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 $W^{ op}$ 直接记为W):  $Z_1 = X \cdot W_1$  $A_1 = f(Z_1)$  $Z_2 = A_1 \cdot W_2$  $A_2 = f(Z_2)$  $Z_3 = A_2 \cdot W_3$  $A_3 = f(Z_3)$ 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: dℓ/d4; • 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: dZ<sub>l</sub>/dW<sub>l</sub> 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  $\mathcal{L}(A_L, Y) = rac{1}{2} \|A_L - Y\|_2^2$ 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  $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}A_{T}} = -\frac{Y}{A_{T}} + \frac{1 - Y}{1 - A_{T}}$ 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:  $A_l = \frac{1}{1 \perp e^{-Z_l}}$ 那么  $rac{\mathrm{d}A_l}{\mathrm{d}Z_l} = -igg(rac{1}{e^{-Z_l}+1}igg)^2 e^{-Z_l}(-1)^2$  $=\frac{e^{-Z_l}}{(e^{-Z_l}+1)^2}$  $=rac{e^{-Z_l}+1-1}{(e^{-Z_l}+1)^2}$  $=rac{e^{-Z_l}+1}{(e^{-Z_l}+1)^2}-rac{1}{(e^{-Z_l}+1)^2}$  $=rac{1}{e^{-Z_l}+1}-rac{1}{(e^{-Z_l}+1)^2}$  $=A_l-A_l^2$  $=A_l(\mathbf{1}-A_l)$ 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  $rac{\mathrm{d}A_l}{\mathrm{d}Z_l} = A_l \circ (\mathbf{1} - A_l)$ 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  $rac{\mathrm{d}A_l}{\mathrm{d}A_{l-1}} = rac{\mathrm{d}A_l}{\mathrm{d}Z_l}rac{\mathrm{d}Z_l}{\mathrm{d}A_{l-1}} = A_l\circ (\mathbf{1}-A_l)\circ W_l$ 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  $rac{\mathrm{d}Z_l}{\mathrm{d}W_l}=A_{l-1}$ 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:  $\mathbb{R}$  $a_1 = x$  $z_l = w_l a_{l-1} + b_l$  $\mathbb{R}$  $a_l = \sigma(z_l)$  $\mathbb{R}$  $z_L = w_L a_{L-1} + b_L$  $\mathbb{R}$  $a_L = \sigma(z_L)$  $\mathbb{R}$  $\mathcal{L} = -y \log a_L - (1-y) \log (1-a_L)$   $\mathbb{R}$ 反向传播  $\frac{\partial \mathcal{L}}{\partial a_L} = -\frac{y}{a_L} + \frac{1-y}{1-a_L}$  $rac{\partial a_L}{\partial z_L} = a_L (1 - a_L)$  $\mathbb{R}$  $rac{\partial \mathcal{L}}{\partial z_L} = rac{\partial \mathcal{L}}{\partial a_L} rac{\partial a_L}{\partial z_L} = a_L - y$  $rac{\partial z_{l+1}}{\partial z_l} = rac{\partial z_{l+1}}{\partial a_l} rac{\partial a_l}{\partial z_l} = w_{l+1} \cdot a_l \cdot (1-a_l)$   $\;\;\mathbb{R}$  $\frac{\partial \mathcal{L}}{\partial z_l} = \frac{\partial \mathcal{L}}{\partial z_{l+1}} \frac{\partial z_{l+1}}{\partial z_l}$  $\mathbb{R}$  $\mathbb{R}$  $\mathbb{R}$  $\frac{\partial \mathcal{L}}{\partial w_{l+1}} = \frac{\partial \mathcal{L}}{\partial z_{l+1}} \frac{\partial z_{l+1}}{\partial w_{l+1}} = \frac{\partial \mathcal{L}}{\partial z_{l+1}} a_l$  $\mathbb{R}$  $\frac{\partial \mathcal{L}}{\partial b_{l+1}} = \frac{\partial \mathcal{L}}{\partial z_{l+1}} \frac{\partial z_{l+1}}{\partial b_{l+1}} = \frac{\partial \mathcal{L}}{\partial z_{l+1}}$  $\mathbb{R}$ 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:  $egin{aligned} rac{\partial z_{l+1}}{\partial z_l} &= egin{bmatrix} rac{\partial z_{l+1}^1}{\partial z_l^1} & rac{\partial z_{l+1}^1}{\partial z_l^2} & rac{\partial z_{l+1}^1}{\partial z_l^3} \ rac{\partial z_{l+1}^1}{\partial z_l^3} & rac{\partial z_{l+1}^1$ (1) $=\begin{bmatrix}W_{l+1}^{1,1}\frac{\partial a_{l}^{1}}{\partial z_{l}^{1}} & W_{l+1}^{1,2}\frac{\partial a_{l}^{2}}{\partial z_{l}^{2}} & W_{l+1}^{1,3}\frac{\partial a_{l}^{3}}{\partial z_{l}^{3}}\\W_{l+1}^{2,1}\frac{\partial a_{l}^{1}}{\partial z^{1}} & W_{l+1}^{2,2}\frac{\partial a_{l}^{2}}{\partial z^{2}} & W_{l+1}^{2,3}\frac{\partial a_{l}^{3}}{\partial z^{3}}\end{bmatrix}=\begin{bmatrix}W_{l+1}^{1,1} & W_{l+1}^{1,2} & W_{l+1}^{1,3}\\W_{l+1}^{2,1} & W_{l+1}^{2,2} & W_{l+1}^{2,3}\end{bmatrix}\circ\begin{bmatrix}\frac{\partial a_{l}^{1}}{\partial z_{l}^{1}} & \frac{\partial a_{l}^{2}}{\partial z_{l}^{2}} & \frac{\partial a_{l}^{3}}{\partial z_{l}^{3}}\end{bmatrix}$ (2) $=W_{_{l\pm1}}^{ op}\circ \left[\,a_{_{l}}^{1}(1-a_{_{l}}^{1})\quad a_{_{l}}^{2}(1-a_{_{l}}^{2})\quad a_{_{l}}^{3}(1-a_{_{l}}^{3})\,
ight]$ (3) $=W_{l+1}^{ op}\circ a_l\circ (1-a_l)$ (4)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:  $egin{bmatrix} W_{l+1}^{1,1} & W_{l+1}^{1,2} & W_{l+1}^{1,3} \ W_{l+1}^{2,1} & W_{l+1}^{2,2} & W_{l+1}^{2,3} \end{bmatrix} \circ egin{bmatrix} rac{\partial a_l^1}{\partial z_l^1} & rac{\partial a_l^2}{\partial z_l^2} & rac{\partial a_l^3}{\partial z_l^3} \end{bmatrix}.$ 是一个 $3 \times 2$ 矩阵和 $3 \times 1$ 矩阵的舒尔积,那么第二个矩阵会被复制乘 $3 \times 2$ ,也就相当于:  $egin{bmatrix} W_{l+1}^{1,1} & W_{l+1}^{1,2} & W_{l+1}^{1,3} \ W_{l+1}^{2,1} & W_{l+1}^{2,2} & W_{l+1}^{2,3} \end{bmatrix} \circ egin{bmatrix} rac{\partial a_l^1}{\partial z_l^1} & rac{\partial a_l^2}{\partial z_l^2} & rac{\partial a_l^3}{\partial z_l^3} \ rac{\partial a_l^1}{\partial z_l^3} & rac{\partial a_l^2}{\partial z_l^3} & rac{\partial a_l^3}{\partial z_l^3} \end{pmatrix} = egin{bmatrix} V_{l+1}^{1,3} & V_{l+1}^{1,3} & V_{l+1}^{1,3} & V_{l+1}^{1,3} & rac{\partial a_l^1}{\partial z_l^3} & rac{\partial a_l^2}{\partial z_l^3} & rac{\partial a_l^3}{\partial z_l^3} \end{bmatrix}$ 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  $Z_{l+1} = \left | egin{array}{c} z_{l+1}^1 \ z_{l+1}^2 \end{array} 
ight |$ 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 (5) $= \left[ egin{array}{cccc} rac{\partial \mathcal{L}}{\partial z_{l+1}^1} & rac{\partial \mathcal{L}}{\partial z_{l+1}^2} 
ight] \cdot \left[ egin{array}{cccc} rac{\partial z_{l+1}^1}{\partial z_l^1} & rac{\partial z_{l+1}^2}{\partial z_l^2} & rac{\partial z_{l+1}^2}{\partial z_l^3} \ rac{\partial z_{l+1}^2}{\partial z_l^2} & rac{\partial z_{l+1}^2}{\partial z_l^2} & rac{\partial z_{l+1}^2}{\partial z_l^3} \end{array} 
ight] \in \mathbb{R}^{1 imes n_l}$ (6)(7)我们再推导损失函数到参数的传播:  $egin{aligned} rac{\partial \mathcal{L}}{\partial W_{l+1}} &= egin{bmatrix} rac{\partial \mathcal{L}}{\partial W_{l+1}^{1,1}} & rac{\partial \mathcal{L}}{\partial W_{l+1}^{1,1}} & rac{\partial \mathcal{L}}{\partial z_{l+1}^1} rac{\partial \mathcal{L}}{\partial W_{l+1}^{1,1}} & rac{\partial \mathcal{L}}{\partial z_{l+1}^2} rac{\partial \mathcal{L}_{l+1}}{\partial W_{l+1}^{1,1}} \ rac{\partial \mathcal{L}}{\partial z_{l+1}^1} rac{\partial \mathcal{L}}{\partial W_{l+1}^{1,1}} & rac{\partial \mathcal{L}}{\partial z_{l+1}^2} rac{\partial \mathcal{L}_{l+1}}{\partial W_{l+1}^{1,2}} \ rac{\partial \mathcal{L}}{\partial z_{l+1}^1} rac{\partial \mathcal{L}}{\partial W_{l+1}^{1,2}} & rac{\partial \mathcal{L}}{\partial z_{l+1}^2} rac{\partial \mathcal{L}_{l+1}}{\partial W_{l+1}^{2,2}} 
ight] \in \mathbb{R}^{n_l imes n_{l+1}} \ rac{\partial \mathcal{L}}{\partial z_{l+1}^1} rac{\partial \mathcal{L}}{\partial W_{l+1}^{1,3}} & rac{\partial \mathcal{L}}{\partial z_{l+1}^2} rac{\partial \mathcal{L}_{l+1}}{\partial W_{l+1}^{2,3}} 
ight] \in \mathbb{R}^{n_l imes n_{l+1}} \end{aligned}$ (8) $=egin{bmatrix} rac{\partial \mathcal{L}}{\partial z_{l+1}^1} a_l^1 & rac{\partial \mathcal{L}}{\partial z_{l+1}^2} a_l^1 \ rac{\partial \mathcal{L}}{\partial z_{l+1}^1} a_l^2 & rac{\partial \mathcal{L}}{\partial z_{l+1}^2} a_l^2 \ rac{\partial \mathcal{L}}{\partial z_{l+1}^1} a_l^3 & rac{\partial \mathcal{L}}{\partial z_l^2} a_l^3 \end{bmatrix} = egin{bmatrix} a_l^1 \ a_l^2 \ a_l^3 \end{bmatrix} egin{bmatrix} rac{\partial \mathcal{L}}{\partial z_{l+1}^1} & rac{\partial \mathcal{L}}{\partial z_{l+1}^2} \end{bmatrix}$ (9) $=a_l^{ op}rac{\partial \mathcal{L}}{\partial z_{l+1}}$ (10) $egin{aligned} rac{\partial \mathcal{L}}{\partial b_{l+1}} = \left[ egin{array}{cc} rac{\partial \mathcal{L}}{\partial b_{l+1}^1} & rac{\partial \mathcal{L}}{\partial b_{l+1}^2} 
ight] = \left[ egin{array}{cc} rac{\partial \mathcal{L}}{\partial z_{l+1}^1} & rac{\partial \mathcal{L}}{\partial z_{l+1}^2} & rac{\partial \mathcal{L}}{\partial z_{l+1}^2} 
ight] = rac{\partial \mathcal{L}}{\partial z_{l+1}^2} \in \mathbb{R}^{1 imes n_{l+1}} \end{aligned}$ (11)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: y = [1, 0, 0, 0]y = [0, 1, 0, 0]y = [0, 0, 1, 0]y = [0, 0, 0, 1]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:  $\mathcal{L} = \sum_k -y_k \log a_{Lk} - (1-y_k) \log (1-a_{Lk})$  $=\sum_{k}-y_k\log a_{Lk}-\sum_{k}(1-y_k)\log(1-a_{Lk})$  $= \left[ y_1, y_2, y_3, y_4 
ight] \cdot \left[ egin{array}{c} \log a_{L1} \ \log a_{L2} \ \log a_{L3} \ \log a_{L3} \end{array} 
ight] - \left[ 1 - y_1, 1 - y_2, 1 - y_3, 1 - y_4 
ight] \cdot \left[ egin{array}{c} \log (1 - a_{L2}) \ \log (1 - a_{L3}) \ \log (1 - a_{L3}) \end{array} 
ight]$  $=y\log a_{\scriptscriptstyle L}^{ op}-(1-y)\log(1-a_{\scriptscriptstyle L})^{ op}$ 因此,损失函数传递到输出层的公式就是  $egin{aligned} rac{\partial \mathcal{L}}{\partial z_L} = \left[ egin{array}{cccc} rac{\partial \mathcal{L}}{\partial z_L^1} & rac{\partial \mathcal{L}}{\partial z_L^2} & rac{\partial \mathcal{L}}{\partial z_L^3} & rac{\partial \mathcal{L}}{\partial z_L^4} \end{array} 
ight] = \left[ egin{array}{cccc} rac{\partial \mathcal{L}}{\partial a_L^1} & rac{\partial a_L^1}{\partial z_L^1} & rac{\partial \mathcal{L}}{\partial a_L^2} & rac{\partial \mathcal{L}}{\partial a_L^3} & rac{\partial a_L^3}{\partial z_L^3} & rac{\partial \mathcal{L}}{\partial a_L^4} & rac{\partial \mathcal{L}}{\partial a_L^4} \end{array} 
ight] \end{aligned}$ (12) $= egin{bmatrix} \left(-rac{y^1}{a_L^1} + rac{1-y^1}{1-a_L^1}
ight) a_L^1(1-a_L^1) \ \left(-rac{y^2}{a_L^2} + rac{1-y^2}{1-a_L^2}
ight) a_L^2(1-a_L^2) \ \left(-rac{y^3}{a_L^3} + rac{1-y^3}{1-a_L^3}
ight) a_L^3(1-a_L^3) \ \left(-rac{y^4}{a_L^4} + rac{1-y^4}{1-a_L^4}
ight) a_L^4(1-a_L^4) \end{bmatrix}$ (13) $= \left[ egin{array}{cccc} a_L^1 - y^1 & a_L^2 - y^2 & a_L^3 - y^3 & a_L^4 - y^4 \end{array} 
ight]$ (14)(15)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, $rac{\mathrm{d} Z_{l+1}}{\mathrm{d} Z_l}$ 以及 $rac{\mathrm{d} Z_l}{\mathrm{d} W_l}$ 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:  $X = egin{bmatrix} x^{(1)} \ x^{(2)} \end{bmatrix} \in \mathbb{R}^{E imes M}, \quad Y = egin{bmatrix} y^{(1)} \ y^{(2)} \end{bmatrix} = egin{bmatrix} 0 & 1 & 0 & 0 \ 0 & 0 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{M imes K}.$ 同时,对于第l层的加权求和和激活之后的变量是:  $A_l = \left | egin{array}{c} a_l^{(1)} \ a_l^{(2)} \end{array} 
ight | \in \mathbb{R}^{M imes n_l}, \quad Z_l = \left | egin{array}{c} z_l^{(1)} \ z_l^{(2)} \end{array} 
ight | \in \mathbb{R}^{M imes n_l},$ 其中上标 $(e) = (1), (2), \ldots, (E)$ 表示第几个数据。 那么损失函数就是:  $\mathcal{L} = rac{1}{E} \sum^{E} \mathcal{L}^{(e)}$ 那么,展开之后就是  $\mathcal{L} = rac{1}{E} \sum_{}^{E} \mathcal{L}^{(e)}$  $y = rac{1}{E} \sum_{i=1}^{E} y^{(e)} \log a_L^{(e)}^ op - (1-y^{(e)}) \log \left(1-a_L^{(e)}
ight)^ op .$ 再对每一个类 $k = 1, \ldots, K$ 展开:  $rac{1}{E}\sum_{i=1}^{E}\sum_{k=1}^{K}y_{k}^{(e)}\log a_{L_{k}^{(e)}}-(1-y_{k}^{(e)})\log (1-a_{L_{k}^{(e)}})$  $= rac{1}{E} \sum_{(a,b)} -Y \circ \log A_L - (1-Y) \circ \log (1-A_L)$ 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:  $rac{\partial \mathcal{L}}{\partial Z_L} = egin{bmatrix} \left(rac{\partial \mathcal{L}}{\partial Z_L}
ight)^{(1)} \ \left(rac{\partial \mathcal{L}}{\partial Z_L}
ight)^{(2)} \end{bmatrix} \in \mathbb{R}^{E imes K}$ 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 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:  $\left| egin{array}{c} rac{\partial \mathcal{L}^{(1)}}{\partial Z_L^{(2)}} \ rac{\partial \mathcal{L}^{(2)}}{\partial Z^{(2)}} \end{array} 
ight| = rac{1}{E} \left[ egin{array}{c} A_L^{(1)} - Y^{(1)} \ A_L^{(2)} - Y^{(2)} \end{array} 
ight] = rac{1}{E} (A_L - Y)$ 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 $\left(rac{\mathrm{d}Z_{l+1}}{\mathrm{d}Z_l}
ight)^{(e)}$ 本身就是一个 $n_{l+1} imes n_l$ 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:  $\left(rac{\mathrm{d} Z_{l+1}}{\mathrm{d} Z_{l}}
ight)^{(e)} = W_{l+1}^{ op} \circ A_{l}^{(e)} \circ (\mathbf{1} - A_{l}^{(e)}) \in \mathbb{R}^{n_{l+1} imes n_{l}}$ 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 $\left(rac{\mathrm{d}Z_{l+1}}{\mathrm{d}Z_l}
ight)^{(e)}$ 是一个2维矩阵,我们仍然把它再拼起来:  $rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_l} = \left| egin{array}{c} \left(rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_l}
ight)^{(1)} \ \left(rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z}
ight)^{(2)} \end{array} 
ight| \in \mathbb{R}^{E imes n_l}$  $= \begin{bmatrix} \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}\right)^{(1)} \left(\frac{\mathrm{d}Z_{l+1}}{\mathrm{d}Z_{l}}\right)^{(1)} \\ \left(\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l}}\right)^{(2)} \left(\frac{\mathrm{d}Z_{l+1}}{\mathrm{d}Z_{l}}\right)^{(2)} \end{bmatrix}$  $egin{aligned} &= \left[ egin{aligned} \left( rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}} 
ight)^{(1)} W_{l+1}^{ op} \circ A_l^{(1)} \circ (\mathbf{1} - A_l^{(1)}) \ \left( rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}} 
ight)^{(2)} W_{l+1}^{ op} \circ A_l^{(2)} \circ (\mathbf{1} - A_l^{(2)}) \end{aligned} 
ight] \end{aligned}$  $=egin{bmatrix} \left(rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}
ight)^{(1)}W_{l+1}^{ op}\ \left(rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}
ight)^{(2)}W_{l+1}^{ op} \end{bmatrix} \circ egin{bmatrix} A_l^{(1)}\circ (\mathbf{1}-A_l^{(1)})\ A_l^{(2)}\circ (\mathbf{1}-A_l^{(2)}) \end{bmatrix}$  $egin{aligned} &= egin{aligned} \left(rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}
ight)^{(1)} \ \left(rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}
ight)^{(2)} \end{aligned} egin{aligned} \cdot W_{l+1}^{ op} &\circ \left[egin{aligned} A_l^{(1)} \circ (\mathbf{1}-A_l^{(1)}) \ A_l^{(2)} \circ (\mathbf{1}-A_l^{(2)}) \end{aligned} 
ight] \end{aligned}$  $\mathbf{x} = rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}} \cdot W_{l+1}^{ op} \circ A_l \circ (\mathbf{1} - A_l)^{ op}$ 推导完毕。 最后是传递到权重 根据之前的结论我们有:  $rac{\mathrm{d}\mathcal{L}}{\mathrm{d}W_{l+1}} = a_l^{ op} rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}$ 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  $rac{\mathrm{d}\mathcal{L}}{\mathrm{d}W_{l+1}} = A_l^{ op} rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}$ 其中 $A_l \in \mathbb{R}^{E imes n_l}$ , $rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}} \in \mathbb{R}^{E imes n_{l+1}}$ ,所以 $rac{\mathrm{d}\mathcal{L}}{\mathrm{d}W_{l+1}} \in \mathbb{R}^{n_l imes n_{l+1}}$ 。 同理,  $rac{\mathrm{d}\mathcal{L}}{\mathrm{d}b_{l+1}} = \mathbf{1}^ op rac{\mathrm{d}\mathcal{L}}{\mathrm{d}Z_{l+1}}$ 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。