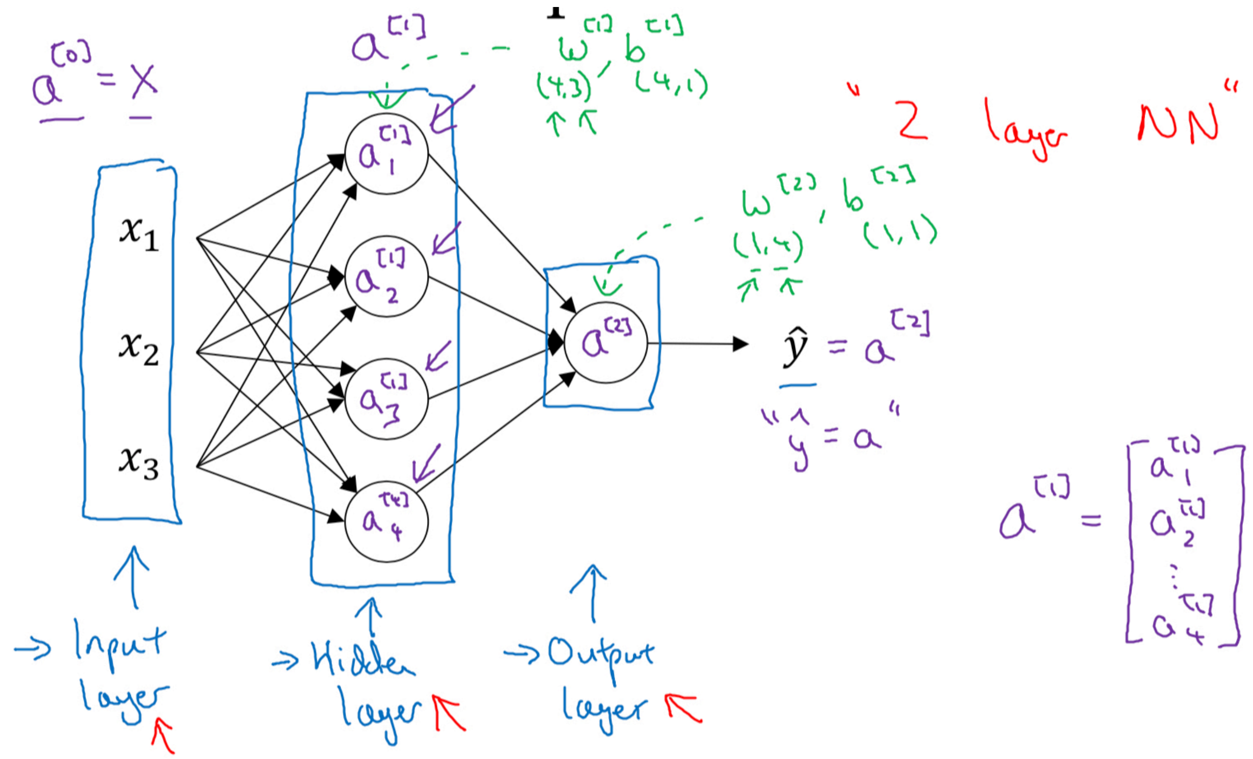
这一周的内容主要是以一个2层神经网络为列，重点讲了如何用向量/矩阵进行前向/反向传播计算。

1. 神经网络表示

一个简单(2层神经网络)的示意图：



说明：

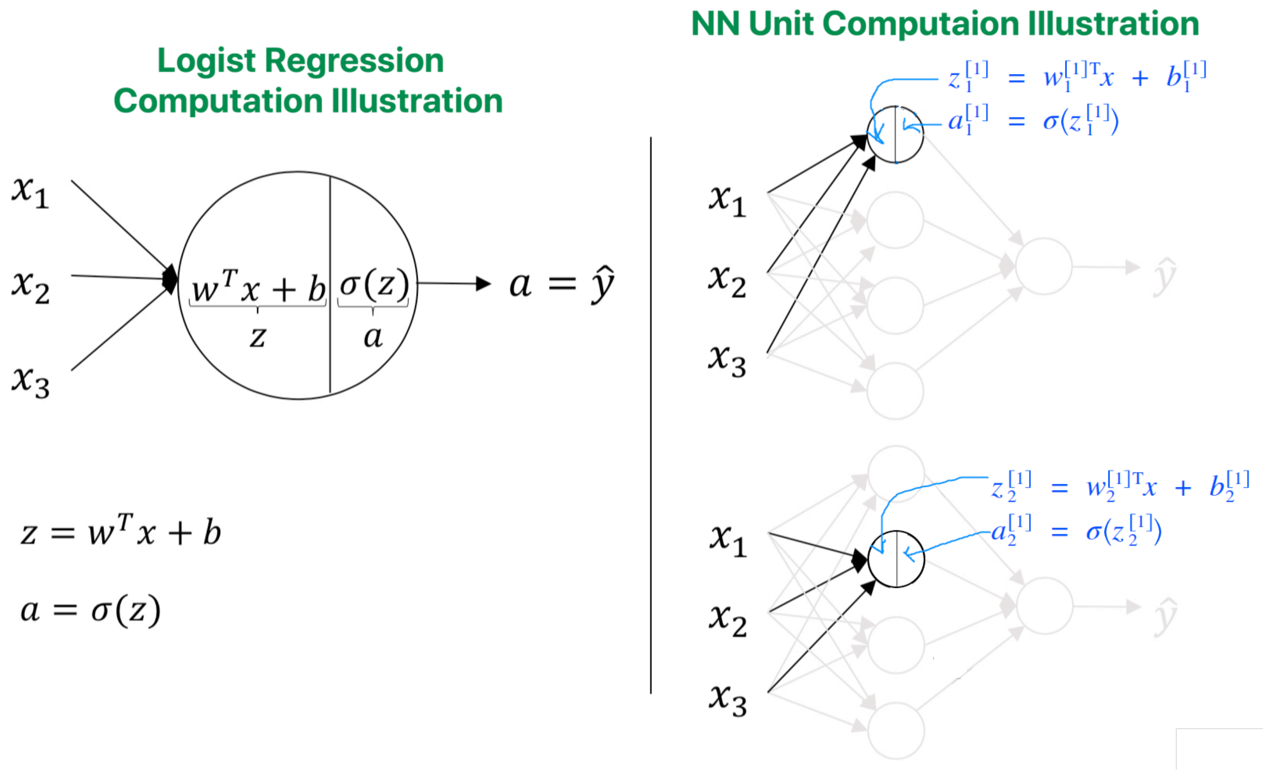
1. 网络结构分Input layer、Hidden layer和Output layer. 但一般地，我们称之为"2 layer NN"，不把Input layer 考虑进去；
2. Input layer —> Hidden layer的权重矩阵和偏置矩阵：
   - $w^{[1]} \in \mathbb{R}^{4\times3}$;
   - $b^{[1]} \in \mathbb{R}^{4\times1}$
3. Hidden layer —> Output layer:
   - $w^{[2]} \in \mathbb{R}^{1\times4}$
   - $b^{[2]} \in \mathbb{R}^{1\times1}$

2. Forward Propagation

上面给出了一个简单的神经网络结构，接下来该如何定义参数矩阵进行计算呢?

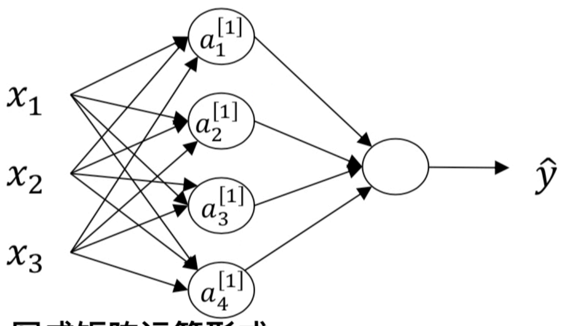
Step1. 考虑单个神经元

单个神经元的计算与Logistic Regression无异，其实整个Neural Network的计算也不过单个unit计算的重复！



可以看到每个神经元的计算的和Logistic Regression是一样的。

Step2. 推广至整个Hidden Layer的计算



$$
\begin{aligned}
z_1^{[1]} &= w_1^{[1]T}x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]}) \\
z_2^{[1]} &= w_2^{[1]T}x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]}) \\
z_3^{[1]} &= w_3^{[1]T}x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]}) \\
z_4^{[1]} &= w_4^{[1]T}x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})
\end{aligned}
$$

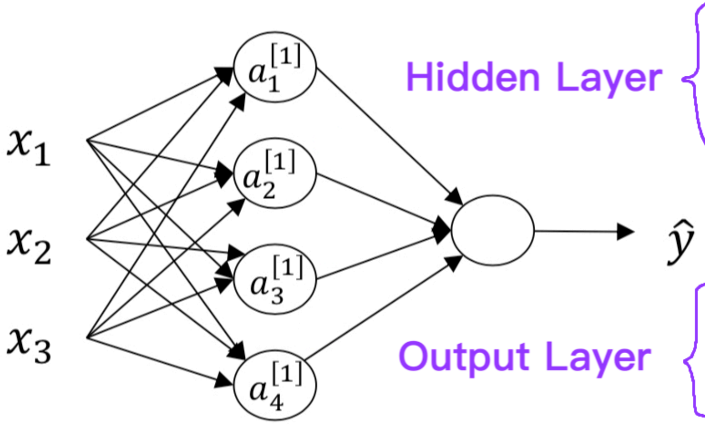
写成矩阵运算形式

$$
z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \underbrace{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}}_{w_1^{[1]}\ (4 \times 3)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T}x + b_1^{[1]} \\ w_2^{[1]T}x + b_2^{[1]} \\ w_3^{[1]T}x + b_3^{[1]} \\ w_4^{[1]T}x + b_4^{[1]} \end{bmatrix}
$$

$$
a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})
$$

Step3. 同时考虑Output Layer

$$
a^{[0]} = x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}
$$



Hidden Layer

$$
\underset{(4,1)}{z^{[1]}} = \underset{(4,3)}{W^{[1]}}\,\underset{(3,1)}{a^{[0]}} + \underset{(4,1)}{b^{[1]}}
$$

$$
\underset{(4,1)}{a^{[1]}} = \underset{(4,1)}{\sigma(z^{[1]})}
$$

Output Layer

$$
\underset{(1,1)}{z^{[2]}} = \underset{(1,4)}{W^{[2]}}\,\underset{(4,1)}{a^{[1]}} + \underset{(1,1)}{b^{[2]}}
$$

$$
\underset{(1,1)}{a^{[2]}} = \underset{(1,1)}{\sigma(z^{[2]})}
$$

Step4. 扩展到多个训练样本

Step1~Step3都是适用于当个样本，那对于m个样本呢?

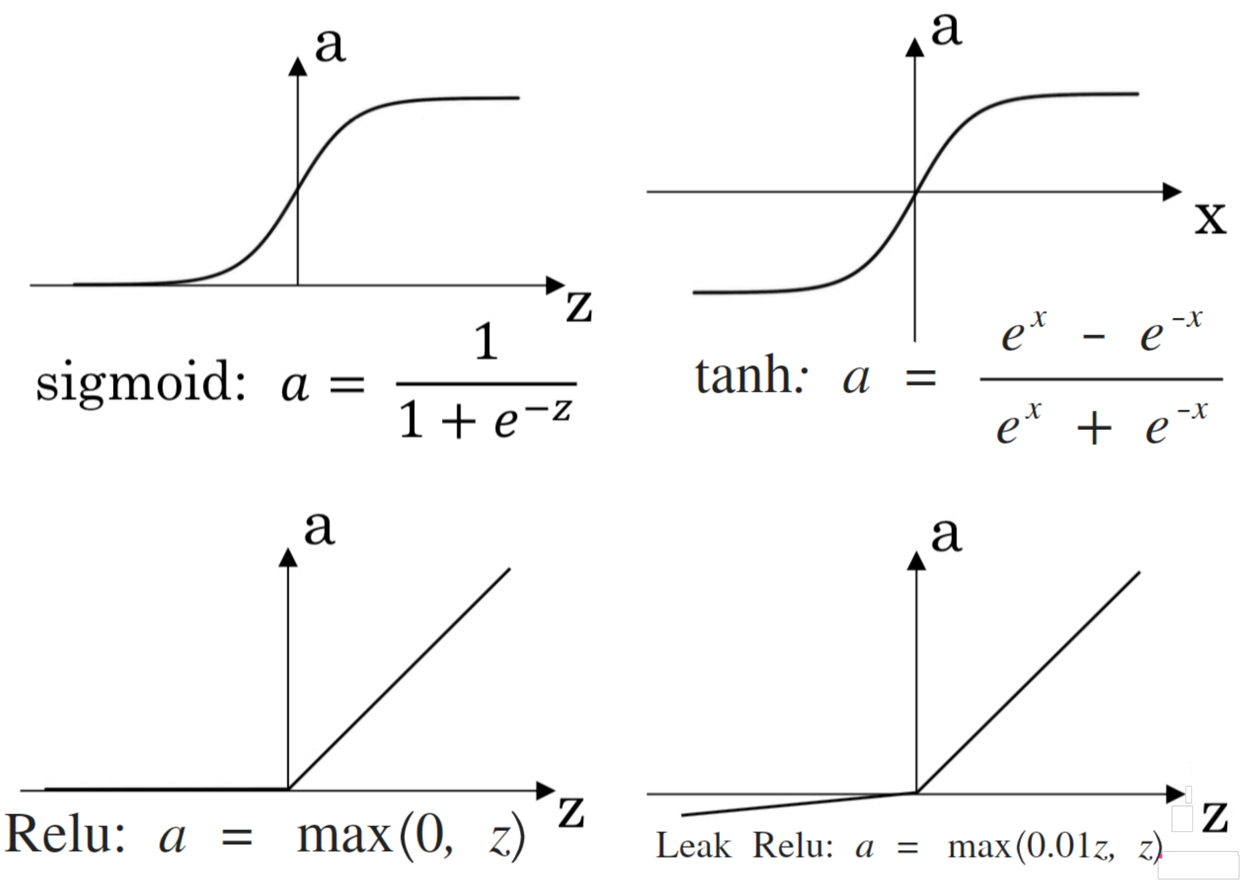
for 循环	向量化
for i = 1 to m:	$Z^{[1]} = W^{[1]}X + b^{[1]}$
$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$	$A^{[1]} = \sigma(Z^{[1]})$
$a^{[1](i)} = \sigma(z^{[1](i)})$	$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$
$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$	$A^{[2]} = \sigma(Z^{[2]})$
$a^{[2](i)} = \sigma(z^{[2](i)})$	

$$X = [x^{(1)}\ x^{(2)} \cdots x^{(m)}]_{(n_x, m)}$$

$$Z^{[1]} = \left.\underbrace{\begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}}_{m}\right\}\text{\# of hidden units}$$

3. 激活函数

3.1 几种常用激活函数



sigmoid函数和tanh函数比较：

- 隐藏层：tanh函数的表现要好于sigmoid函数，因为tanh取值范围为$[-1,+1]$，输出分布在0值的附近，均值为0，从隐藏层到输出层数据起到了归一化（均值为0）的效果。
- 输出层：对于二分类任务的输出取值为{0,1}，故一般会选择sigmoid函数。

然而sigmoid和tanh函数在$|z|$很大的时候，梯度会很小，在依据梯度的算法中，更新在后期会变得很慢。在实际应用中，要使$|z|$尽可能的落在0值附近。

ReLU弥补了前两者的缺陷，当$z>0$时，梯度始终为1，从而提高神经网络基于梯度算法的运算速度。然而当$z<0$时，梯度一直为0，但是实际的运用中，该缺陷的影响不是很大。

Leaky ReLU保证在$z<0$的时候，梯度仍然不为0。

在选择激活函数的时候，如果在不知道该选什么的时候就选择ReLU，当然也没有固定答案，要依据实际问题在交叉验证集合中进行验证分析。

3.2 激活函数的导数

1. Sigmoid

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

2. tanh

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

3. Relu

$$a = g(z) = \max(0, z)$$
$$g'(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

4. Leaky Relu

$$a = g(z) = \max(0.01z, z)$$
$$g'(z) = \begin{cases} 1 & z \geq 0 \\ 0.01 & z < 0 \end{cases}$$

4. GD for Neural Network

需要更新的参数有：

$$\underset{(n^{[1]}, n^{[0]})}{W^{[1]}}, \quad \underset{(n^{[1]}, 1)}{b^{[1]}}, \quad \underset{(n^{[2]}, n^{[1]})}{W^{[2]}}, \quad \underset{(n^{[2]}, 1)}{b^{[2]}}$$

$n_x = n^{[0]}$ $\quad$ $n^{[1]}$ 第一层神经元个数

损失函数记为：

$$J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

用矩阵表示的前向/后向传播可表示为：

Forward propagation:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$

X	(n_x,m)	$n_x=n^{[0]}$
$W^{[1]}$	$(n^{[1]},n_x)$	
$b^{[1]}$	$(n^{[1]},1)$	
$Z^{[1]}$	$(n^{[1]},m)$	
$A^{[1]}$	$(n^{[1]},m)$	
$W^{[2]}$	$(n^{[2]},n^{[1]})$	
$b^{[2]}$	$(n^{[2]},1)$	
$Z^{[2]}$	$(n^{[2]},m)$	
$A^{[2]}$	$(n^{[2]},m)$	

Backward propagation:

$$dZ^{[2]} = A^{[2]} - Y \quad (1,m)$$

$$\underset{(n^{[2]},n^{[1]})}{dW^{[2]}} = \frac{1}{m}dZ^{[2]}A^{[1]T}$$

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

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

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

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

5. 随机初始化

隐藏层的$W^{[1]}$不能全部初始化为0，如果都初始化为0，可以理解隐藏层的所有神经元函数都是一样的。

```
# 假设X为二维向量，Hidden Layer有2个神经元
W = np.random.rand((2,2))* 0.01
b = np.zero((2,1))
```

这里我们将W的值乘以0.01是为了尽可能使得权重W初始化为较小的值，这是因为如果使用sigmoid函数或者tanh函数作为激活函数时，W比较小，则$Z = WX + b$所得的值也比较小，处在0的附近，0点区域的附近梯度较大，能够大大提高算法的更新速度。而如果W设置的太大的话，得到的梯度较小，训练过程因此会变得很慢。

ReLU和Leaky ReLU作为激活函数时，不存在这种问题，因为在大于0的时候，梯度均为1。