# 深度学习之前向传播和反向传播

Input: $a^{[L-1]}$ Output: $a^{[L]}$, cache$(z^{[L]})$

**Forward:**

$$\overset{n\times m}{z^{[L]}} = W^{[L]} \overset{m\times 1}{a^{[L-1]}} + \overset{n\times 1}{b^{[L]}}$$

$$\underset{n\times 1}{a^{[L]}} = g^{[L]}(z^{[L]})$$

(vectorized)

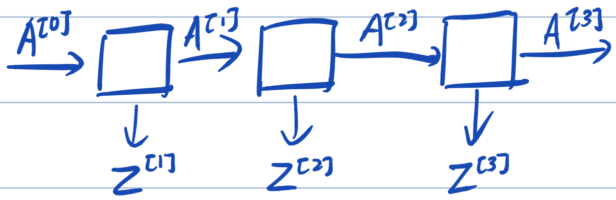
$$Z^{[L]} = W^{[L]} A^{[L-1]} + b^{[L]}$$

$$A^{[L]} = g^{[L]}(Z^{[L]})$$

$A_0 = \boxed{m\times n_0}$ $W_0 = n_1\times n_0$ $B_0$

$A_1 = m\times n_1$ $W_1 = n_2\times n_1$ $B_1$

$$\overset{m\times n_1}{(\underset{m\times n_0}{A_0}\cdot \underset{n_1\times n_0}{W^T})} + \overset{1\times n_1}{B_0}$$




**Backpropagation:**

$$\frac{dJ}{dz^{[L]}} = \frac{dJ}{da^{[L]}} \odot \frac{da^{[L]}}{dz^{[L]}} = \frac{dJ}{da^{[L]}} \odot g^{[L]\prime}(z^{[L]})$$

$$\frac{dJ}{dW^{[L]}} = \frac{dJ}{dz^{[L]}} \cdot a^{[L-1]T}$$

$$\frac{dJ}{db^{[L]}} = \frac{dJ}{dz^{[L]}}$$

$$\frac{dJ}{da^{[L-1]}} = W^{[L]T} \frac{dJ}{dz^{[L]}}$$

$x: (m, n_1)$

$w: (n_2, n_1)$

$b: (n_2, 1)$

forward: $\overset{m\times n_2}{Z} = \overset{m\times n_1}{X}\, \overset{n_1\times n_2}{W^T} + \overset{1\times n_2}{b^T}$ $n_2\times m$

backward: $\underset{n_2\times 1}{db} = dZ \cdot \frac{dZ}{db} = (dZ.T,\ \text{axis}=1)$ np.mean

$$\underset{m\times n_1}{dx} = dz \cdot W \quad m\times n_2 \cdot n_2\times n_1$$

$$\underset{n_2\times n_1}{dw} = dz.T \cdot X \quad (n_2\times m)\cdot(m\times n_1)$$

## softmax公式和求导:

公式: $\underset{n\times 1}{z} \rightarrow \underset{n\times 1}{y}$ (向量)

$$y_i = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}$$

反向传播:

结果是一个n×n的矩阵

$$\begin{bmatrix} \frac{dy_1}{dz_1} & \cdots & \frac{dy_n}{dz_1} \\ \frac{dy_1}{dz_2} & & \vdots \\ \vdots & & \\ \frac{dy_1}{dz_n} & & \frac{dy_n}{dz_n} \end{bmatrix}$$

其中 $$\frac{dy_i}{dz_j} = \frac{d\left(\frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}\right)}{d z_j}$$

当i=j时 $$\frac{dy_i}{dz_j} = \frac{e^{z_i}\left(\sum_{i=1}^{n} e^{z_i}\right) - e^{z_i}\cdot e^{z_i}}{\left(\sum_{i=1}^{n} e^{z_i}\right)^2}$$

$$= \frac{e^{z_i}}{\left(\sum_{i=1}^{n} e^{z_i}\right)^2} - \left(\frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}\right)^2$$

$$= y_i - y_i^2$$

$= y_i - y_i^2$

当$i \neq j$时, $\frac{dy_i}{dz_j} = -\frac{e^{z_i}}{(\sum_{i=1}^{n} e^{z_i})^2} \cdot e^{z_j}$

$= -y_i y_j$

综上所述:

$$\frac{dy_i}{dz_j} = \begin{cases} y_i(1-y_i) & i=j \\ -y_i y_j & i \neq j \end{cases}$$

coss-entropy求导:

$$H(\hat{y}, y) = -\hat{y}^T \log y = -\sum_{t=1}^{m} \hat{y}_t \log y_t$$

$$\frac{dH(\hat{y}, y)}{dy} = \frac{1}{y} \odot -\hat{y} = -\frac{\hat{y}}{y}$$

coss-entropy + softmax 一起求导:

$$z \to \overset{y}{\text{softmax}(z)} \to -\hat{y}^T \log y$$

$$\frac{dH(\hat{y}, y)}{dz_j} = \sum_{i=1}^{m} \frac{\partial H(\hat{y}, y)}{\partial y_i}$$

$$x \to \log x \to y^T x$$

一种想法: $\frac{d(y^T x)}{dx} = \frac{1}{x} \odot y$

(哈达玛乘积)

另一种想法: $\frac{d(y^T x)}{dx} = \begin{pmatrix} \frac{1}{x_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{x_2} & \cdots & 0 \\ \vdots & & & \\ 0 & \cdots & \cdots & \frac{1}{x_n} \end{pmatrix}^T \cdot \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \frac{1}{x} \odot y$

$$g(x) = \frac{1}{1+e^{-x}}$$

$$g'(x) = -\frac{1}{(1+e^{-x})^2} \cdot e^{-x} \cdot (-1)$$

$$= \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= g(x) \cdot (1-g(x))$$

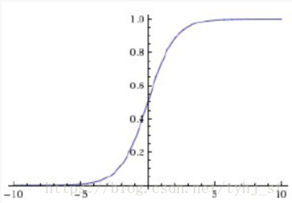
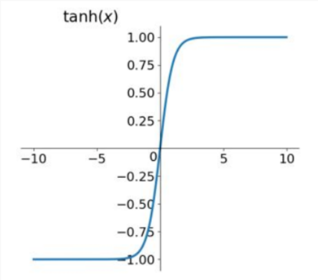
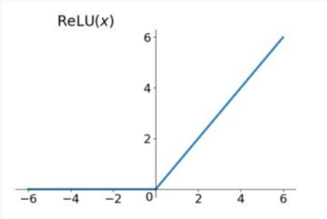
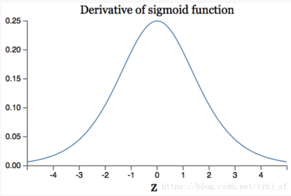
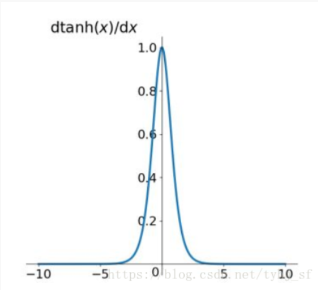
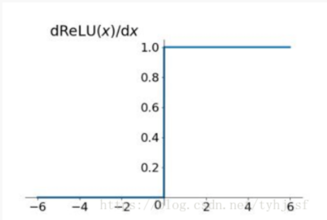
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\tanh'(x) = \frac{(e^x+e^{-x})(e^x+e^{-x}) - (e^x-e^{-x})(e^x-e^{-x})}{(e^x+e^{-x})^2}$$

$$= \frac{(e^x+e^{-x})^2 - (e^x-e^{-x})^2}{(e^x+e^{-x})^2}$$

$$= 1 - \tanh^2$$

常见的3类激活函数的比较:

| 常用函数 | sigmoid函数 | tanh函数 | relu函数 |
| --- | --- | --- | --- |
| 表达式 | $sigmoid(x) = \frac{1}{1+e^{-x}}$ | $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ | $relu(x) = max(0, x)$ |
| 图像 |  |  |  |
| 导数 | $g'(x) = g(x)(1 - g(x))$ | $tanh'(x) = 1 - tanh^2(x)$ | $relu'(x) = 1 * (x > 0) * x$ |
| 导数图像 |  |  |  |
| 特点 | 它能够把输入的连续实值变换为0和1之间的输出，特别的，如果是非常大的负数，那么输出就是0；如果是非常大的正数，输出就是1. | tanh读作Hyperbolic Tangent，它解决了Sigmoid函数的不是zero-centered输出问题，然而，梯度消失（gradient vanishing）的问题和幂运算的问题仍然存在。 | ReLU函数其实就是一个取最大值函数，注意这并不是全区间可导的，但是我们可以取sub-gradient，如上图所示。ReLU虽然简单，但却是近几年的重要成果，有以下几大优点：<br>1）解决了gradient vanishing问题 (在正区间)<br>2）计算速度非常快，只需要判断输入是否大于0<br>3）收敛速度远快于sigmoid和tanh |

| 常用函数 | sigmoid函数 | tanh函数 | relu函数 |
| --- | --- | --- | --- |
| 缺点 | 缺点1：在深度神经网络中梯度反向传递时导致梯度爆炸和梯度消失，其中梯度爆炸发生的概率非常小，而梯度消失发生的概率比较大。<br>缺点2：Sigmoid 的 output 不是0均值（即 zero-centered）。这是不可取的，因为这会导致后一层的神经元将得到上一层输出的非0均值的信号作为输入。 产生的一个结果就是：如 $x > 0, f = w^T x + bx,$ 那么对w求局部梯度则都为正，这样在反向传播的过程中w要么都往正方向更新，要么都往负方向更新，导致有一种捆绑的效果，使得收敛缓慢。 当然了，如果按batch去训练，那么那个batch可能得到不同的信号，所以这个问题还是可以缓解一下的。因此，非0均值这个问题虽然会产生一些不好的影响，不过跟上面提到的梯度消失问题相比还是要好很多的。<br>缺点3：其解析式中含有幂运算，计算机求解时相对来讲比较耗时。对于规模比较大的深度网络，这会较大地增加训练时间。 | tanh读作Hyperbolic Tangent，它解决了Sigmoid函数的不是zero-centered输出问题，然而，梯度消失（gradient vanishing）的问题和幂运算的问题仍然存在。 | ReLU也有几个需要特别注意的问题：<br>1）ReLU的输出不是zero-centered<br>2）Dead ReLU Problem，指的是某些神经元可能永远不会被激活，导致相应的参数永远不能被更新。有两个主要原因可能导致这种情况产生: (1) 非常不幸的参数初始化，这种情况比较少见 (2) learning rate太高导致在训练过程中参数更新太大，不幸使网络进入这种状态。解决方法是可以采用Xavier初始化方法，以及避免将learning rate设置太大或使用adagrad等自动调节learning rate的算法。 |