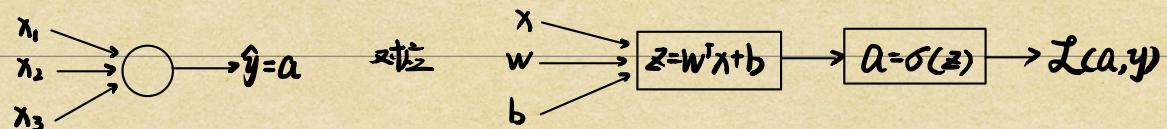
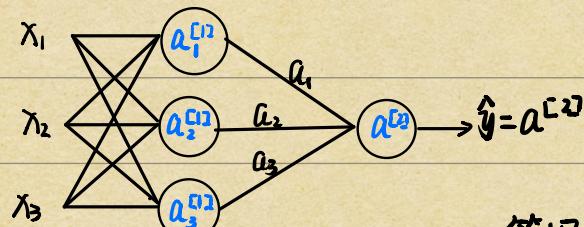


1.

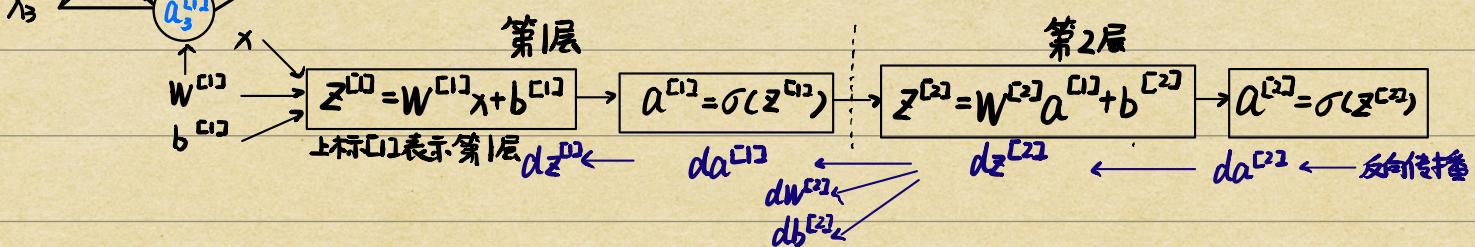
对于单个神经元：



对于神经网络，每个节点进行的计算与单个神经元类似。

输入层 $a^{[0]}$ 隐藏层 $a^{[1]}$ 输出层

注：通常称这是一个2层NN



2. NN的计算方法

⇒ 向量化时，一层中有多个神经元时，就将其堆叠起来。

例： $z^{[0]} = \begin{bmatrix} z_1^{[0]} \\ z_2^{[0]} \\ z_3^{[0]} \\ z_4^{[0]} \end{bmatrix}$, $W^{[0]} = \begin{bmatrix} -w_{11}^{[0]T} \\ -w_{21}^{[0]T} \\ -w_{31}^{[0]T} \\ -w_{41}^{[0]T} \end{bmatrix}$, $X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, $b^{[0]} = \begin{bmatrix} b_1^{[0]} \\ b_2^{[0]} \\ b_3^{[0]} \\ b_4^{[0]} \end{bmatrix}$, $a^{[0]} = \begin{bmatrix} a_1^{[0]} \\ a_2^{[0]} \\ a_3^{[0]} \\ a_4^{[0]} \end{bmatrix} = \sigma(z^{[0]})$

则计算时： $z^{[0]} = W^{[0]}X + b^{[0]}$, $a^{[0]} = \sigma(z^{[0]})$

$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$, $a^{[1]} = \sigma(z^{[1]})$

当有多个样本时，

$$x^{(1)} \rightarrow a^{1} = \hat{y}^{(1)}$$

$$x^{(2)} \rightarrow a^{[1](2)} = \hat{y}^{(2)}$$

$$\vdots$$

$$x^{(m)} \rightarrow a^{[1](m)} = \hat{y}^{(m)}$$

计算方法：① for 循环遍历 $i=1$ to m

② 向量化： $X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{bmatrix}_{n_x, m}$

$$\text{计算: } Z^{[1]} = W^{[1]}X + b^{[1]}, A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}, A^{[2]} = \sigma(Z^{[2]})$$

则

$$Z^{[1]} = \begin{bmatrix} z_1^{1} & z_1^{[1](2)} & \dots & z_1^{[1](m)} \end{bmatrix}$$

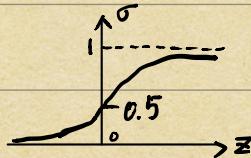
$$A^{[1]} = \begin{bmatrix} a_1^{1} & a_1^{[1](2)} & \dots & a_1^{[1](m)} \end{bmatrix}$$

横向为不同训练样本的值

纵向为同一层
不同节点的值

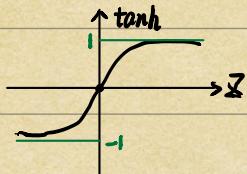
3. 激活函数

① Sigmoid 函数:



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

② tanh 函数:
双曲正切函数



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

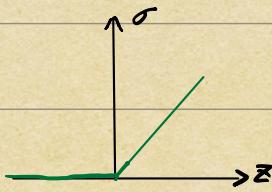
说明: 在神经网络中, 隐藏层使用 tanh 作为激活函数的效果几乎始终比 Sigmoid 好
但在二元分类输出层使用 Sigmoid 比较合适, 原因是输出值处于 0~1 间, 相当于概率,
比较容易理解.

→ Sigmoid 与 tanh 的缺点: 当 z 很大或很小时, 函数的导数很小, 即斜率很小.

这会拖慢算法, 因此引入“修正线性单元 ReLu”
亦称整流

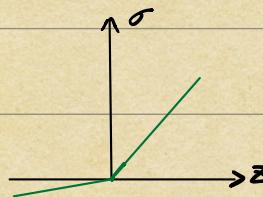
③ ReLu 函数:

推荐!



$$\sigma = \max(0, z)$$

带泄漏的 ReLu 函数:



$$\sigma = \max(0.01z, z)$$

← 可调参数

说明: (1) z=0 处 ReLu 函数导数未定义. 但实际上 z=0.0000... 的几率很小.

可以在 z=0 时给导数赋值. (0、1 皆可)

可以将 NN 中除了输出层以外所有单元设为 ReLU 函数，而输出层为 Sigmoid 函数

激活函数的选择：(1) 当输出值为 0 或 1 时（二元分类），使用 Sigmoid 作为输出层。其它所有单元都用 ReLU

(2) Sigmoid 建议只在二元分类的输出层使用。

tanh 函数几乎在任何场合下都更优越。

ReLU 函数是最一般的默认选择

注：为什么需要非线性的 Activation function？

→ 当计算 $a^{[i]} = g^{[i]}(z^{[i]})$ 时，若 $g^{[i]}$ 只是线性激活函数，则输出只是输入的线性组合。此时网络再深也没有用。

4. 激活函数的导数

① Sigmoid 函数： $g'(z) = g(z)[1 - g(z)]$

② tanh 函数： $g'(z) = 1 - [\tanh(z)]^2$

③ ReLU : $g'(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \\ \text{未定义}, & z=0 \end{cases}$

4. NN 梯度下降

① 参数： $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$

$n_1 = n^{[0]}$ 个输入特征， $n^{[1]}$ 个隐藏单元， $n^{[2]}$ 个输出单元 ($= 1$)

则： $W^{[2]} \in (n^{[1]}, n^{[2]})$, $b^{[2]} \in (n^{[2]}, 1)$

$W^{[2]} \in (n^{[2]}, n^{[1]})$, $b^{[2]} \in (n^{[2]}, 1)$

② 代价函数： $J(W^{[2]}, b^{[2]}, W^{[1]}, b^{[1]}) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}_i - y_i)$, 其中 $\hat{y} = a^{[2]}$

二分类时， $\mathcal{L}(\hat{y} - y)$ 与 Sigmoid 函数的相同

③ 梯度下降：

随机初始化。

重复直至收敛：

计算 $\text{predict}(\hat{y}^{(i)}, i=1, 2, \dots, m)$

计算 $dw^{(i)}, db^{(i)}, \dots$

$$w^{(i)} = w^{(i)} - \alpha dw^{(i)}$$

$$b^{(i)} = b^{(i)} - \alpha db^{(i)}$$

...

前向传播：

$$\begin{cases} z^{(1)} = W^{(1)}X + b^{(1)} & , A^{(1)} = g(z^{(1)}) \\ z^{(2)} = W^{(2)}A^{(1)} + b^{(2)} & , A^{(2)} = g(z^{(2)}) \end{cases}$$

反向传播：

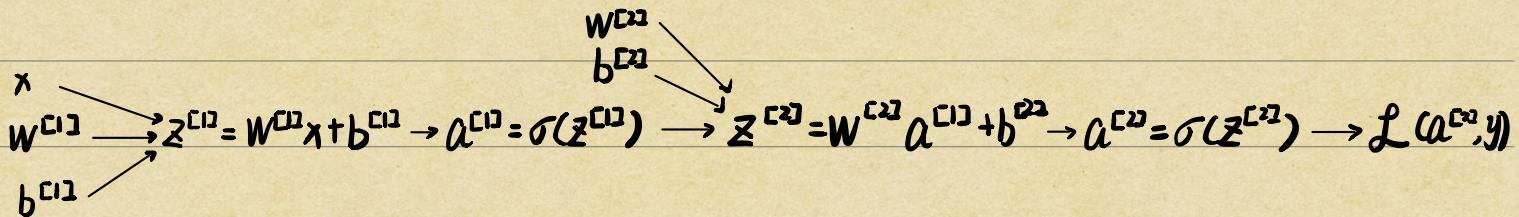
$$\begin{cases} dz^{(2)} = A^{(2)} - Y & , Y = [y^{(1)} \ y^{(2)} \ \dots \ y^{(m)}] \\ dW^{(2)} = \frac{1}{m} dz^{(2)} A^{(1)T} & \\ db^{(2)} = \frac{1}{m} \cdot \text{np.sum}(dz^{(2)}, axis=1, keepdims=True) & \text{防止numpy输出秩1的数组。} \\ dz^{(1)} = W^{(2)T} dz^{(2)} * \underbrace{g^{(1)'}(z^{(1)})}_{\text{元素对应相乘}} & \text{激活函数的导数} | \text{链式法则: } dz = da \cdot g'(z) \end{cases}$$

$$dW^{(1)} = \frac{1}{m} dz^{(1)} X^T$$

$$db^{(1)} = \frac{1}{m} \cdot \text{np.sum}(dz^{(1)}, axis=1, keepdims=True)$$

$dA^{(2)}, dA^{(1)}$ 的计算被合为 dz 计算中的一步

计算图：



5. 随机初始化

①若初始参数 $W^{[1]} = \vec{0}$ \Rightarrow 引发对称权重问题

则隐藏层都是由同一函数来计算的. (W 参数相同)

\therefore 隐藏层中的激活项 a 全部相等, 则 $a_1^{[1]} = a_2^{[1]}, dz_1^{[1]} = dz_2^{[1]}$, 也全部相等

则可得后向传播时偏导数项也相等 \Rightarrow 每次梯度下降后, 对应输入的隐藏层的参数都相等

\Rightarrow 此时神经网络高度冗余. $dW = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$

②随机初始化: 使每个 W 的元素 随机生成在集一区间内.

$W^{[1]} = np.random.randn(2, 2) * 0.01$ $\xrightarrow{\text{赋予非常小的随机初始值.}} \text{Sigmoid 函数在 } z \rightarrow \infty \text{ 时斜率很小.}$

$b^{[1]} = np.zeros(2, 1) \longrightarrow b$ 全为 0 不会造成对称问题.

$W^{[2]} = \sim$

$b^{[2]} = \sim$