3.3 计算一个神经网络的输出（Computing a Neural Network's output）

首先，回顾下只有一个隐藏层的简单两层**神经网络结构**：

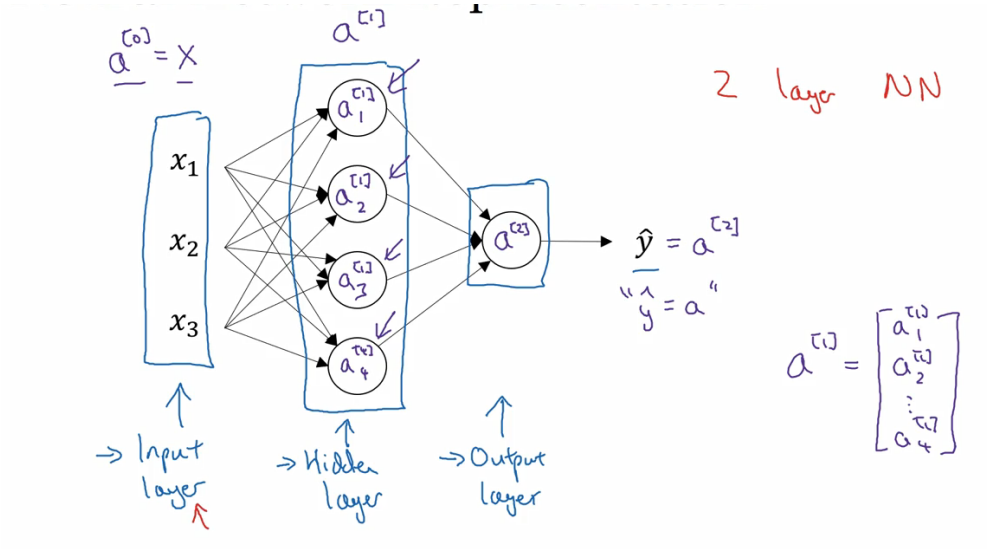
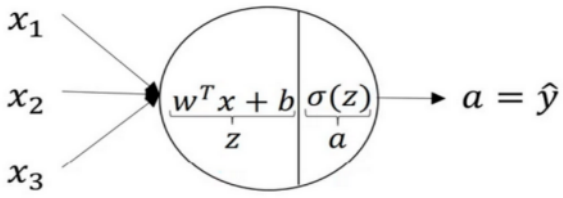


图 3.3.1

其中，x表示输入特征，a表示每个神经元的输出，W表示特征的权重，上标表示神经网络的层数（隐藏层为 1），下标表示该层的第几个神经元。这是神经网络的**符号惯例**，下同。

神经网络的计算

关于神经网络是怎么计算的，从我们之前提及的逻辑回归开始，如下图所示。用圆圈表示神经网络的计算单元，逻辑回归的计算有两个步骤，首先你按步骤计算出z，然后在第二步中你以 **sigmoid** 函数为激活函数计算z（得出a），一个神经网络只是这样子做了好多次重复计算。



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

$$a = \sigma(z)$$

图 3.3.2

回到两层的神经网络，我们从隐藏层的第一个神经元开始计算，如上图第一个最上面的箭头所指。从上图可以看出，输入与逻辑回归相似，这个神经元的计算与逻辑回归一样分为两步，小圆圈代表了计算的两个步骤。

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$a_1^{[1]}, a_1^{[1]} = \sigma(z_1^{[1]})$。

隐藏层的第二个以及后面两个神经元的计算过程一样，只是注意符号表示不同，最终分别得到$a_2^{[1]}$、$a_3^{[1]}$、$a_4^{[1]}$，详细结果见下:

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

向量化计算 如果你执行神经网络的程序，用 **for** 循环来做这些看起来真的很低效。所以接下来我们要做的就是把这四个等式向量化。向量化的过程是将神经网络中的一层神经元参数纵向堆积起来，例如隐藏层中的w纵向堆积起来变成一个(4,3)的矩阵，用符号$W^{[1]}$表示。另一个看待这个的方法是我们有四个逻辑回归单元，且每一个逻辑回归单元都有相对应的参数——向量w，把这四个向量堆积在一起，你会得出这 4×3 的矩阵。 因此， 公式 3.8：

$z^{[n]} = w^{[n]}x + b^{[n]}$

公式 3.9：

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

详细过程见下: 公式 3.10：

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

公式 3.11：

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

对于神经网络的第一层，给予一个输入x，得到$a^{[1]}$，x可以表示为$a^{[0]}$。通过相似的衍生

你会发现，后一层的表示同样可以写成类似的形式，得到$a^{[2]}$，$\hat{y} = a^{[2]}$，具体过程见公式 3.8、3.9。

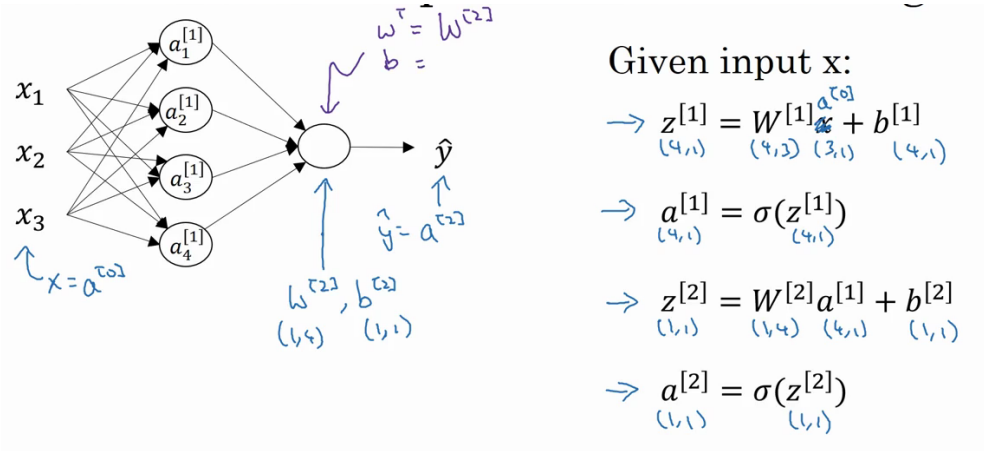


图 3.3.3

如上图左半部分所示为神经网络，把网络左边部分盖住先忽略，那么最后的输出单元就相当于一个逻辑回归的计算单元。当你有一个包含一层隐藏层的神经网络，你需要去实现以计算得到输出的是右边的四个等式，并且可以看成是一个向量化的计算过程，计算出隐藏层的四个逻辑回归单元和整个隐藏层的输出结果，如果编程实现需要的也只是这四行代码。

总结：通过本视频，你能够根据给出的一个单独的输入特征向量，运用四行代码计算出一个简单神经网络的输出。接下来你将了解的是如何一次能够计算出不止一个样本的神经网络输出，而是能一次性计算整个训练集的输出。

3.4 多样本向量化（Vectorizing across multiple examples）

逻辑回归是将各个训练样本组合成矩阵，对矩阵的各列进行计算。神经网络是通过对逻辑回归中的等式简单的变形，让神经网络计算出输出值。这种计算是所有的训练样本同时进行的，以下是实现它具体的步骤：

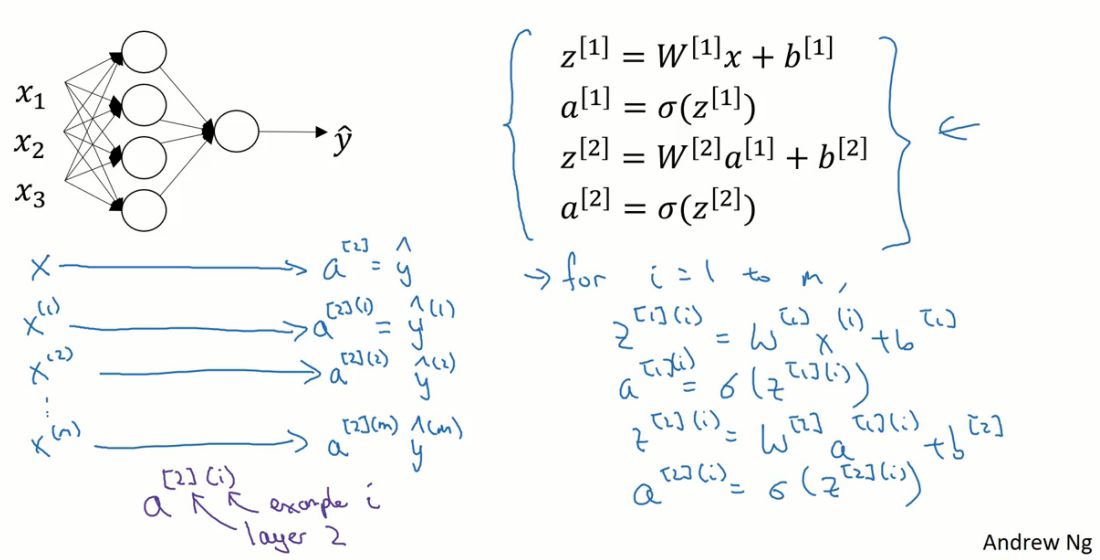


图 3.4.1

上一节视频中得到的四个等式。它们给出如何计算出$z^{[1]}$，$a^{[1]}$，$z^{[2]}$，$a^{[2]}$。

对于一个给定的输入特征向量X，这四个等式可以计算出$a^{[2]}$等于$\hat{y}$。这是针对于单一的训练样本。如果有m个训练样本,那么就需要重复这个过程。

用第一个训练样本$x^{[1]}$来计算出预测值$\hat{y}^{[1]}$，就是第一个训练样本上得出的结果。

然后，用$x^{[2]}$来计算出预测值$\hat{y}^{[2]}$，循环往复，直至用$x^{[m]}$计算出$\hat{y}^{[m]}$。

用激活函数表示法，如上图左下所示，它写成$a^{[2](1)}$、$a^{[2](2)}$和$a^{[2](m)}$。

【注】：$a^{[2](i)}$，(i)是指第i个训练样本而[2]是指第二层。

如果有一个非向量化形式的实现，而且要计算出它的预测值，对于所有训练样本，需要让i从 1 到m实现这四个等式：

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

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

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

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

对于上面的这个方程中的 $^{(i)}$，是所有依赖于训练样本的变量，即将(i)添加到x，z和a。

如果想计算m个训练样本上的所有输出，就应该向量化整个计算，以简化这列。

如何向量化这些：

公式 3.12：

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

公式 3.13：

$$Z^{[1]} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

公式 3.14：

$$A^{[1]} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \alpha^{[1](1)} & \alpha^{[1](2)} & \cdots & \alpha^{[1](m)} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

公式 3.15：

$$\left.\begin{aligned} z^{[1](i)} &= W^{[1](i)}x^{(i)} + b^{[1]} \\ \alpha^{[1](i)} &= \sigma(z^{[1](i)}) \\ z^{[2](i)} &= W^{[2](i)}\alpha^{[1](i)} + b^{[2]} \\ \alpha^{[2](i)} &= \sigma(z^{[2](i)}) \end{aligned}\right\} \Longrightarrow \begin{cases} A^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} = \sigma(z^{[2]}) \end{cases}$$

以此类推，从小写的向量x到这个大写的矩阵X，只是通过组合x向量在矩阵的各列中。

同理，$z^{[1](1)}$，$z^{[1](2)}$等等都是$z^{[1](m)}$的列向量，将所有m都组合在各列中，就的到矩阵$Z^{[1]}$。

同理，$a^{[1](1)}$，$a^{[1](2)}$，……，$a^{[1](m)}$将其组合在矩阵各列中，如同从向量x到矩阵X，以及从向量z到矩阵Z一样，就能得到矩阵$A^{[1]}$。

同样的，对于$Z^{[2]}$和$A^{[2]}$，也是这样得到。

这种符号其中一个作用就是，可以通过训练样本来进行索引。这就是**水平索引对应于不同的训练样本的原因**，这些训练样本是从左到右扫描训练集而得到的。

在垂直方向，这个垂直索引对应于神经网络中的不同节点。例如，这个节点，该值位于矩阵的最左上角对应于激活单元，它是位于第一个训练样本上的第一个隐藏单元。它的下一个值对应于第二个隐藏单元的激活值。它是位于第一个训练样本上的，以及第一个训练示例中第三个隐藏单元，等等。

当垂直扫描，是索引到隐藏单位的数字。当水平扫描，将从第一个训练示例中从第一个隐藏的单元到第二个训练样本，第三个训练样本……直到节点对应于第一个隐藏单元的激活值，且这个隐藏单元是位于这m个训练样本中的最终训练样本。

从水平上看，矩阵A代表了各个训练样本。从竖直上看，矩阵A的不同的索引对应于不同的隐藏单元。

对于矩阵Z，X情况也类似，水平方向上，对应于不同的训练样本；竖直方向上，对应不同的输入特征，而这就是神经网络输入层中各个节点。

神经网络上通过在多样本情况下的向量化来使用这些等式。

在下一个视频中，将证明为什么这是一种正确向量化的实现。这种证明将会与逻辑回归中的证明类似。

3.5 向量化实现的解释（Justification for vectorized implementation）

我们先手动对几个样本计算一下前向传播，看看有什么规律：

公式 3.16： $z^{[1](1)} = W^{[1]}x^{(1)} + b^{[1]}$

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

$$z^{[1](3)} = W^{[1]}x^{(3)} + b^{[1]}$$

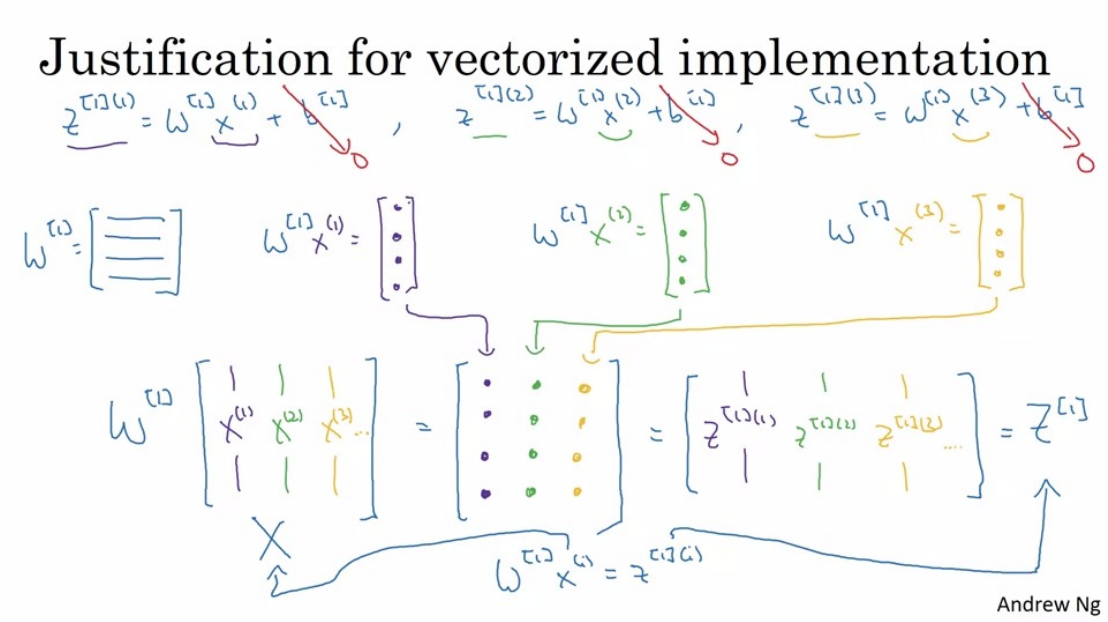
这里，为了描述的简便，我们先忽略掉 $b^{[1]}$后面你将会看到利用 **Python** 的广播机制，可以很容易的将$b^{[1]}$ 加进来。

现在 $W^{[1]}$ 是一个矩阵，$x^{(1)}, x^{(2)}, x^{(3)}$都是列向量，矩阵乘以列向量得到列向量：

$$W^{[1]}x = \begin{bmatrix} \cdots \\ \cdots \\ \cdots \end{bmatrix} \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ x^{(1)} & x^{(2)} & x^{(3)} & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ w^{(1)}x^{(1)} & w^{(1)}x^{(2)} & w^{(1)}x^{(3)} & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} =$$

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} = Z^{[1]}$$

视频中，吴恩达老师很细心的用不同的颜色表示不同的样本向量，及其对应的输出。所以从图中可以看出，当加入更多样本时，只需向矩阵X中加入更多列。



所以从这里我们也可以了解到，为什么之前我们对单个样本的计算要写成 $z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$ 这种形式，因为当有不同的训练样本时，将它们堆到矩阵X的各列中，那么它们的输出也就会相应的堆叠到矩阵 $Z^{[1]}$ 的各列中。现在我们就可以直接计算矩阵 $Z^{[1]}$ 加上$b^{[1]}$，因为列向量 $b^{[1]}$ 和矩阵 $Z^{[1]}$的列向量有着相同的尺寸，而 **Python** 的广播机制对于这种矩阵与向量直接相加的处理方式是，将向量与矩阵的每一列相加。 所以这一节只是说明了为什么公式 $Z^{[1]} = W^{[1]}X + b^{[1]}$是前向传播的第一步计算的正确向量化实现，但事实证明，类似的分析可以发现，前向传播的其它步也可以使用非常相似的逻辑，即如果将输入按列向量横向堆叠进矩阵，那么通过公式计算之后，也能得到成列堆叠的输出。

最后，对这一段视频的内容做一个总结:

由公式 3.12、公式 3.13、公式 3.14、公式 3.15 可以看出，使用向量化的方法，可以不需要显示循环，而直接通过矩阵运算从X就可以计算出 $A^{[1]}$，实际上X可以记为 $A^{[0]}$，使用同样的方法就可以由神经网络中的每一层的输入 $A^{[i-1]}$ 计算输出 $A^{[i]}$。其实这些方程有一定对称性，其中第一个方程也可以写成$Z^{[1]} = W^{[1]}A^{[0]} + b^{[1]}$，你看这对方程，还有这对方程形式其实很类似，只不过这里所有指标加了 1。所以这样就显示出神经网络的不同层次，你知道大概每一步做的都是一样的，或者只不过同样的计算不断重复而已。这里我们有一个双层神经网络，我们在下周视频里会讲深得多的神经网络，你看到随着网络的深度变大，基本上也还是重复这两步运算，只不过是比这里你看到的重复次数更多。在下周的视频中将会讲解更深层次的神经网络，随着层数的加深，基本上也还是重复同样的运算。

以上就是对神经网络向量化实现的正确性的解释，到目前为止，我们仅使用 **sigmoid** 函数作为激活函数，事实上这并非最好的选择，在下一个视频中，将会继续深入的讲解如何使用更多不同种类的激活函数。