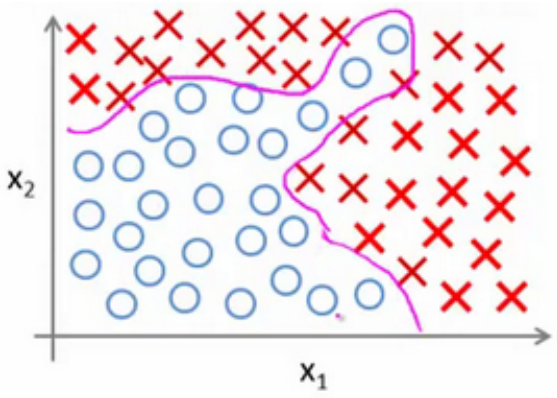
八、神经网络：表述

第八、神经网络：表述(Neural Networks: Representation)

8.1 非线性假设

我们之前学的，无论是线性回归还是逻辑回归都有这样一个缺点，即：**当特征太多时，计算的负荷会非常大。**

下面是一个例子：



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

当我们使用x_1, x_2 的多次项式进行预测时，我们可以应用的很好。
之前我们已经看到过，使用非线性的多项式项，能够帮助我们建立更好的分类模型。假设我们有非常多的特征，例如大于100个变量，我们希望用这100个特征来构建一个非线性的多项式模型，结果将是数量非常惊人的特征组合，即便我们只采用两两特征的组合$(x_1x_2 + x_1x_3 + x_1x_4 + \dots + x_2x_3 + x_2x_4 + \dots + x_{99}x_{100})$，我们也会有接近5000个组合而成的特征。这对于一般的逻辑回归来说需要计算的特征太多了。

一、背景：线性模型的局限性

之前我们学过：

- **线性回归（Linear Regression）**：$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$
- **逻辑回归（Logistic Regression）**：$h_\theta(x) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\cdots+\theta_n x_n)}}$

这两种模型的本质都是**线性的**：它们只会在输入特征上做线性组合。

二、非线性模型的必要性

线性模型无法拟合复杂的模式，比如：

- 一条曲线边界；
- 圆形、椭圆形的分类边界；
- 图像、语音等复杂数据的结构。

所以我们需要引入**非线性特征**。
例如，在二维情况下（特征 (x_1, x_2)）：

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2$$

这样模型就能画出**曲线型的决策边界**，效果更好。

三、维度爆炸（Curse of Dimensionality）

但问题来了：

- 如果只有两个特征 ((x_1, x_2))，你可以轻松地加入 $(x_1^2, x_2^2, x_1 x_2)$；
- 但如果你有 100 个特征，哪怕只考虑两两组合，也有：

  $\frac{100 \times 99}{2} = 4950$ 个新特征
- 如果是 2500 个像素（例如一张 50×50 灰度图），那么：

  $\frac{2500 \times 2499}{2} \approx 3,000,000$

  也就是三百万个特征！

  这就导致了计算量极大，参数太多，训练困难，过拟合严重。

四、神经网络的引入

神经网络（Neural Network）可以被看作是一种**自动学习非线性组合的模型**。

它不需要我们手动去生成所有多项式特征，而是通过**隐藏层（Hidden Layers)**来“学习”这些复杂关系：

$$a^{(2)} = g(W^{(1)} x + b^{(1)}),$$
$$a^{(3)} = g(W^{(2)} a^{(2)} + b^{(2)}), \ldots$$

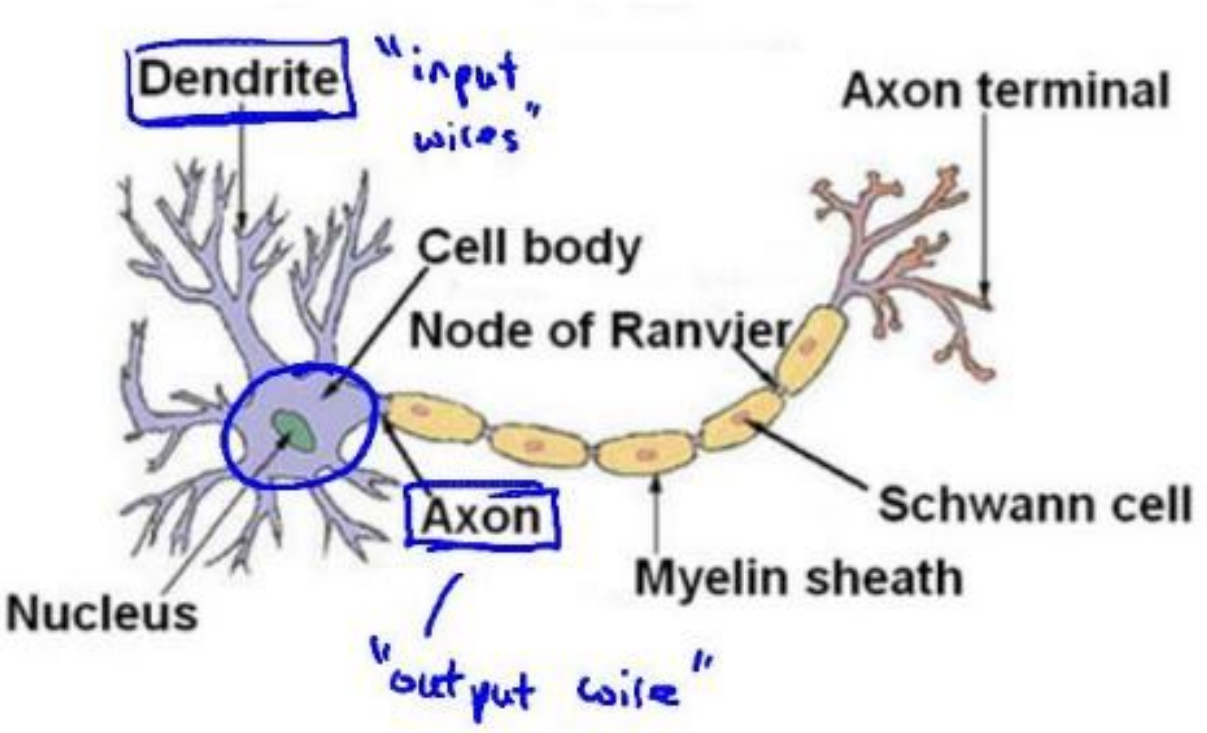
隐藏层中的神经元就像是在帮我们自动地创建：

- $(x_1 x_2)$
- $(x_1^2 x_3)$
- $(\sin(x_4 x_5))$
- 等各种非线性组合

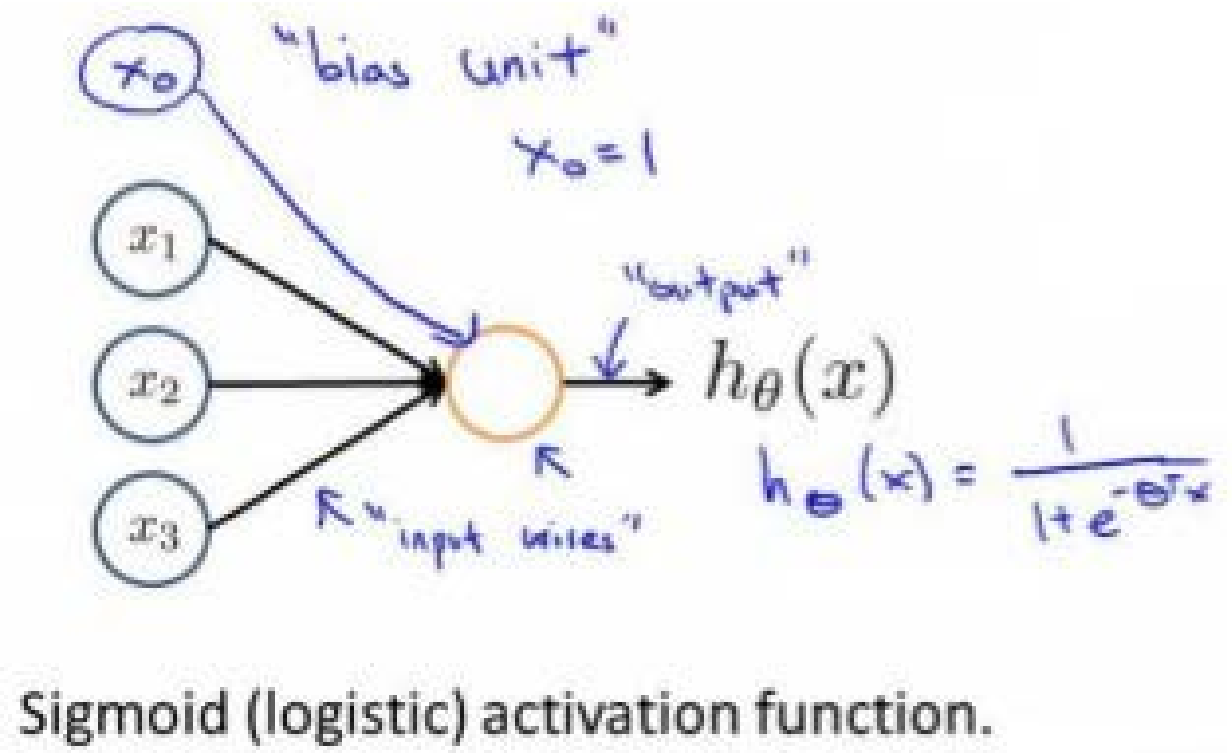
这样模型能“自己学出”复杂的模式，而不需要我们显式地去定义这些组合。

8.3 模型表示1

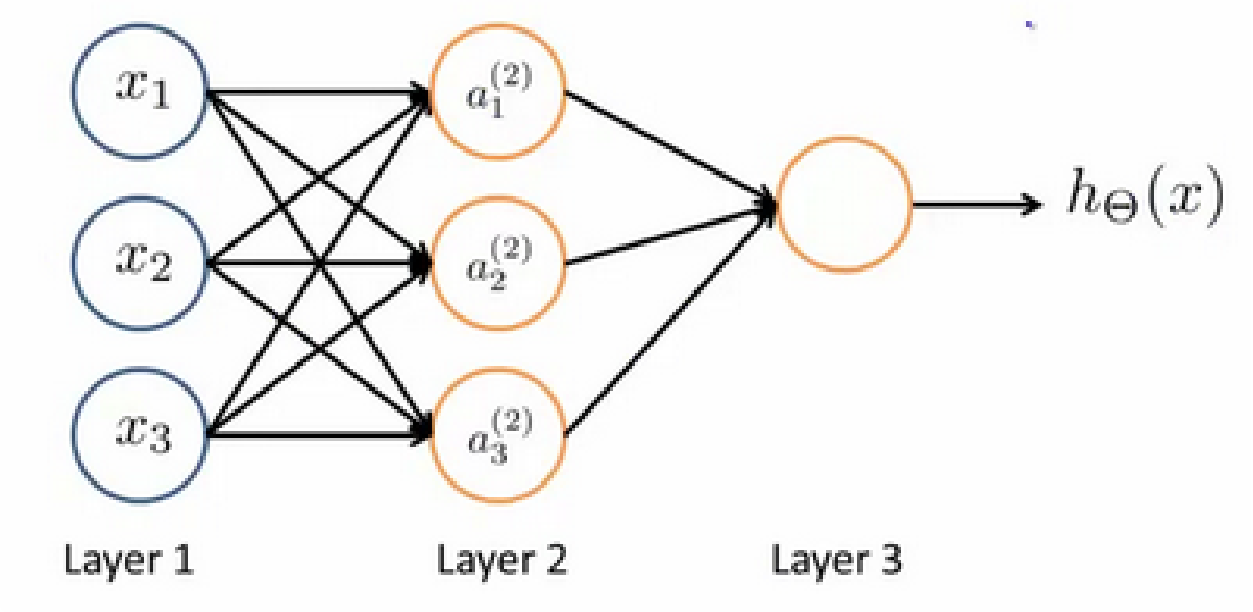
每一个神经元都可以被认为是一个处理单元/神经核（**processing unit/Nucleus**），它含有许多输入/树突（**input/Dendrite**），并且有一个输出/轴突（**output/Axon**）。神经网络是大量神经元相互链接并通过电脉冲来交流的一个网络。



神经网络模型建立在很多神经元之上，每一个神经元又是一个个学习模型。这些神经元（也叫激活单元，**activation unit**）采纳一些特征作为输出，并且根据本身的模型提供一个输出。下图是一个以逻辑回归模型作为自身学习模型的神经元示例，在神经网络中，参数又可被成为权重（**weight**）。

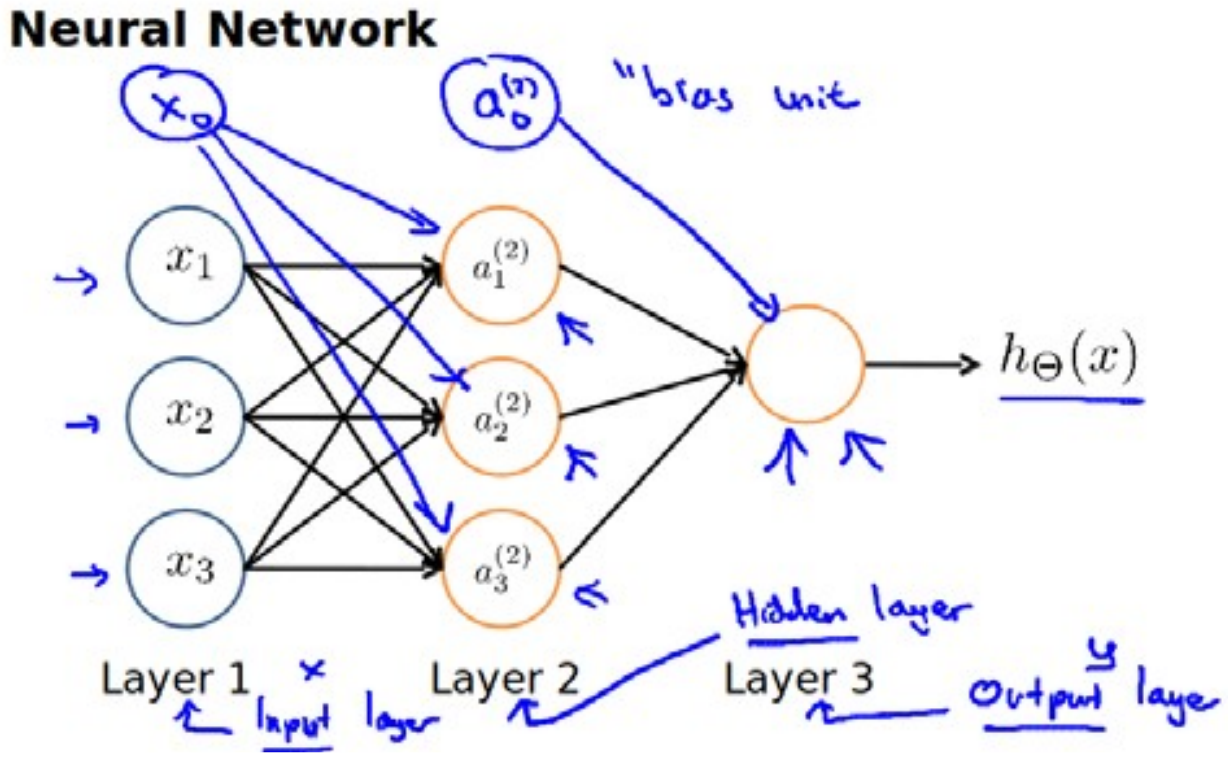


我们设计出了类似于神经元的神经网络，效果如下：



其中x_1, x_2, x_3是输入单元（**input units**），我们将原始数据输入给它们。
a_1, a_2, a_3是中间单元，它们负责将数据进行处理，然后呈递到下一层。
最后是输出单元，它负责计算$h_\theta(x)$。

神经网络模型是许多逻辑单元按照不同层级组织起来的网络，每一层的输出变量都是下一层的输入变量。下图为一个3层的神经网络，第一层成为输入层（**Input Layer**），最后一层称为输出层（**Output Layer**），中间一层成为隐藏层（**Hidden Layers**）。我们为每一层都增加一个偏差单位（**bias unit**）：



下面引入一些标记法来帮助描述模型：
$a_i^{(j)}$ 代表第j 层的第 i 个激活单元。$\theta^{(j)}$代表从第 j 层映射到第 $j+1$层时的权重的矩阵，例如$\theta^{(1)}$代表从第一层映射到第二层的权重的矩阵。其尺寸为：以第 $j+1$层的激活单元数量为行数，以第 j 层的激活单元数加一为列数的矩阵。例如：上图所示的神经网络中$\theta^{(1)}$的尺寸为 3*4。

对于上图所示的模型，激活单元和输出分别表达为：

$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$
$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$
$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$
$h_\Theta(x) = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$

上面进行的讨论中只是将特征矩阵中的一行（一个训练实例）喂给了神经网络，我们需要将整个训练集都喂给我们的神经网络算法来学习模型。

我们可以知道：每一个a都是由上一层所有的x和每一个x所对应的决定的。

（我们把这样从左到右的算法称为前向传播算法(**FORWARD PROPAGATION**)）

把x, θ, a 分别用矩阵表示：

$$X = \begin{matrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{matrix}, \quad \theta = \begin{matrix} \theta_{10} & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \theta_{33} \end{matrix}, \quad a = \begin{matrix} a_1 \\ a_2 \\ a_3 \end{matrix}$$

我们可以得到$\theta \cdot X = a$。

8.4 模型表示2

(**FORWARD PROPAGATION**)
相对于使用循环来编码，利用向量化的方法会使得计算更为简便。以上面的神经网络为例，试着计算第二层的值：

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} x$$
$$a^{(2)} = g(z^{(2)})$$

$$g\left(\begin{bmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = g\left(\begin{bmatrix} \theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3 \\ \theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3 \\ \theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3 \end{bmatrix}\right) = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

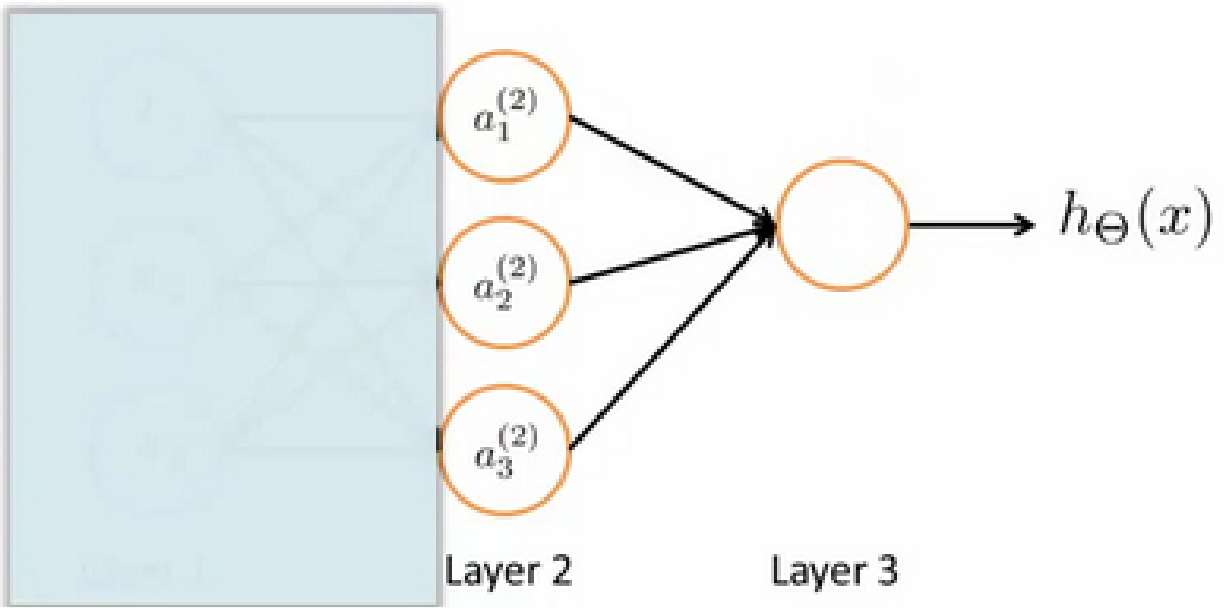
我们令 $z^{(2)} = \theta^{(1)} x$，则 $a^{(2)} = g(z^{(2)})$ ，计算后添加 $a_0^{(2)} = 1$。 计算输出的值为：

$$g\left(\begin{bmatrix} \theta_{10}^{(2)} & \theta_{11}^{(2)} & \theta_{12}^{(2)} & \theta_{13}^{(2)} \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}\right) = g\left(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)}\right) = h_\theta(x)$$

我们令 $z^{(3)} = \theta^{(2)} a^{(2)}$，则 $h_\theta(x) = a^{(3)} = g(z^{(3)})$。
这只是针对训练集中一个训练实例所进行的计算。如果我们要对整个训练集进行计算，我们需要将训练集特征矩阵进行转置，使得同一个实例的特征都在同一列里。即：
$z^{(2)} = \Theta^{(1)} \times X^T$

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

为了更好了了解**Neuron Networks**的工作原理，我们先把左半部分遮住：



右半部分其实就是以a_0, a_1, a_2, a_3, 按照**Logistic Regression**的方式输出$h_\theta(x)$：

其实神经网络就像是**logistic regression**，只不过我们把**logistic regression**中的输入向量$[x_1 \sim x_3]$变成了中间层的$\left[a_1^{(2)} \sim a_3^{(2)}\right]$, 即: $h_\theta(x) = g\left(\Theta_0^{(2)}a_0^{(2)} + \Theta_1^{(2)}a_1^{(2)} + \Theta_2^{(2)}a_2^{(2)} + \Theta_3^{(2)}a_3^{(2)}\right)$
我们可以把a_0, a_1, a_2, a_3看成更为高级的特征值，也就是x_0, x_1, x_2, x_3的进化体，并且它们是由 x与θ决定的，因为是梯度下降的，所以a是变化的，并且变得越来越厉害，所以这些更高级的特征值远比仅仅将 x次方厉害，也能更好的预测新数据。
这就是神经网络相比于逻辑回归和线性回归的优势。

8.5 特征和直观理解1

从本质上讲，神经网络能够通过学习得出其自身的一系列特征。

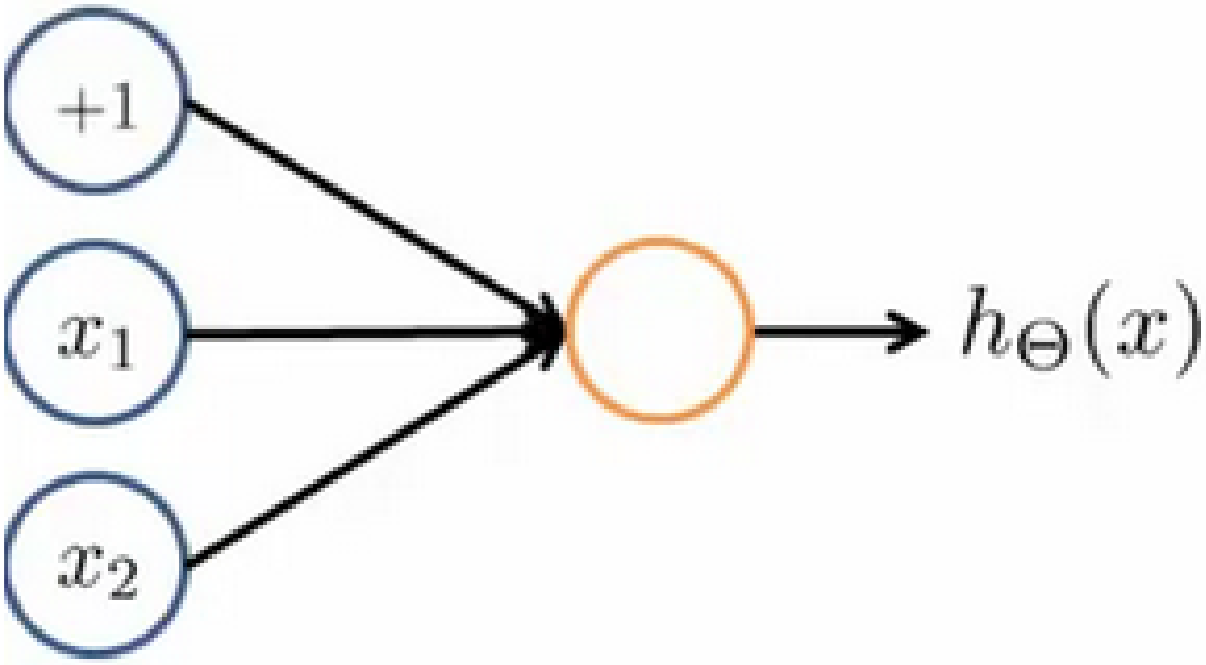
- 在普通的逻辑回归中，我们被限制为使用数据中的原始特征$x_1, x_2, \ldots, x_n$，我们虽然可以使用一些二项式项来组合这些特征，但是我们仍然受到这些原始特征的限制。
- 在神经网络中，原始特征只是输入层，在我们上面三层的神经网络例子中，第三层也就是输出层做出的预测利用的是第二层的特征，而非输入层中的原始特征，我们可以认为第二层中的特征是神经网络通过学习后自己得出的一系列用于预测输出变量的新特征。

神经网络中，单层神经元（无中间层）的计算可用来表示逻辑运算，比如逻辑与(**AND**)、逻辑或(**OR**)。

举例说明：逻辑与(**AND**)；下图中左半部分是神经网络的设计与**output**层表达式，右边上部分是**sigmod**函数，下半部分是真值表。

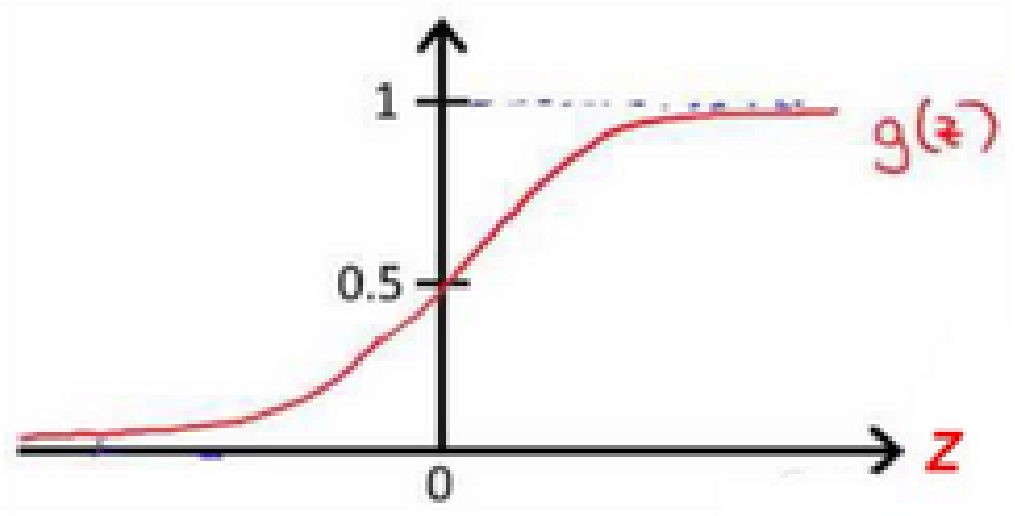
我们可以用这样的一个神经网络表示**AND** 函数：

$$x_1, x_2 \in \{0, 1\}$$
$$y = x_1 \text{ AND } x_2$$



其中$\theta_0 = -30, \theta_1 = 20, \theta_2 = 20$
我们的输出函数$h_\theta(x)$即为：$h_\Theta(x) = g\left(-30 + 20x_1 + 20x_2\right)$

我们知道$g(x)$的图像是：



x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

所以我们有：$h_\Theta(x) \approx \mathrm{x}_1 \mathrm{AND}\, \mathrm{x}_2$

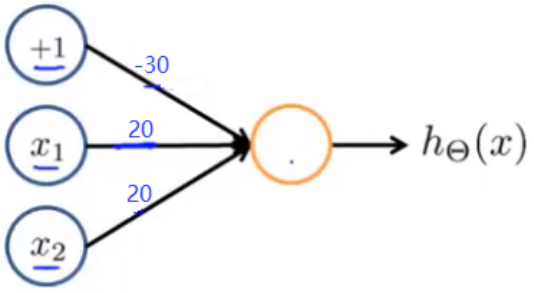
所以我们的：$h_\Theta(x)$

这就是**AND**函数。

8.6 样本和直观理解II

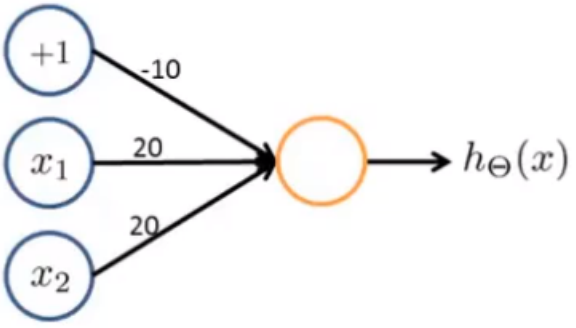
二元逻辑运算符（**BINARY LOGICAL OPERATORS**）当输入特征为布尔值（0或1）时，我们可以用一个单一的激活层可以作为二元逻辑运算符，为了表示不同的运算符，我们只需要选择不同的权重即可。

下图的神经元（三个权重分别为-30，20，20）可以被视为作用同于逻辑与（**AND**）：



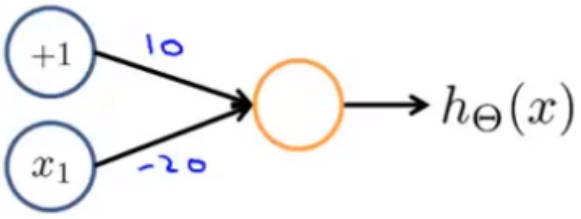
x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

下图的神经元（三个权重分别为-10，20，20）可以被视为作用等同于逻辑或（**OR**）：



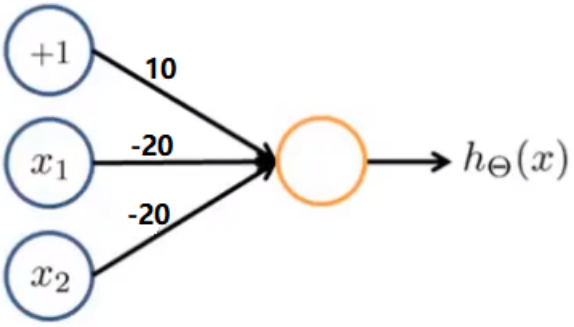
x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$

下图的神经元（两个权重分别为 10，-20）可以被视为作用等同于逻辑非（**NOT**）：



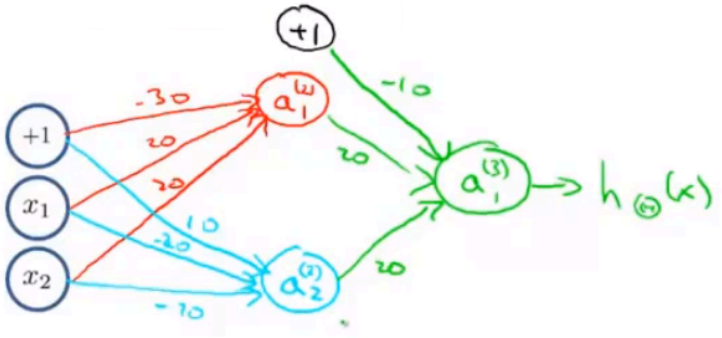
x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

我们可以利用神经元来组合成更为复杂的神经网络以实现更复杂的运算。例如我们要实现**XNOR** 功能（输入的两个值必须一样，均为1或均为0），即 $\mathrm{XNOR} = (\mathrm{x}_1 \,\mathrm{AND}\, \mathrm{x}_2) \,\mathrm{OR}\, ((\mathrm{NOT}\, \mathrm{x}_1) \mathrm{AND}\, (\mathrm{NOT}\, \mathrm{x}_2))$
首先构造一个能表达$(\mathrm{NOT}\, \mathrm{x}_1) \mathrm{AND}\, (\mathrm{NOT}\, \mathrm{x}_2)$部分的神经元：



x_1	x_2	$h_\Theta(x)$
0	0	$g(10) \approx 1$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(-30) \approx 0$

然后将表示 **AND** 的神经元和表示$(\mathrm{NOT}\, \mathrm{x}_1) \mathrm{AND}\, (\mathrm{NOT}\, \mathrm{x}_2)$的神经元以及表示 OR 的神经元进行组合：



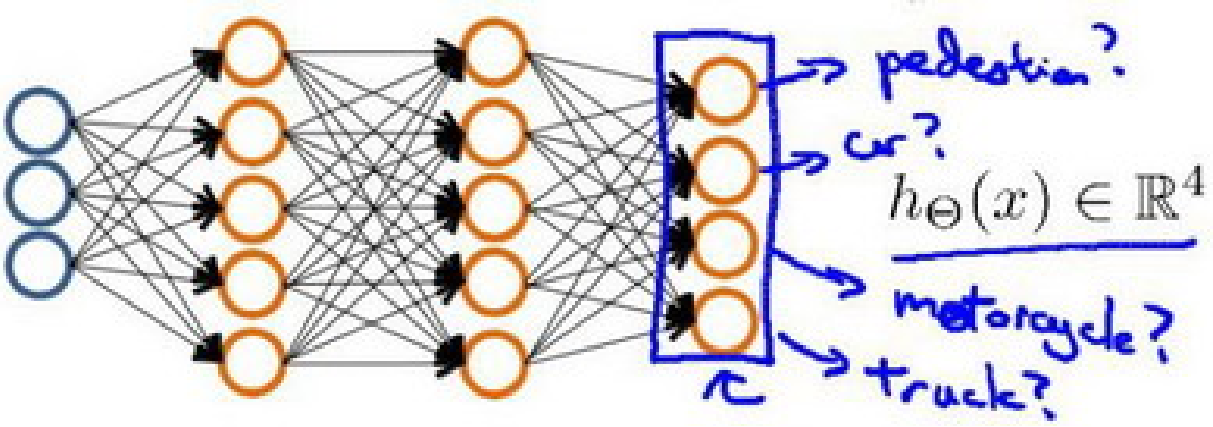
x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

我们就得到了一个能实现 XNOR 运算符功能的神经网络。

8.7 多类分类

当我们有不止两种分类时（也就是$y = 1, 2, 3 \ldots$），比如以下这种情况，该怎么办？如果我们要训练一个神经网络算法来识别路人、汽车、摩托车和卡车，在输出层我们应该有4个值。例如，第一个值为1或0用于预测是否是行人，第二个值用于判断是否为汽车。

输入向量x有三个维度，两个中间层，输出层4个神经元分别用来表示4类，也就是每一个数据在输出层都会出现$[a\ b\ c\ d]^T$，且a, b, c, d中仅有一个为1，表示当前类。下面是该神经网络的可能结构示例：



Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian　　when car　　when motorcycle

神经网络算法的输出结果为四种可能情形之一：

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$