L = \sigma(z_L)$	$\mathbb{R}$
$\mathcal{L} = -y \log a_L - (1-y)\log(1-a_L)$	$\mathbb{R}$

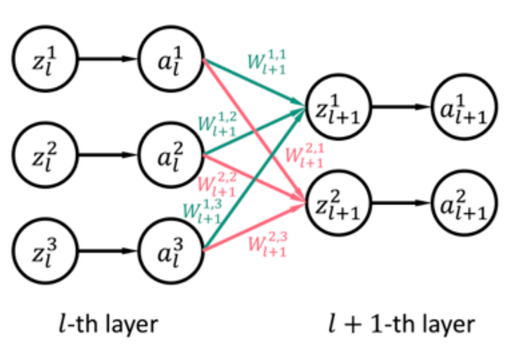
反向传播

$\frac{\partial \mathcal{L}}{\partial a_L} = -\frac{y}{a_L} + \frac{1-y}{1-a_L}$	$\mathbb{R}$
$\frac{\partial a_L}{\partial z_L} = a_L(1-a_L)$	$\mathbb{R}$
$\frac{\partial \mathcal{L}}{\partial z_L} = \frac{\partial \mathcal{L}}{\partial a_L}\frac{\partial a_L}{\partial z_L} = a_L - y$	$\mathbb{R}$
$\frac{\partial z_{l+1}}{\partial z_l} = \frac{\partial z_{l+1}}{\partial a_l}\frac{\partial a_l}{\partial z_l} = w_{l+1} \cdot a_l \cdot (1-a_l)$	$\mathbb{R}$
$\frac{\partial \mathcal{L}}{\partial z_l} = \frac{\partial \mathcal{L}}{\partial z_{l+1}}\frac{\partial z_{l+1}}{\partial z_l}$	$\mathbb{R}$
$\frac{\partial z_l}{\partial w_l} = a_{l-1}$	$\mathbb{R}$
$\frac{\partial z_l}{\partial b_l} = 1$	$\mathbb{R}$
$\frac{\partial \mathcal{L}}{\partial w_{l+1}} = \frac{\partial \mathcal{L}}{\partial z_{l+1}}\frac{\partial z_{l+1}}{\partial w_{l+1}} = \frac{\partial \mathcal{L}}{\partial z_{l+1}} a_l$	$\mathbb{R}$
$\frac{\partial \mathcal{L}}{\partial b_{l+1}} = \frac{\partial \mathcal{L}}{\partial z_{l+1}}\frac{\partial z_{l+1}}{\partial b_{l+1}} = \frac{\partial \mathcal{L}}{\partial z_{l+1}}$	$\mathbb{R}$

用上述的公式就可以求出任意w_l和b_l的梯度了。

I神经元，1个数据，2个分类

这里需要一个示意图来帮助推导：



层与层之间的传递：

$$\begin{aligned} \frac{\partial z_{l+1}}{\partial z_l} &= \begin{bmatrix} \frac{\partial z_{l+1}^1}{\partial z_l^1} & \frac{\partial z_{l+1}^1}{\partial z_l^2} & \frac{\partial z_{l+1}^1}{\partial z_l^3} \\ \frac{\partial z_{l+1}^2}{\partial z_l^1} & \frac{\partial z_{l+1}^2}{\partial z_l^2} & \frac{\partial z_{l+1}^2}{\partial z_l^3} \end{bmatrix} = \begin{bmatrix} \frac{\partial z_{l+1}^1}{\partial a_l^1}\frac{\partial a_l^1}{\partial z_l^1} & \frac{\partial z_{l+1}^1}{\partial a_l^2}\frac{\partial a_l^2}{\partial z_l^2} & \frac{\partial z_{l+1}^2}{\partial a_l^2}\frac{\partial a_l^3}{\partial z_l^3} \\ \frac{\partial z_{l+1}^2}{\partial a_l^1}\frac{\partial a_l^1}{\partial z_l^1} & \frac{\partial z_{l+1}^2}{\partial a_l^2}\frac{\partial a_l^2}{\partial z_l^2} & \frac{\partial z_{l+1}^1}{\partial a_l^3}\frac{\partial a_l^3}{\partial z_l^3} \end{bmatrix} \in \mathbb{R}^{n_{l+1} \times n_l} \tag{1} \\ &= \begin{bmatrix} W_{l+1}^{1,1}\frac{\partial a_l^1}{\partial z_l^1} & W_{l+1}^{1,2}\frac{\partial a_l^2}{\partial z_l^2} & W_{l+1}^{1,3}\frac{\partial a_l^3}{\partial z_l^3} \\ W_{l+1}^{2,1}\frac{\partial a_l^1}{\partial z_l^1} & W_{l+1}^{2,2}\frac{\partial a_l^2}{\partial z_l^2} & W_{l+1}^{2,3}\frac{\partial a_l^3}{\partial z_l^3} \end{bmatrix} = \begin{bmatrix} W_{l+1}^{1,1} & W_{l+1}^{1,2} & W_{l+1}^{1,3} \\ W_{l+1}^{2,1} & W_{l+1}^{2,2} & W_{l+1}^{2,3} \end{bmatrix} \circ \begin{bmatrix} \frac{\partial a_l^1}{\partial z_l^1} & \frac{\partial a_l^2}{\partial z_l^2} & \frac{\partial a_l^3}{\partial z_l^3} \end{bmatrix} \tag{2} \\ &= W_{l+1}^\top \circ \begin{bmatrix} a_l^1(1-a_l^1) & a_l^2(1-a_l^2) & a_l^3(1-a_l^3) \end{bmatrix} \tag{3} \\ &= W_{l+1}^\top \circ a_l \circ (1 - a_l) \tag{4} \end{aligned}$$

这里又牵扯到2个额外的知识。第一个就是“圈乘”（○），学名叫舒尔积或者阿达玛积，指的是两个维度一样的矩阵的每个元素对应相乘。

第二个点就是broad casting，也就是说，当两个变量运算时，如果维度缺失，一些编程语言会自动补充缺失维度。再缺失的维度上会自动复制，比如：

$$\begin{bmatrix} W_{l+1}^{1,1} & W_{l+1}^{1,2} & W_{l+1}^{1,3} \\ W_{l+1}^{2,1} & W_{l+1}^{2,2} & W_{l+1}^{2,3} \end{bmatrix} \circ \begin{bmatrix} \frac{\partial a_l^1}{\partial z_l^1} & \frac{\partial a_l^2}{\partial z_l^2} & \frac{\partial a_l^3}{\partial z_l^3} \end{bmatrix}$$

是一个3×2矩阵和3×1矩阵的舒尔积，那么第二个矩阵会被复制乘3×2，也就相当于：

$$\begin{bmatrix} W_{l+1}^{1,1} & W_{l+1}^{1,2} & W_{l+1}^{1,3} \\ W_{l+1}^{2,1} & W_{l+1}^{2,2} & W_{l+1}^{2,3} \end{bmatrix} \circ \begin{bmatrix} \frac{\partial a_l^1}{\partial z_l^1} & \frac{\partial a_l^2}{\partial z_l^2} & \frac{\partial a_l^3}{\partial z_l^3} \\ \frac{\partial a_l^1}{\partial z_l^1} & \frac{\partial a_l^2}{\partial z_l^2} & \frac{\partial a_l^3}{\partial z_l^3} \end{bmatrix}$$

这里有趣的一点在于出现了舒尔积的运算。相比于矩阵乘积，舒尔积的运算其实是稀疏的，因为只有对应元素相乘，也就是说一乘数矩阵的一个元素只会影响结果中的一个元素。之所以会这样，是因为神经网络的拓扑结构，例如图中的$W_{l+1}^{1,1}$，$W_{l+1}^{1,2}$，$W_{l+1}^{1,3}$元素只和z_{l+1}^1相关，而和z_{l+1}^2无关。这样，当矩阵

$$Z_{l+1} = \begin{bmatrix} z_{l+1}^1 \\ z_{l+1}^2 \end{bmatrix}$$

当Z_{l+1}对$W_{l+1}^{1,1}$，$W_{l+1}^{1,2}$，$W_{l+1}^{1,3}$求导时，就等价于只有z_{l+1}^1对它们求导，其他的都是0。

当Z_{l+1}对$W_{l+1}^{2,1}$，$W_{l+1}^{2,2}$，$W_{l+1}^{2,3}$求导时，就等价于只有z_{l+1}^2对它们求导，其他的都是0。

同理，A_l对Z_l求导时，相当于a_l^1对z_l^1单独求导，a_l^2对z_l^2单独求导，a_l^3对z_l^3单独求导，而不是A_l的每一个元素对Z_l中的每一个元素分别单独求导。（本质上应该是这样的，但是由于网络的结构，非对应元素的求导都为0）。

然后我们推导损失函数到输出层的公式

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial z_l} &= \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_l^1} & \frac{\partial \mathcal{L}}{\partial z_l^2} & \frac{\partial \mathcal{L}}{\partial z_l^3} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_{l+1}^1}\frac{\partial z_{l+1}^1}{\partial z_l^1} + \frac{\partial \mathcal{L}}{\partial z_{l+1}^2}\frac{\partial z_{l+1}^2}{\partial z_l^1} & \frac{\partial \mathcal{L}}{\partial z_{l+1}^1}\frac{\partial z_{l+1}^1}{\partial z_l^2} + \frac{\partial \mathcal{L}}{\partial z_{l+1}^2}\frac{\partial z_{l+1}^2}{\partial z_l^2} & \frac{\partial \mathcal{L}}{\partial z_{l+1}^1}\frac{\partial z_{l+1}^1}{\partial z_l^3} + \frac{\partial \mathcal{L}}{\partial z_{l+1}^2}\frac{\partial z_{l+1}^2}{\partial z_l^3} \end{bmatrix} \tag{5} \\ &= \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_{l+1}^1} & \frac{\partial \mathcal{L}}{\partial z_{l+1}^2} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial z_{l+1}^1}{\partial z_l^1} & \frac{\partial z_{l+1}^1}{\partial z_l^2} & \frac{\partial z_{l+1}^1}{\partial z_l^3} \\ \frac{\partial z_{l+1}^2}{\partial z_l^1} & \frac{\partial z_{l+1}^2}{\partial z_l^2} & \frac{\partial z_{l+1}^2}{\partial z_l^3} \end{bmatrix} \in \mathbb{R}^{1 \times n_l} \tag{6} \\ &= \frac{\partial \mathcal{L}}{\partial z_{l+1}} \cdot \frac{\partial z_{l+1}}{\partial z_l} \tag{7} \end{aligned}$$

我们再推导损失函数到参数的传播：

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial W_{l+1}} &= \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial W_{l+1}^{1,1}} & \frac{\partial \mathcal{L}}{\partial W_{l+1}^{2,2}} \\ \frac{\partial \mathcal{L}}{\partial W_{l+1}^{1,2}} & \frac{\partial \mathcal{L}}{\partial W_{l+1}^{2,1}} \\ \frac{\partial \mathcal{L}}{\partial W_{l+1}^{1,3}} & \frac{\partial \mathcal{L}}{\partial W_{l+1}^{2,3}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_{l+1}^1}\frac{\partial z_{l+1}^1}{\partial W_{l+1}^{1,1}} & \frac{\partial \mathcal{L}}{\partial z_{l+1}^2}\frac{\partial z_{l+1}^2}{\partial W_{l+1}^{2,1}} \\ \frac{\partial \mathcal{L}}{\partial z_{l+1}^1}\frac{\partial z_{l+1}^1}{\partial W_{l+1}^{1,2}} & \frac{\partial \mathcal{L}}{\partial z_{l+1}^2}\frac{\partial z_{l+1}^2}{\partial W_{l+1}^{2,2}} \\ \frac{\partial \mathcal{L}}{\partial z_{l+1}^1}\frac{\partial z_{l+1}^1}{\partial W_{l+1}^{1,3}} & \frac{\partial \mathcal{L}}{\partial z_{l+1}^2}\frac{\partial z_{l+1}^2}{\partial W_{l+1}^{2,3}} \end{bmatrix} \in \mathbb{R}^{n_l \times n_{l+1}} \tag{8} \\ &= \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_{l+1}^1}a_l^1 & \frac{\partial \mathcal{L}}{\partial z_{l+1}^2}a_l^1 \\ \frac{\partial \mathcal{L}}{\partial z_{l+1}^1}a_l^2 & \frac{\partial \mathcal{L}}{\partial z_{l+1}^2}a_l^2 \\ \frac{\partial \mathcal{L}}{\partial z_{l+1}^1}a_l^3 & \frac{\partial \mathcal{L}}{\partial z_{l+1}^2}a_l^3 \end{bmatrix} = \begin{bmatrix} a_l^1 \\ a_l^2 \\ a_l^3 \end{bmatrix} \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_{l+1}^1} & \frac{\partial \mathcal{L}}{\partial z_{l+1}^2} \end{bmatrix} \tag{9} \\ &= a_l^\top \frac{\partial \mathcal{L}}{\partial z_{l+1}} \tag{10} \\ \frac{\partial \mathcal{L}}{\partial b_{l+1}} &= \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial b_{l+1}^1} & \frac{\partial \mathcal{L}}{\partial b_{l+1}^2} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_{l+1}^1}\frac{\partial z_{l+1}^1}{\partial b_{l+1}^1} & \frac{\partial \mathcal{L}}{\partial z_{l+1}^2}\frac{\partial z_{l+1}^2}{\partial b_{l+1}^2} \end{bmatrix} = \frac{\partial \mathcal{L}}{\partial z_{l+1}} \in \mathbb{R}^{1 \times n_{l+1}} \tag{11} \end{aligned}$$

I神经元，1个数据，K个分类

对于K类的分类问题，y将是一个One-Hot encoding成的K维变量，比如当$K=4$时：

$$\begin{aligned} y &= [1, 0, 0, 0] \\ y &= [0, 1, 0, 0] \\ y &= [0, 0, 1, 0] \\ y &= [0, 0, 0, 1] \end{aligned}$$

分别代表第1类，第2类，第3类和第4类。

交叉熵（cross entropy）损失函数就是：

$$\mathcal{L} = \sum_k -y_k \log a_{Lk} - (1 - y_k)\log(1 - a_{Lk})$$

其中下标$k = 1, \ldots, K$表示y和a_L的第几个元素。 cross entropy意义是让$y_k = 0$的项对应的输出尽可能减小，让$y_k = 1$的项对应的输出尽可能增大。那么可以得到：

$$\begin{aligned} \mathcal{L} &= \sum_k -y_k \log a_{Lk} - (1 - y_k)\log(1 - a_{Lk}) \\ &= \sum_k -y_k \log a_{Lk} - \sum_k (1 - y_k)\log(1 - a_{Lk}) \\ &= [y_1, y_2, y_3, y_4] \cdot \begin{bmatrix} \log a_{L1} \\ \log a_{L2} \\ \log a_{L3} \\ \log a_{L4} \end{bmatrix} - [1 - y_1, 1 - y_2, 1 - y_3, 1 - y_4] \cdot \begin{bmatrix} \log(1 - a_{L1}) \\ \log(1 - a_{L2}) \\ \log(1 - a_{L3}) \\ \log(1 - a_{L4}) \end{bmatrix} \\ &= y \log a_L^\top - (1 - y)\log(1 - a_L)^\top \end{aligned}$$

因此，损失函数传递到输出层的公式就是

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial z_L} &= \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_L^1} & \frac{\partial \mathcal{L}}{\partial z_L^2} & \frac{\partial \mathcal{L}}{\partial z_L^3} & \frac{\partial \mathcal{L}}{\partial z_L^4} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial a_L^1}\frac{\partial a_L^1}{\partial z_L^1} & \frac{\partial \mathcal{L}}{\partial a_L^2}\frac{\partial a_L^2}{\partial z_L^2} & \frac{\partial \mathcal{L}}{\partial a_L^3}\frac{\partial a_L^3}{\partial z_L^3} & \frac{\partial \mathcal{L}}{\partial a_L^4}\frac{\partial a_L^4}{\partial z_L^4} \end{bmatrix} \tag{12} \\ &= \begin{bmatrix} \left(-\frac{y^1}{a_L^1} + \frac{1-y^1}{1-a_L^1}\right) a_L^1(1 - a_L^1) \\ \left(-\frac{y^2}{a_L^2} + \frac{1-y^2}{1-a_L^2}\right) a_L^2(1 - a_L^2) \\ \left(-\frac{y^3}{a_L^3} + \frac{1-y^3}{1-a_L^3}\right) a_L^3(1 - a_L^3) \\ \left(-\frac{y^4}{a_L^4} + \frac{1-y^4}{1-a_L^4}\right) a_L^4(1 - a_L^4) \end{bmatrix}^\top \tag{13} \\ &= \begin{bmatrix} a_L^1 - y^1 & a_L^2 - y^2 & a_L^3 - y^3 & a_L^4 - y^4 \end{bmatrix} \tag{14} \\ &= a_L - y \in \mathbb{R}^{1 \times K} \tag{15} \end{aligned}$$

由于这一小节相对于上一小节只是把输出从1个变成K个，而隐藏层和输入层没有任何变化，因此，$\frac{\mathrm{d}Z_{l+1}}{\mathrm{d}Z_l}$以及$\frac{\mathrm{d}Z_l}{\mathrm{d}W_l}$的公式没有任何变化。

I神经元，E个数据，K个分类

这就是一般的神经网络了。为了方便直观的推导，我们假设有$E=2$个数据，并且仍然假设$K=4$，那么：

$$X = \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} \in \mathbb{R}^{E \times M}, \quad Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{M \times K}$$

同时，对于第l层的加权求和和激活之后的变量是：

$$A_l = \begin{bmatrix} a_l^{(1)} \\ a_l^{(2)} \end{bmatrix} \in \mathbb{R}^{M \times n_l}, \quad Z_l = \begin{bmatrix} z_l^{(1)} \\ z_l^{(2)} \end{bmatrix} \in \mathbb{R}^{M \times n_l}$$

其中上标$(e) = (1), (2), \ldots, (E)$表示第几个数据。

那么损失函数就是：

$$\mathcal{L} = \frac{1}{E}\sum_e^E \mathcal{L}^{(e)}$$

那么，展开之后就是

$$\begin{aligned} \mathcal{L} &= \frac{1}{E}\sum_e^E \mathcal{L}^{(e)} \\ &= \frac{1}{E}\sum_e^E y^{(e)} \log {a_L^{(e)}}^\top - (1 - y^{(e)})\log\left(1 - a_L^{(e)}\right)^\top \end{aligned}$$

再对每一个类$k = 1, \ldots, K$展开：

$$\begin{aligned} &\frac{1}{E}\sum_e^E \sum_k^K y_k^{(e)} \log {a_L}_k^{(e)} - (1 - y_k^{(e)})\log\left(1 - {a_L}_k^{(e)}\right) \\ =&\frac{1}{E}\sum_{(e,k)} -Y \circ \log A_L - (1 - Y) \circ \log(1 - A_L) \end{aligned}$$

这里就可以直接用A_L和Y来计算损失函数，而不需要逐个数据迭代的运算。要注意，变化前后在数学意义上一样，但是在代码中，矩阵的直接运算往往比循环要快得多。这里面牵扯到许多底层的优化和并行运算的问题。

注意这里从形式上比之前多了一个$\frac{1}{E}$。

因此，从损失函数传递到输出层的公式就是：

$$\frac{\partial \mathcal{L}}{\partial Z_L} = \begin{bmatrix} \left(\frac{\partial \mathcal{L}}{\partial Z_L}\right)^{(1)} \\ \left(\frac{\partial \mathcal{L}}{\partial Z_L}\right)^{(2)} \end{bmatrix} \in \mathbb{R}^{E \times K}$$

由于每个数据的损失$\mathcal{L}^{(e)}$只和自己的forward过程中的变量有关，因此它就等于

$$\begin{bmatrix} \frac{\partial \mathcal{L}^{(1)}}{\partial Z_L^{(1)}} \\ \frac{\partial \mathcal{L}^{(2)}}{\partial Z_L^{(2)}} \end{bmatrix}$$

每一行只关乎于一条数据。那么这就相当于是上一小节的内容，因此可以表示为：

$$\begin{bmatrix} \frac{\partial \mathcal{L}^{(1)}}{\partial Z_L^{(1)}} \\ \frac{\partial \mathcal{L}^{(2)}}{\partial Z_L^{(2)}} \end{bmatrix} = \frac{1}{E}\begin{bmatrix} A_L^{(1)} - Y^{(1)} \\ A_L^{(2)} - Y^{(2)} \end{bmatrix} = \frac{1}{E}(A_L - Y)$$

层与层之间的传递：

层与层之间的传递本身并不复杂，但是从上面我们可以看到，当情况从1个数据扩展到E个数据时，每一个变量都多了一个E的维度。由于对于每个数据对应的$\left(\frac{\mathrm{d}Z_{l+1}}{\mathrm{d}Z_l}\right)^{(e)}$本身就是一个$n_{l+1} \times n_l$的矩阵，再多出一个数据就超出了一般意义上的二维矩阵。我们可以叫它高维矩阵，也可以叫它张量（tensor），这也是为什么PyTorch里面的基本数据类型就是`torch.tensor`的名字来源。

我们暂时只针对每一条数据来求反向传播的公式，那么它就应该和上一小节一模一样，也就是：

$$\left(\frac{\mathrm{d}Z_{l+1}}{\mathrm{d}Z_l}\right)^{(e)} = W_{l+1}^\top \circ A_l^{(e)} \circ (\mathbf{1} - A_l^{(e)}) \in \mathbb{R}^{n_{l+1} \times n_l}$$

输出层到某一层到传递

这里我们额外推导一个输出层到第l层到传播公式，因为这里面牵扯到高纬度的数据。 尽管$\left(\frac{\mathrm{d}Z_{l+1}}{\mathrm{d}Z_l}\right)^{(e)}$是一个2维矩阵，我们仍然把它再拼起来：

$$\begin{aligned} \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_l} &= \begin{bmatrix} \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_l}\right)^{(1)} \\ \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_l}\right)^{(2)} \end{bmatrix} \in \mathbb{R}^{E \times n_l} \\ &= \begin{bmatrix} \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}\right)^{(1)} \left(\frac{\mathrm{d}Z_{l+1}}{\mathrm{d}Z_l}\right)^{(1)} \\ \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}\right)^{(2)} \left(\frac{\mathrm{d}Z_{l+1}}{\mathrm{d}Z_l}\right)^{(2)} \end{bmatrix} \\ &= \begin{bmatrix} \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}\right)^{(1)} W_{l+1}^\top \circ A_l^{(1)} \circ (\mathbf{1} - A_l^{(1)}) \\ \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}\right)^{(2)} W_{l+1}^\top \circ A_l^{(2)} \circ (\mathbf{1} - A_l^{(2)}) \end{bmatrix} \\ &= \begin{bmatrix} \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}\right)^{(1)} W_{l+1}^\top \\ \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}\right)^{(2)} W_{l+1}^\top \end{bmatrix} \circ \begin{bmatrix} A_l^{(1)} \circ (\mathbf{1} - A_l^{(1)}) \\ A_l^{(2)} \circ (\mathbf{1} - A_l^{(2)}) \end{bmatrix} \\ &= \begin{bmatrix} \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}\right)^{(1)} \\ \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}\right)^{(2)} \end{bmatrix} \cdot W_{l+1}^\top \circ \begin{bmatrix} A_l^{(1)} \circ (\mathbf{1} - A_l^{(1)}) \\ A_l^{(2)} \circ (\mathbf{1} - A_l^{(2)}) \end{bmatrix} \\ &= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}} \cdot W_{l+1}^\top \circ A_l \circ (\mathbf{1} - A_l) \end{aligned}$$

推导完毕。

最后是传递到权重

根据之前的结论我们有：

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}W_{l+1}} = {a_l}^\top \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}$$

由于只针对一条数据，它们的维度分别为$\mathcal{L} \in \mathbb{R}$，$W_{l+1} \in \mathbb{R}^{n_l \times n_{l+1}}$，$a_l \in \mathbb{R}^{1 \times n_l}$，$Z_{l+1} \in \mathbb{R}^{1 \times n_{l+1}}$。并且$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}W_{l+1}} \in \mathbb{R}^{n_l \times n_{l+1}}$，$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}} \in \mathbb{R}^{1 \times n_{l+1}}$。

在多个数据的情况下，上式中的a_l和$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}$同时被扩展了1个维度$E$，也就是说

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}W_{l+1}} = {A_l}^\top \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}$$

其中$A_l \in \mathbb{R}^{E \times n_l}$，$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}} \in \mathbb{R}^{E \times n_{l+1}}$，所以$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}W_{l+1}} \in \mathbb{R}^{n_l \times n_{l+1}}$。

同理，

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}b_{l+1}} = \mathbf{1}^\top \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}$$

其中$\mathbf{1} \in \mathbb{R}^{E \times 1}$。

总结

这一章我们推导神经网络的反向传播，可以发现，所有的梯度都可以简洁高效地得到。这也就是神经网络发展迅速的原因。

下面我们会用自己的代码（没有PyTorch）来训练一个神经网络，再后面，我们会用PyTorch来训练神经网络。