深度学习-第三章

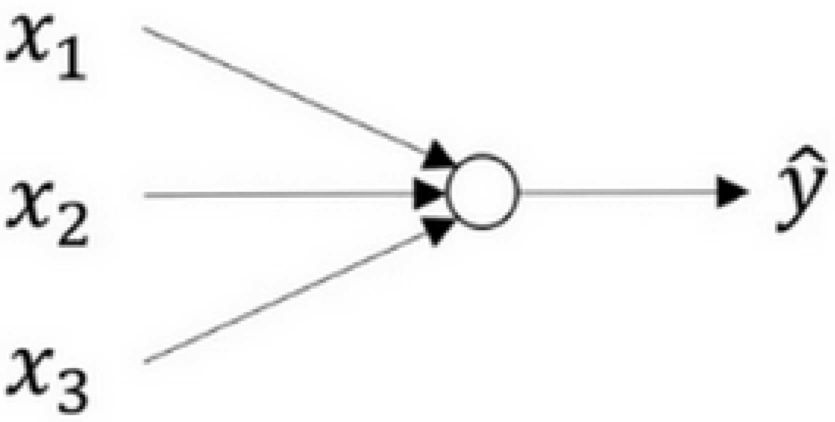
黄海广 副教授

2021年03月

本章目录

1.神经网络的概念

神经网络的概念

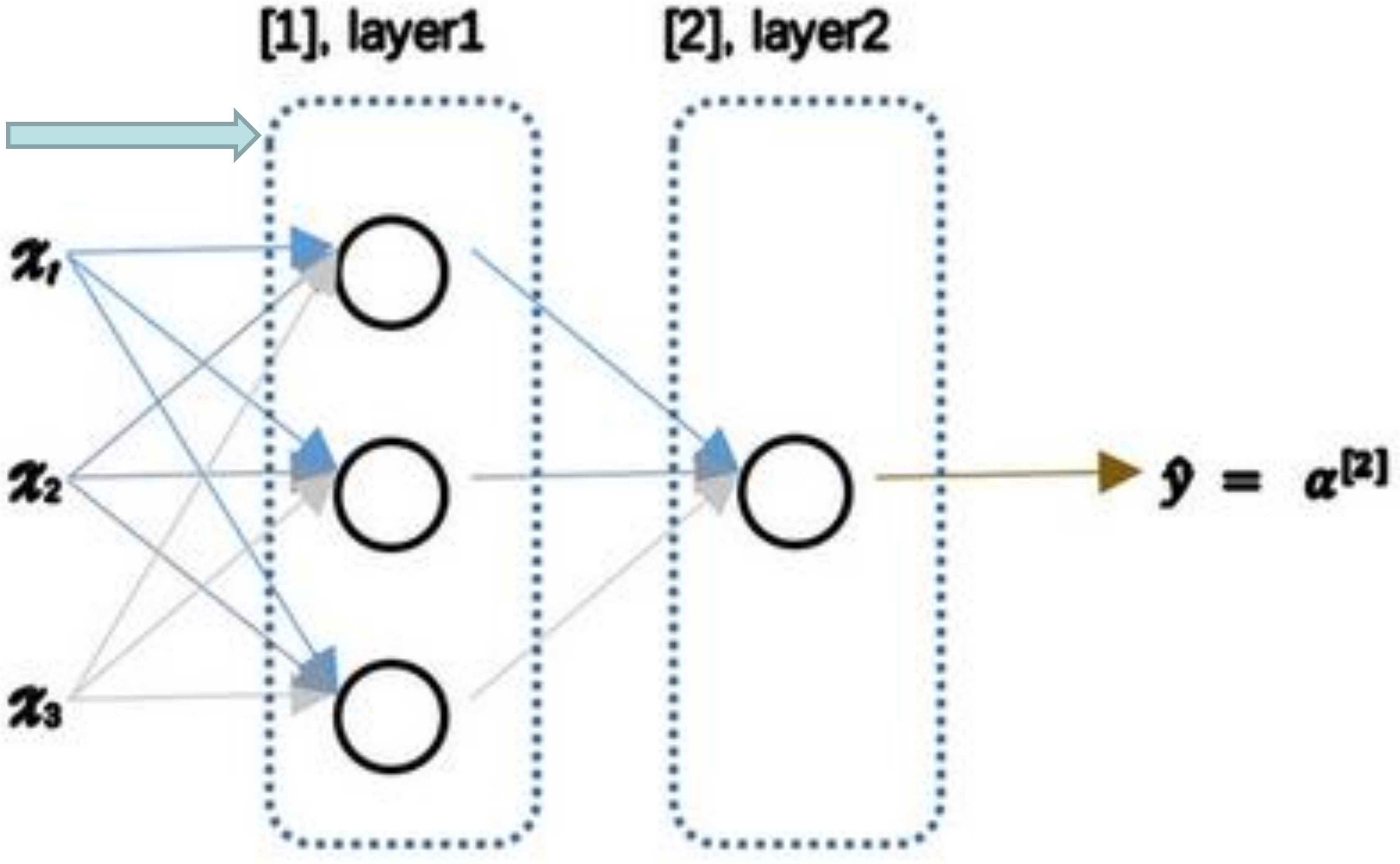


$$\left.\begin{array}{l} x \\ w \\ b \end{array}\right\} \Longrightarrow z = w^T x + b \Longrightarrow \alpha = \sigma(z)$$

$$\Longrightarrow L(a, y)$$

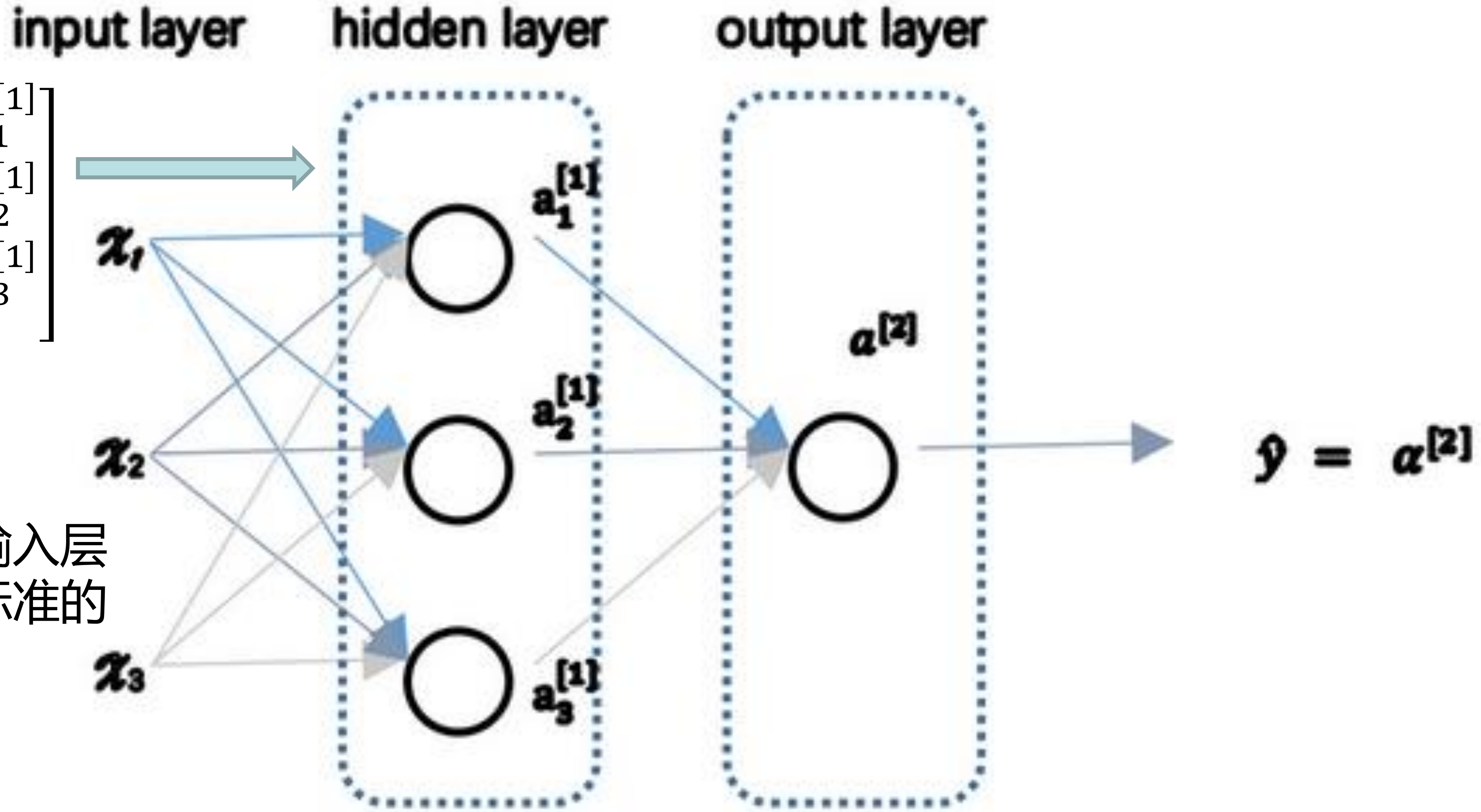
神经网络的概念

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix}$$



神经网络的概念

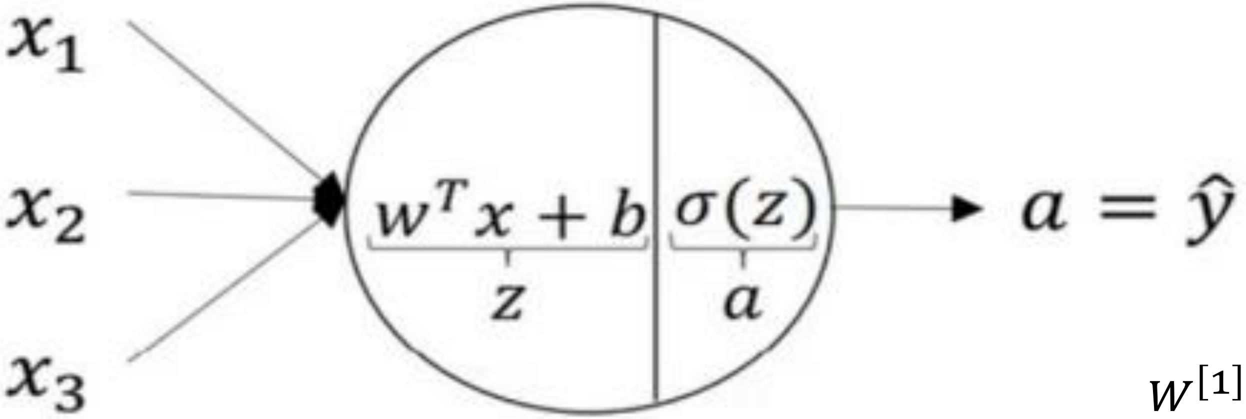
$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix}$$



我们不将输入层看作一个标准的层。

2.神经网络的向量化

2.神经网络的向量化



$$z = w^T x + b$$

$$a = \sigma(z)$$

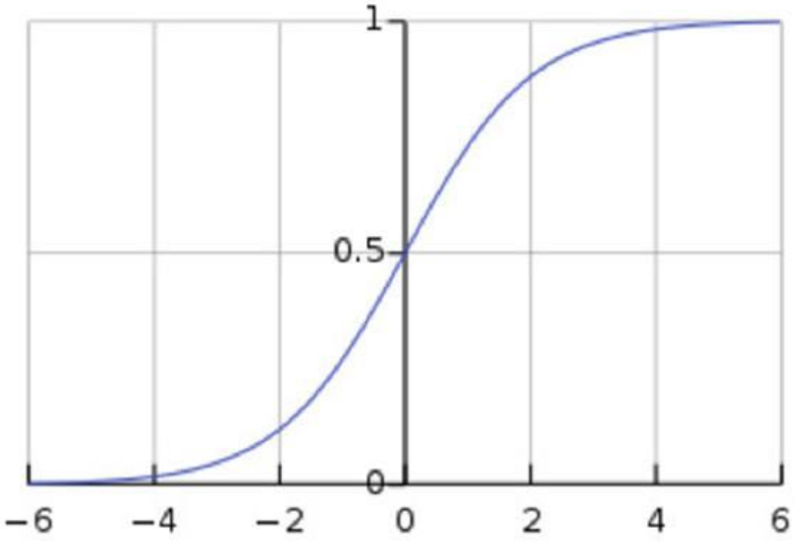
$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \overbrace{\begin{bmatrix} \ldots W_1^{[1]T} \ldots \\ \ldots W_2^{[1]T} \ldots \\ \ldots W_3^{[1]T} \ldots \\ \ldots W_4^{[1]T} \ldots \end{bmatrix}}^{W^{[1]}} * \overbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}^{input} + \overbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}^{b^{[1]}}$$

3.激活函数

3.激活函数

Sigmoid函数

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



sigmoid 函数

$$\frac{d}{dz}g(z) = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$
$$= g(z)(1 - g(z))$$

当$\sigma(z)$大于等于0.5时，预测 y=1

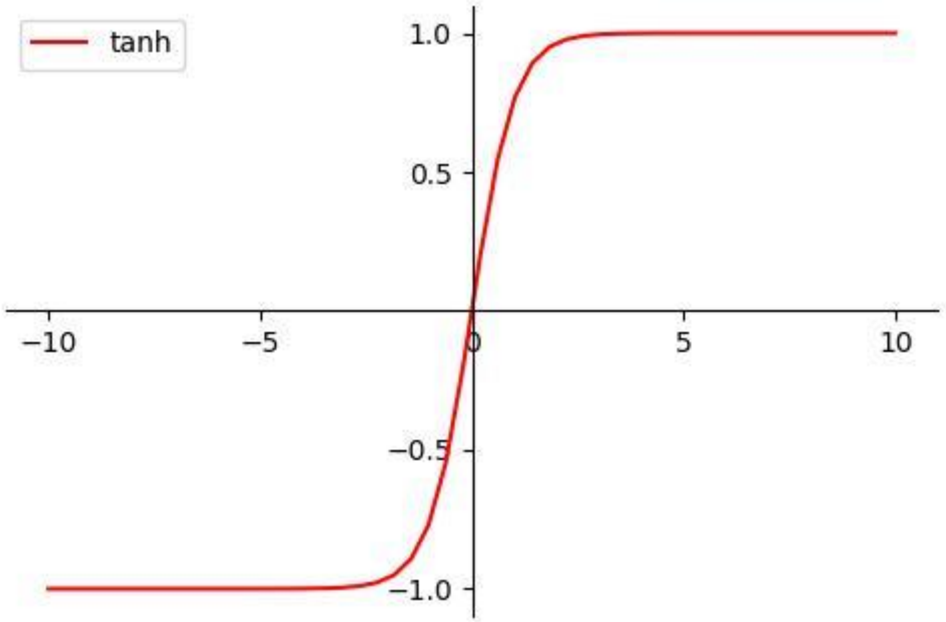
当$\sigma(z)$小于0.5时，预测 y=0

3.激活函数

tanh函数

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

$$\frac{d}{dz} g(z) = 1-(tanh(z))^{2}$$



tanh函数是**sigmoid**的向下平移和伸缩后的结果。对它进行了变形后，穿过了(0,0)点，并且值域介于+1和-1之间。

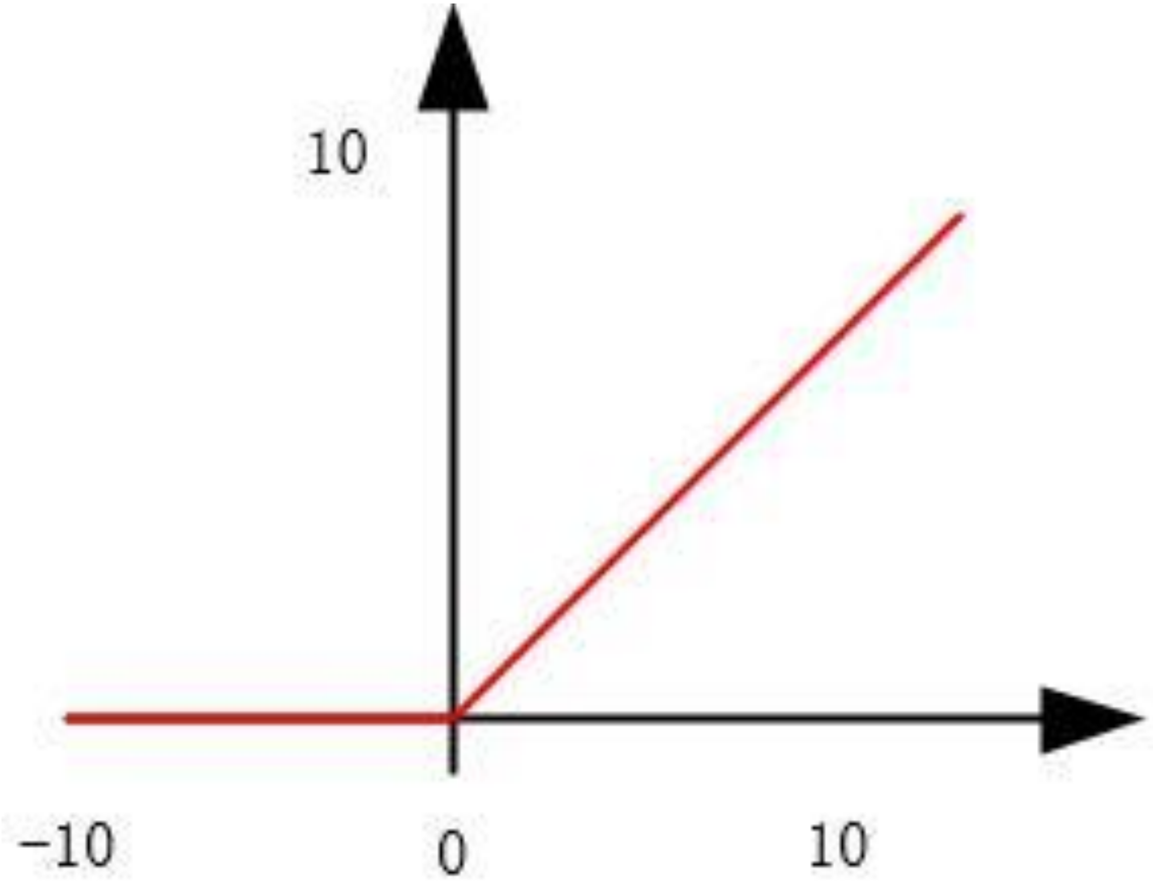
tanh函数是总体上都优于**sigmoid**函数的激活函数。

3.激活函数

ReLu函数

$$a = max(0, z)$$

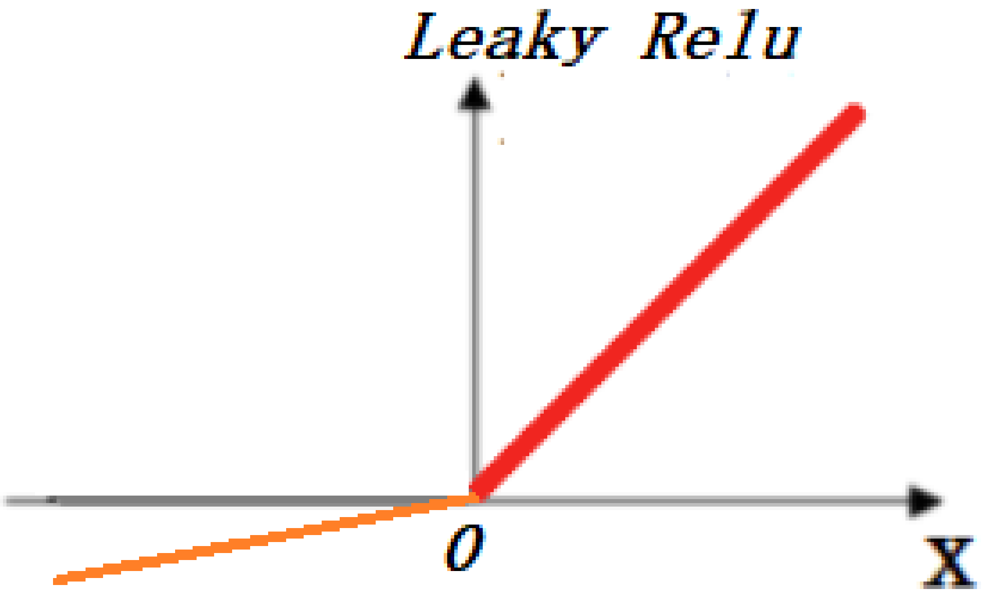
在输入是负值的情况下，它会输出0，那么神经元就不会被激活。这意味着同一时间只有部分神经元会被激活，从而使得网络很稀疏，进而对计算来说是非常有效率的。



3.激活函数

Leaky ReLu函数

$$a = max(0.01z, z)$$



Leaky ReLu通常比**Relu**激活函数效果要好，

尽管在实际中**Leaky ReLu**使用的并不多。

3.激活函数的使用场景

Sigmoid激活函数：除了输出层是一个二分类问题基本不会用它。

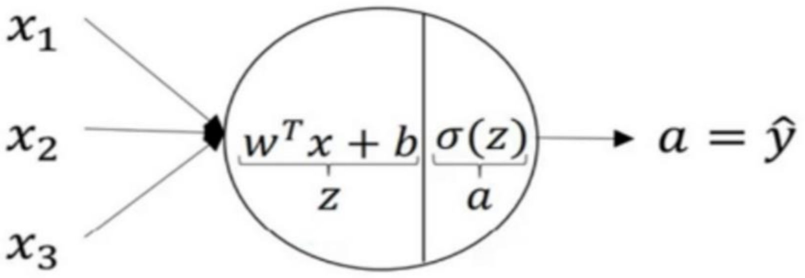
Tanh激活函数：**tanh**是非常优秀的，几乎适合所有场合。

ReLu激活函数：最常用的默认函数，，如果不确定用哪个激活函数，就使用**ReLu**或者**Leaky ReLu**。

4.反向传播算法

4.反向传播算法

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



$$\left.\begin{matrix}x\\w\\b\end{matrix}\right\} \Longleftarrow z = w^T x + b$$

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

$$dw = dz\cdot x, db = dz$$

$$\Longleftarrow \alpha = \sigma(z) \Longleftarrow L(a,y)$$

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

4.反向传播算法

因为$\frac{dL(a,y)}{dz}=\frac{dL}{dz}=(\frac{dL}{da})\cdot(\frac{da}{dz})$

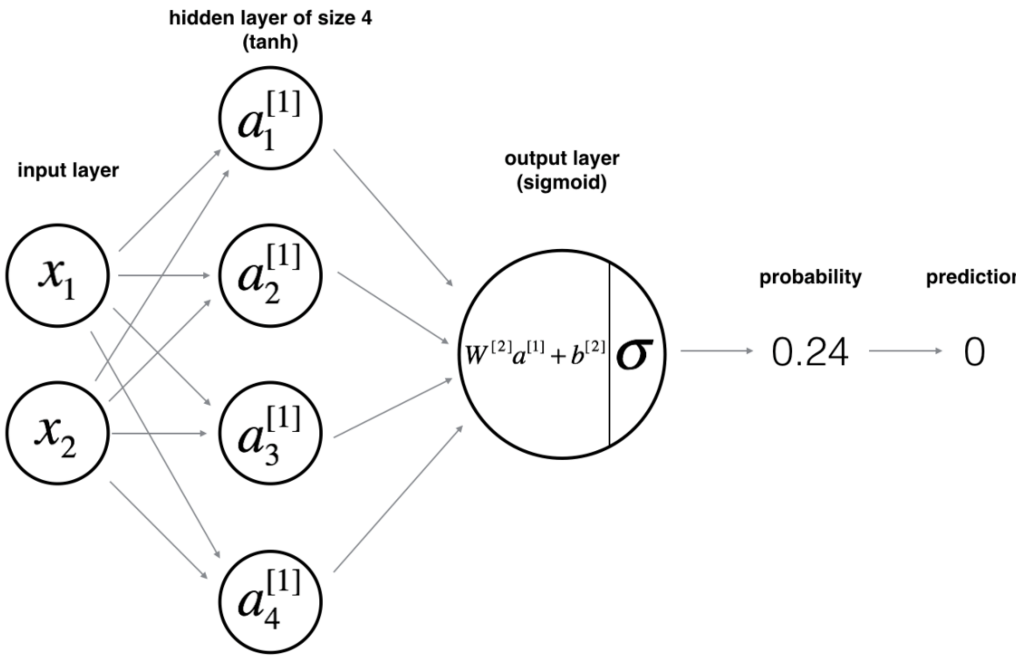
并且$\frac{da}{dz}=a\cdot(1-a)$,

而$\frac{dL(a,y)}{da}=\frac{dL}{da}=(-\frac{y}{a}+\frac{(1-y)}{(1-a)})$

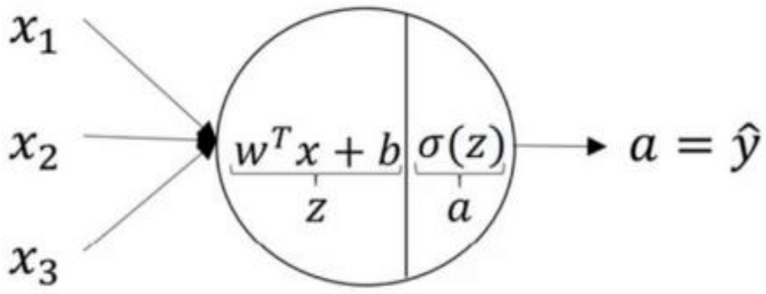
因此将这两项相乘，得到:

$$dz=\frac{dL(a,y)}{dz}=\frac{dL}{dz}=\left(\frac{dL}{da}\right)\cdot\left(\frac{da}{dz}\right)$$

$$=(-\frac{y}{a}+\frac{(1-y)}{(1-a)})\cdot a(1-a)=a-y$$



4.反向传播算法



$z = w^Tx + b$

$a = \sigma(z)$

第一步，计算$z_1^{[1]}, z_1^{[1]} = w_1^{[1]T}x + b_1^{[1]}$。

第二步，通过激活函数计算$a_1^{[1]}, a_1^{[1]} = \sigma(z_1^{[1]})$。

$a_2^{[1]}$、$a_3^{[1]}$、$a_4^{[1]}$的计算方式跟$a_1^{[1]}$相似

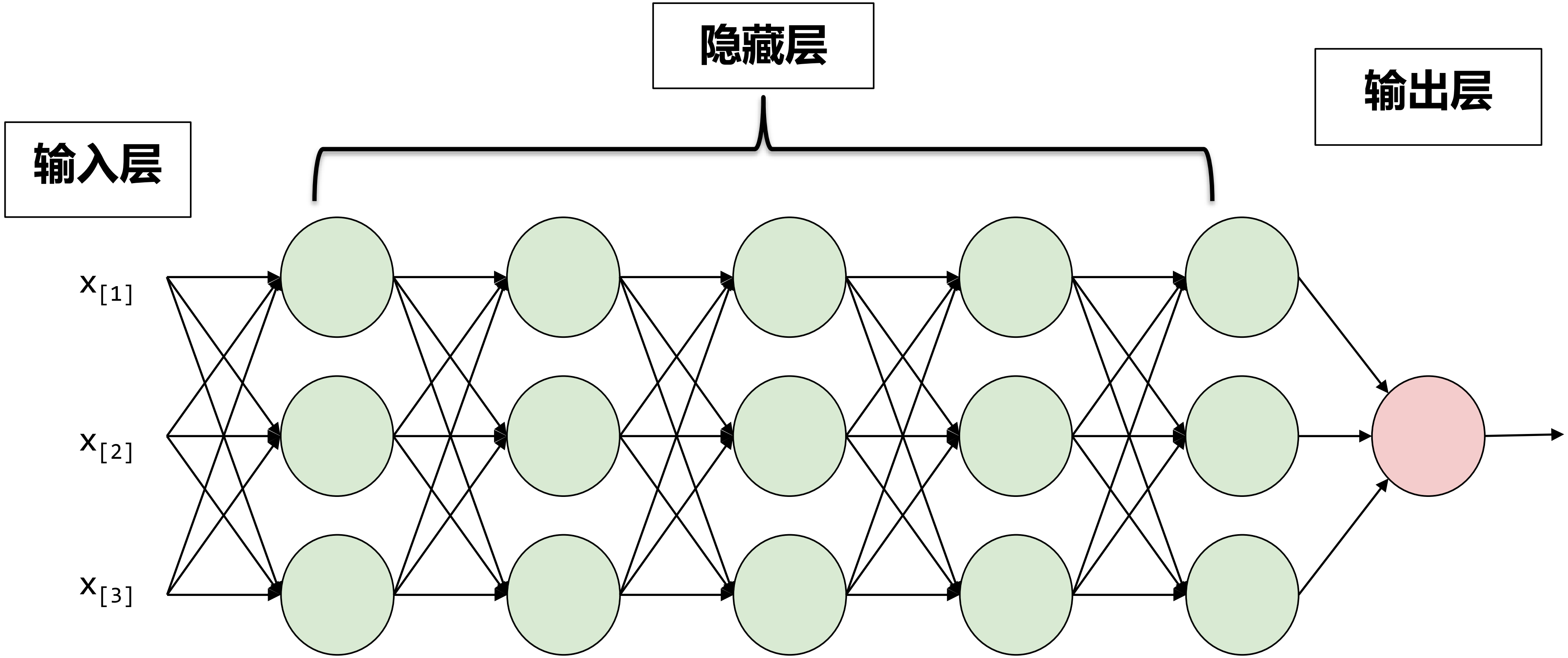
$$z_1^{[1]} = w_1^{[1]T}x + b_1^{[1]}, a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T}x + b_2^{[1]}, a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T}x + b_3^{[1]}, a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T}x + b_4^{[1]}, a_4^{[1]} = \sigma(z_4^{[1]})$$

4.反向传播算法



前向传播：

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

4.反向传播算法

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：

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

4.反向传播算法

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

$dz^{[1]} = W^{[2]T}dz^{[2]} * g[1]'(z^{[1]})$ 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：

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。

4.反向传播算法

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

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。

4.反向传播算法

主要的推导过程： $dZ^{[2]} = A^{[2]} - Y$ ， $dW^{[2]} = \frac{1}{m} dZ^{[2]} {A^{[1]}}^T$

$$L = \frac{1}{m} \sum_i^m L(\hat{y}, y)$$

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

$$\underbrace{dZ^{[1]}}_{(n^{[1]},m)} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]},m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]},m)}$$

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

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

4.反向传播算法

总结

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

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

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

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

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

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

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

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

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

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

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

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

参考文献

1. IAN GOODFELLOW等，《深度学习》，人民邮电出版社，2017
2. Andrew Ng，http://www.deeplearning.ai

谢 谢！