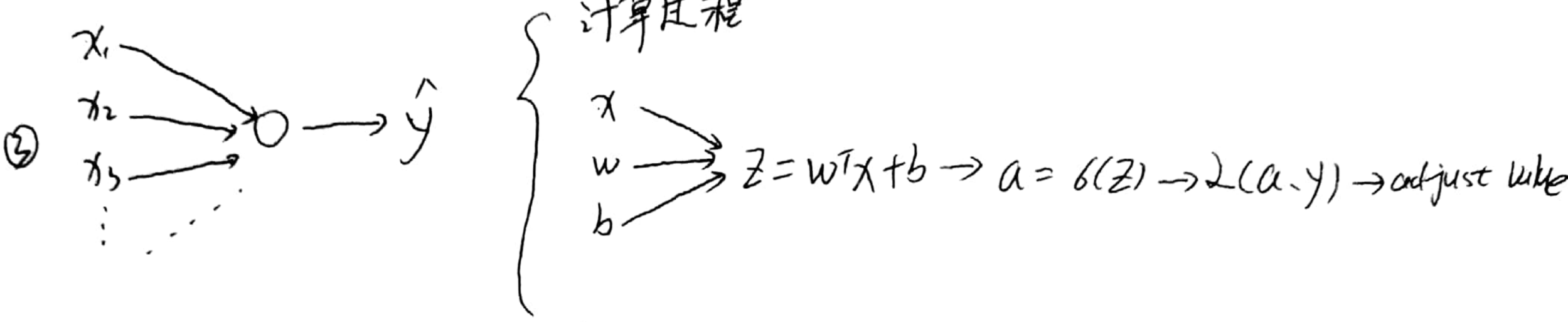
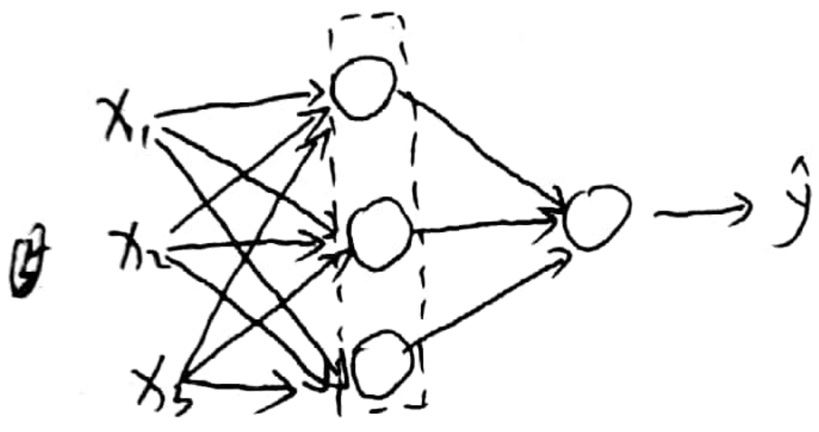
3.1 神经网络概览

①在这一节中(第三单元)我们将学习如何搭建一个神经网络模型。在这之前，我们先来看一下神经网络的大概，有些看不懂没关系，后面会有解释

②在上一周，我们学习了logistic回归，它的模型如下



③之前我们介绍过，神经网络模型像下面这样



④很明显，神经网络模型就是一个个logistic回归堆叠起来的，它的每个节点都包含两步运算

即①$z=w^Tx+b$ ②$a=\sigma(z)$

⑤之后我们会使用[ ]标记的方式来标识神经网络中的层级。

比如，在③图中虚线部分为神经网络第一层，有关它的计算可以表示为

用方括号来区分训练集中的样本表示。

$$z^{[1]}=W^{[1]}x+b^{[1]}$$

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

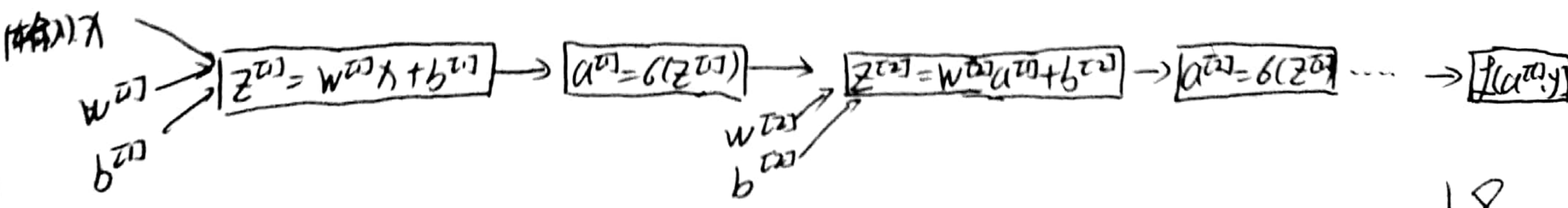
⑥在计算完成$a^{[1]}$之后，我们就可以计算下一层，即第[2]层

$$z^{[2]}=W^{[2]}a^{[1]}+b^{[2]} \qquad a^{[2]}=\sigma(z^{[2]})$$

⑦在所有层的运算结束之后，我们会计算损失函数

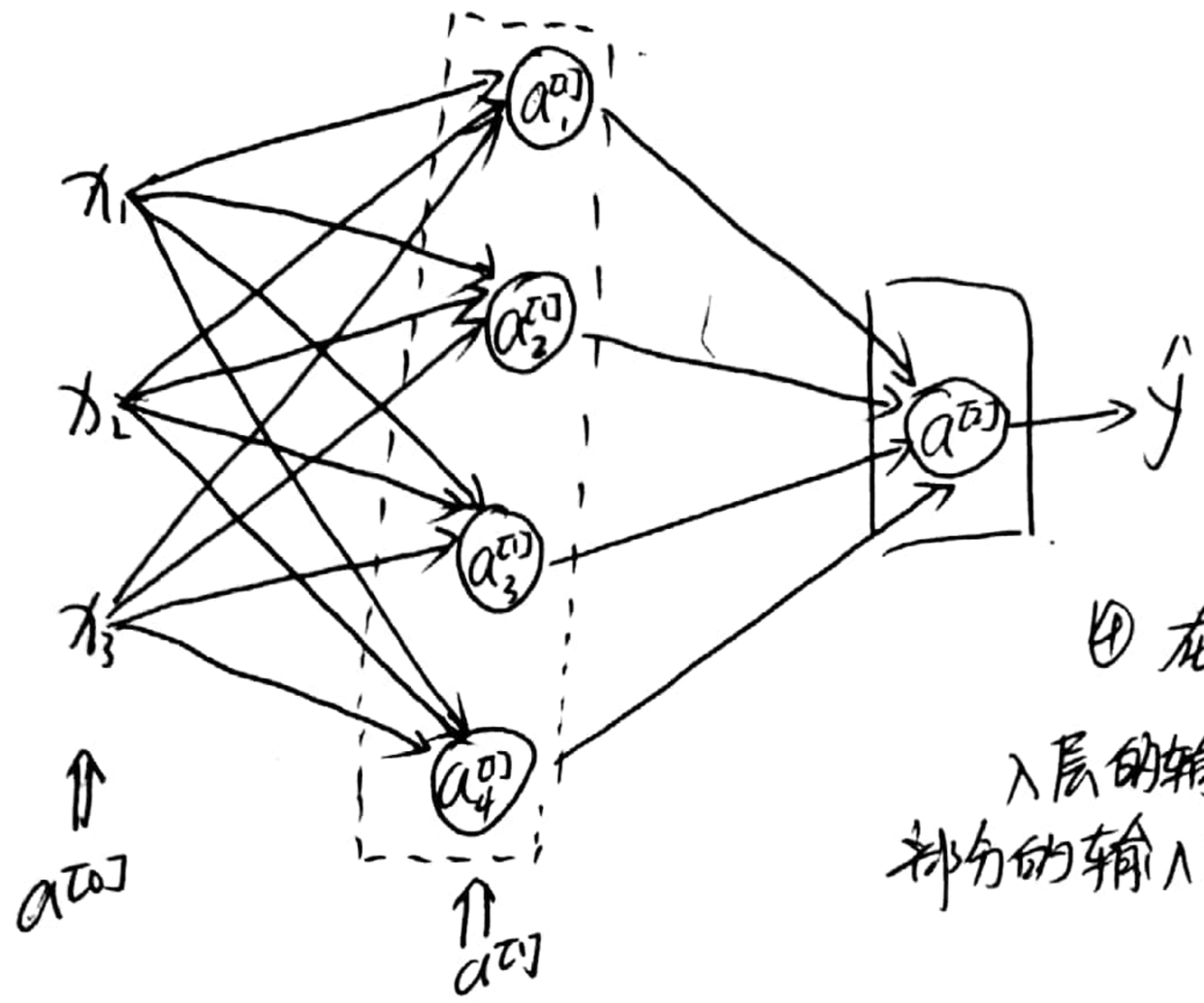
$$\mathcal{L}(a^{[last\ layer]},y)$$

那上面的运算可以用下面的流程图来表示。



3.2 神经网络表示

① 我们先来看一个只有一个隐藏层的神经网络.



① x_1、x_2、x_3 (输入层)

② ⬚ 隐藏层

③ □ 输出层

④ 在我们运行模型时，我们能看到输入层的输入，能看到输出层的输出，但是中间虚线部分的输入输出我们是看不到的，因此称为 - - —

⑤ 我们应该清楚，上面的每一层都是带有参数的，

即第一层有 $w^{[1]}$ 和 $b^{[1]}$.

第二层有 $w^{[2]}$ 和 $b^{[2]}$

⋮

⑥ 在上面的例子中，$w^{[1]}$ 是一个 4×3 矩阵，$b^{[1]}$ 是 4×1 矩阵

$w^{[1]}$ 是 4×3 矩阵，其 4 是因为第一层有四个节点

3 是因为有 3 个输入特征

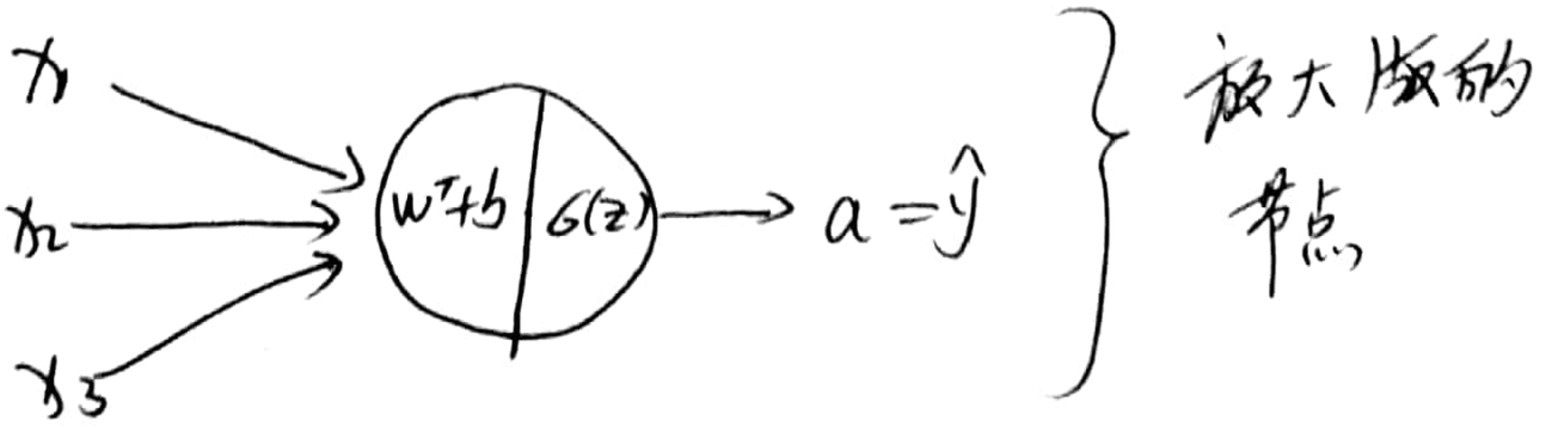
同理：$w^{[2]}$ 是一个 1×3 矩阵，$b^{[2]}$ 是 1×1 矩阵

- - - - — -

3.3 计算神经网络的输出

承接 3.2 中的图.

在 logistic 回归中，节点的运算如下.



对隐藏层第一个节点，计算过程如下

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

同理，对于第二个节点，计算过程如下：

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

} 和Logistic回归很像.

下面是第一个隐藏层的所有节点的计算公式

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

一个想法自然是使用for循环计算各节点值.
但使用向量化会使计算更加快速，让我们看看怎么向量化它们

要向量化，首先应该想，第一层的作用是什么？它最终生的数据格式应该是什么样的

① 第一层作用就是对输入层数据做一次操作，生输出作为下一层的输入.

既然是作为下一层的输入，那么格式应该和输入层数据格式一样

因此，最终，第一层的输出应该像下面这样

$$\begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \sigma\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix} \cdot x + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

因此，我们知道，在第一个隐藏

$w_1^{[1]} = \begin{bmatrix} w_{11} \\ w_{12} \\ w_{13} \end{bmatrix}$ 是3x1矩阵，而 $w_1^{[1]T}$ 就是1x3矩阵.

这样向量化的 $W^{[1]}$ 即为4x3矩阵

$$W^{[1]} = \begin{bmatrix} \cdots w_1^{[1]T} \cdots \\ \cdots w_2^{[1]T} \cdots \\ \cdots w_3^{[1]T} \cdots \\ \cdots w_4^{[1]T} \cdots \end{bmatrix}$$

向量化后的 b 为4x1矩阵

$$b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

也就是，在 $W^{[1]}$ 中，行数是节点数，列数是输入特征维度.

因此对于4.2节中的两层神经网络来说，计算步骤如下

$$z^{[1]}_{[4\times1]} = W^{[1]}_{[4\times3]} x_{[3\times1]} + b^{[1]}_{[4\times1]}$$

$$a^{[1]}_{[4\times1]} = \sigma(z^{[1]}_{[4\times1]})$$

$$z^{[2]}_{[1\times1]} = W^{[2]}_{[1\times4]} a^{[1]}_{[4\times1]} + b^{[2]}_{[1\times1]}$$

$$a^{[2]}_{[1\times1]} = \sigma(z^{[2]}_{[1\times1]})$$

没错，就是这么简单！

其中 $z^{[1]}$ (4×1) $W^{[1]}$ $[4\times3]$ x (3×1) $b^{[1]}$ (4×1)

$z^{[2]}$ (1×1) $W^{[2]}$ (1×4) $a^{[1]}$ (4×1) $a^{[2]}$ (1×1)

为了公式更加统一，也可以把输入层看作是 $a^{[0]}$，因此 x 即为 $a^{[0]}$。

3.4 多个例子中的向量化

在上一节中，我们看一下输入 x，了解到它是一个 3×1 的矩阵，很明显这是一个单一样本的输入。在之前章节中，对于单个节点 logistic 回归，我们已经做到了将多个 (m) 个样本专门装到一个向量里面 $(n_x \times m)$，这样做避免了显式的 for 循环，提升了运算效率。那么让我们看一下在多层神经网络中怎么将每个输入 x 封装起来，达到向量化的目的。

怎么向量化？先回忆一下 logistic 回归中的情况。

对于输入 $x^1, x^2 \cdots x^m$，我们期望的输出为 $y^1, y^2 \cdots y^m$，同时由于 $z^{[1]}$ 是 $(n_h \times 1)$ 矩阵，很明显我们不可能将 $z^{[1](i)}$ 封装成下面的样子

$$\begin{bmatrix} z^1 \\ z^2 \\ \vdots \\ z^m \end{bmatrix}$$

如果这样封装的话，区分 $z^1 - z^m$ 就成为一个很麻烦的事情。

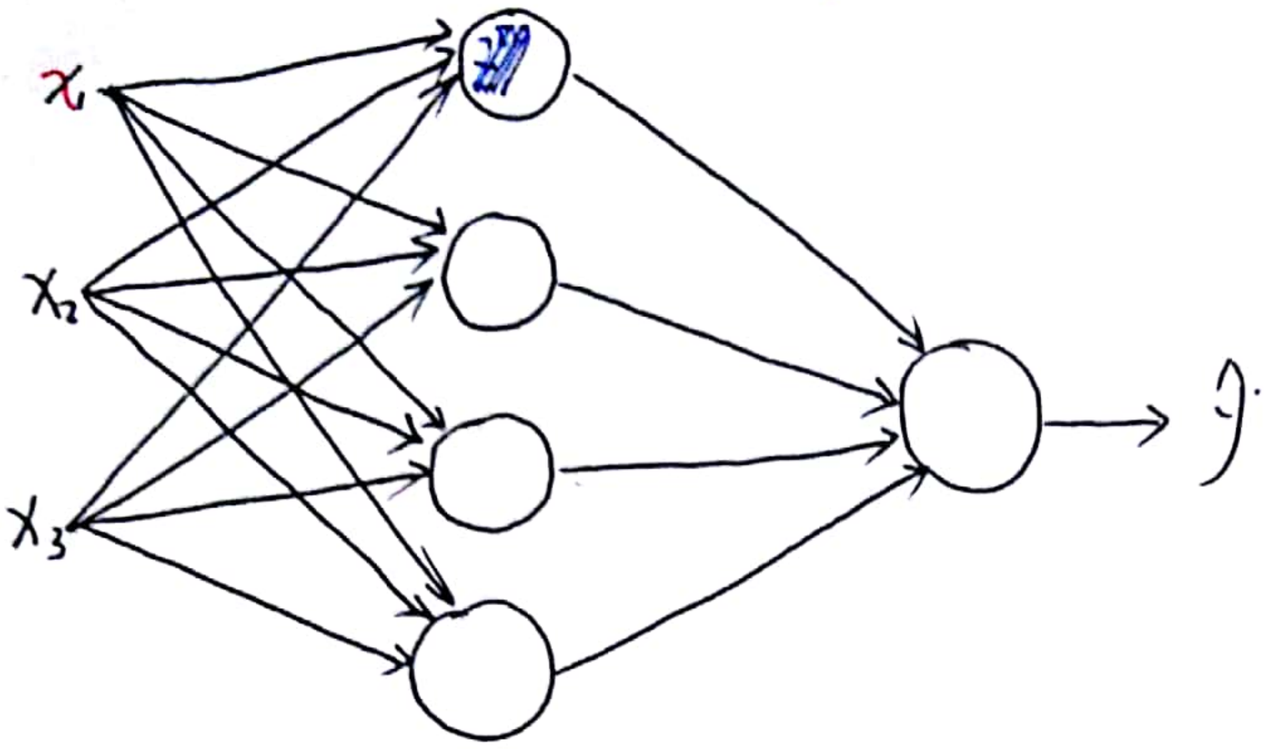
因此，理想的封装就应该和 logistic 回归中的封装方式一样。

$$X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

让我们看一下，这样封装后，

我们上面的两层神经网络的计算结果是什么？

在此之前，把之前的神经网络计算总结如下



$$z^{[1]} = W^{[1]}x + b^{[1]}$$

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

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

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

之后让我们用输入 X 来进行计算，先计算第一层。

$$z^{[1]} = W^{[1]}X + b^{[1]}$$

$$\begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix} \cdot \begin{bmatrix} X \end{bmatrix} + b^{[1]} \Rightarrow \begin{bmatrix} w_1^{[1]T}X \\ w_2^{[1]T}X \\ w_3^{[1]T}X \\ w_4^{[1]T}X \end{bmatrix} + b^{[1]}$$

我们来研究 $w_1^{[1]T}X$ 的结果是什么？

$$w_1^{[1]T} \cdot X \Rightarrow w_1^{[1]T} \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix} \Rightarrow \begin{bmatrix} w_1^{[1]T}x^{(1)} & w_1^{[1]T}x^{(2)} & \cdots & w_1^{[1]T}x^{(m)} \end{bmatrix}$$

其中 $w_1^{[1]T}x^{(1)}$ 就是第一层网络的第1个节点对输入第1个样本的输入 $z_1^{[1](1)}$

同此 $w_1^{[1]T}X \Rightarrow \begin{bmatrix} z_1^{[1](1)} & z_1^{[1](2)} & \cdots & z_1^{[1](m)} \end{bmatrix}$

因此可知

$$w^{[1]} \cdot X \Rightarrow \begin{bmatrix} z_1^{[1](1)} & z_1^{[1](2)} & \cdots & z_1^{[1](m)} \\ z_2^{[1](1)} & z_2^{[1](2)} & \cdots & z_2^{[1](m)} \\ z_3^{[1](1)} & z_3^{[1](2)} & \cdots & z_3^{[1](m)} \\ z_4^{[1](1)} & z_4^{[1](2)} & \cdots & z_4^{[1](m)} \end{bmatrix}$$

其中红线部分 $\begin{bmatrix} z_1^{[1](1)} \\ z_2^{[1](1)} \\ z_3^{[1](1)} \\ z_4^{[1](1)} \end{bmatrix}$ 即为对第一个样本的输出即 $z^{[1](1)}$

因此，最终的输出为

$$W^{[1]} \cdot X = [Z^{[1](1)} \; Z^{[1](2)} \; \cdots \; Z^{[1](m)}] = Z^{[1]} \quad (\text{大写})$$

因此最终的第一层输出为

$$\sigma(W^{[1]} X) = [a^{[1](1)} \; a^{[1](2)} \; \cdots \; a^{[1](m)}] = A^{[1]} \quad (\text{大写})$$

再来看第二层网络的输出(计算)

第二层计算公式如下：

$$Z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]} \quad \text{将 } a^{[1]} \text{ 换作 } A^{[1]}$$

有 $W^{[2]} \cdot A^{[1]} \Rightarrow W^{[2]} \cdot [a^{[1](1)} \; a^{[1](2)} \; \cdots \; a^{[1](m)}]$

有 $\Rightarrow [W^{[2]} \cdot a^{[1](1)} \quad W^{[2]} a^{[1](2)} \quad \cdots \quad W^{[2]} a^{[1](m)}]$

最终两处 $b^{[2]}$ 根据上面的公式有

$$W^{[2]} A^{[1]} + b^{[2]} \Rightarrow [Z^{[2](1)} \; Z^{[2](2)} \; \cdots \; Z^{[2](m)}] = Z^{[2]} \quad (\text{大写?})$$

最终有

$$\begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix} \Rightarrow [a^{[2](1)} \; a^{[2](2)} \; \cdots \; a^{[2](m)}] \Rightarrow [\hat{y}^{(1)} \; \hat{y}^{(2)} \; \cdots \; \hat{y}^{(m)}]$$

可以看到，和之前 logistic 回归中的输入输出形式是一样的。这也就印证我们的想法是正确的。

即 $X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$ 为 $n_x \times m$ 的矩阵。

因此，从上面的总结中，我们可以看到一个规律，对于各层的输出来说，

它的横向各个项是对应每个节点的输出

而 竖向各 项是对应每个训练样本对各节点的输出

就拿第一层的 $A^{[1]}$ 来说

$$A^{[1]} = \begin{bmatrix} a_1^{[1](1)} & a_1^{[1](2)} & \cdots & a_1^{[1](m)} \\ a_2^{[1](1)} & a_2^{[1](2)} & \cdots & a_2^{[1](m)} \\ a_3^{[1](1)} & a_3^{[1](2)} & \cdots & a_3^{[1](m)} \\ a_4^{[1](1)} & a_4^{[1](2)} & \cdots & a_4^{[1](m)} \end{bmatrix}$$

第一个节点 对 m 个样本的输出

第一个样本 4 个节点的输出

最终我们的神经网络计算公式如下：

$$Z^{[1]}_{(4\times m)} = W^{[1]}_{(4\times 3)} X_{(3\times m)} + b^{[1]}_{(4\times 1)}$$

$$A^{[1]}_{(4\times m)} = \sigma(Z^{[1]})_{(4\times m)}$$

$$Z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$\xrightarrow{X = A^{[0]}}$$

$$Z^{[2]}_{(1\times m)} = W^{[2]}_{(1\times 4)} A^{[1]}_{(4\times m)} + b^{[2]}_{(1\times 1)}$$

$$A^{[2]}_{(1\times m)} = \sigma(Z^{[2]})_{(1\times m)}$$

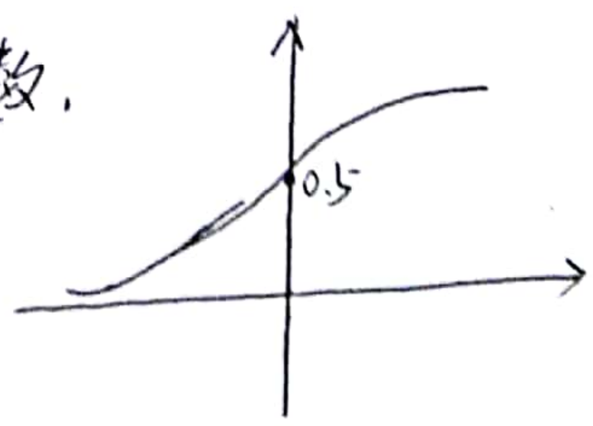
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

3.6 激活函数

在你搭建神经网络的时候，你可能会想使用什么激活函数，因为激活函数有很多种，到目前为止，我们使用的都是 Sigmoid 函数，

即 $\sigma(z) = \dfrac{1}{1+e^{-z}}$

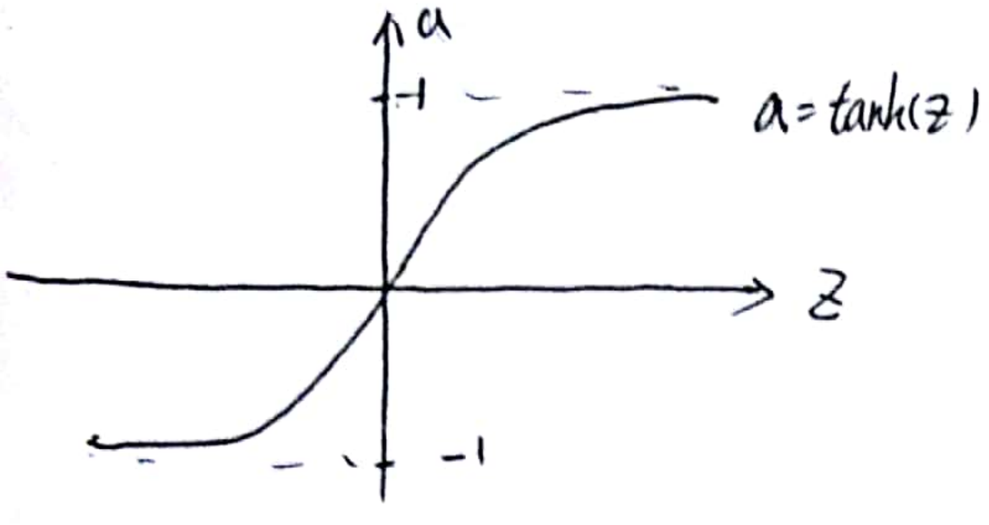


在上面提到的神经网络中，我们有两处用到了激活函数，即

$A^{[1]} = \sigma(Z^{[1]})$　　$A^{[2]} = \sigma(Z^{[2]})$

在一般情况下我们用 $g(z)$ 来表示激活函数.

$g(z)$ 不一定是 Sigmoid 函数.. 更多情况下它可以是 tanh 函数、即双曲正切函数



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

事实证明. 如果是隐藏单元、让 $g(z)=\tanh(z)$ 效果会更好.

因为此时激活函数 (-1,1) 之间. 它的平均值为零. 使用 tanh 函数

而不是 6 函数有类似数据中心化的效果. 使数据平均值更接近0而不是0.5

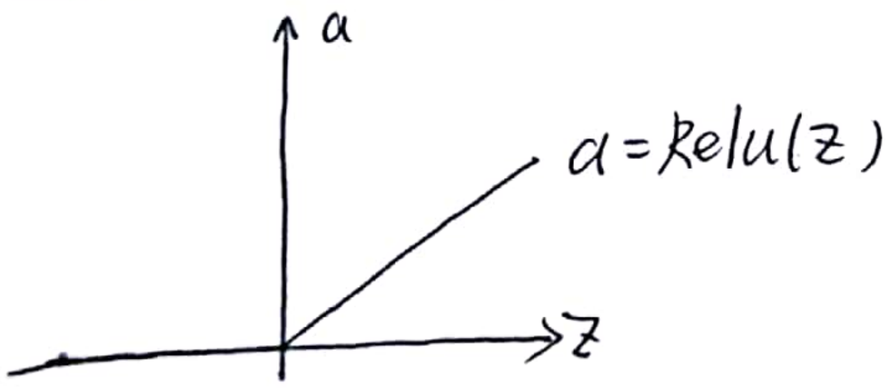
实际上让下一层的学习更方便一点。

tanh 在很多场合都比 6 函数更为优秀. 但有一个例外. 就是在输出层.

如果输出层考虑二分分类的话. 很明显使用 6 函数要更为适合(因为可以表示为概率

对于 tanh 函数来说、它和 Sigmoid 函数一样有一个共同的缺点. 就是当我们使用梯度下降法来进行计算的时候.. 当 z 非常大时. 效率会很低(因为斜率趋于零)

在机器学习领域最受欢迎的一个函数是 Relu 函数



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

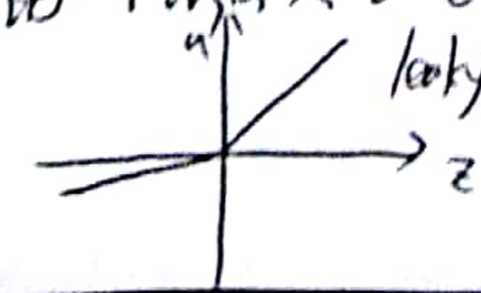
在选择激活函数时有一些经验法则

① 如果你的输出值是 0 和 1、那么 6 函数很适合做输出层的激活函数

② 之后其他单元用 Relu

③ 如果不确定隐藏层用什么激活函数. 就用 Relu 函数

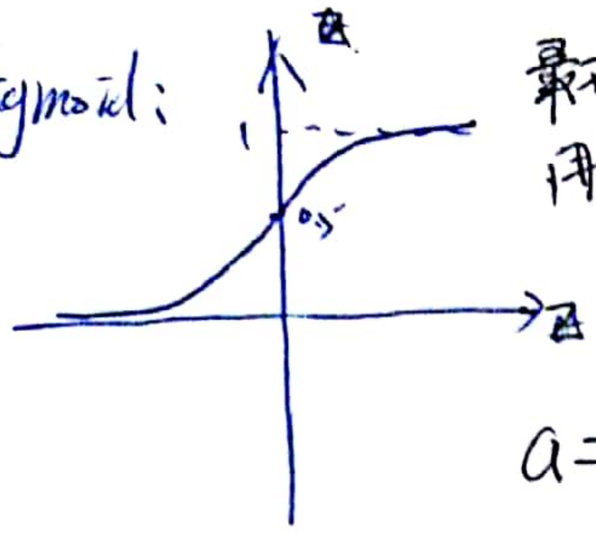
Relu 的一个缺点是在 $z<0$ 时. 导数为零. 因此也有一种带泄漏的 Relu、即 Leaky Relu



Leaky Relu. 不过这个函数实际中使用频率却不高

下面来看这四个函数

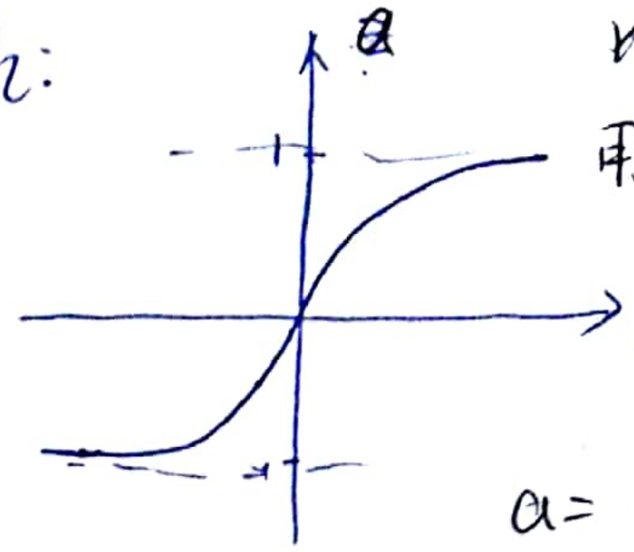
Sigmoid：



最不常用
用于二分分类

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

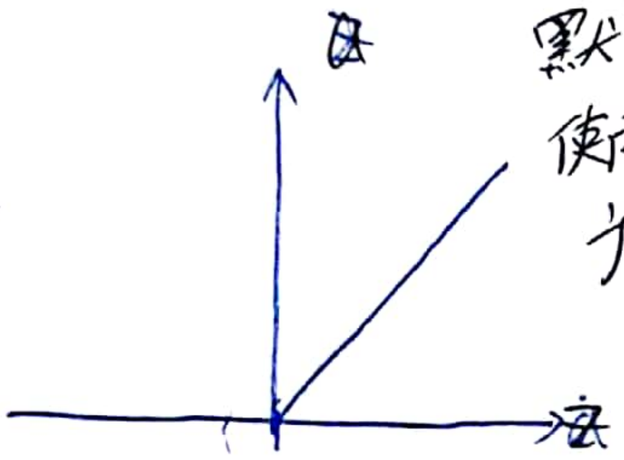
tanh：



比6更优秀
用6可以用它
来替换

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

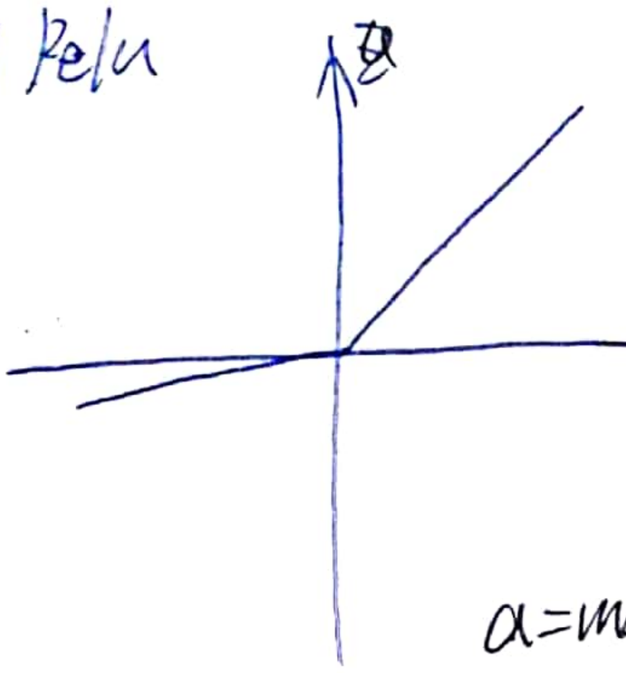
为ReLu



默认使用.
使用最为
广泛

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

Leaky ReLu



或许效果更
好，但很少
有人用

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

神经网络一个特点就是可以有很多不同的选择. 比如.

① 几个隐藏层

② 层中有几个隐藏节点

③ 使用什么样的激活函数

④ 怎么初始化节点参数（权重）

……

3.7. 为什么需要非线性修正单元（激活函数）

让我们来看看神经网络的计算过程

$$z^{[1]} = W^{[1]} \cdot X + b^{[1]}$$

$$a^{[1]} = g(z^{[1]})$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = g(z^{[2]})$$

那么，如果计算中的激活函数是线性的话会怎么样. 让我们以最简单的线性函数来为例

即 $g(z) = z$

这个函数也被称为 恒等激活函数.

让我们来看看会发生什么？

$z^{[1]} = W^{[1]} \cdot x + b^{[1]}$

$a^{[1]} = z^{[1]}$

$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$

$a^{[2]} = z^{[2]}$

那么有 $a^{[2]} = W^{[2]} x + b^{[2]}$

因为 $a^{[1]} = z^{[1]}$ 则有

$a^{[2]} = W^{[2]} \cdot (W^{[1]} x + b^{[1]}) + b^{[2]}$

$a^{[2]} = (W^{[2]} W^{[1]} / x + (W^{[2]} b^{[1]} + b^{[2]})$

我们可以看到上式中 $a^{[2]}$ 就相当于 $a^{[2]} = W' x + b'$

那么这和只有一节点的 logistic 回归有什么区别呢？

很明显毫无区别。因此：如果我们的激活函数都使用线性激活函数的话，那么隐藏层的存在就会变得毫无意义

因此只有引入非线性激活函数，才能让神经网络富于变化

否则，即使你有再多的隐藏层也是毫无意义

只有一种情况，就是在研究某些线性回归问题的时候可以使用，就比如第一节中研究的房价问题。

但其实在这个问题中我们使用的函数是 ReLU 函数更为适合。

3.8 激活函数的导数

sigmoid：

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

导数：$\frac{d}{dz} g(z) = \frac{1}{1+e^{-z}} \cdot (1 - \frac{1}{1+e^{-z}})$

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

在神经网络中，$g(z) = a$，因此，就

因此 $g'(z) = a \cdot (1-a)$

tanh.

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

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

因此在神经网络中 $a=g(z)$ 因此对tanh来说 $g'(z)=1-a^2$

对于Relu函数 不用过多解释.

$$g'(z)=\begin{cases}0 & \text{if } z<0\\ 1 & \text{if } z\geq 0\end{cases}$$

leaky Relu.

$$g'(z)=\begin{cases}0.01 & \text{if } z<0\\ 1 & \text{if } z\geq 0\end{cases}$$

$$\text{leaky Relu}(z)=\max(0.01z,\ z)$$

39. 神经网络的梯度下降法

在单隐层神经网络中 我们有如下参数.

$[W^{[1]},\ b^{[1]},\ W^{[2]},\ b^{[2]}]$

因此、要调整这些参数、使用梯度下降法. 就需要知道它们的导数

我们也用n来表示输出参数的维度.

例如 $n^{[1]}$ 代表输出X的维度. $n^{[1]}$ 代表第一层的输出维度

$n^{[2]}$ 代表第二层的输出维度

因此、对于第一层来说.

$W^{[1]}$的维度为 $(n^{[1]},\ n^{[0]})$ b为 $(n^{[1]},\ 1)$

对于第二层来说

$W^{[2]}$的维度为 $(n^{[2]},\ n^{[1]})$ b为 $(n^{[2]},\ 1)$

} 在只有一个隐藏层的单元来说. $n^{[0]}=1$

要使用梯度下降法. 就需要一个成本函数. 这里我们假定在做二分类

因此 Cost function $J(W^{[1]},\ W^{[2]},\ b^{[1]},\ b^{[2]})=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y},\ y)$ 很明显 $\hat{y}=a^{[2]}$

因此、根据三句话来看下面的计算

① 计算 $\hat{y}$、$\Rightarrow (\hat{y}^{(1)}\ \hat{y}^{(2)} \cdots \hat{y}^{(m)})$

② 计算 第一层参数导数、更新参数

$$dw^{[1]} = \frac{dJ}{dw^{[1]}} \quad 、 \quad db^{[1]} = \frac{dJ}{db^{[1]}}$$

$$dw^{[1]} \quad W^{[1]} = W^{[1]} - \alpha \cdot dw^{[1]}$$

$$b^{[1]} -= \alpha \cdot db^{[1]}$$

③ 计算第二层参数导数更新参数

$$dw^{[2]} = \frac{dJ}{dw^{[2]}} \qquad db^{[2]} = \frac{dJ}{db^{[2]}}$$

$$W^{[2]} -= \alpha dw^{[2]}$$

$$b^{[2]} -= \alpha db^{[2]}$$

④ 重复上面 ② — ③ 的计算

$$\begin{cases} Z^{[1]} = W^{[1]} X + b^{[1]} \\ A^{[1]} = g(Z^{[1]}) \\ Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]} \\ A^{[2]} = \sigma(Z^{[2]}) \\ \Uparrow \\ (\hat{y}^{(1)}, \hat{y}^{(2)} \cdots \hat{y}^{(m)}) \end{cases}$$

这里面最重要的就是计算

$dw^{[1]}$, $dw^{[2]}$, $db^{[1]}$, $db^{[2]}$

下面直接给出结论.

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

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

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

和logistic回归很像. 因为拿前者A输出 后的活. 就是一个logistic模型

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} \odot g'^{[1]}(Z^{[1]})$$

($W^{[2]T}$: $(n^{[1]}, m)$; $g'^{[1]}(Z^{[1]})$: $(n^{[1]}, m)$)

这点是简单的乘法. 所以才有 $dZ^{[1]}$ 维度为 $(n^{[1]}, m)$

(之前少写了个括号. 推导后发现结果不一致 糊心拉人吧!)

$$= np.dot(W^{[2]T}, dZ^{[2]}) \odot g'^{[1]}(Z^{[1]})$$

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

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

3.10 向传播导数的计算

$x_{[n^{[0]},1]}$, $W^{[1]}_{[n^{[1]},n^{[0]}]}$, $b^{[1]}_{[n^{[1]},1]}$ → $z^{[1]} = W^{[1]}x + b^{[1]}$ → $a^{[1]} = g(z^{[1]})$ → $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$ → $a^{[2]} = \sigma(z^{[2]})$ → $\mathcal{L}(a^{[2]}, y)$

$W^{[2]}_{[n^{[2]},n^{[1]}]}$, $b^{[2]}_{[n^{[2]},1]}$

$z^{[1]}_{[n^{[1]},1]}$, $a^{[1]}_{[n^{[1]},1]}$, $z^{[2]}_{[n^{[2]},1]}$, $a^{[2]}_{[n^{[2]},1]}$

首先因为是 σ 函数做为输出，对于 $dz^{[2]}$ 有 $dz^{[2]} = \hat{y} - y$ 有 $dz^{[2]}$ 的维度为 $[n^{[2]}, 1]$

根据公式 $dW^{[2]} = dz^{[2]} \cdot \frac{\partial z^{[2]}}{\partial W^{[2]}} = dz^{[2]} \cdot a^{[1]}$ 根据维度、结果应该是 $[n^{[2]}, n^{[1]}]$ 因此 $dW^{[2]} = dz^{[2]}_{[n^{[2]},1]} \cdot a^{[1]T}_{[1,n^{[1]}]}$ (维度匹配)

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

其次计算 $dz^{[1]}$ 有 $dz^{[1]} = dz^{[2]} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} * \frac{da^{[1]}}{dz^{[1]}}$ (链式法则)

$$\Rightarrow dz^{[2]}_{[n^{[2]},1]}, W^{[2]}_{[n^{[2]},n^{[1]}]} * g'(z^{[1]})_{[n^{[1]},1]}$$

根据维度匹配原则：$dz^{[1]}$ 维度应是 $[n^{[1]}, 1]$ 因此 $dz^{[1]} = W^{[2]T}_{[n^{[1]},n^{[2]}]} \cdot dz^{[2]}_{[n^{[2]},1]} * g'(z^{[1]})_{[n^{[1]},1]}$ 维度匹配

因此 $dW^{[1]} = dz^{[1]} \cdot X$ 根据维度匹配：$dW^{[1]} = dz^{[1]} \cdot X^T$

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

因此，将 m 个样本考虑进去之后结果如下：

$dz^{[2]} = A^{[2]} - Y$ $\quad dW^{[2]} = \frac{1}{m} np.dot(dz^{[2]}, A^{[2]T})$ $\quad db^{[2]} = \frac{1}{m} np.sum(dz^{[2]}, axis=1, keepdims=True)$

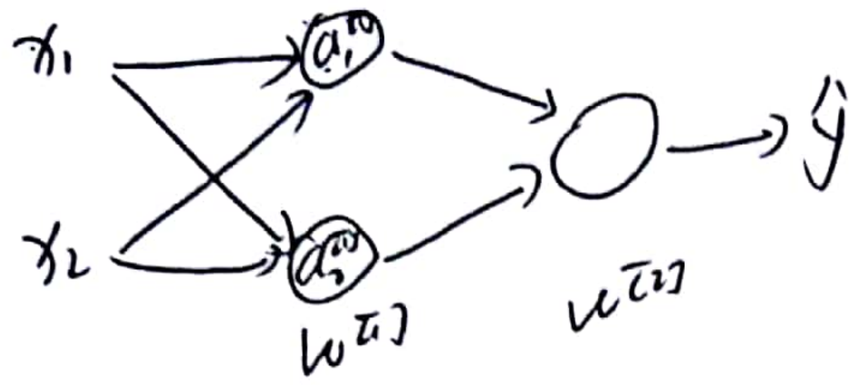
~~$dz^{[1]} = np.dot(dz^{[2]}, W$~~

$dz^{[1]} = np.dot(W^{[2]T}, dz^{[2]}) * g'(z^{[1]})$

$dW^{[1]} = \frac{1}{m} np.dot(dz^{[1]}, X^T)$ $\quad db^{[1]} = \frac{1}{m} np.sum(dz^{[1]}, axis=1, keepdims=True)$

3.11. 随机初始化

为什么要随机初始化参数呢？看下面的例子



如果 $w^{[1]}$ 中 $w_1^{[1]}$、$w_2^{[1]}$ 的参数相同 会出现什么情况呢？

结果是 $a_1^{[1]}$ 和 $a_2^{[1]}$ 计算结果相同. 最后导致它们计算出来的导数也相同. 当我们用梯度下降法来看亲行时 它们的值依然会相同 也就是说它们会永远相同、这样就起不到作用了

因此需要随机初始化

如 $W^{[1]}$ = ~~np.dot~~ np.random.randn((2,2)) * 0.01.

$W^{[2]}$ = np.random.randn((1,2)) * 0.01

$b^{[1]}$ = np.zero((2,1)).

$b^{[2]}$ = 0

上面乘以 0.01 是因为我们应该让初始化参数尽量小、

这样计算出的 $z^{[1]}$ 也就相对较小、如果使用 Sigmoid 函数的话. 会保证较高的学习效率