3.10（选修）直观理解反向传播（Backpropagation intuition）

这个视频主要是推导反向传播。

下图是逻辑回归的推导：

回想一下逻辑回归的公式(参考公式 3.2、公式 3.5、公式 3.6、公式 3.15) 公式 3.38：

$$\left.\begin{matrix}x\\w\\b\end{matrix}\right\}\Rightarrow z=w^Tx+b\Rightarrow a=\sigma(z)\Rightarrow L(a,y)$$

所以回想当时我们讨论逻辑回归的时候，我们有这个正向传播步骤，其中我们计算z，然后a，然后损失函数L。

公式 3.39：

$$\left.\begin{matrix}x\\w\\b\end{matrix}\right\}\quad\Leftarrow\quad z=w^Tx+b$$

$$dw=dz\cdot x,db=dz \qquad dz=da\cdot g'(z),g(z)=\sigma(z),\frac{dL}{dz}=\frac{dL}{da}\cdot\frac{da}{dz},\frac{d}{dz}g(z)=g'(z)$$

$$\Leftarrow\quad a=\sigma(z)\Leftarrow L(a,y)$$

$$da=\frac{d}{da}L(a,y)=(-y\log a-(1-y)\log(1-a))'=-\frac{y}{a}+\frac{1-y}{1-a}$$

神经网络的计算中，与逻辑回归十分类似，但中间会有多层的计算。下图是一个双层神经网络，有一个输入层，一个隐藏层和一个输出层。

前向传播：

计算$z^{[1]}$，$a^{[1]}$，再计算$z^{[2]}$，$a^{[2]}$，最后得到 **loss function**。

反向传播：

向后推算出$da^{[2]}$，然后推算出$dz^{[2]}$，接着推算出$da^{[1]}$，然后推算出$dz^{[1]}$。我们不需要对x求导，因为x是固定的，我们也不是想优化x。向后推算出$da^{[2]}$，然后推算出$dz^{[2]}$的步骤可以合为一步：

公式 3.40： $dz^{[2]}=a^{[2]}-y$ ， $dW^{[2]}=dz^{[2]}a^{[1]^T}$ (注意：逻辑回归中；为什么$a^{[1]T}$多了个转置：dw中的W(视频里是$W_i^{[2]}$)是一个列向量，而$W^{[2]}$是个行向量，故需要加个转置);

公式 3.41： $db^{[2]}=dz^{[2]}$

公式 3.42：$dz^{[1]}=W^{[2]T}dz^{[2]}*g[1]'(z^{[1]})$ 注意：这里的矩阵：$W^{[2]}$的维度是：$(n^{[2]},n^{[1]})$。

$z^{[2]}$ ， $dz^{[2]}$的维度都是：$(n^{[2]},1)$，如果是二分类，那维度就是$(1,1)$。

$z^{[1]}$，$dz^{[1]}$的维度都是：$(n^{[1]},1)$。

证明过程： 见公式 3.42，

其中$W^{[2]T}dz^{[2]}$维度为：$(n^{[1]},n^{[2]})$、$(n^{[2]},1)$相乘得到$(n^{[1]},1)$，和$z^{[1]}$维度相同，$g[1]'(z^{[1]})$的维度为$(n^{[1]},1)$，这就变成了两个都是$(n^{[1]},1)$向量逐元素乘积。

实现后向传播有个技巧，就是要保证矩阵的维度相互匹配。最后得到$dW^{[1]}$和$db^{[1]}$，公式 3.43： $dW^{[1]}=dz^{[1]}x^{T},db^{[1]}=dz^{[1]}$

可以看出$dW^{[1]}$ 和$dW^{[2]}$ 非常相似，其中x扮演了$a^{[0]}$的角色，x^{T} 等同于$a^{[0]T}$。

由： $Z^{[1]}=W^{[1]}x+b^{[1]}$ ， $a^{[1]}=g^{[1]}(Z^{[1]})$ 得到： $Z^{[1]}=W^{[1]}x+b^{[1]},A^{[1]}=g^{[1]}(Z^{[1]})$

$$Z^{[1]}=\begin{bmatrix}\vdots & \vdots & \vdots & \vdots\\ z^{[1](1)} & z^{[1](2)} & \vdots & z^{[1](m)}\\ \vdots & \vdots & \vdots & \vdots\end{bmatrix}$$

注意：大写的$Z^{[1]}$表示$z^{[1](1)},z^{[1](2)},z^{[1](3)}\ldots z^{[1](m)}$的列向量堆叠成的矩阵，以下类同。

下图写了主要的推导过程：

公式 3.44： $dZ^{[2]}=A^{[2]}-Y$ ， $dW^{[2]}=\frac{1}{m}dZ^{[2]}A^{[1]^{T}}$

公式 3.45： $L=\frac{1}{m}\sum_{i}^{n}L(\hat{y},y)$

公式 3.46： $db^{[2]}=\frac{1}{m}np.sum(dZ^{[2]},axis=1,keepdims=True)$

公式 3.47： $\underset{(n^{[1]},m)}{dZ^{[1]}}=\underset{(n^{[1]},m)}{W^{[2]T}dZ^{[2]}}*\underset{(n^{[1]},m)}{g[1]'(Z^{[1]})}$

公式 3.48： $dW^{[1]}=\frac{1}{m}dZ^{[1]}x^{T}$

公式 3.49： $db^{[1]}=\frac{1}{m}np.sum(dZ^{[1]},axis=1,keepdims=True)$

吴恩达老师认为反向传播的推导是机器学习领域最难的数学推导之一，矩阵的导数要用链式法则来求，如果这章内容掌握不了也没大的关系，只要有这种直觉就可以了。还有一点，就是初始化你的神经网络的权重，不要都是 0，而是随机初始化，下一章将详细介绍原因。

Summary of gradient descent

$dz^{[2]} = a^{[2]} - y$

$dW^{[2]} = dz^{[2]} a^{[1]T}$

$db^{[2]} = dz^{[2]}$

$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$

$(n^{[1]}, 1)$

$dW^{[1]} = dz^{[1]} x^T$

$db^{[1]} = dz^{[1]}$

$dZ^{[2]} = A^{[2]} - Y$

$J(\dots) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$

$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$

$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$

$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$

$(n^{[1]}, m)$ $(n^{[1]}, m)$ $(n^{[1]}, m)$ — elementwise product

$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$

$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$

Andrew Ng