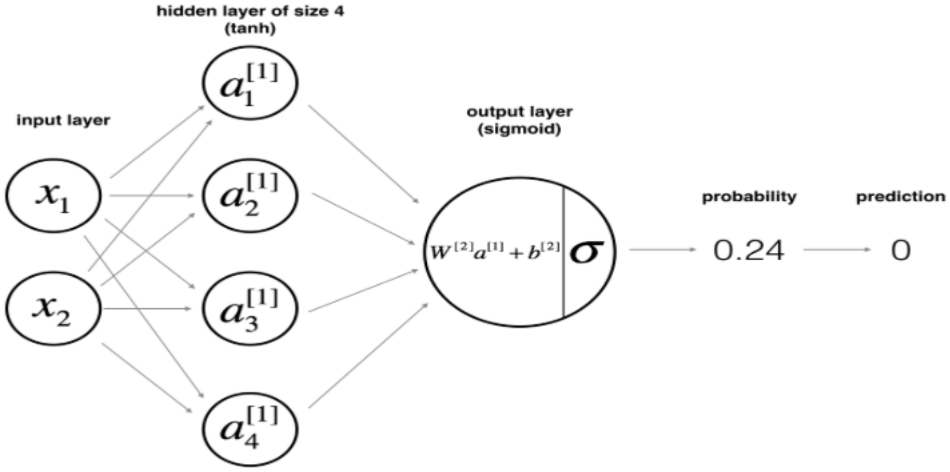
（一）双层神经网络表示



x_1 x_2 x_3：输入层A[0]，指的是单个样本的输入值

中间四个神经元：隐藏层$A^{[1]}$

右侧的单个神经元：输出层$A^{[2]}$

在单次训练过程中，首先训练样本分别对隐藏层的各神经元的参数（w向量和b值）进行计算得到$z^{[1]}$，各神经元的z放到一起组成$Z^{[1]}$，$z^{[1]}$激活后得到a，各神经元的a放到一起组成$A^{[1]}$。

各神经元的$A^{[1]}$再作为训练样本对对输出层的单个神经元的参数（w向量和b值）进行计算得到$z^{[2]}$，$z^{[2]}$激活得到$a^{[2]}$，也就等于是yhat。这是正向传播。

然后进行反向传播，从输出结果到第二层到第一层依次计算对成本函数的导数，达到对各个w、b的迭代、训练效果。

直观上来看，每一次训练实际上是：训练样本对第一层神经元进行训练，得到四个神经元的判断结果。这四个判断结果再对第二层神经元进行训练，得到第一层四个神经元的判断权重（也就是他们四个谁的判断更有说服力），这个权重记录在第二层神经元的w参数里。

第一层神经元的w向量维数 = 训练样本特征向量维数， 第二层神经元的w向量维数 = 第一层神经元的个数， 第n+1层神经元的w向量维数 = 第n层神经元的个数。 像这样层层训练，就可以得到比单个神经元更合理、更灵活的模型，得到更准确的预测结果。

（二）双层神经网络向量化计算流程

训练样本集：X = [$x^{(1)}$,$x^{(2)}$,$x^{(3)}$, ... ,$x^{(m)}$]，其中$x^{[i]}$是训练样本的特征列向量

第一层神经元的w参数集：

$$W^{[1]}=\begin{bmatrix} w_1^{(1)} & w_2^{(1)} & w_3^{(1)} & \cdots & w_{n_x}^{(1)} \\ w_1^{(2)} & w_2^{(2)} & w_3^{(2)} & \cdots & w_{n_x}^{(2)} \\ w_1^{(3)} & w_2^{(3)} & w_3^{(3)} & \cdots & w_{n_x}^{(3)} \\ \vdots & \vdots & \vdots & & \vdots \\ w_1^{(m)} & w_2^{(m)} & w_3^{(m)} & \cdots & w_{n_x}^{(m)} \end{bmatrix}$$

第一层神经元的b参数集：

$$b^{[1]}=\begin{bmatrix} b^{(1)} \\ b^{(2)} \\ b^{(3)} \\ \vdots \\ b^{(m)} \end{bmatrix}$$

###由于第二层只有一个神经元，所以虽然用大写字母W表示，但实际上是一个向量###

第二层神经元的w参数集：

$$W^{[2]}=[w_1 \quad w_2 \quad w_3 \quad w_4]$$

第二层神经元的b参数集：

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

激活函数：

$$g^{[1]}(x)=undefined$$
$$g^{[2]}(x)=undefined$$

计算步骤：

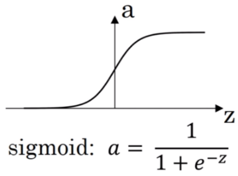
$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]})
\end{aligned}
$$

$X.shape = (n_x, m) W^{[1]}.shape = (n_h, n_x) Z^{[1]}.shape = (n_h, m) A^{[1]}.shape = (n_h, m) W^{[2]}.shape = (n_y, n_h) Z^{[2]}.shape = (n_y, m) A^{[2]}.shape$

（三）激活函数

1. sigmoid：只可能用于二元分类的输出层。

$$
a = \frac{1}{1+e^{-z}}
$$
$$
\frac{da}{dz} = a(1-a)
$$



2. tanh：几乎在所有情况下优于sigmoid函数。（计算速度更快）

$$
a = \frac{e^z - e^{-z}}{e^z + e^{-z}}
$$
$$
\frac{da}{dz} = 1 - a^2
$$

3. ReLU(Rectified Linear Unit)：最常用的默认激活函数

$$
a = max(0, z)
$$
$$
\frac{da}{dz} = \begin{cases} 0, z < 0 \\ 1, z > 0 \\ undefined, z = 0 \end{cases}
$$

4. leaky ReLU：有人认为这个比ReLU好

$$
a = max(\alpha z, z), \alpha\ usually\ less\ than\ 1
$$
$$
\frac{da}{dz} = \begin{cases} \alpha, z < 0 \\ 1, z > 0 \\ undefined, z = 0 \end{cases}
$$

sigmoid函数的图像让我明白了：第二周实验作业中对输入像素作/=255处理是为了让z更接近0，使得da/dz更大，迭代的速度更快

（四）为什么要使用非线性激活函数？

如果使用线性激活函数，由以下推导以及数学归纳法：

$$
a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}
$$
$$
a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}
$$
$$
a^{[2]} = W^{[2]}(\underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}}) + b^{[2]}
$$
$$
= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}
$$
$$
= W'x + b'
$$

可得所有的隐藏层都没有效果了，多层网络最终等效于一层网络。

只有一种情况可能使用线性激活函数：在输出层。

（五）神经网络的梯度下降法

参数

训练样本维数：$n^{[0]}$隐藏层神经元个数：$n^{[1]}$输出层神经元个数：$n^{[2]} = 1 W^{[1]} : (n^{[1]}, n^{[0]}), b^{[1]} : (n^{[1]}, 1) W^{[2]} : (n^{[2]}, n^{[1]}), b^{[2]} : (n^{[2]}, 1)$成本函数：$J(W^{[1]}, l$

梯度下降

$$dW^{[i]} = \frac{\partial J}{\partial W^{[i]}}, db^{[i]} = \frac{\partial J}{\partial b^{[i]}} W^{[i]} = W^{[i]} - \alpha dW[i] b^{[i]} = b^{[i]} - \alpha db[i] i = 1, 2$$

全过程公式

$$\#\#Forward\ Propagation\#\#Z^{[1]} = W^{[1]}X + b^{[1]}A^{[1]} = g^{[1]}(Z^{[1]})Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}A^{[2]} = g^{[2]}(Z^{[2]})\#\#Backward\ Propagation\#\#dZ^{[2]}$$

吴恩达梯度下降总结：

Summary of gradient descent

$dz^{[2]} = a^{[2]} - y$	$dZ^{[2]} = A^{[2]} - Y$
$dW^{[2]} = dz^{[2]}a^{[1]^T}$	$dW^{[2]} = \frac{1}{m}dZ^{[2]}A^{[1]^T}$
$db^{[2]} = dz^{[2]}$	$db^{[2]} = \frac{1}{m}np.sum(dZ^{[2]}, axis = 1, keepdims = True)$
$dz^{[1]} = W^{[2]T}dz^{[2]} * g^{[1]\prime}(z^{[1]})$	$dZ^{[1]} = W^{[2]T}dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$
$dW^{[1]} = dz^{[1]}x^T$	$dW^{[1]} = \frac{1}{m}dZ^{[1]}X^T$
$db^{[1]} = dz^{[1]}$	$db^{[1]} = \frac{1}{m}np.sum(dZ^{[1]}, axis = 1, keepdims = True)$

（六）为什么要对W随机初始化？

> 双层神经网络表示：
>
> 直观上来看，每一次训练实际上是：训练样本对第一层神经元进行训练，得到四个神经元的判断结果。这四个判断结果再对第二层神经元进行训练，得到第一层四个神经元的判断权重（也就是他们四个谁的判断更有说服力），这个权重记录在第二层神经元的w参数里。

如果把W初始化为全部为0，那么第一层上的神经元训练后都将是相同的，其下一层的神经元对上一层的判断权重也是完全相同的，同时这一层的神经元也会是完全相同的。由归纳法，每一层上的神经元都是完全相同的。这样就丧失了多层神经网络的判断性能优势。

术语：

- 完全对称：节点计算完全一样的函数

另外，初始化时应该使W中的数字尽量小，以使得sigmoid或tanh计算导数时处于导数较大的区域，以保证迭代学习的速度