第三章 超参数调试 batch正则化和程序框架

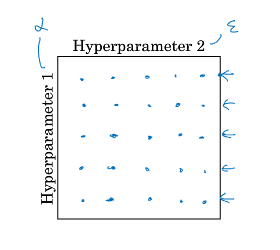
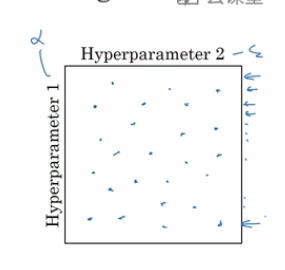
3.1 调试处理

调整神经网络的过程 包含了对很多不同超参数的设置，如何为这些参数找到合适的设定值呢。分享一些指导原则以及系统化进行超参数调优的技巧，它们将帮助你更有效地获得合适的超参数。

在深度神经网络训练中面对大量超参数：

|  |  |
| --- | --- |
|  | 学习速率 |
| β | Momentum梯度下降超参数 |
| 和ε | Adam优化算法 |
| L | 网络层数 |
| The number of units | 每层网络中隐藏单元的数量 |
| Decay\_rate | 学习率衰减衰减率，学习速率 |
| Mini-Batch的大小 | Mini-batch梯度下降法设置每个batch大小 |

大多数学习算法应用，学习速率α 是需要调优的超参数中最重要的一个，没有之一。接下来会调整的一些超参数，也许是动量项β，0.9是一个不错的默认值，还会调整Mini-Batch的大小，来保证最优化算法的运行效率，以及调试隐藏单元数量。这三个是我认为重要性仅次于学习速率α的超参数。接下来第三重要的就是：网络层数，有时候对结果起到重要作用；学习率衰减有时也一样；当使用Adam优化算法时 我几乎不调节 和ε，几乎都是用0.9 0.999和。

但这也不是死规定，可能也有自己的观点。如果你要调整一些超参数，该怎样选择这些值的组合呢？在早期的机器学习算法中，如果你有两个超参数，假设是超参数1和超参数2，人们经常会像这样 在一个网格中对点进行取样，然后系统化地尝试这些点所代表的值。也就是说，相当于一个网格，在网格点上的数进行尝试。在这个例子中当你尝试过所有25个点后，选择最优的超参数，当超参数的数量相对较少时，这样的取参方法较为实用。但是在深度学习中，我推荐你采取另一种方法：随机采样。像这样随机选择一些点，同样的我们选择25个点，然后在这些随机选取的点中尝试所有的超参数。这样做的原因是事先你不知道到底哪个参数才是重要的，也许另外的参数并不是很重要，那么你的25个组合就只有5个关于某个参数的值调整起到了作用，也就相当于你训练了25个模型，实际上只尝试了5个关于α的重要参数值。所以通过随机取样，即使其中有个参数不影响结果，但是可能也尝试了25个关于重要的参数的值，并且看到了不同的效果，关于重要的那个参数找到理想值的概率就大些。

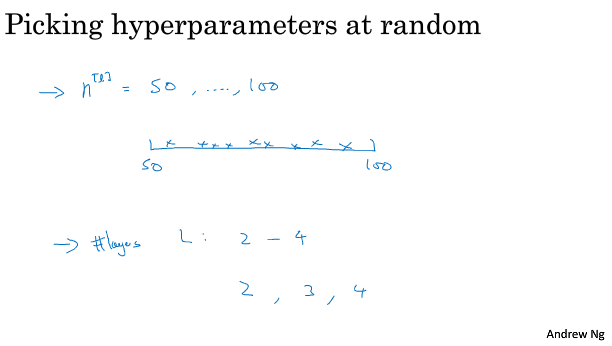
上面的例子只有两个超参数，实际上你可能会面对更多的超参数，比如说你有3个超参数 那么你将不是在平面正方形中寻找，而是在三维立方体中寻找超参数3，即为第三个维度，通过在这个三维立方体中抽样，你需要尝试更多这三个超参数的组合值。在实践中你要寻找的可能不仅仅是三个超参数，真正的困难在于，你需要事先知道哪一个超参对于你的模型更重要，这种情况下使用随机取样。而不是在网格中规则抽样，随机抽样将帮助你更充分地为最重要的超参数，尝试尽可能多的值的组合。

无论最重要的超参数是哪个，另一个常见的做法是使用区域定位的抽样方案，也就是说你在二维参数中发现某个点产生最好的效果，并且周围的点表现的也不错，那么久对这些点所在的区域进行限定，然后在这个区域内进行密度更高的抽样，或者依然选择随机抽样，但是需要把更多资源限定在你后来选择的小区域中进行搜索，前提是你大体能确定这个区域内取的值能产生最优结果。

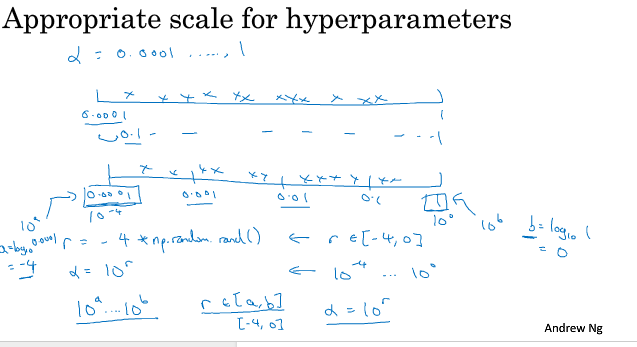
所以这一节重点就是关于很多的超参数，应该大概知道重要性排序，以及如何设置合适值，使用随机抽样，而不是在网格中定点搜索，或使用区域定位的搜索过程，除此以外还有很多超参数探寻的方法，下一节学习如何正确选择超参数取样的范围。

3.2 为超参数选择合适的范围

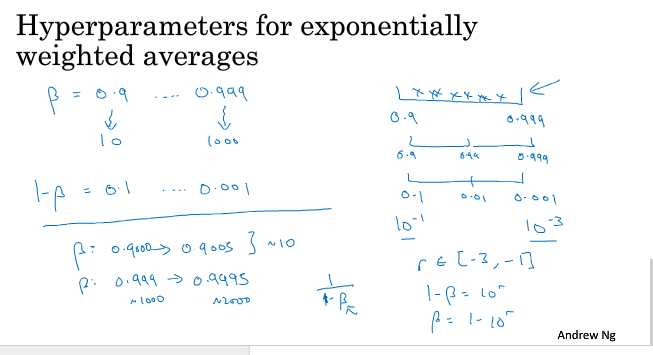
超参数值域的随机抽样能让你更有效地搜索超参数空间，但实际上,随机抽样并不意味着在有效值范围内的均匀随机抽样(sampleing uniformly at random)。相反,更重要的是选取适当的尺度(scale)，用以研究这些超参数。接下来展示如何改变尺度从而可以完成选择性的在比较可靠的范围内进行随机抽样。假设你现在要选择第l层隐藏单元的个数，你认为50~100是个不错的范围，那么,参考数轴上从50~100的范围，也许你会在这个线段上随机地取一些数值，这是搜索这个超参数的非常直观的方法。或者如果你要决定神经网络的层数L，认为这个层数应该在2~4之间，那么均匀随机的抽样,即2,3,4,这样也是合理的，或者甚至用网格搜索,显式计算。这两个关于如何选择单元层隐藏单元数以及神经网络层数使用均匀随机抽样是合理的，在我们的关注域中。但这种方法并不是对所有的参数都有效。



例如，正在搜索超参数即学习率，你认为它的范围应该是0.0001到1 比较合适，现在画出从0.0001~1的数轴，并均匀随机地抽取样本值，那么90%的样本值将落在0.1~1的范围内，即你用90%的资源搜索0.1~1，只有10%的资源用于搜索0.0001~0.1范围内的值，看起来不大对，更合理的方法似乎应该以对数尺度(log scale)来搜索，而不是用线性尺度(linear scale)。将0.0001到1化为对数，则是关于10的-4到0次方，因此这个数轴应该为-4~0，接下来你可以在这个对数尺度上均匀随机的取样，可以有更多的资源致力于搜索。用Python实现就是,然后随机取。第一行运行之后,r为-4~0的随机数，值在~之间。更一般的来说，如果你要在10^a~10^b的范围内取样，根据对数的定义，将a和b的值解出来，要做的就是在a~b的范围内均匀随机取样。整理一下,要基于对数尺度取样,首先取得下限值，取其对数得到a，再取上限值,取其对数得到b，然后在对数尺度上在10^a~10^b范围内取样。



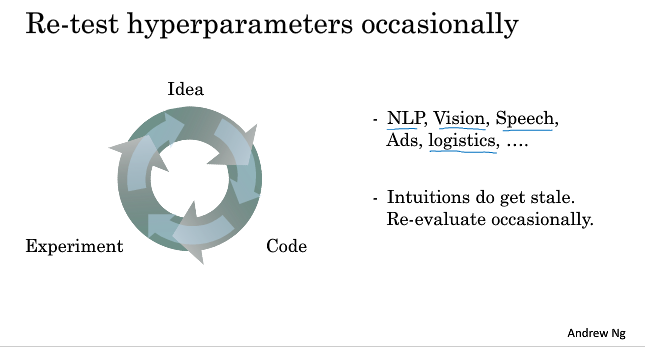
另一个棘手的问题就是关于使用到指数加权平均时的，假设你认为beta值应该在0.9~0.999之间，也许这是你要搜索的值空间，要记住的是,用0.9计算指数加权平均值，相当于计算最后10个值的平均值, 使用0.999就相当于计算1000个值的平均值。,如果你要搜索，0.9~0.999的范围,线性尺度的取样，即均匀的，随机的，0.9至0.999范围内的搜索没什么意义。那么对于这个问题，最好的办法就是将这个范围展开为

1. ，得到0.1~0.001的范围，那么就是用刚才学到的关于对α的处理方法，0.1~0.001即是到，注意之前的数轴是从左到右递增，现在反过来，那么你要做的就是在-3~-1的范围内均匀随机的取样。并且有关系 即 。 就得到了这个超参数在适当尺度上的随机取样值。在探索0.9~0.99和0.99~0.999的范围时，用了同样数量的资源。为什么以线性尺度取样是个坏主意呢，因为在指数加权平均中，我们有个公式为，表达的计算结果是对多少个样本的平均计算，从公示可以看到，随着趋近于1，当趋近于0时，整个结果比较大，影响也比较大，而结果与0相差很大的范围内变化，对结果并没有造成很大的影响。所以当β趋近于1时,它对β的改变非常敏感，所以整个取样过程应该做的就是在β趋近于1的区域内以更大的密度抽样。也就是在1-β趋近于0的时候，能得到更高效的样本分布，在搜索可能的超参数空间时更有效率。
2. 

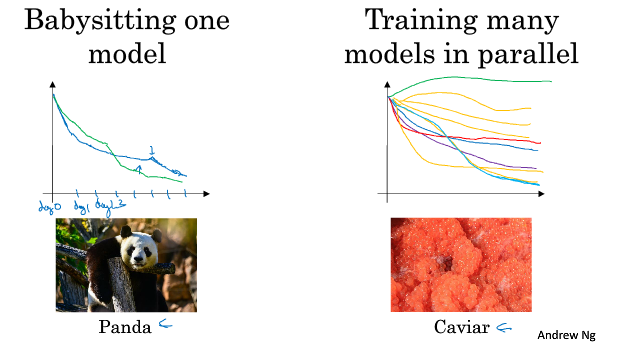
讲完在对超参数取样时选择正确的尺度，如果对于某个超参数你最后选择的尺度是不对的，也不用太担心，即使在存在更优尺度的情况下，你依然选择了均匀尺度(uniform scale)，你仍然可能得到不错的结果，尤其是如果你采取从粗到精(coarse to fine)的搜索策，在后期的迭代中，你将更专注于在最有用的超参数值域取样。

2.3 超参数训练的实践：Pandas VS Caviar（熊猫vs鱼子酱）

前面学习了如何探寻合适的超参数，在对各种探寻方式总结之前，分享一些小技巧。深度学习已经被应用在生活中的方方面面，而涉及某一个领域的超参数设置，对于其它领域来说，可能碰巧适合但也可能不适合。有许多不同领域可以抽象出很多共同点，例如，为计算机视觉识别领域建立模型的某些思路，比如Confonets和ResNets 同样适用于演讲领域以及在演讲领域发挥作用的构思，同样被成功应用于NLP。在超参数设置的问题上，无论是直觉还是思路，都应该与时俱进，哪怕你只研究某一个问题，例如逻辑学，并且已经找了到最优超参数设置，用来持续优化你的算法，而在过去一段时间服务器数据发生了变化， 原先那些被你定义为最优的超参数，可能就不合适了。 所以我建议至少每隔几个月重新检测或者重新评估一次你认为最优的超参数，以此来确保这些数值依然是最优解。



关于探寻超参数的两种途径：Pandas VS Caviar，意思是熊猫以及鱼子酱。Pandas指的是你精心照料一个模型，通常你需要处理一个非常庞大的数据集，但没有充足的计算资源，比如没有很多CPU 没有很多GPU，那你只能一次训练一个或者非常少量的模型。例如某天首先你开始把参数随机初始化，开始训练，紧盯着你的学习曲线，也许是损失函数J，或者是数据集误差或其他曲线，第一天曲线呈下降趋势，有点满意了。第二天再看看增加一点学习率，是不是表现的更好，再后面或者又尝试一下Momentum或者减少一点学习率等等。就象这样你每天都在照看你的模型，尝试微调参数，照看模型，观察性能曲线。这通常发生在你没有足够的计算资源同时训练几个模型的情况下，需要像个熊猫照顾它仅有的一个孩子一样细心。另外一种情形就是并行训练许多个模型，这种情况下你可能设置一些超参数，然后让模型自己运行一天或几天，然后你可以得到像这样的学习曲线，这可能是损失函数J或者训练的错误率，或者数据集错误率，基本都是你会关心的指标，与此同时你可能会使用完全不同的超参数，开始运行另一个不同的模型。而第二个模型可能会反馈一条不同的学习曲线，你同时运行的其他模型可能还有更多的学习曲线，同时运行许多不同的模型。用这样的方式就可以尝试不同的超参数设置，这可以让超参数选择变得简单，只要找一个最终结果最好的就行了。这种方式就像鱼一次产卵无数个，只需要有一个或者一部分能够存活。所以这分别就是Pandas和Caviar。



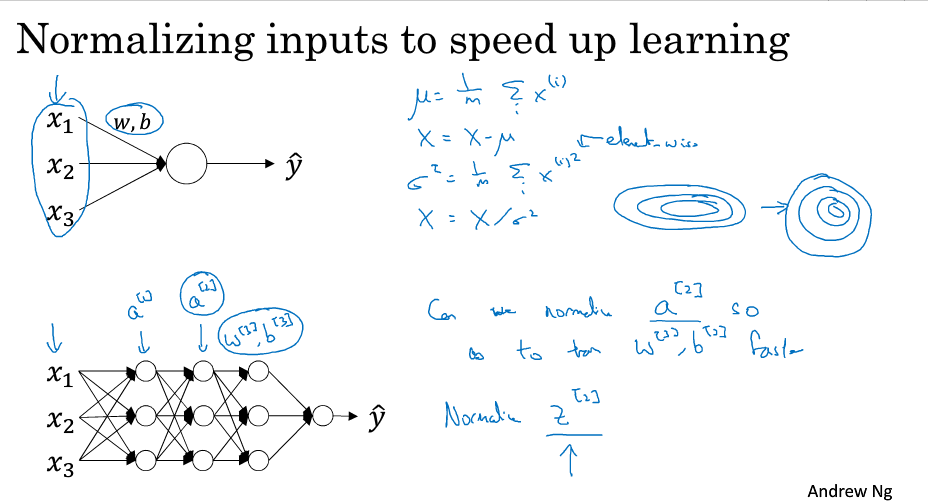
那么如何挑选适合你的模式呢？这取决于你有多少计算资源，如果你有足够的计算机来并行训练很多模型，那不用考虑别的，采用鱼子酱模式就行了，尝试大量不同的超参数 看看结果如何。但是在某些应用领域，例如在线广告设置，以及计算机视觉识别，都有海量的数据和大量的模型需要去训练，而同时训练大量模型是极其困难的事情，这实际上是由行业性质所决定的。熊猫一次只精心照顾一个孩子，但是一生也有很多个孩子，判断一个模型是否可行后，可能发现需要重新初始化一个新的模型，重新照料。

还以另外一种技巧能够使你的神经网络变得更加坚实，它并不是对所有的神经网络都适用，但是一旦适用，可以使超参数搜索变得更容易并且加速试验过程。

3.4 正则化网络的激活函数

深度学习不断兴起，重要的创新之一，batch normalization（批量归一化），可以让你的超参搜索变得很简单，让你的神经网络变得更加具有鲁棒性，可以让你的神经网络对于超参数的选择上不再那么敏感，而且可以让你更容易地训练非常深的网络。

看看batch normalization如何工作，之前讲过训练一个模型比如说逻辑回归模型时，对输入特征进行归一化可以加速学习过程，即归一化输入特征。公式为：



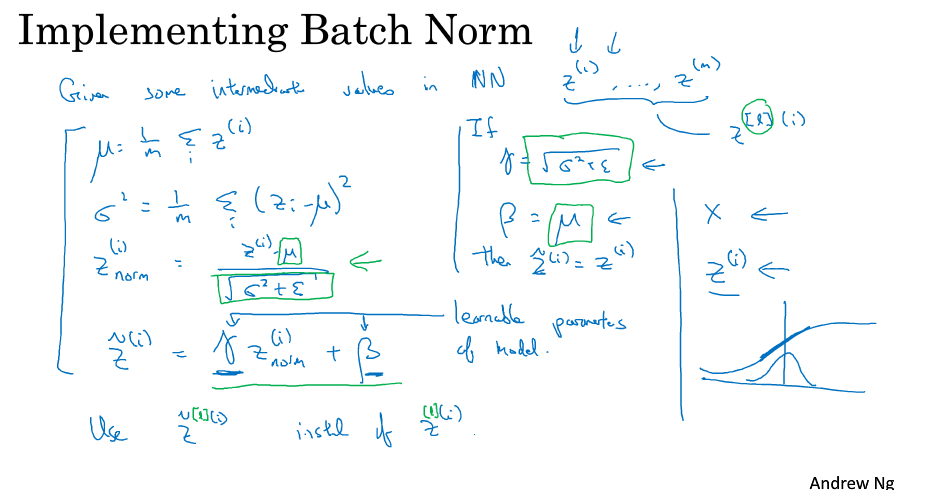
这是前面讲过的如何让机器学习的图形从扁的变成圆的，会便于类似梯度下降的算法去做优化，所以对于神经网络或者逻辑回归，针对输入值的归一化处理是有用的。那么对于层数更多的模型呢？你不仅有输入特征值x，在每一层你还有激活函数的输出结果a，例如下图中的四层神经网络，如果想训练参数和，就得考虑上一层的激活函数输出结果，除非你能对进行标准差归一化，否则对于和的训练都不会太有效率。 在逻辑回归的例子里 我们看到了对x1x2 x3做归一化可以对对w和b的训练更有效，所以这里的问题就是，对于任何一个深层神经网络的隐藏层我们是否也可以对激活函数的输出结果做归一化处理呢。这就是batch normalization，实际上我们我们归一化时针对的 并不是]而是，在激活函数使用前做归一化，还是使用后归一化，这一点还是有争议的，但是对做归一化比较普遍。

Batch normalization实现：

假设有一些神经网络中的中间值，隐藏单元到,来源于某个隐藏层，给你这些值时，你要做的是像这样计算平均值，所有这些都是针对某一层，计算方差并且对每个去归一化。公式为：

(4)中为了避免在某些情况下分母为0，通常在分母加上ε，经过上面3个步骤。就把归一化为一组均值0方差1的值了，并且每一组都是均值为0，方差为1。但是我们并不希望所有的隐藏单元都是这样的，也许本身它们的分布就有不同，所以可以完成第（4）步，对每组隐藏单元的均值和方差再做调整，γ和β值可以从你的模型中学习。就可以使用梯度下降算法，或者其他类似算法比如momentum的梯度下降算法，或者atom算法 来更新γ和β的值，就像更新神经网络的权重一样。

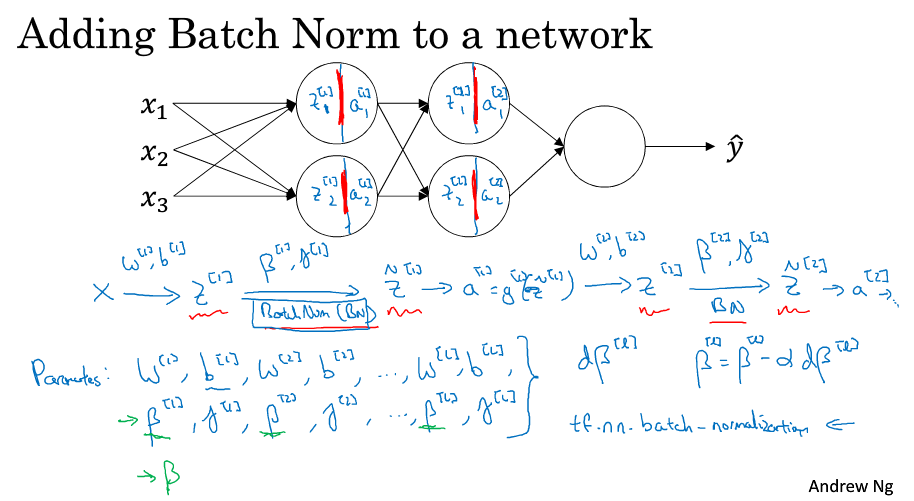
如果 并且，那么，通过选择不同的γ和β 可以让隐藏单元的呈现不同的分布，让你的隐藏层可以是其他的任何均值或者方差，使就是通过设置恰当的γ和β。关于神经网络激活函数的归一化就是上面四个步骤，用代替再进行后面的计算。



这一节我们学到的是如何对输入特征X做归一化来帮助神经网络的训练，以及batch norm所做的就是不仅仅在输入层，而且在一些隐藏层上也做归一化。你使用这种归一化方法，对某些隐藏单元的值z做归一化，但是输入层和隐藏层的归一化还有一点不同，就是隐藏层归一化后并不一定是均值0方差1，通过γ和β参数可以进行任意的修改，比如激活函数是sigmoid时，不希望归一化之后都聚集在原点附近，希望有更大的方差，以便于更好的利用sigmoid函数非线性的特性，而不是所有的值都在中间这段近似直线的区域上，这就是为什么通过设置γ和β，你可以控制在你希望的范围内。这一节你对如何实现batch norm有了初步的了解，至少是在单层神经网络上，下一节展示如何在神经网络中使用batch norm，包括深度神经网络，以及如何用数据训练它，以及如何使它在不同的层都能起到作用。

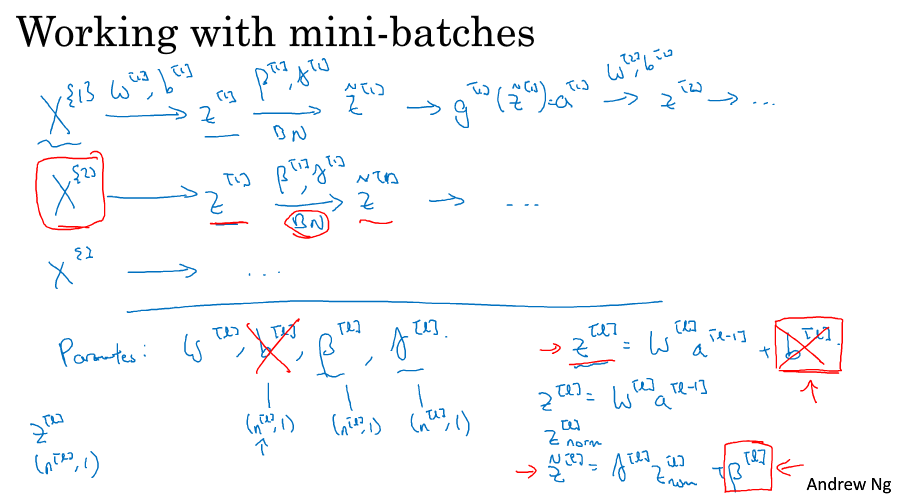
2.5 将batch normalization 拟合进神经网络

上一节的四个公式已经学习了单隐藏层的Batch norm公式，这一节看看如何在深度网络中使用。假设有这么一个三层神经网络如下：



每一个节点即圆圈都可以看作是在计算两个事情，如果不使用batch norm，那么就跟之前过程一样，由，如果加入了batch norm那么整个过程则是，上图中的流程图所示。再作为输入的后面，进行norm变化，这个过程缩写为BN，由参数以及决定，产生新的归一值,再把它作为输入给激活函数来获取，这样就完成了第一层的计算，总的来说，batch norm被应用于由Z到A的计算过程中。下一步，用来计算，由参数和决定，之后再进行BN过程，由参数以及决定，以此类推。直觉是与其使用没有归一化的Z值，不如用均值和方差都归一化之后的值，第二层同理。所以我们的参数就是：

对每个采用BN算法的网络层都要如此，需要注意的是，这里的参数，等于Momentum等算法中用于计算各种加权平均值的超参数β无关。以上就是你算法的新参数，接下来你可以使用任何想用的一种优化法。例如，采用梯度下降法的话，就要计算，然后更新参数，，也可以用Adam或RMSprop或Momentum算法来更新参数。如果你正在使用深度学习编程框架，通常你不必自己去实现BN算法，把batch归一化步骤应用于batch归一化层，仅仅是一行代码直接实现，例如在TensorFlow框架中，可以用这个函数来实现batch归一化，。在实践中不必自己操作所有这些具体的细节。到目前为止我们已经讲过了在整个训练集上利用批量梯度下降（batch gradient decent）训练的BN算法，实际上通常都是使用mini-batch BN算法，也就是说BN 批量归一化与小批量梯度下降配合使用。过程如下图，其实就是把每一个子样本都进行BN操作。



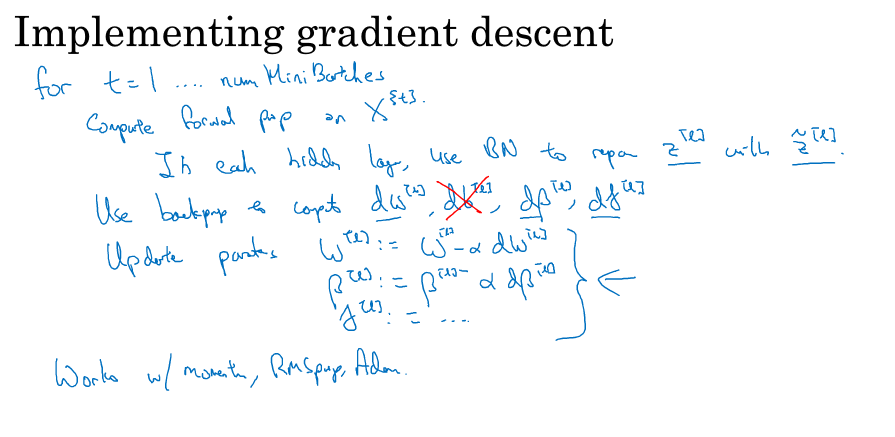
有个细节需要注意，就是前面说过关于BN使用到神经网络中时有上面那些参数，但是其中关于b的参数都是可以省略的，原本，但归一化做的是在这个mini-batch上，把先归一化为均值为0和方差为1，再由重新缩放，这就说明无论是多少，都是要被减去的，因为归一化过程要先计算均值，再减去平均值，在计算时候增加任何常数数值都不会改变。所以你使用BN的话，可以消除b参数。那么

因为BN算法使层级中各个的均值为0，我们就没有理由保留参数，所以就把它忽略了，相应地被所代替，也就是一个用来控制最终偏移量影响的参数。最后记住关于的维度是,那么也是，为隐藏层的单元数，被用于调整每一个隐藏单元的均值和方差，这是由网络决定的。

汇总一下，如何使用mini-batch梯度下降法以及BN算法：

|  |
| --- |
| 对每个隐藏层应用正向prop      计算完了梯度就可以使用梯度下降法更新参数了 |

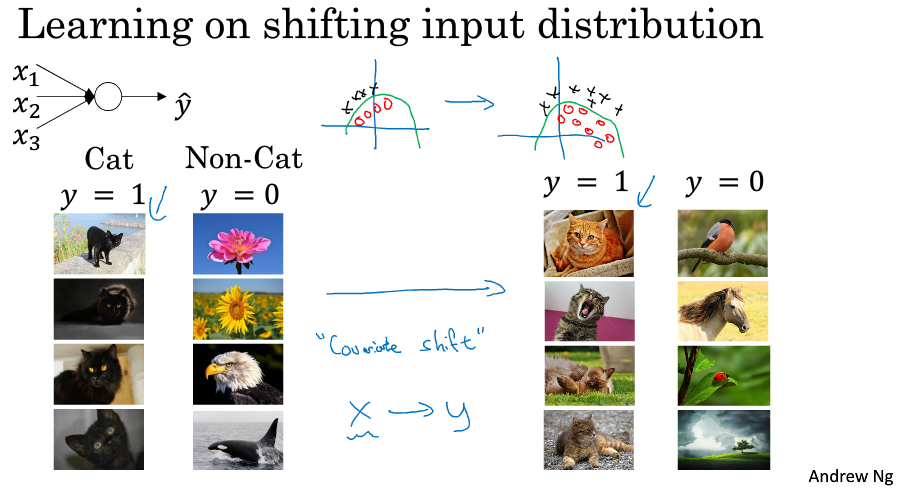
除了使用梯度下降法，也适用于Momentum，RMSprop，Adam梯度下降法。语气使用梯度下降法更新参数，也可以使用其他的算法来更新。也可以使用其他的优化算法来更新batch归一化添加到算法中的β和γ参数。

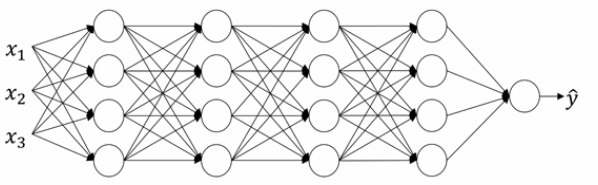


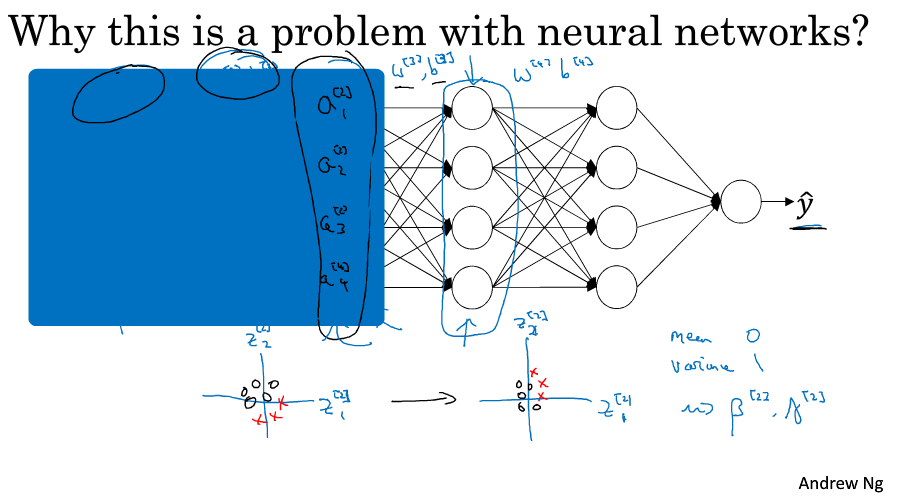
如果你正在使用某种深度学习编程框架，希望能够调用编程框架中的方法，来轻松实现BN算法，下节了解BN算法为什么会有效提高训练速度。

3.6 Batch norm为什么奏效

Batch norm到底是如何起作用的呢。原因1，经过归一化的输入特征(用X表示) 将大幅加速学习过程，与其含有某些在0到1范围内变动的特征，或在1到1000范围内变动的特征，通过归一化所有输入特征X，让它们都拥有相同的变化范围将加速学习，所以 BN算法有效的一个原因是。BN同样如此，只不过它应用于隐藏层的值，而不是这里输入特征。原因2，BN可以使权重，比你的网络更滞后或者更深层，比如，第10层的权重更经受得住变化相比于神经网络中浅层的权重，也就是说前面层的参数变化，对第10层的影响相对减小。举个例子，比如在某个网络上训练猫咪检测深度网络，假设我们的数据集是在所有黑猫图片上训练得到的，现在我们将其应用于所有的正确结果但并不像左侧黑猫那样的彩色猫咪图片上，这时候，正面例子也就是猫图的样本除了黑白猫图还有彩色猫图，那么分类将会出错，原先的模型可能适用的不是很好。又比如说，在图像中，你的训练集是如图红色代表正面例子，黑色反面例子，你试图把它们都统一在一个数据集，无法期待在前者数据集分布上训练的很好的模型同样在后者也运行的很好。尽管这里可能存在着相同的有效函数，但你就不要指望你的学习算法能找到那条绿色决策边界。我们的数据分布随着某种东西在变化，就是covariate shift，协变量。具体想法是，如果我们学习了某种x-y映射，如果x的分布变化了，那么我们就得重新训练学习算法，这是一定的。这种想法同样适用于如果由X到Y映射保持不变，正如此例中，因为真实函数就是确定一个图片是否是猫图，训练你的函数变得更加迫切。但是如果真实函数改变，情况将会变得糟糕。Covariate shift的问题是怎么体现在神经网络中的呢？

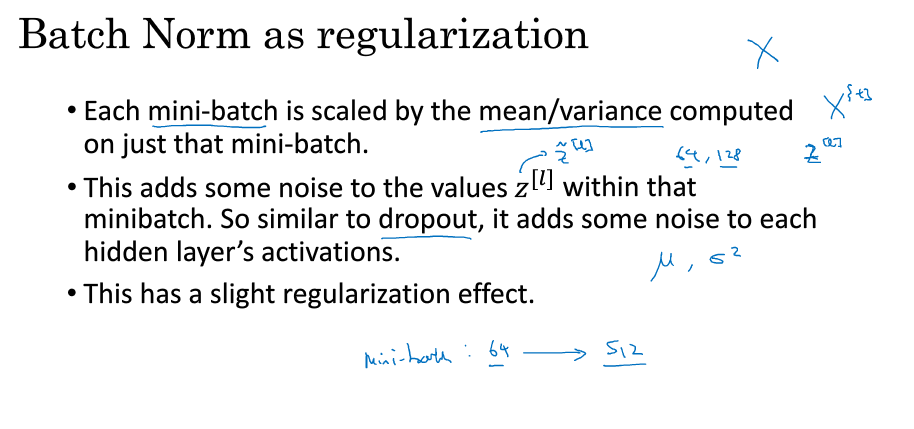


试想有这么一个神经网络，从第三层来分析一下学习过程，假设网络已经学习得出了参数和，从前面层获得了某些值，会做一些处理使得输出值更接近于真实值y。从第三层角度来看，获得了某些值，假设为，这些值可能和特征值x1 x2 x3 x4一样，第三级隐藏层的作用就是以这些值做输入然后找到某种将它们映射到的方法，假设后面还有做梯度衰减，第三层的左边同样会学习参数和等，所以当这些参数变化时，等的隐藏单元值会发生变化，所以受协变量问题的影响。所以BN算法所做的就是减少了这些隐藏单元值的分布的不稳定性，当神经网络刷新前几层参数时它们的确会变化，第二层的Z值肯定会变化，但BN算法确保的是无论它怎么变，的均值和方差都不会变，均值为0，方差为1 ，当然也可能是受β和γ影响的其他均值和方差，这两个参数由神经网络选定。所以BN限制了先前层中参数的更新，对第三层也就是现在所看到和要学习的值的分布的影响。BN算法减少输入值变化所产生的问题，所以神经网络的后层可以有更加稳固的基础。尽管输入分布变化了一点，它变化的更少 实际是尽管前几层继续学习，后面层适应前面层变化的力量被减弱。原因2， BN算法削弱了前面层参数和后层参数之间的耦合，允许网络的每一层独立学习，有一点独立于其它层的意思，有效提升整个网络学习速度。从神经网络某一后层角度来看，前面的层的影响并不会很大，因为它们被同一均值和方差所限制，所以这使后层的学习工作变得更加简单。



原因3，BN算法还有轻微的正则化的效果，mini-batch是BN算法中一个模糊的概念，在该min-batch上计算了均值和方差，而不是在整个数据集上计算，所以该均值和方差包含有噪声，因为它是由相对较少的数据集评估得来。所以归一化计算的过程也会有噪声，因为它是用带有一定噪声的均值和方差来计算的。所以和dropout算法类似，它会为每个隐藏层的激活函数增加一些噪声。dropout有噪声的原因是，它将隐藏单元以一定概率乘以0，以一定概率乘以1，所以dropout有着许多噪声。然而BN算法有着许多噪声，均值和标准方差的评估值都含有噪声，所以 和dropout类似，BN算法因此具有轻微的正则化效果，通过在隐藏单元上增加噪声，它使后续的隐藏单元不要过度依赖其它隐藏单元。因为加入的噪声太小，这没有强大的正则化效果，如果你想获得更强大的正则化效果的话，可以BN和dropout混合使用。

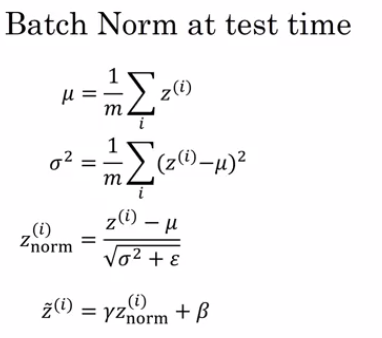
原因4，另一个模糊的效果，如果我们采用更大尺寸的mini-batch，假设尺寸为512而不是64，通过使用更大尺寸的mini-batch，我们可以减少噪声同时也减少了正则化效果。通过使用更大尺寸的min-batch，将会削弱正则化效果，肯定不会用BN算法作为正则化器，但是有时候它对我们的学习算法有着些额外的有意或无意的影响。所以不要把BN看成正则化方法，而是用来归一化隐藏单元激活函数用以加速学习的方法。



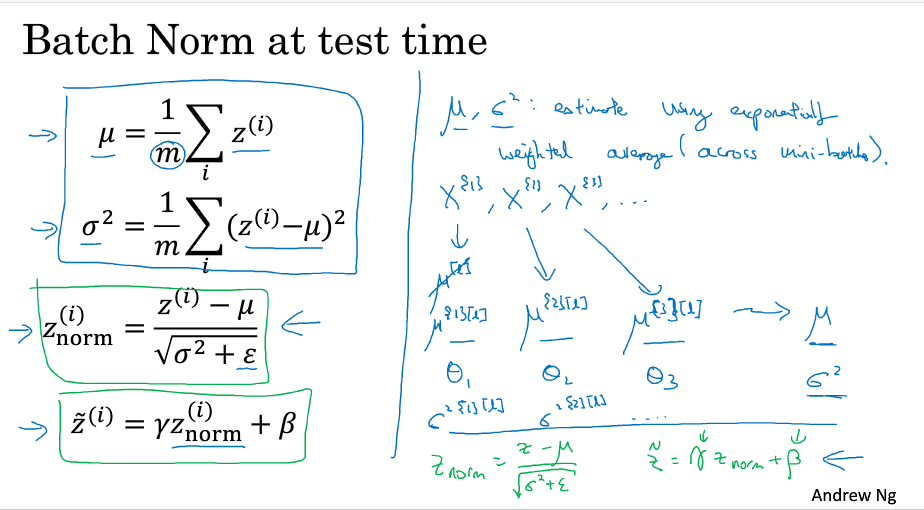
BN算法中一个mini-batch处理一次数据，在该mini-batch上计算均值和方差，所以在测试时当你尝试做出预测和评估神经网络时，你可能没有包含多个例子的min-batch，你可能每次只处理一个例子，在测试时你需要做些变化，来确保你做出的预测准确，下一节讨论使用BN算法训练神经网络以正确做出预测所要做的细节。

3.7 测试时的batch norm

Batch norm将你的数据以mini-batch的形式逐一处理，但在测试时，你可能需要对每一个样本逐一处理，如何修改神经网络来实现这一功能。在训练时需要用到的Batch归一化等式如下：



在一个mini-batch中，将求和，计算均值，只要把每一个mini-batch的样本都加起来，用m表示这个mini-batch中的样本数量，而不是整个训练集，然后再计算方差，以及，再用均值和标准差来调整，加上是为了数值稳定性，是用和再次调整。注意用于调节计算的和是在整个mini-batch上计算的，但是在测试时，可能不能将一个mini-batch中的6428或2506个样本同时处理，所以需要其他方式来得到和。而且如果你只有一个样本，一个样本的均值和方差没有意义，实际上，为了你的神经网络运用于测试需要单独估算和。需要用一个指数加权平均来估算，这个平均涵盖了所有的mini-batch。

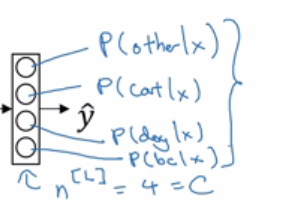


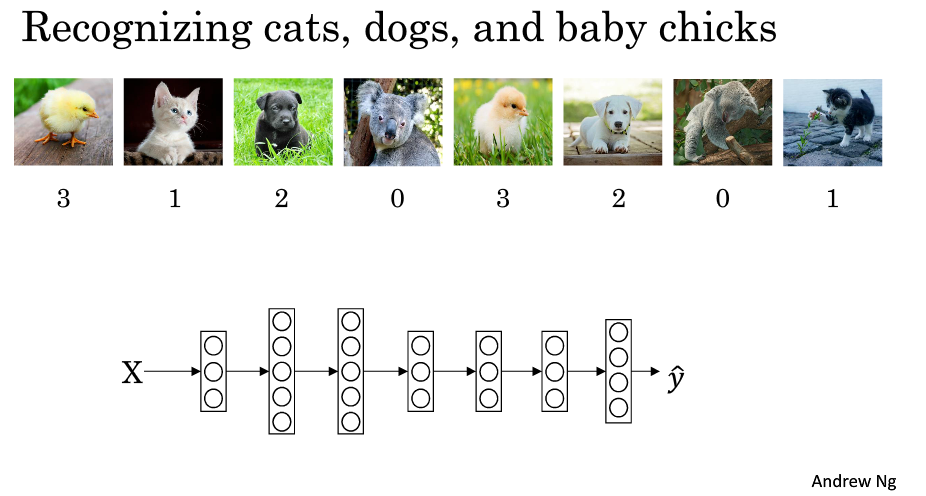
选择L层，假设我们有mini-batch , 以及它们相对应的Y值, 在用训练L层时，会得到一个，用第二个最小批来训练这一层时，得到，根据后面每一个mini-batch，对应得到等。当我们在试图计算当前温度的指数加权平均数的时候，我们是如何用指数加权平均数来计算，就用同样的方法来对和进行估算。因此在用不同的mini-batch训练神经网络的同时，能够得到你所查看的每一层的和的平均数的实时数值。最后在测试时，对应计算的这个等式只需要用Z值以及计算出来的和的指数加权平均来计算，，再用公式使用到之前训练神经网络时学到的和参数值来计算一个测试实例的。所以重点就是在训练时我们是用一整个mini-batch， 但是在测试时，我们可能会需要处理单个测试实例，那么处理的方式就是通过训练集来估算和，我们有很多方式来做估算。理论上我们可以用我们最后的网络运行整个训练集，但是实际上人们通常会实现某种指数加权平均粗略的估算。如果你在用一个深度学习框架，通常会有一些默认的方式来估算和，这些方式一般效果也会比较好，但是实际上任何合理的方式来估算隐藏神经元的均值和方差，应该在测试时都是效果可以的。应用批标准化我相信你们可以训练更深的神经网络，并且让你们的学习算法运行的更快。

3.8 Softmax回归

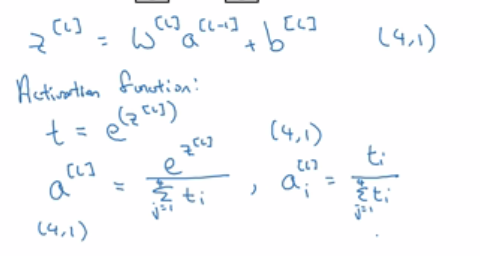
已经讨论过的分类示例使用了，二分类方法,也就是当你只有两类标签的情况，如果我们有多种可能的分类目标呢? 有一种更普遍的逻辑回归的方法叫做softmax回归，这种方法能够让你试图预测一个多种分类中的类别时做出预测,而不是识别两类中的类别。

下图中的例子就是一个需要识别4种分类，猫叫做类别1,狗就是类别2,小鸡就是类别3，其他则为第4类，大写的C表示你将要预测的总的类别数，这里C=4，那么当你有4个类别时 类别序号就是，从0到，换句话说,就是0,1,2,3。这种情况下,我们就需要构建一个新的神经网络 其中输出层是4个，或者说有C个输出单元。输出层L的单元数, 想得到的是输出层的单元分别表示了每个类别的概率，因此输出层从上到下的4个单元分别表示了当输入为x的情况下，输出是其他类别的概率、输出概率为猫的概率、输出x是狗的概率以及输出x是小鸡的概率。这四个输出的概率加和应该为1，所以输出层的总和等于1。





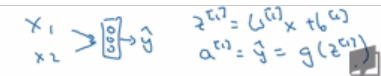
标准化的做法就是使你的网络使用softmax层,以及生成这些输出的层，在网络的最后一层，需要像往常那样计算每层的线性部分，到达最后一层时，首先计算得到，接下来就要用到softmax激活函数，这个激活函数对于Softmax层有些不同，计算一个临时的值t，适用于的每个元素，是个（4,1）的向量，所以也是一个（4,1）的向量，对所有元素求幂，那么继续计算就需要将t向量归一化，也就是得到概率值。,也就是t向量除以里面四个数的和，分别求得四个数占的比重，得到的即为是这个对应类别的概率。总的来说，Softmax几号函数就是通过向量计算出总和为1的概率。从到的计算过程，即计算幂得到临时变量，再做归一化，这个过程总结为一个softmax激活函数。



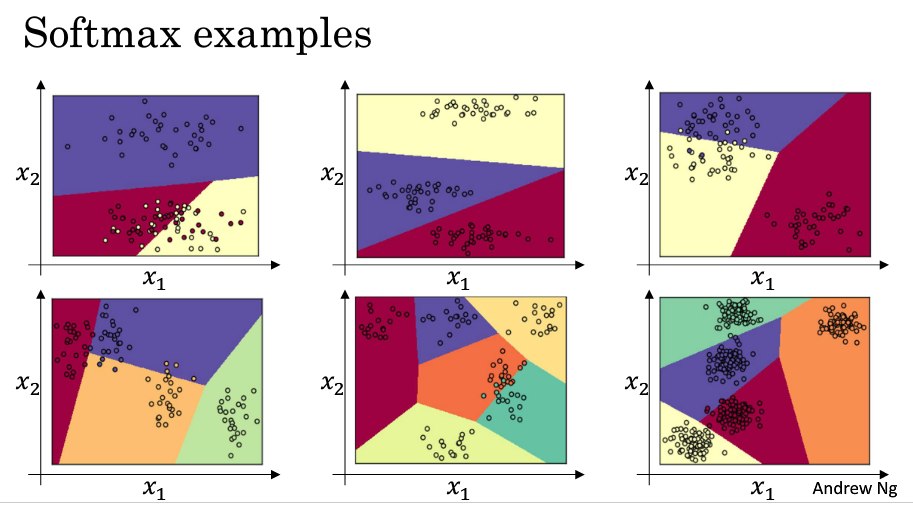
|  |
| --- |
|  |

假设从到，是通过激活函数，那么这个激活函数不同之处在于这个函数g需要输入一个4\*1的向量，也会输出一个4\*1的向量，以前我们的激活通常是接收单行输入，比如 sigmoid函数和ReLU函数就是接收一个实数输入，然后输出一个实数的输出。softmax函数的不同之处就是，由于它需要把输出归一化，以及输入输出都是向量。

因此softmax分类器还能代表什么呢?下面展示一个例子，其中输入为x1，x2，然后直接送入softmax，中间没有隐藏层，我们可以知道这是做的线性运算。Softmax层做的就是计算以及，或者说就是在上使用softmax激活函数，所以这个没有隐藏层的神经网络应该可以让你有一个关于softmax函数意义的直观感觉。



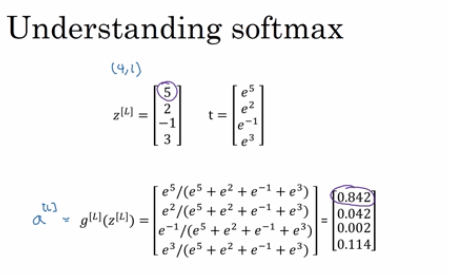
下图中是关于输入为x1和x2，并且直接输入softmax层，输出类别分别C=3，C=4，C=5，C=6的几个实例，这种形式的决策边界请注意这是一种线性决策边界，但是可以把数据区别为我们想要的那么多类C类，我们所做的就是，选择图中展示的训练数据，使用数据的C种标签训练这个softmax分类器。图中的颜色显示了分类器输出的阈值，输入的颜色是基于三种输出中概率最高的那种，这就是一种泛化的逻辑回归，它使用一类似种线性的决策边界，具体体现在每两类之间的决策边界是线性的。这些图展示了softmax分类器在没有隐藏层的时候可以做什么，当然有的网络会更深有更多的隐藏层，以及更多的隐藏单元，然后你就可以发现更复杂的非线性决策平面来区分。



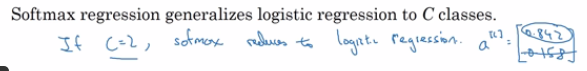
这节讲解了softmax激活函数的工作原理，下一节我们来看一下如何训练一个带有softmax层的神经网络。

3.9 训练一个softmax分类器

加深对softmax分类的理解，并学习如何训练采用了softmax层的模型。上一节中我们举过一个例子如下图，注意到中最大的元素是5，而中UI大概率值饿依然是第一个元素。



Softmax这个名称的来源于所谓的hardmax相对，hardmax将矢量映射到这个矢量， hardmax函数遍历中的元素，将中最大的元素对应的位置置1， 其余位置置0。与之相对的是softmax中Z到这些概率值的映射要平和些。另一个需要提示一下的就是softmax激活函数将logistic激活函数从2分类推广到C分类，也就是说 如果C=2 那么C=2的softmax，实际上将简化成logistic回归。在这个视频里给出证明 但是大致的证明思路是，当C=2时 如果你应用softmax，那么输出层将输出2个数，也许是0.842和0.158，这2个数的和总是1，因为这2个数的和必须为1， 他们实际上是冗余的，只需要计算其中一个就可以了。你计算这个数的方法将会推导成，logistic回归的计算方法 即最后只需要计算一个输出。softmax回归是logistic回归从2分类到多分类的推广。

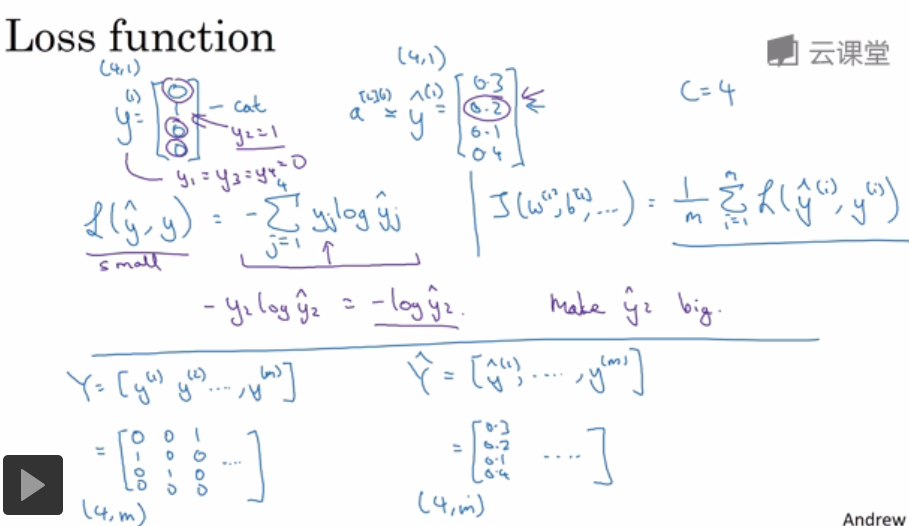


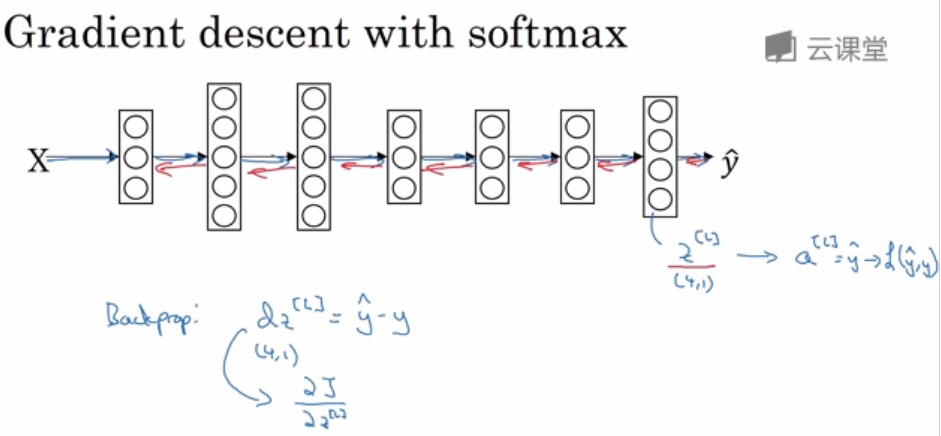
看看实际上训练一个包含softmax输出层的神经网络要怎么做，首先，应该先定义用于训练神经网络的损失函数，举个例子，关于上一节的识别分类，这个例子中训练集的目标输出(target output) 即标签真实值是，说明输入的图片是个猫图，属于第1类，假设你的神经网络的输出是，这个预测结果是个概率矢量，其元素和为1。对于y与可以看书，这个神经网络对这个样例的预测并不好，因为它实际上是猫但是在猫这一分类只得到了20%概率，所以它在这个样例上表现并不好。那计算机上是如何判断损失的，损失函数如何定义？softmax分类典型的损失计算：

表示意思是y向量与向量对应元素先取对数再乘积加和，最后取相反数，对这个损失函数的理解：要使模型分类效果好，即损失函数值应该小，那么对于后面求和部分应该更大，根据0乘任何数得0，则上例中最后的结果为，要小，那么就应该最大，与最初的我们想要预测出结果为第1类，使其概率值最大符合。如果你的学习算法试图将损失变小，因为你会用梯度下降法来减少训练集的损失，那么唯一使损失变小的方法是使对应类别的概率预测值尽可能的大。更一般地 损失函数的功能是查看训练集的真实分类值，并令其对应的概率值尽可能的大。如果你熟悉最大似然估计统计学习方法，这实际上也是某种形式的最大似然估计。这只是单个训练样例的损失，整个训练集的损失要如何处理呢？基于某个训练集的损失函数包括权重(weights) 偏差(bias)等等，

把训练算法对所有样本的预测都加起来，定义为整个训练集的损失的和，即所有训练样例的预测值与真实值所反映的损失。你要做的是使用梯度下降法使损失(cost)最小化，最后说一个实现细节，因为C=4，所以都是一个(4,1)的向量，并且对于整个训练集，矩阵形式都为m个样本的横向堆叠，所以最后Y以及 都是一个(4,m)的矩阵。最后看看梯度下降法在包含softmax输出层的神经网络上的实现，在得到最后一层的和后，获得了输出并且计算损失，那么反向传播也就是梯度下降法如何实现？实际上关键步骤或者说关键方程式的初始化则为：

其中的变量都是(C,m)的维度，这是的计算。方式，是损失函数对的偏导，从计算开始你的反向传播过程。



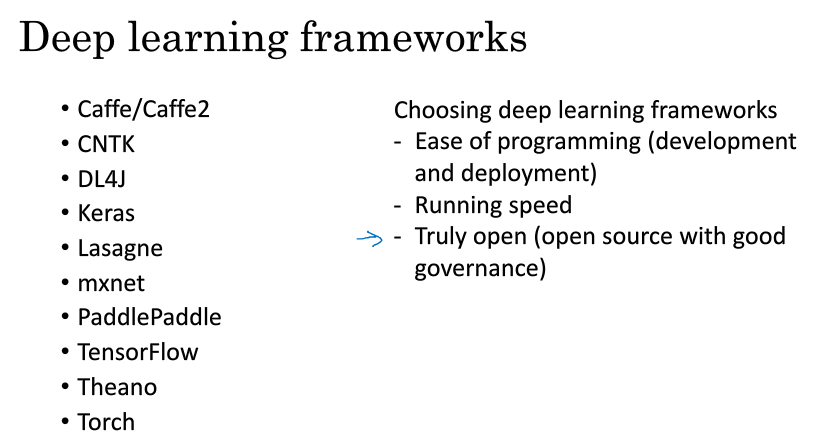


如果你使用一个深度学习的编程框架 而这些基本框架，通常都只需要你专注于正确地实现正向传播，只要你指定它为一个基本框架(primary framwork)<br />并制定前向传播过程，该框架将自动为你实现反向传播过程。以上就是softmax分类，你可以用它实现的不仅仅是二分类，还包括C分类。

3.10 深度学习框架

通过使用Pyhton和NumPy差不多能够从头开始完成深度学习的算法，并且大概理解这些深度学习的算法到底在做什么，但你会发现当你要实现复杂的模型时例如卷积神经网络(CNN)或循环神经网络(RNN)，或你着手的复杂模型时，你会发现越来越难操作，对于大多数人不会再自己完成所有细节编程，而是借助深度学习框架，帮助实现这些模型。例如，矩阵的乘法可能自己知道也会变成，但是真正在应用时，直接借助数值线性代数库可以更高效地帮你完成计算。深度学习已经发展到一个成熟的阶段，现在使用深度学习框架 可以使一些任务变得更实际更高效。

在此列出一些领先的框架，每个框架都有自己忠诚的用户和开发人员社区，这里的每一个框架，在一些应用中都是可靠的选择。如何选择深度学习框架的标准，其中一个重要条件是编程的简便性，便于编程，这主要体现在两个方面，开发神经网络对它进行迭代改善，以及在生产环境中进行实战布署。第二个重要标准是运行速度，特别是在大数据集上进行训练时，有些框架比起其他框架能够让你运行和训练神经网络时更为高效。还有一个标准是这个框架是否真正的开源，作为一个真正开放的框架，不仅需要开放源码，还需要良好的管理，不幸的是在软件产业中，曾有一些企业虽然将软件开源但却独占着领导权。并且还要注意是否会保持开源，即使现在它是开源的，在以后可能会因各种原因所关闭。但至少在短期内取决于你对程序语言的偏好，你是喜欢Python Java C++或其他，还取决于你在构建的是什么应用，是计算机视觉，或进行自然语言处理，或者在线广告或其他，其中许多框架都会是不错的选择。



3.11 TensorFlow

关于TensorFlow是python的一个库，需要进行安装，并且有个问题，python2不支持TensorFlow，所以之前用的Anaconda2是python2的环境，所以在Anaconda官网上<https://repo.continuum.io/archive/>下载Anaconda3-4.1.0-Windows-x86\_64.exe，进行安装，并且安装在Anaconda2/envs下，新建py3文件夹作为安装路径，安装完成后，cmd环境中启动py3，使用命令activate py3(这里的py3即刚才的文件名)，启动好python3环境后，安装TensorFlow库，使用命令pip install tendorflow。然后就可以使用啦。遇到问题，直接pip安装的TensorFlow版本太新，import时候报错，那就pip uninstall卸载重新安装旧的版本。另外，上面说的pip install tendorflow是在控制台里直接网上下载安装，如果你已经有TensorFlow的包，那么就cmd进入这个包在的目录，输入命令pip install tensorflow-1.0.1-cp35-cp35m-win\_amd64.whl。问题就解决啦。

目前有很多非常好的深度学习编程框架其中一个就是TensorFlow，这一节将展示TensorFlow的基础结构。先举个例子来说明，使用TensorFlow的动机，假设有个代价函数J 我们希望将其最小化:

这就是代价函数，其实就是，应该在W=5时取得最小值，假设我们并不知道这个点，而且你只有这个代价函数表达式看看如何使用TensorFlow来最小化这个表达式，因为有一个非常类似的程序可以用来训练神经网络。而根据你的神经网络的所有参数，你的神经网络可能会有非常复杂的代价函数J(w,b)，同样也可以使用TensorFlow，自动地寻找w和b的值来最小化代价函数的值。代码实现以及理解都写在jupyter notebook里面了。