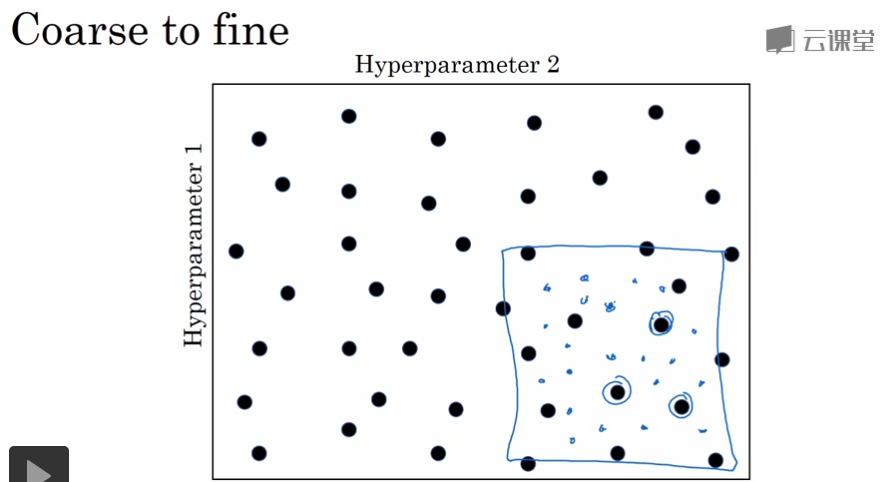
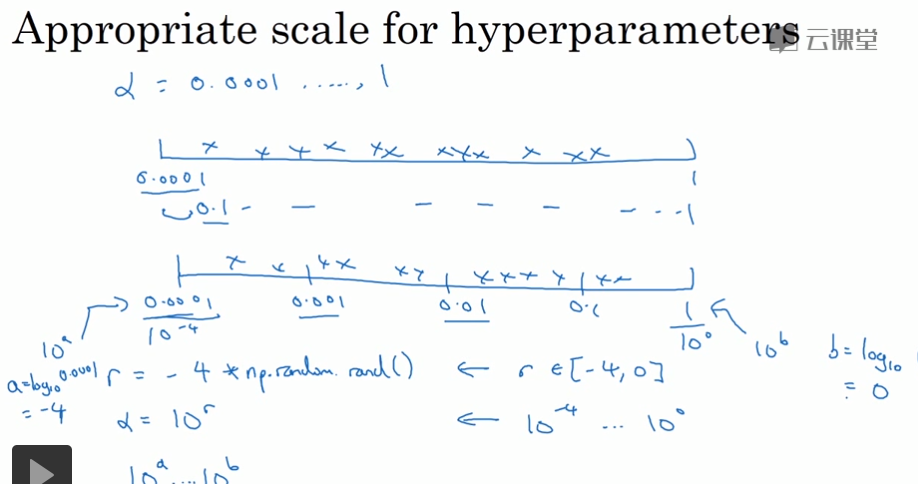
**1.超参数调试处理**

在机器学习领域，超参数比较少的情况下，我们之前利用设置网格点的方式来调试超参数；但在深度学习领域，超参数较多的情况下，不是设置规则的网格点，而是随机选择点进行调试。这样做是因为在我们处理问题的时候，是无法知道哪个超参数是更重要的，所以随机的方式去测试超参数点的性能，更为合理，这样可以探究更超参数的潜在价值。如果在某一区域找到一个效果好的点，将关注点放到点附近的小区域内继续寻找。



**2. 为超参数选择合适的范围**

在超参数选择的时候，一些超参数是在一个范围内进行均匀随机取值，如隐藏层神经元结点的个数、隐藏层的层数等。但是有一些超参数的选择做均匀随机取值是不合适的，这里需要按照一定的比例在不同的小范围内进行均匀随机取值，以学习率α的选择为例，在0.001,…,1范围内进行选择：



如上图所示，如果在 0.001,…,10.001,…,1 的范围内进行进行均匀随机取值，则有90%的概率 选择范围在 0.1∼10.1∼1 之间，而只有10%的概率才能选择到0.001∼0.10.001∼0.1之间，显然是不合理的。

所以在选择的时候，在不同比例范围内进行均匀随机取值，如0.0001∼0.001、0.001∼0.01、 0.01∼0.1、0.1∼1 范围内选择：

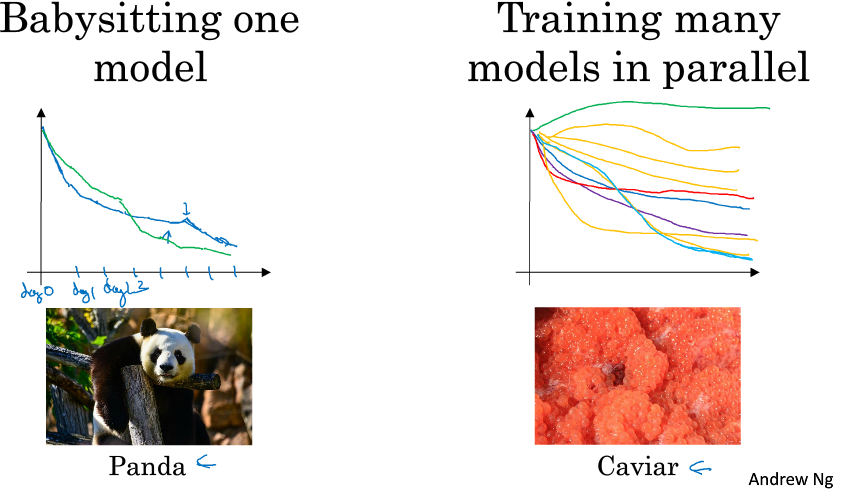
r = -4 \* np.random.rand() ；

learning\_rate = 10 \*\* r ；

一般的，如果在10a∼10b之间的范围内进行按比例的选择，则r∈[a,b]，α=10r。同样，在使用指数加权平均的时候，超参数β也需要用上面这种方向进行选择。

**3.超参数训练的实践**

两种方法：



在计算资源有限的情况下，使用第一种，仅调试一个模型，每天不断优化；

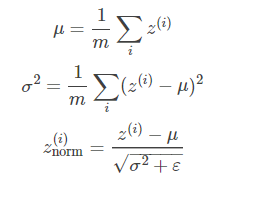
在计算资源充足的情况下，使用第二种，同时并行调试多个模型，选取其中最好的模型。

**4.正则化网络的激活函数**

常用的方式是将隐藏层的经过激活函数前的z[l]进行归一化。

**Batch Norm 的实现：**

以神经网络中某一隐藏层的中间值为例：z(1),z(2),…,z(m)：



这里加上ε是为了保证数值的稳定。

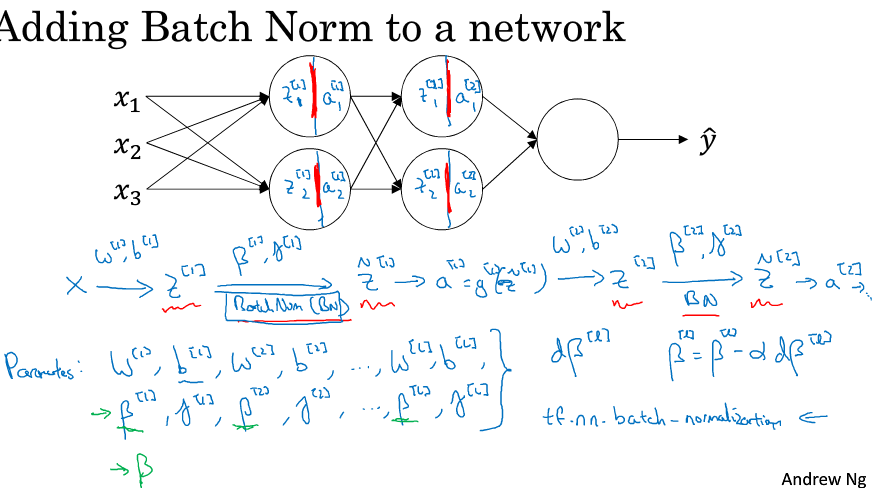
到这里所有z的分量都是平均值为0和方差为1的分布，但是我们不希望隐藏层的单元总是如此，也许不同的分布会更有意义，所以我们再进行计算：



这里γ和β是可以更新学习的参数，如神经网络的权重w一样，两个参数的值来确定z˜(i)所属的分布。

**5.将Batch Norm拟合进神经网络**

在深度神经网络中应用Batch Norm:



**实现梯度下降:**

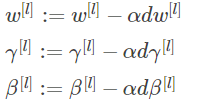
for t = 1 … num （这里num 为Mini Batch 的数量）：

在每一个 Xt上进行前向传播（forward prop）的计算：

在每个隐藏层都用 Batch Norm 将z[l]替换为z˜[l];

使用反向传播计算各个参数的梯度：dw[l]、dγ[l]、dβ[l];

更新参数：



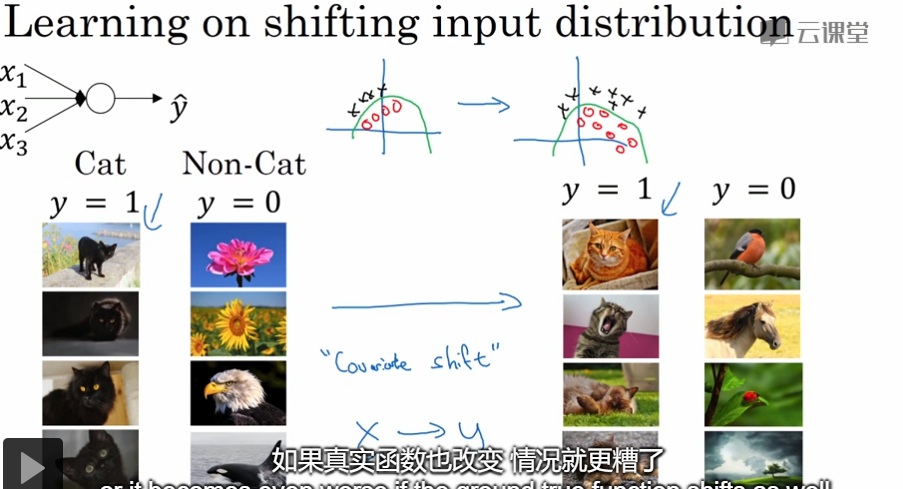
**6. Batch Norm 为什么起作用**

1、首先Batch Norm 可以加速神经网络训练的原因和输入层的输入特征进行归一化，从而改变Cost function的形状，使得每一次梯度下降都可以更快的接近函数的最小值点，从而加速模型训练过程的原理是有相同的道理。

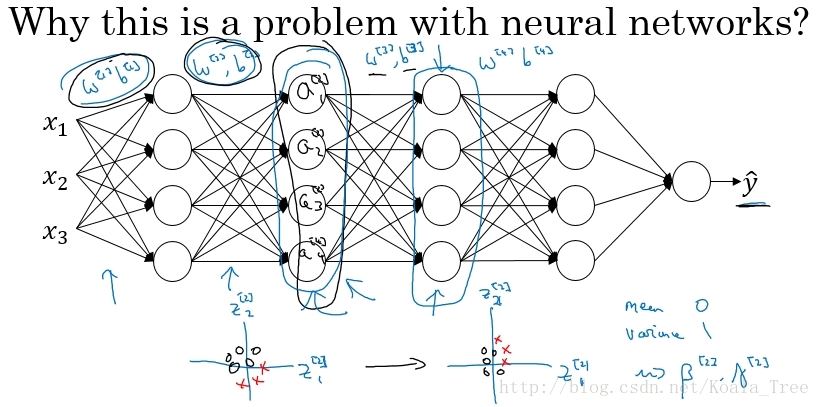
只是Batch Norm 不是单纯的将输入的特征进行归一化，而是将各个隐藏层的激活函数的激活值进行的归一化，并调整到另外的分布。

2、下面是一个判别是否是猫的分类问题，假设第一训练样本的集合中的猫均是黑猫，而第二个训练样本集合中的猫是各种颜色的猫。如果我们将第二个训练样本直接输入到用第一个训练样本集合训练出的模型进行分类判别，那么我们在很大程度上是无法保证能够得到很好的判别结果。

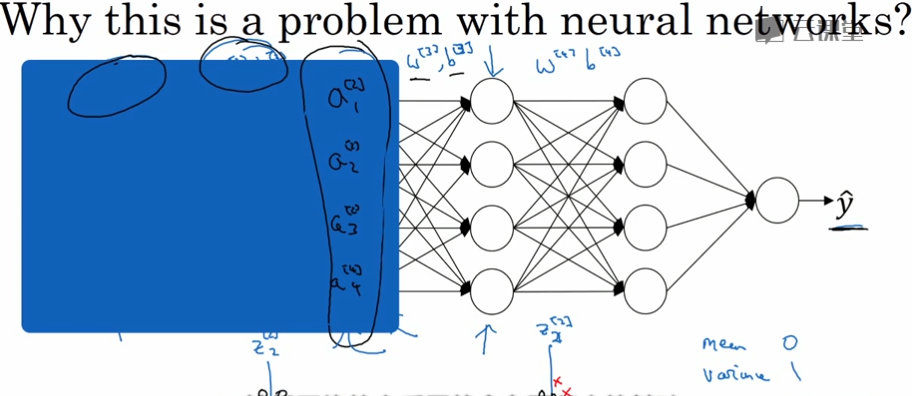
这是因为第一个训练集合中均是黑猫，而第二个训练集合中各色猫均有，虽然都是猫，但是很大程度上样本的分布情况是不同的，所以我们无法保证模型可以仅仅通过黑色猫的样本就可以完美的找到完整的决策边界。第二个样本集合相当于第一个样本的分布的改变，称为：Covariate shift。如下图所示：



那么存在Covariate shift的问题如何应用在神经网络中？就是利用Batch Norm来实现。



网络的目的是通过不断的训练，最后输出一个更加接近于真实值的y^。现在以第2个隐藏层为输入来看：



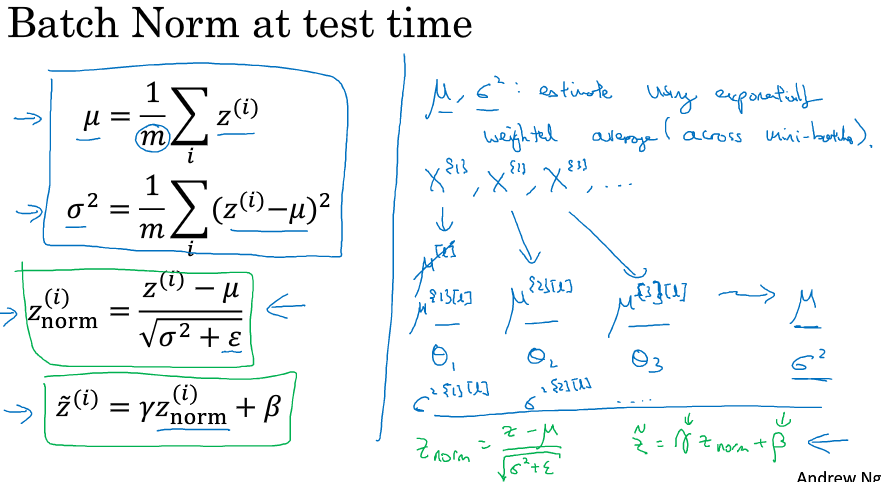
对于后面的神经网络，是以第二层隐层的输出值a[2]作为输入特征的，通过前向传播得到最终的y^，但是因为我们的网络还有前面两层，由于训练过程，参数w[1]，w[2]是不断变化的，那么也就是说对于后面的网络，a[2]的值也是处于不断变化之中，所以就有了Covariate shift的问题。

那么如果对z[2]使用了Batch Norm，那么即使其值不断的变化，但是其均值和方差却会保持。那么Batch Norm的作用便是其限制了前层的参数更新导致对后面网络数值分布程度的影响，使得输入后层的数值变得更加稳定。

**7.测试时的Batch Norm**

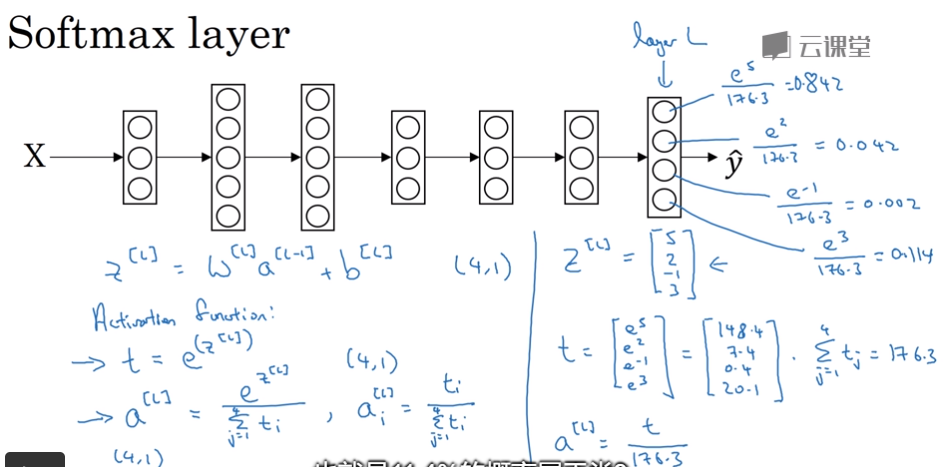
训练过程中，我们是在每个Mini-batch使用Batch Norm，来计算所需要的均值μ和方差σ2。但是在测试的时候，我们需要对每一个测试样本进行预测，无法计算均值和方差。

此时，我们需要单独进行估算均值μ和方差σ2。通常的方法就是在我们训练的过程中，对于训练集的Mini-batch，使用指数加权平均，当训练结束的时候，得到指数加权平均后的均值μ和方差σ2，而这些值直接用于Batch Norm公式的计算，用以对测试样本进行预测。



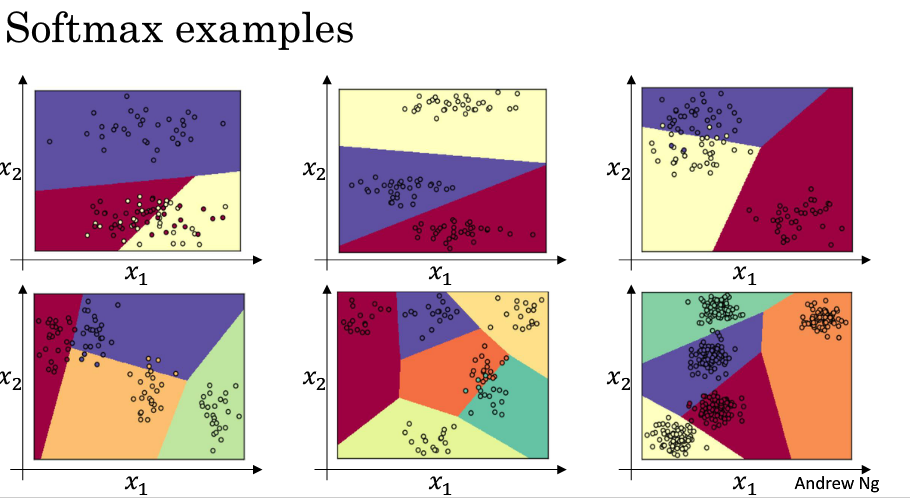
**8.Softmax 回归**

Softmax回归可以将多分类任务的输出转换为各个类别可能的概率，从而将最大的概率值所对应的类别作为输入样本的输出类别。

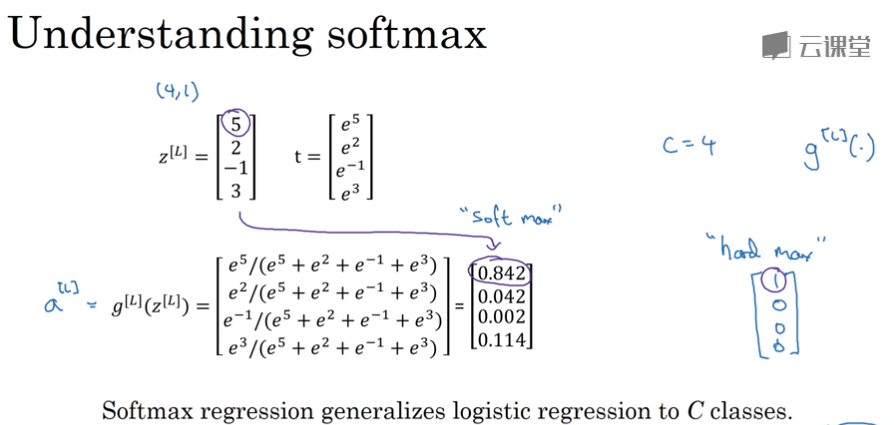


可以看出Softmax通过向量z[L]计算出总和为1的四个概率。

在没有隐藏隐藏层的时候，直接对Softmax层输入样本的特点，则在不同数量的类别下，Sotfmax层的作用：

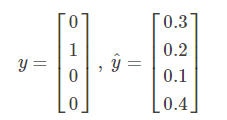


**9.训练Softmax分类器**



**Softmax 的Loss function**

在使用Sotfmax层时，对应的目标值y以及训练结束前某次的输出的概率值y^分别为：



Sotfmax使用的Loss function为：

  
在训练过程中，我们的目标是最小化Loss function，由目标值我们可以知道，y1=y3=y4=0， y2=1，所以代入L(y^,y)中，有：



所以为了最小化Loss function，我们的目标就变成了使得y^2的概率尽可能的大。

也就是说，这里的损失函数的作用就是找到你训练集中的真实的类别，然后使得该类别相应的概率尽可能地高，这其实是最大似然估计的一种形式。

对应的Cost function如下：



在Softmax层的梯度计算公式为：

