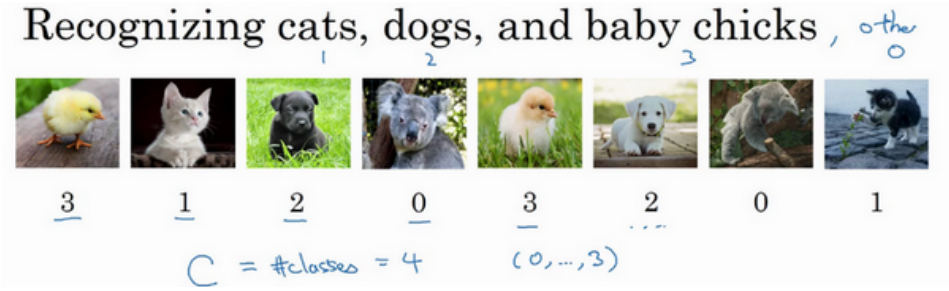
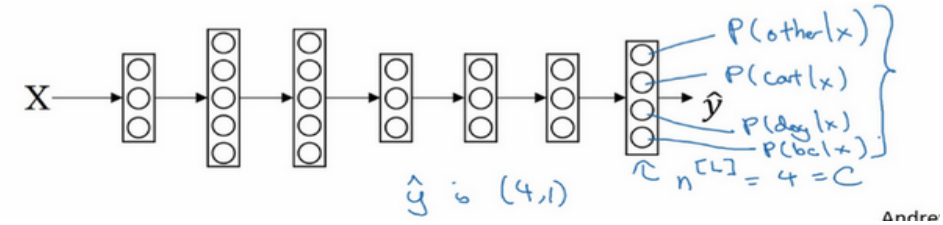
3.8 Softmax 回归（Softmax regression）

到目前为止，我们讲到过的分类的例子都使用了二分分类，这种分类只有两种可能的标记 0 或 1，这是一只猫或者不是一只猫，如果我们有多种可能的类型的话呢？有一种 **logistic** 回归的一般形式，叫做 **Softmax** 回归，能让你在试图识别某一分类时做出预测，或者说是多种分类中的一个，不只是识别两个分类，我们来一起看一下。



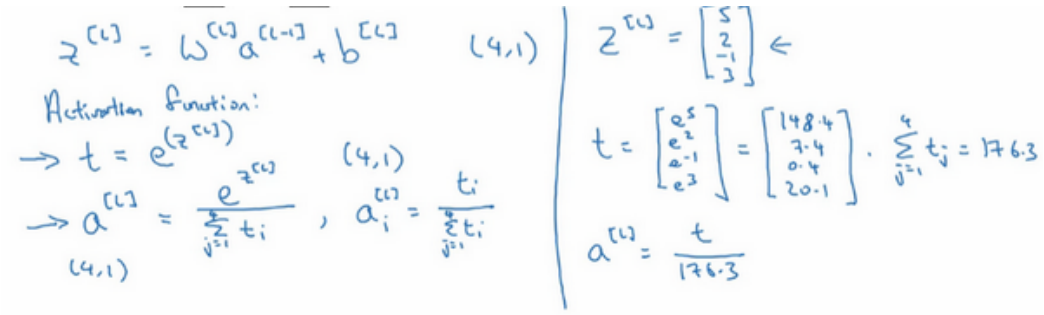
假设你不单需要识别猫，而是想识别猫，狗和小鸡，我把猫加做类 1，狗为类 2，小鸡是类 3，如果不属于以上任何一类，就分到“其它”或者说“以上均不符合”这一类，我把它叫做类 0。这里显示的图片及其对应的分类就是一个例子，这幅图片上是一只小鸡，所以是类 3，猫是类 1，狗是类 2，我猜这是一只考拉，所以以上均不符合，那就是类 0，下一个类 3，以此类推。我们将会用符号表示，我会用大写的C来表示你的输入会被分入的类别总个数，在这个例子中，我们有 4 种可能的类别，包括“其它”或“以上均不符合”这一类。当有 4 个分类时，指示类别的数字，就是从 0 到$C-1$，换句话说就是 0、1、2、3。



在这个例子中，我们将建立一个神经网络，其输出层有 4 个，或者说C个输出单元，因此n，即输出层也就是L层的单元数量，等于 4，或者一般而言等于C。我们想要输出层单元的数字告诉我们这 4 种类型中每个的概率有多大，所以这里的第一个节点(最后输出的第 1 个方格+圆圈)输出的应该是或者说我们希望它输出“其它”类的概率。在输入X的情况下，这个(最后输出的第 2 个方格+圆圈)会输出猫的概率。在输入X的情况下，这个会输出狗的概率(最后输出的第 3 个方格+圆圈)。在输入X的情况下，输出小鸡的概率（最后输出的第 4 个方格+圆圈），我把小鸡缩写为 **bc**（**baby chick**）。因此这里的$\hat{y}$将是一个4×1维向量，因为它必

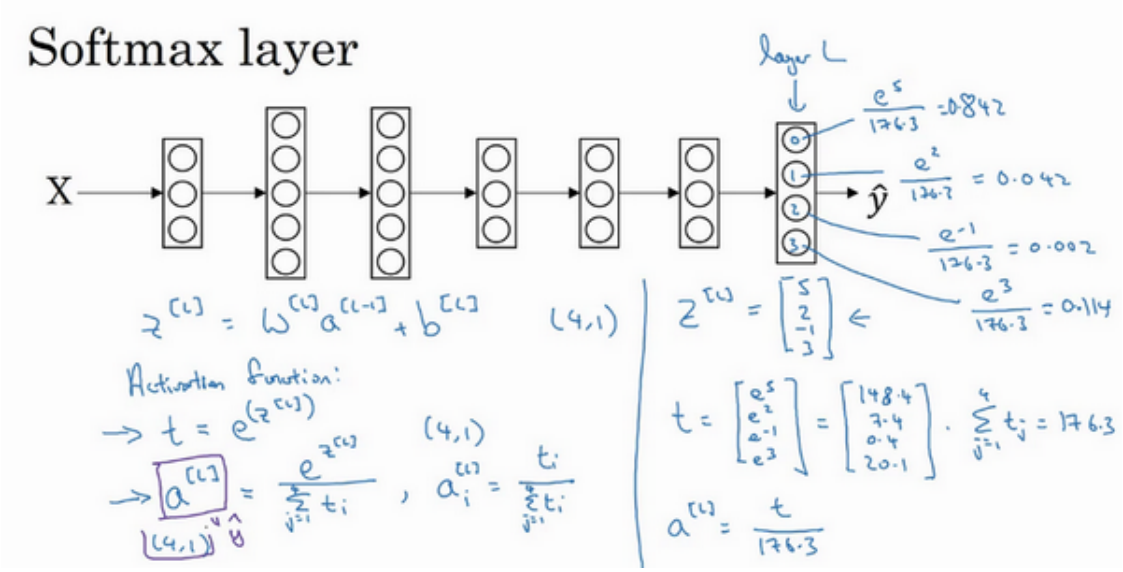
须输出四个数字，给你这四种概率，因为它们加起来应该等于 1，输出中的四个数字加起来应该等于 1。

让你的网络做到这一点的标准模型要用到 **Softmax** 层，以及输出层来生成输出，让我把式子写下来，然后回过头来，就会对 **Softmax** 的作用有一点感觉了。

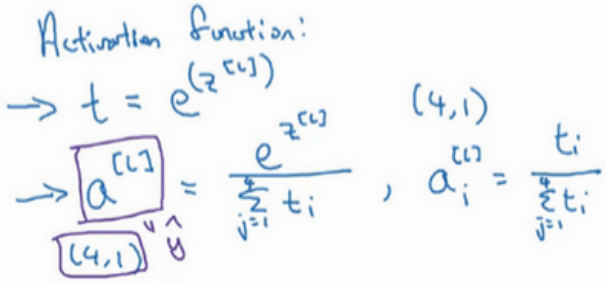


在神经网络的最后一层，你将会像往常一样计算各层的线性部分，$z^{[l]}$这是最后一层的z变量，记住这是大写L层，和往常一样，计算方法是$z^{[l]} = W^{[l]}a^{[L-1]} + b^{[l]}$，算出了$z$之后，你需要应用 Softmax 激活函数，这个激活函数对于 **Softmax** 层而言有些不同，它的作用是这样的。首先，我们要计算一个临时变量，我们把它叫做 t，它等于$e^{z^{[l]}}$，这适用于每个元素，而这里的$z^{[l]}$，在我们的例子中，$z^{[l]}$是 4×1 的，四维向量$t = e^{z^{[l]}}$，这是对所有元素求幂，t也是一个 4×1 维向量，然后输出的$a^{[l]}$，基本上就是向量t，但是会归一化，使和为 1。因此$a^{[l]} = \frac{e^{z^{[l]}}}{\sum_{j=1}^{4} t_i}$，换句话说，$a^{[l]}$也是一个 4×1 维向量，而这个四维向量的第$i$个元素，我把它写下来，$a_i^{[l]} = \frac{t_i}{\sum_{j=1}^{4} t_i}$，以防这里的计算不够清晰易懂，我们马上会举个例子来详细解释。

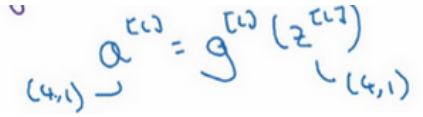
我们来看一个例子，详细解释，假设你算出了$z^{[l]}$，$z^{[l]}$是一个四维向量，假设为$z^{[l]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$，我们要做的就是用这个元素取幂方法来计算$t$，所以$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$，如果你按一下计算器就会得到以下值$t = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$，我们从向量$t$得到向量$a^{[l]}$就只需要将这些项目归一化，使总和为 1。如果你把$t$的元素都加起来，把这四个数字加起来，得到 176.3，最终$a^{[l]} = \frac{t}{176.3}$。



例如这里的第一个节点，它会输出$\frac{e^5}{176.3} = 0.842$，这样说来，对于这张图片，如果这是你得到的$z$值($\begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$)，它是类 0 的概率就是 84.2%。下一个节点输出$\frac{e^2}{176.3} = 0.042$，也就是 4.2%的几率。下一个是$\frac{e^{-1}}{176.3} = 0.002$。最后一个是$\frac{e^3}{176.3} = 0.114$，也就是 11.4%的概率属于类 3，也就是小鸡组，对吧？这就是它属于类 0，类 1，类 2，类 3 的可能性。



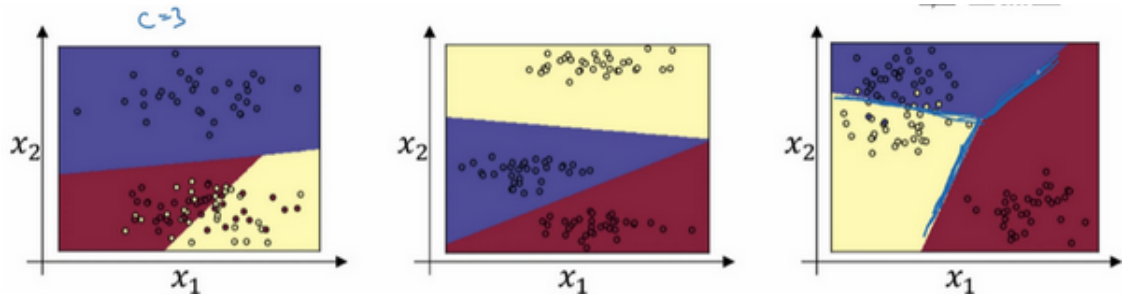
神经网络的输出$a^{[l]}$，也就是$\hat{y}$，是一个 4×1 维向量，这个 4×1 向量的元素就是我们算出来的这四个数字($\begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$)，所以这种算法通过向量$z^{[l]}$计算出总和为 1 的四个概率。



如果我们总结一下从$z^{[l]}$到$a^{[l]}$的计算步骤，整个计算过程，从计算幂到得出临时变量t，再归一化，我们可以将此概括为一个 **Softmax** 激活函数。设$a^{[l]} = g^{[l]}(z^{[l]})$，这一激活函数的与众不同之处在于，这个激活函数$g$ 需要输入一个 4×1 维向量，然后输出一个 4×1 维向量。之前，我们的激活函数都是接受单行数值输入，例如 **Sigmoid** 和 **ReLu** 激活函数，输入一个实数，输出一个实数。**Softmax** 激活函数的特殊之处在于，因为需要将所有可能的输出

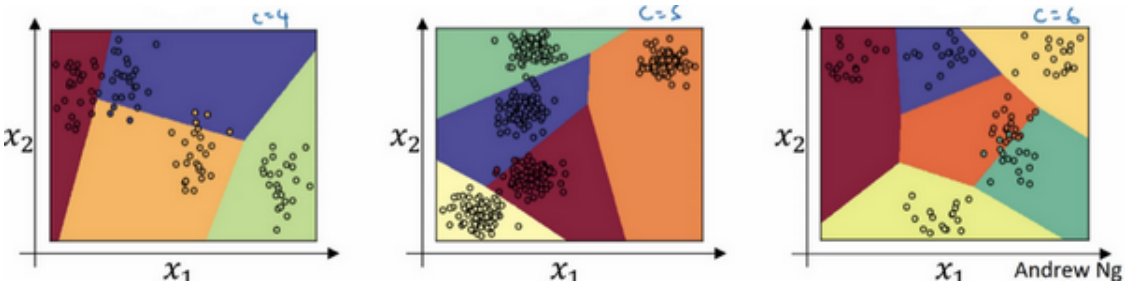
归一化，就需要输入一个向量，最后输出一个向量。

那么 **Softmax** 分类器还可以代表其它的什么东西么？我来举几个例子，你有两个输入x_1，x_2，它们直接输入到 **Softmax** 层，它有三四个或者更多的输出节点，输出$\hat{y}$，我将向你展示一个没有隐藏层的神经网络，它所做的就是计算$z^{[1]} = W^{[1]}x + b^{[1]}$，而输出的出$a^{[l]}$，或者说$\hat{y}$，$a^{[l]} = y = g(z^{[1]})$，就是$z^{[1]}$的 **Softmax** 激活函数，这个没有隐藏层的神经网络应该能让你对 **Softmax** 函数能够代表的东西有所了解。



这个例子中（左边图），原始输入只有x_1和x_2，一个$C = 3$个输出分类的 **Softmax** 层能够代表这种类型的决策边界，请注意这是几条线性决策边界，但这使得它能够将数据分到 3 个类别中，在这张图表中，我们所做的是选择这张图中显示的训练集，用数据的 3 种输出标签来训练 **Softmax** 分类器，图中的颜色显示了 **Softmax** 分类器的输出的阈值，输入的着色是基于三种输出中概率最高的那种。因此我们可以看到这是 **logistic** 回归的一般形式，有类似线性的决策边界，但有超过两个分类，分类不只有 0 和 1，而是可以是 0，1 或 2。

这是（中间图）另一个 **Softmax** 分类器可以代表的决策边界的例子，用有三个分类的数据集来训练，这里（右边图）还有一个。对吧，但是直觉告诉我们，任何两个分类之间的决策边界都是线性的，这就是为什么你看到，比如这里黄色和红色分类之间的决策边界是线性边界，紫色和红色之间的也是线性边界，紫色和黄色之间的也是线性决策边界，但它能用这些不同的线性函数来把空间分成三类。



我们来看一下更多分类的例子，这个例子中（左边图）$C = 4$，因此这个绿色分类和 **Softmax** 仍旧可以代表多种分类之间的这些类型的线性决策边界。另一个例子（中间图）是$C = 5$类，最后一个例子（右边图）是$C = 6$，这显示了 **Softmax** 分类器在没有隐藏层的情况

下能够做到的事情，当然更深的神经网络会有x，然后是一些隐藏单元，以及更多隐藏单元等等，你就可以学习更复杂的非线性决策边界，来区分多种不同分类。

我希望你了解了神经网络中的 **Softmax** 层或者 **Softmax** 激活函数有什么作用，下一个视频中，我们来看一下你该怎样训练一个使用 **Softmax** 层的神经网络。

3.9 训练一个 Softmax 分类器（Training a Softmax classifier）

上一个视频中我们学习了 **Softmax** 层和 **Softmax** 激活函数，在这个视频中，你将更深入地了解 **Softmax** 分类，并学习如何训练一个使用了 **Softmax** 层的模型。

(4,1)

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \qquad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

回忆一下我们之前举的的例子，输出层计算出的$z^{[l]}$如下，$z^{[l]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$我们有四个分类$C = 4$，$z^{[l]}$可以是 4×1 维向量，我们计算了临时变量$t$，$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$，对元素进行幂运算，最后，如果你的输出层的激活函数$g^{[L]}()$是 **Softmax** 激活函数，那么输出就会是这样的：

$$a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.84 \\ 0.04 \\ 0.00 \\ 0.11 \end{bmatrix}$$

简单来说就是用临时变量t将它归一化，使总和为 1，于是这就变成了$a^{[L]}$，你注意到向量z中，最大的元素是 5，而最大的概率也就是第一种概率。

Understanding softmax

(4,1)

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \qquad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

C=4 $g^{[L]}(\cdot)$

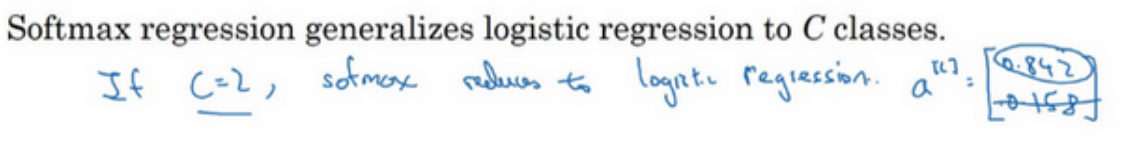
$$a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

"hard max"

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

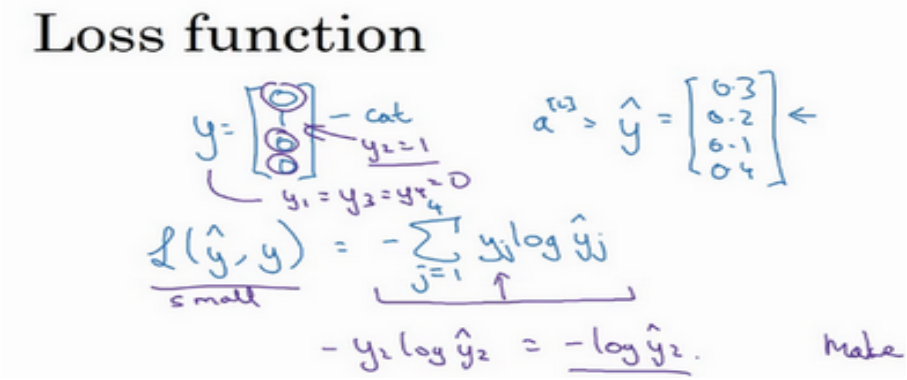
Softmax 这个名称的来源是与所谓 **hardmax** 对比，**hardmax** 会把向量z变成这个向量$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$，**hardmax** 函数会观察$z$的元素，然后在$z$中最大元素的位置放上 1，其它位置放上 0，所这是

一个 **hard max**，也就是最大的元素的输出为 1，其它的输出都为 0。与之相反，**Softmax** 所做的从z到这些概率的映射更为温和，我不知道这是不是一个好名字，但至少这就是 **softmax** 这一名称背后所包含的想法，与 **hardmax** 正好相反。



有一点我没有细讲，但之前已经提到过的，就是 **Softmax** 回归或 **Softmax** 激活函数将 **logistic** 激活函数推广到C类，而不仅仅是两类，结果就是如果$C = 2$，那么$C = 2$的 **Softmax** 实际上变回了 **logistic** 回归，我不会在这个视频中给出证明，但是大致的证明思路是这样的，如果$C = 2$，并且你应用了 **Softmax**，那么输出层$a^{[L]}$将会输出两个数字，如果$C = 2$的话，也许输出 0.842 和 0.158，对吧？这两个数字加起来要等于 1，因为它们的和必须为 1，其实它们是冗余的，也许你不需要计算两个，而只需要计算其中一个，结果就是你最终计算那个数字的方式又回到了 **logistic** 回归计算单个输出的方式。这算不上是一个证明，但我们可以从中得出结论，**Softmax** 回归将 **logistic** 回归推广到了两种分类以上。

接下来我们来看怎样训练带有 **Softmax** 输出层的神经网络，具体而言，我们先定义训练神经网络使会用到的损失函数。举个例子，我们来看看训练集中某个样本的目标输出，真实标签是$\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$，用上一个视频中讲到过的例子，这表示这是一张猫的图片，因为它属于类 1，现在我们假设你的神经网络输出的是$\hat{y}$，$\hat{y}$是一个包括总和为 1 的概率的向量，$y = \begin{bmatrix}0.3\\0.2\\0.1\\0.4\end{bmatrix}$，你可以看到总和为 1，这就是$a^{[l]}$，$a^{[l]} = y = \begin{bmatrix}0.3\\0.2\\0.1\\0.4\end{bmatrix}$。对于这个样本神经网络的表现不佳，这实际上是一只猫，但却只分配到 20%是猫的概率，所以在本例中表现不佳。



那么你想用什么损失函数来训练这个神经网络？在 **Softmax** 分类中，我们一般用到的损失函数是$L(\hat{y},y)=-\sum_{j=1}^{4}y_j log\,\hat{y}_j$，我们来看上面的单个样本来更好地理解整个过程。注意在这个样本中$y_1=y_3=y_4=0$，因为这些都是 0，只有$y_2=1$，如果你看这个求和，所有含有值为 0 的y_j的项都等于 0，最后只剩下$-y_2 t log\,\hat{y}_2$，因为当你按照下标j全部加起来，所有的项都为 0，除了$j=2$时，又因为$y_2=1$，所以它就等于$-log\,\hat{y}_2$。 $L(\hat{y},y)=-\sum_{j=1}^{4}y_j \log\hat{y}_j=-y_2\ log\,\hat{y}_2=-log\,\hat{y}_2$

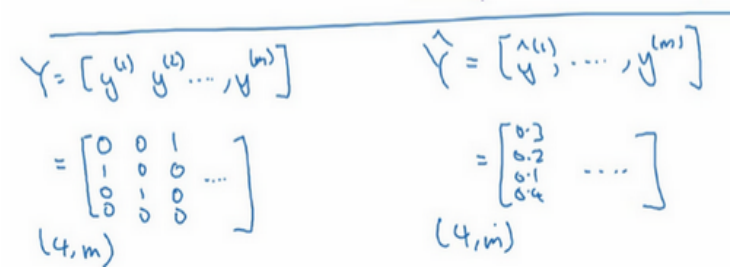
这就意味着，如果你的学习算法试图将它变小，因为梯度下降法是用来减少训练集的损失的，要使它变小的唯一方式就是使$-\log\hat{y}_2$变小，要想做到这一点，就需要使$\hat{y}_2$尽可能大，因为这些是概率，所以不可能比 1 大，但这的确也讲得通，因为在这个例子中x是猫的图片，你就需要这项输出的概率尽可能地大（$y=\begin{bmatrix}0.3\\0.2\\0.1\\0.4\end{bmatrix}$中第二个元素）。

概括来讲，损失函数所做的就是它找到你的训练集中的真实类别，然后试图使该类别相应的概率尽可能地高，如果你熟悉统计学中最大似然估计，这其实就是最大似然估计的一种形式。但如果你不知道那是什么意思，也不用担心，用我们刚刚讲过的算法思维也足够了。

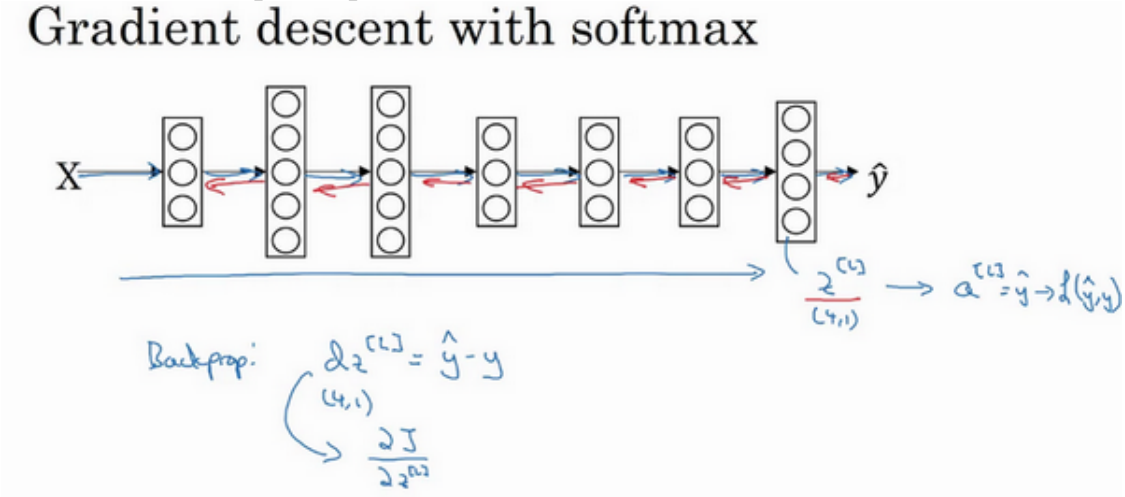
这是单个训练样本的损失，整个训练集的损失J又如何呢？也就是设定参数的代价之类的，还有各种形式的偏差的代价，它的定义你大致也能猜到，就是整个训练集损失的总和，把你的训练算法对所有训练样本的预测都加起来，

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

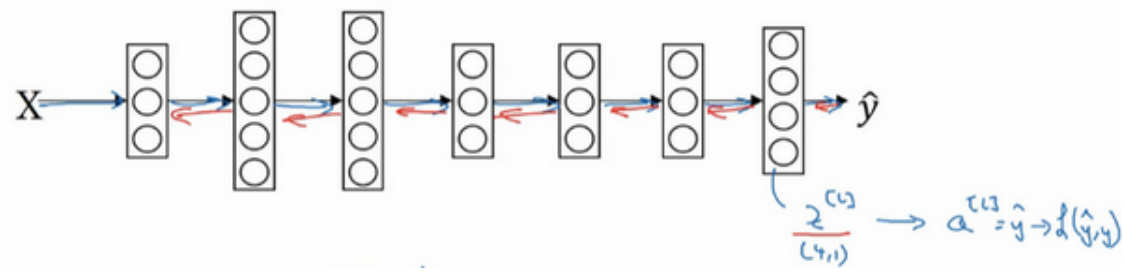
因此你要做的就是用梯度下降法，使这里的损失最小化。



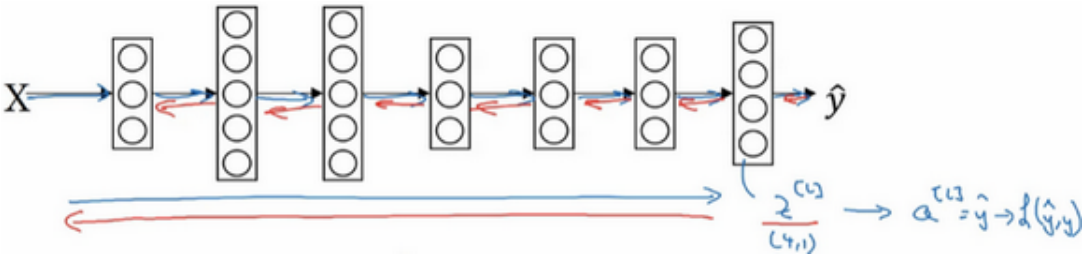
最后还有一个实现细节，注意因为$C = 4$，y是一个 4×1 向量，y也是一个 4×1 向量，如果你实现向量化，矩阵大写Y就是$[y^{(1)}y^{(2)} \ldots\ldots y^{(m)}]$，例如如果上面这个样本是你的第一个训练样本，那么矩阵$Y = \begin{bmatrix} 0 & 0 & 1 & \ldots \\ 1 & 0 & 0 & \ldots \\ 0 & 1 & 0 & \ldots \\ 0 & 0 & 0 & \ldots \end{bmatrix}$，那么这个矩阵$Y$最终就是一个$4 \times m$维矩阵。类似的，$\hat{Y} = [\hat{y}^{(1)}\ \hat{y}^{(2)} \ldots\ldots \hat{y}^{(m)}]$，这个其实就是$\hat{y}^{(1)}$（$a^{[l](1)} = y^{(1)} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$），或是第一个训练样本的输出，那么$\hat{Y} = \begin{bmatrix} 0.3 & \ldots \\ 0.2 & \ldots \\ 0.1 & \ldots \\ 0.4 & \ldots \end{bmatrix}$，$\hat{Y}$本身也是一个$4 \times m$维矩阵。



最后我们来看一下，在有 **Softmax** 输出层时如何实现梯度下降法，这个输出层会计算$z^{[l]}$，它是$C \times 1$维的，在这个例子中是 4×1，然后你用 **Softmax** 激活函数来得到$a^{[l]}$或者说y，然后又能由此计算出损失。我们已经讲了如何实现神经网络前向传播的步骤，来得到这些输出，并计算损失，那么反向传播步骤或者梯度下降法又如何呢？其实初始化反向传播所需要的关键步骤或者说关键方程是这个表达式$dz^{[l]} = \hat{y} - y$，你可以用$\hat{y}$这个 4×1 向量减去y这个 4×1 向量，你可以看到这些都会是 4×1 向量，当你有 4 个分类时，在一般情况下就是$C \times 1$，这符合我们对dz的一般定义，这是对$z^{[l]}$损失函数的偏导数（$dz^{[l]} = \frac{\partial J}{\partial z^{[l]}}$），如果你精通微积分就可以自己推导，或者说如果你精通微积分，可以试着自己推导，但如果你需要从零开始使用这个公式，它也一样有用。



有了这个，你就可以计算$dz^{[l]}$，然后开始反向传播的过程，计算整个神经网络中所需要的所有导数。



但在这周的初级练习中，我们将开始使用一种深度学习编程框架，对于这些编程框架，通常你只需要专注于把前向传播做对，只要你将它指明为编程框架，前向传播，它自己会弄明白怎样反向传播，会帮你实现反向传播，所以这个表达式值得牢记（$dz^{[l]} = \hat{y} - y$），如果你需要从头开始，实现 **Softmax** 回归或者 **Softmax** 分类，但其实在这周的初级练习中你不会用到它，因为编程框架会帮你搞定导数计算。

Softmax 分类就讲到这里，有了它，你就可以运用学习算法将输入分成不止两类，而是C个不同类别。接下来我想向你展示一些深度学习编程框架，可以让你在实现深度学习算法时更加高效，让我们在下一个视频中一起讨论。