第二篇第一周

1、训练神经网络的时候，我们需要做很多抉择：比如：神经网络分多少层，每层含有多少个隐藏层单元，学习率是多少，各层采用哪些激活函数，以及一些超参数等等。实际上应用型机器学习是一个高度迭代的过程，也就是不断地去尝试找到更好的方案。创建高质量的训练集，测试集和验证集，有助于提高循环效率，机器学习常用的划分数据集的方法是6,2,2，也就是训练集 60%，验证集20%，测试集20%，如果没有验证集的话就是7,3开，但是大数据时代通常不这么做了，通常验证集合测试集都会占比很少。

例子：加入图像识别猫咪，可能存在这样子一个问题，就是猫咪的像素可能存在着很大的差异，那么有这样子的一个建议，就是尽量使得验证集和测试集处于同样的分布，因为你要用验证集评价不同的模型，尽可能的优化，如果验证集和测试集来自相同的分布，假如你不需要无偏估计，就可以不用测试集，因为验证集就包含了测试的效果，

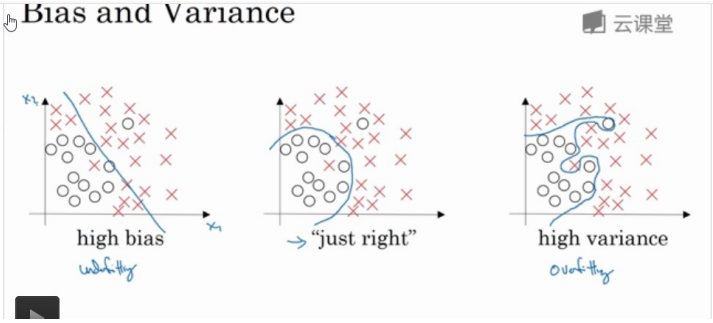
2、偏差和方差：

探究偏差和方差的权衡问题

第一种情况是高偏差的情况，也就是欠拟合。

第二种情况是分类器方差比较高，属于过度拟合。

第三种情况是复杂程度适中，数据拟合程度适度的分类器。也就是适度拟合。

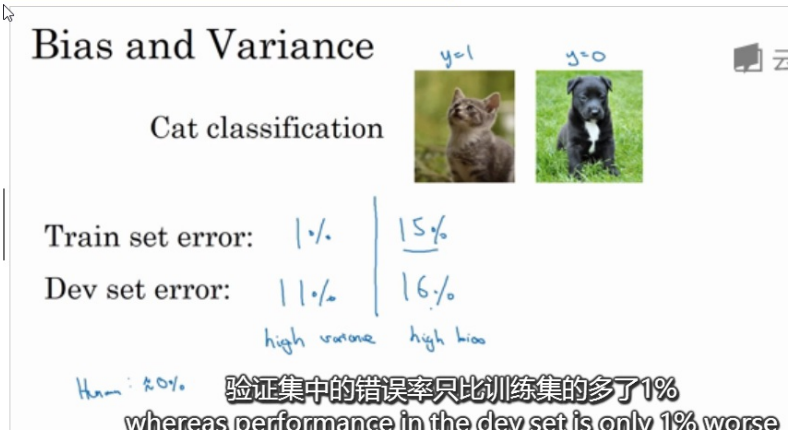


研究方差和偏差的指标：

理解偏差和方差的两个关键数据是：训练集误差和验证集误差

例子：如果训练误差是1%，测试误差是11%，由此可见，训练集设置相当好，而验证集测试相对差，我们可能过度拟合了训练集，从某种意义上来说，验证集并没有起到充分利用交叉验证集的作用，我们称之为高方差，通过检测训练集误差和测试集方差，我们就可以诊断算法是否具有高方差，

如果算法在训练集上并没有得到和好的训练，也就是训练数据的拟合程度不高，也就是欠拟合，可以说这种算法偏差比较高，如下图所示15%跟16%的那组数据，偏差较高，不能很好的拟合数据。



还有一种情况就是训练数据15%但是验证数据上的表现更惨，30%，这种情况就是偏差高，方差也很高。

还有一种情况就是偏差和方差都很低，比如0.5%和1%,说说明效果很好。

最优误差也叫做贝叶斯误差，也就是接近0%。

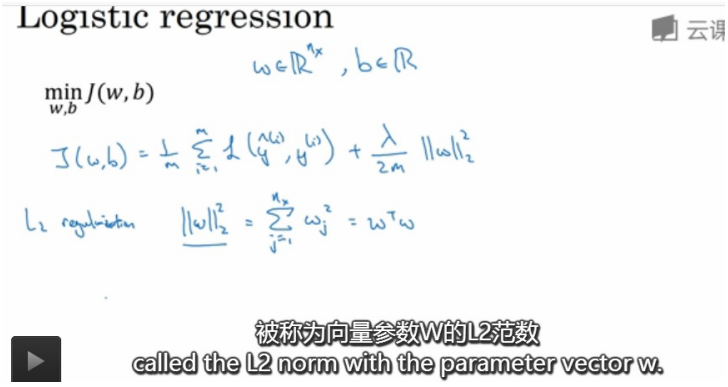
3、训练神经网络的基本方法： 1、首先，看看偏差是不是很高，如果很高的话就考虑换个模型，比如更多隐藏层或者隐藏单元的神经网络，或者花更多的时间或者选择更加优秀的算法，

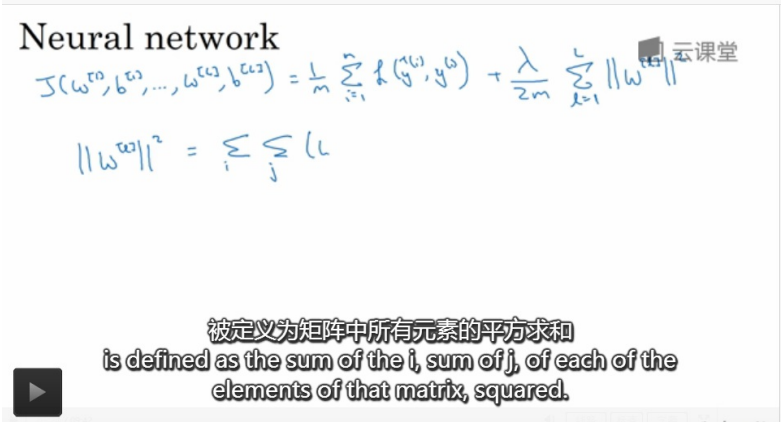
偏差合理之后，就可以看方差，如果方差过高，可以采集更多的数据，如果不能够采集更多的数据，我们也可以通过正

则化来减少过拟合，还可以反复尝试，找到更加合理的模型，这样就可能一箭双雕，偏差和方差都得到解决。

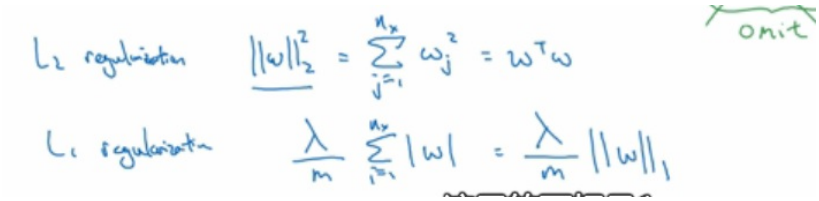
4、神经网络过度拟合了数据，也就是高方差，最先想到的方法就是正则化，另外一个解决高方差的问题就是准备更多的数据，如果无法有效的获得更多的数据，那么正则化则可以有助于避免过度拟合，有效的减少方差，同时也不会使得偏差增大很多。

正则化原理：





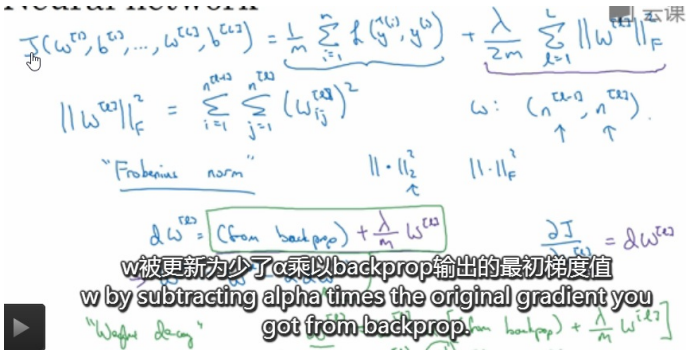
为什么只正则化参数w，不加上b，通常省略b，因为通常w是一个高纬参数矢量，已经可以表达高偏差问题，w可能包含很多参数，我们不可能拟合所有参数，而b只是一个数字，所以w几乎涵盖了所有参数，而不是b，如果加了参数b，也没有太大影响，因为b只是众多参数中的一个，所以通常省略不计，加上也没关系，L2是常见的正则化类型（参数的平方和也），还有一个就是L1正则化（参数的和）。



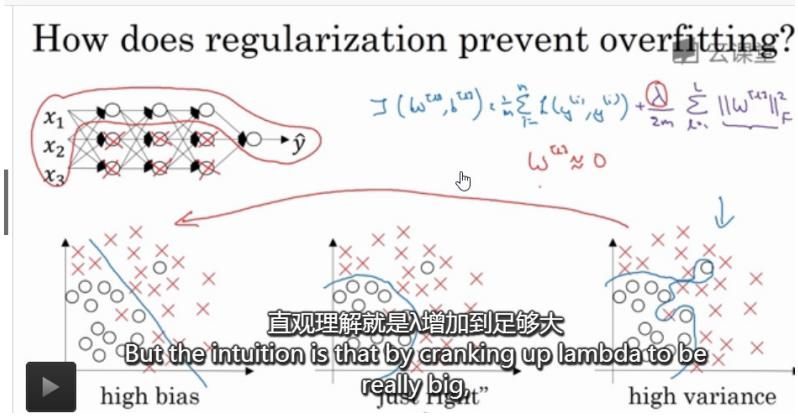
如果用L1正则化，W最终会很稀疏，也就是说w向量中将会有很多0，这样有利于压缩模型，因为集合中的参数均为0，存储该模型所占用的内存更少，但是却没有降低太多内存的，所以吴恩达不认为压缩了模型，人们在训练神经网络时，往往倾向于使用L2正则化。

λ是正则化的参数，是另外一个需要被调整的超参数：通常用验证集或者交叉验证集来配置这个参数，尝试各种数据，寻找最佳参数，我们要考虑训练集之间的平衡，把参数正常值设置为较小值，这样可以避免过拟合。

L2也被称为权重衰减法，因为它很像梯度下降：

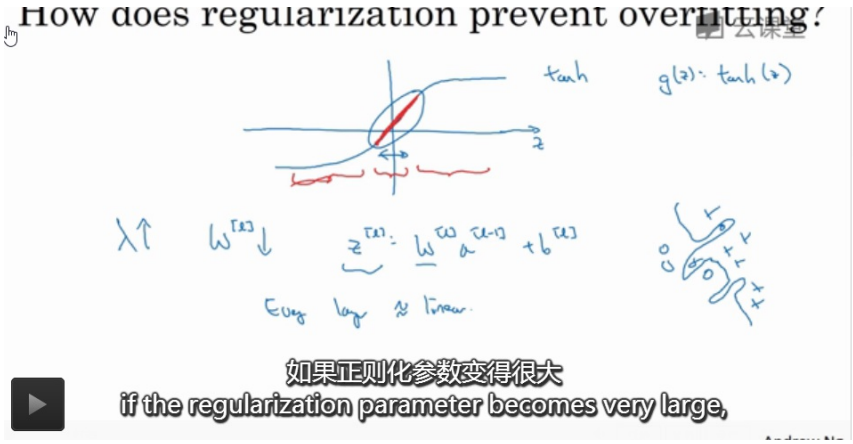


5、为什么正则化可以减少过拟合，为什么他可以减少方差问题，



直观理解就是当λ增加到足够大时，W会接近于0，实际上不会发生这种情况，我们尝试或者减小许多隐藏单元的影响，最终这个网络变得很简单，这个神经网络越来越接近逻辑回归，我们直观上认为大量隐藏层单元被消灭了，其实不是，实际上依然存在，只是他们的影响变得更小了，神经网络变得更加简单，上面只是直观上来说的。

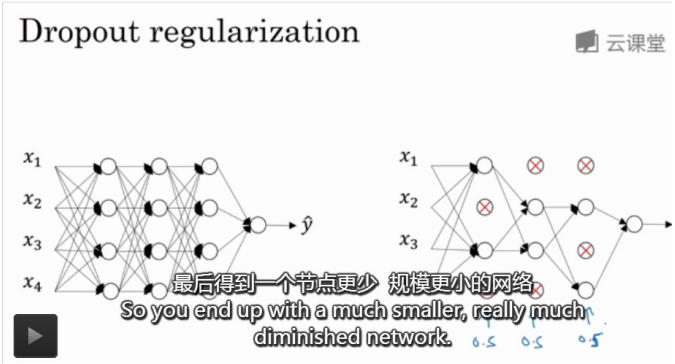
现在我们摒弃这个直觉：如果正则参数λ很大，激活函数的参数w会相对较小，因为代价函数的参数变大了，如果w很小，由于z=wA+b，w很小，z就会很小，最终z的值就会在很小的范围，g(z)大致会落在如下红色的大致呈现线性，每层几乎都是线性的，和线性回归函数一样，我们知道，如果每层都是线性的，那么整个网络就是一个线性网络，即使是一个非常深的深层，因为具有线性激活函数的特征，最终我们只能计算线性函数，因此他不适合非常复杂的决策，以及过度拟合数据集的非线性决策边界。



总结一下，如果正则化参数变得很大，参数w会很小，z也会变得相对较小，此时忽略b的影响，z会相对变小，实际上，z的取值范围很小，这个激活函数，也就是曲线函数，会相对呈现线性，整个神经网络会计算离线性函数近的值，这个线性函数非常简单，并不是一个极复杂高度非线性函数，不会发生过拟合。

6、dropout（随机失活、随机丢弃）的工作原理：

假如你在训练左图这样的神经网络，它存在过拟合，这就是要dropout要处理的，dropout会遍历神经网络的每一层，并设置消除神经网络中节点的概率，假设，神经网络的每一层，每个节点都以抛硬币的方式设置概率，每个节点得以保留和消除的概率都是0.5，设置完节点的概率。我们会消除一些节点，然后删除从该节点进出的连线，如下图：

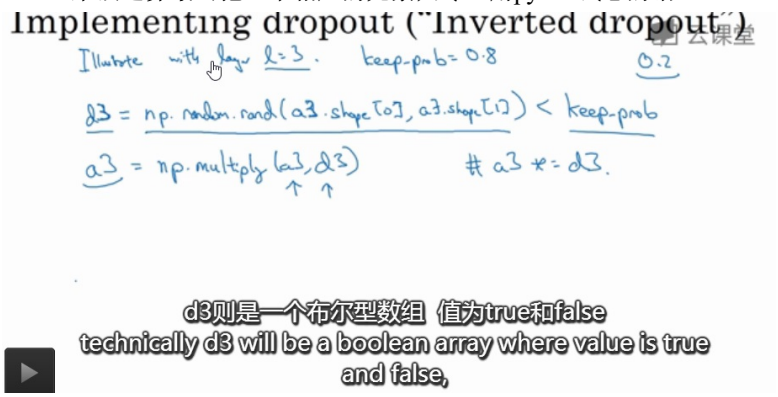


得到一个节点更少，规模更小的网络，然后用backprop方法进行训练，右边的神经网络时精简之后的一个样本。对应其他样本，我们照样子以抛硬币的方式设置概率，保留一类节点的集合，删除其他类型节点的集合，对于每个训练样本，我们都会采用精简之后的神经网络训练它，这种方式纯粹是遍历节点，编码也是随机的，可他真的有效，不过可想而知，我们针对每个样本训练极小规模的网络时，最后你可能会认识到为什么要使用正则化网络，因为我们在训练极小的网络。

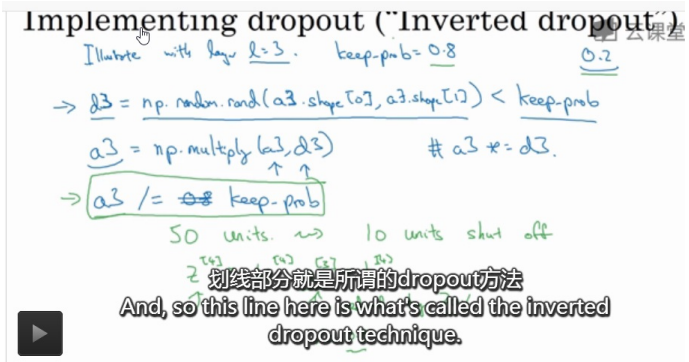
如何实施dropout：

1、inverted dropout(反向随机失活):

首先定义向量d，d3表示三层的的dropout向量，d3=np.random.rand，参数都是a3.shape,看它是否小于某个数，我们称之为keep-prob,keep-prob是一个具体的数字，上个示例它是0.5，而本例中它是0.8，他表示保留某个隐藏层的概率，它意味着消除任意一个隐藏层单元的概率是0.2，它的作用就是生成随机矩阵，如果对a3进行因子分解，效果也是一样的，d3是一个矩阵，每个样本和隐藏层单元，其在d3中对应值为1的概率是0.8，对应值为0 的概率是0.2，然后从第三层中获取激活函数，我们叫它a3,a3含有要计算的激活函数，a3=a3*d3,它的作用是过来d3中所有等于0的元素，而各个元素等于0的概率是20%，乘法运算最终把d3中相应的元素归零，用python试想的话：



接着向外扩展a3,a3=a3/keep-prob，我们假设第三隐藏层上有50个节点，在这一维上a3是50，我们通过因子分解将它拆分成50*m维的，保留或者删除的概率分别是80%和20%，所以最后被删除或者归零的单元平均有10个，z(4)=w(4)a(3)+b(4),我们的预期是a3减少20%，为了不影响在z(4)的期望值，我们需要将w(4)a(3)除以0.8，他会修正或者弥补我们所需的那20%，a(3)的期望不会变。绿色圈子内的就是所谓的dropout方法，



他的功能是，无论keep-prob的值是多少，反向随机失活方法通过消除keep-prob,确保a3期望值不变，事实证明，在测试阶段，在评估一个神经网络时，方向随机失活方法，使得测试阶段变得更加容易，因为他的数据扩展问题变少。

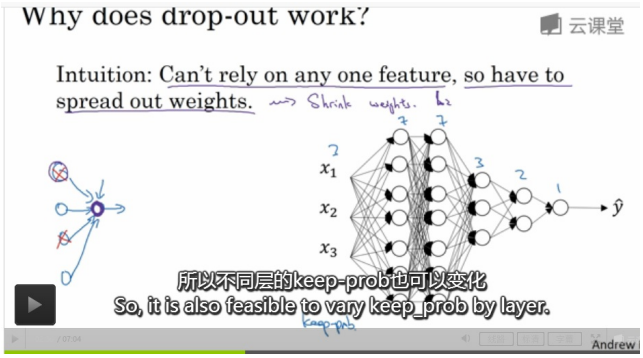
对应d3，你会发现对于不同的训练数据，消除的隐藏单元也是不同的。

7、理解dropout:

上节课的直观认识：每次迭代之后，神经网络都会变得比以前更小，因此采用较小的神经网络好像和使用正则化的效果

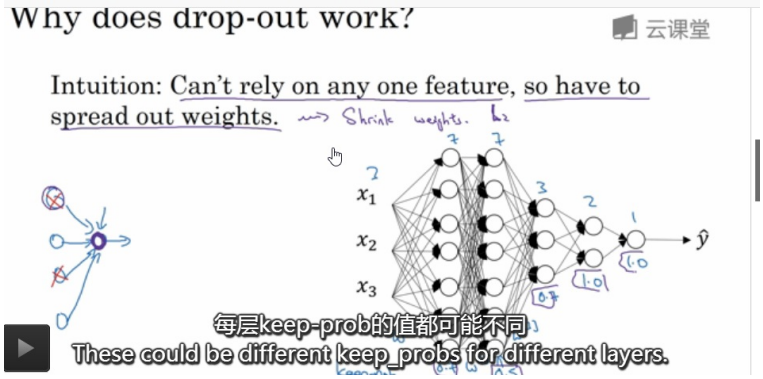
是一样的，

对于神经元来说，就是从输入得到输出3，使用dropout，使得输入特征有可能被随机删除，我们不愿意吧所有赌注放在一个节点上，不愿意给任何输出加上太多的权重，因为它可能随时被删除，实施dropout的结果就是它会压缩权重（这跟L2是类似的），并完成一下预防过拟合的外层正则化，事实证明，dropout被正式作为一种正则化的代替形式，L2对不同权重的衰减是不同的，它取决于倍增的激活函数的大小，总结一下，dropout的功能类似于L2正则化，与L2不同的是，被应用的方式不同，dropout也会有所不同，甚至更适用于不同的输入范围。



第一层W[1]权重矩阵是3*7，第二层W[2]是7*7，第三层是7*3，第四层是3*2，第五层是2*1，W[2]是最大的权重矩阵，为了预防矩阵过拟合，对于第二层，它的keep-prob（每一层上保留节点的概率）的值应该相对较低，假设为0.5，对于其他层，过拟合的问题就可能没那么严重，它的keep-prob的值可以相对较高，比如为0.7，如果在某一层，我们不必担心其过拟合问题，那么keep-prob的值可以为1,。如下图紫线圈出来的：

注意：keep-prob的值为1，意味着保留所有单元，并且不再这一层使用dropout（随机丢弃）。



对于可能出现过拟合，而且含有诸多参数的层，我们可以把keep-prob设置成更小的值，以便应用更强大的dropout，有点像处理L2正则化的λ，我们尝试对某些层实施更多的正则化。

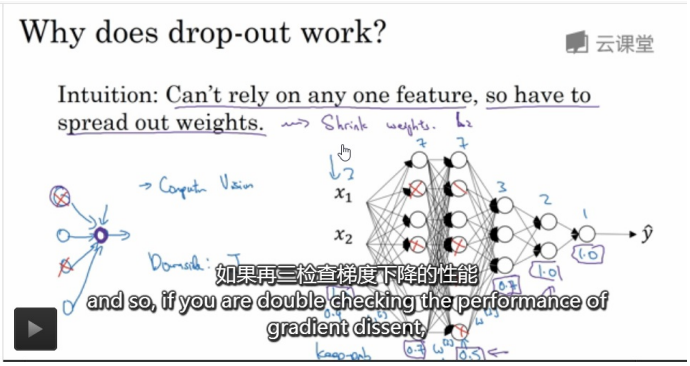
从技术上将，我们可以对输入层使用dropout，我们可以删除一个或者多个特征，但是我们通常不那么做，keep-prob为1是非常常用的输入值，也可以设置为更大的值1.9，但是消除一半的输入特征是不太可能的，所以如果你对输入层使用dropout，keep-prob的值要接近1,。

总结一下，如果你担心某些层比其它层更容易发生过拟合，就把某些层的keep-prob设置得比其它层更低，缺点是如果你使用交叉验证，你要搜索更多的超参数，另外一种方式是在一些层上应用dropout，而有些层不用，应用dropout的层只含有一个超级参数，就是keep-prob。

成功经验：实施dropout，在计算机视觉领域有很多成功的第一次，计算机视觉中的输入非常大，输入了太多像素，以至于没有足够的数据，所以它在计算机视觉中使用的非常频繁，在计算机视觉领域，成为默认的选择。

记住一点，dropout是一种正则化方法，它有助于防止过拟合，因此，除非算法过拟合，不然我是不会使用dropout的，所以它在其他领域应用的比较少，主要存在于计算机视觉，因为我们通常没有足够的数据，所以一直存在着过拟合，这就是计算机视觉研究员为何如此钟情于dropout的原因。

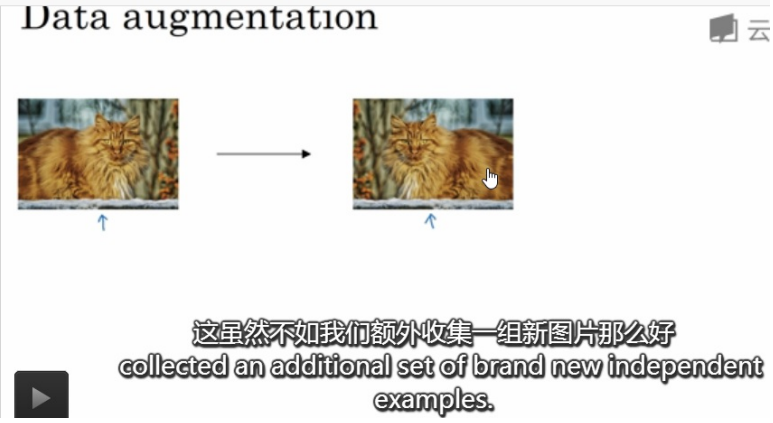
dropout的最大缺点就是代价函数J不再被明确定义，每次迭代，都会移除一些节点，如果再三检查梯度下降性能，实际上很难进行复查，定义明确的代价函数J，每次迭代都会下降，因为没有定义，所以很难计算，所以我们失去了调试工具，所以通常关闭dropout，将keep_prob的值设置为1，运行代码，确保J函数单调递减，然后再打开dropout，在dropout过程中，代码并未引入bug，：



8、防止过拟合的其他方法

一：扩充数据集

1、如果无法额外添加一些训练数据，通过水平翻转图片，可以使得训练集增大一倍。



2、还可以随意剪裁图片：

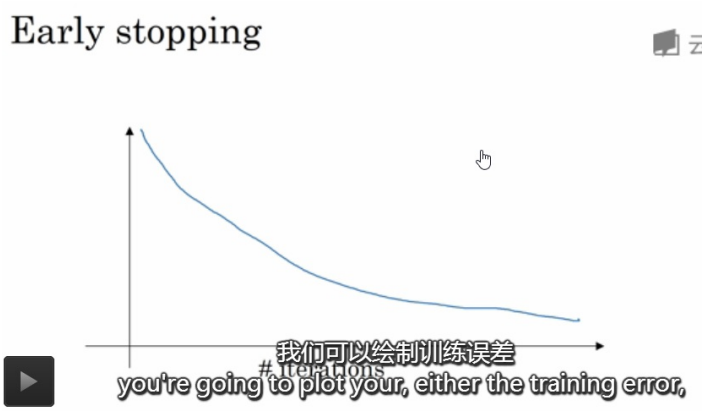
上面那样做存在对抗代价，是假的训练数据集。

3、对于字符，我们可以进行变形，如下图

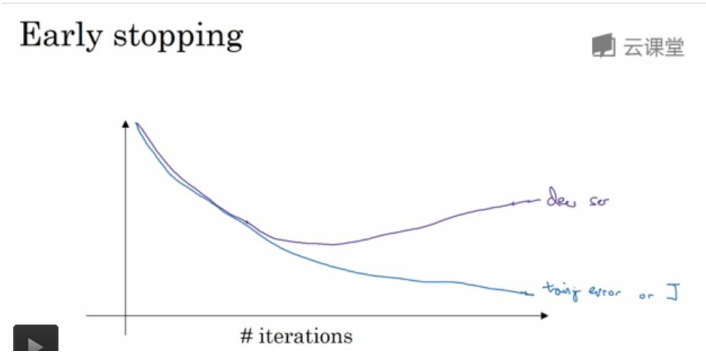
二：early stopping（提早停止训练神经网络）

运行梯度下降时，我们可以绘制训练误差

或者只绘制代价函数J的优化过程，



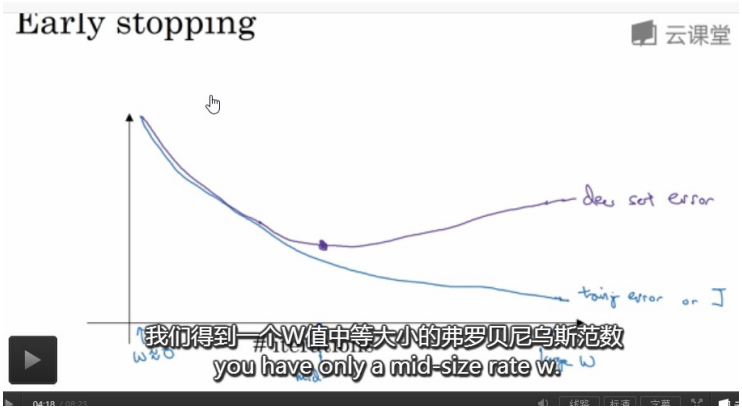
然后绘制验证集误差：



验证集误差通常是先下降后到达某个点之后，上升。

Early stopping的意思是，神经网络已经在这个迭代过程中表现的很好了，我们就提前停止训练吧，得到验证误差最小的点，刚开始迭代的时候，参数W接近0，因为随机初始化w时，它的值可能都是比较小的随机值，所以在你长期训练神经网络之前，W依然很小，在迭代和训练过程中，W的值会变得越来越大。

疑问：W应该是指参数的个数吧？

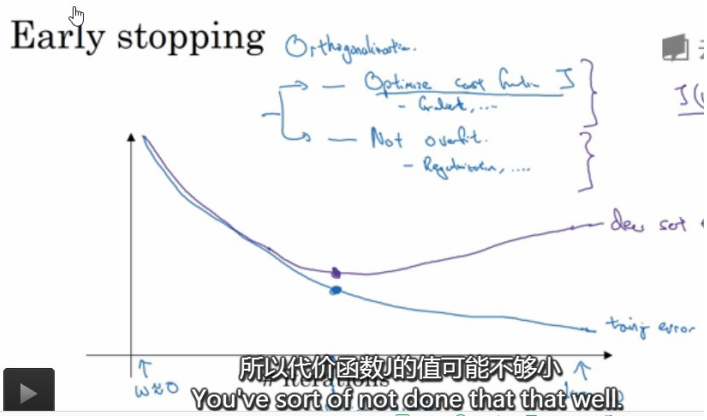


与L2正则化相似，选择参数W范数较小的神经网络。

Early stopping意思是提前停止训练神经网络。

机器学习的第一步是：选择一个算法来优化代价函数，如梯度下降，Momentum，RMSprop，Adam等等。但是优化代价函数之后，我不想发生过拟合。解决过拟合问题可以采用正则化，扩展数据等等，在机器学习中，超级参数激增，选出可行的算法变得越来越复杂，优化代价函数，机器学习就会变得更加简单，在重点优化代价函数J时，你只需要注意W和b，但是预防过拟合，还有其他任务，就是减小方差，用正交化来减小方差，思路就是在一个时间做一个任务，

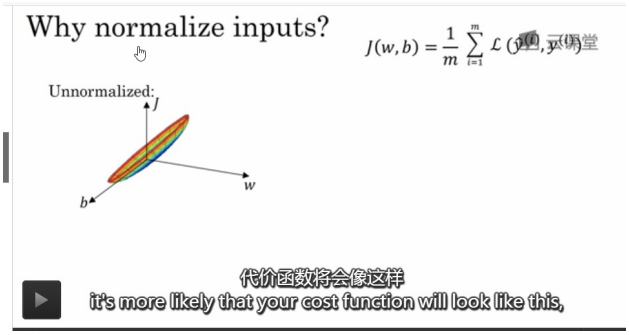
Early stopping的缺点是你不能独立的处理上面的优化代价函数和解决过拟合这两个问题，因为提早停止梯度下降，也就是停止了优化代价函数J，所以：



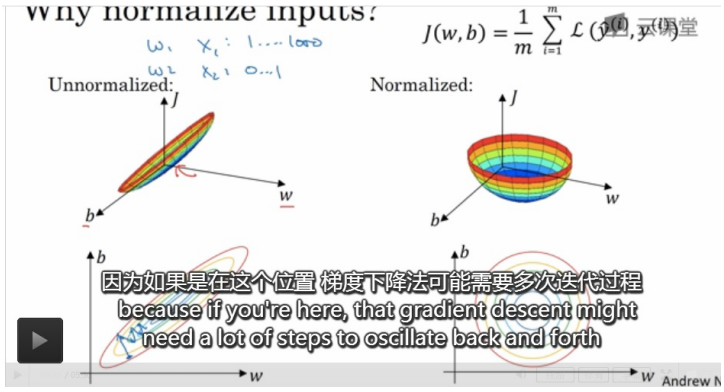
9、正则化输入

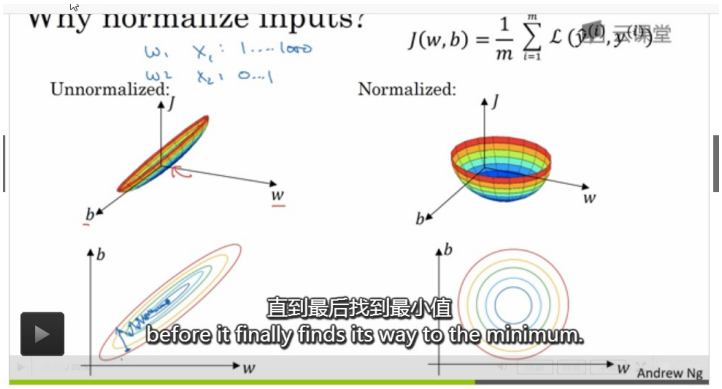
训练神经网络时，加速训练的方法是归一化输入。

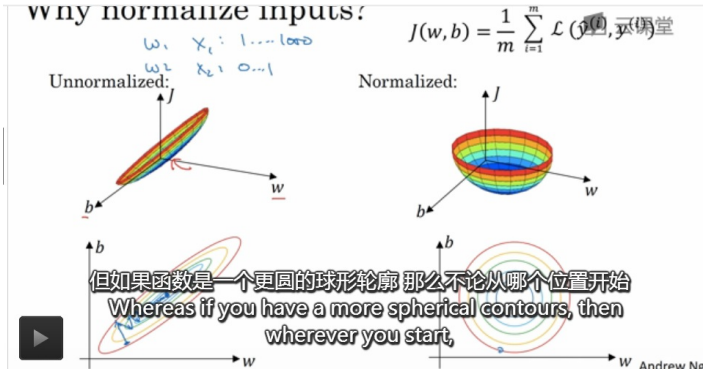
归一化输入需要两个步骤，第一步就是零均值化，第二步就是归一化方差，如果你是用非归一化的输入特征：

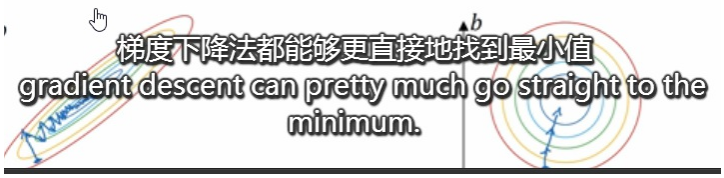


这是一个非常细长狭小的代价函数，假设x1是0到1000，x2是0到1，结果是参数w1和参数w2的比率，将会非常不同：



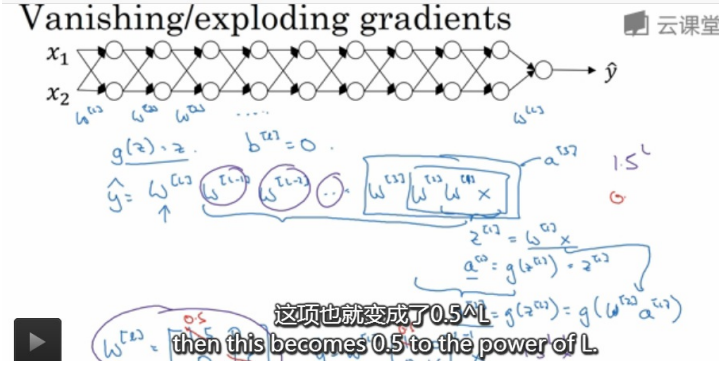


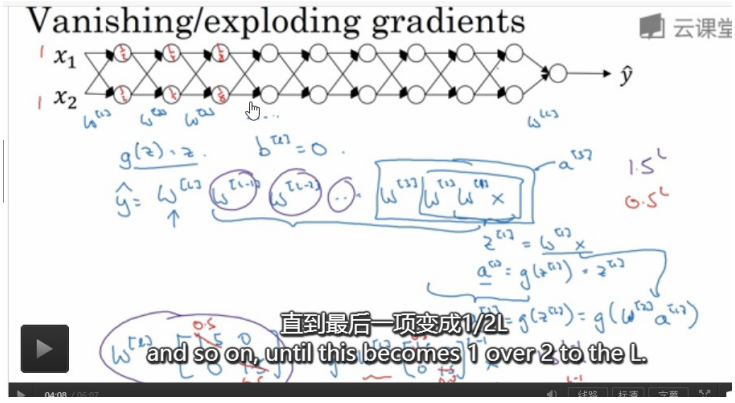


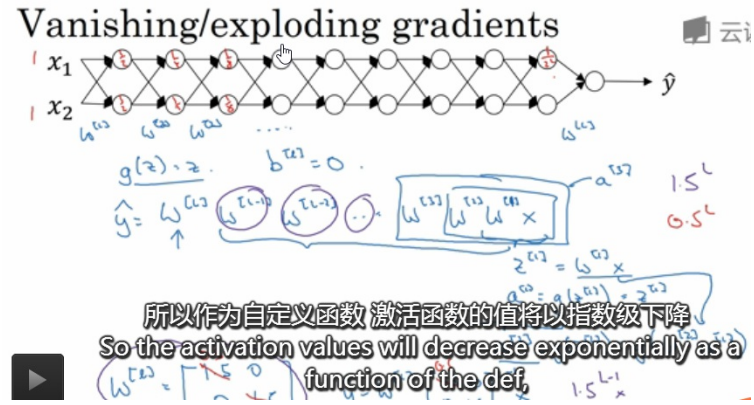


10、梯度消失或者梯度爆炸

训练神经网络的时候，导数或者坡度会变得非常大或者非常小，以指数方式变小。

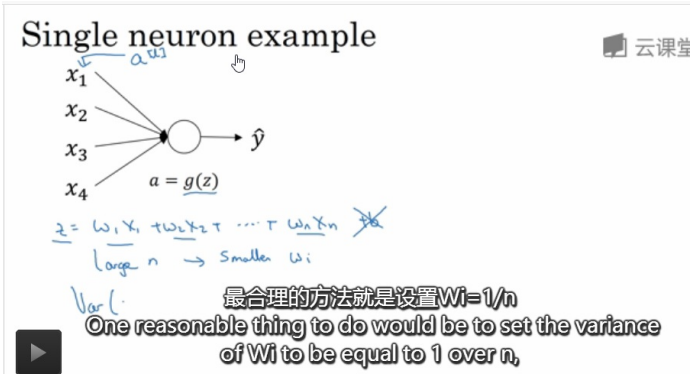




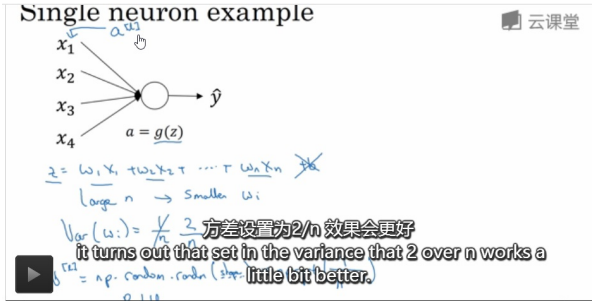


所以，权重W[1]比1略大或者说只比单位矩阵大一点，深度神经网络的集激活函数将以爆炸式的速度增长，如果比1略小，深度神经网络将以指数式的速度递减，

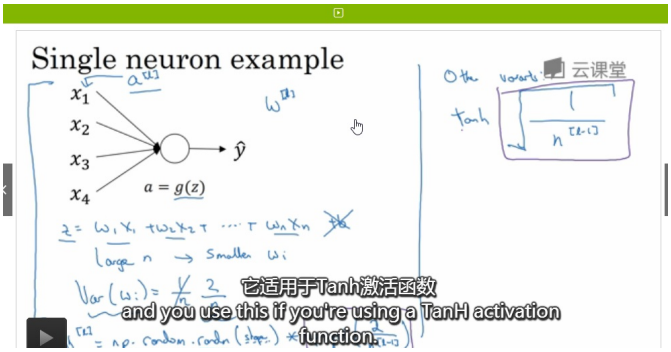
11、神经网络的权重初始化。



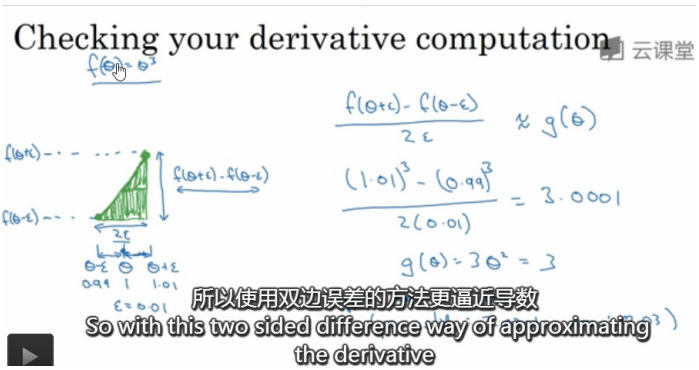
n为神经元输入特征数量，实际上，你要做的就是设置某层权重矩阵W，W=np.random.randn(shape)*square(该层每个神经元的特征数量分之一)，假设激活函数是relu是，方差设置为2/n效果更佳

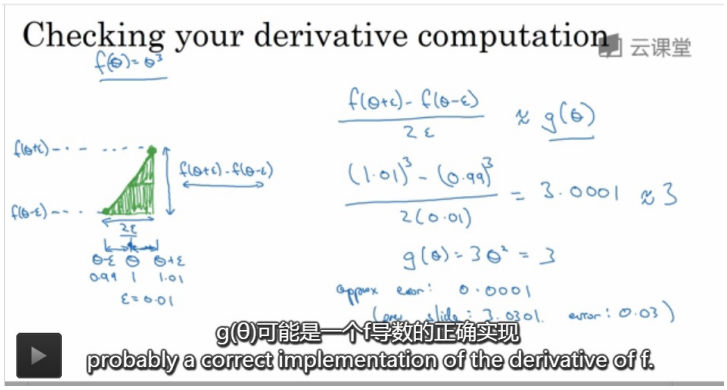


所以，如激活函数的输入特征被零均值，标准化方差，方差为1，z也会调节到相应的范围，但是它确实减低了梯度消失或者梯度爆炸问题，因为它给权重矩阵设置了合理的取值范围，你也知道，它不能比1大太多或者小太多。



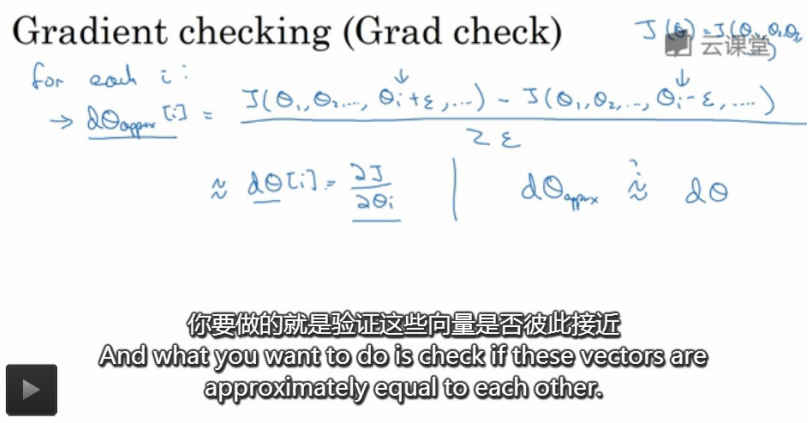
12、





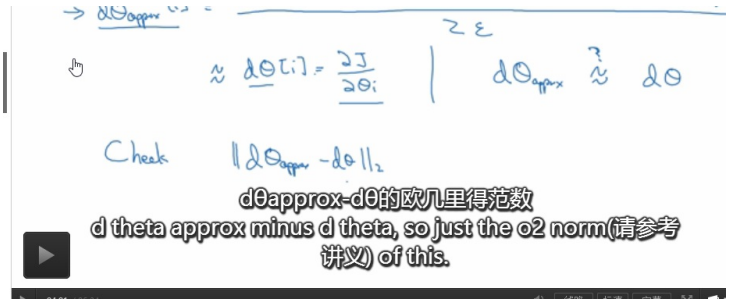
这种方法非常值得使用。因为它的结果更加准确。双边误差公式的结果更加准确。

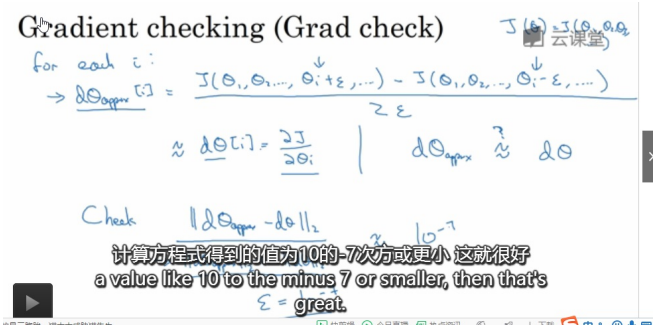
13、

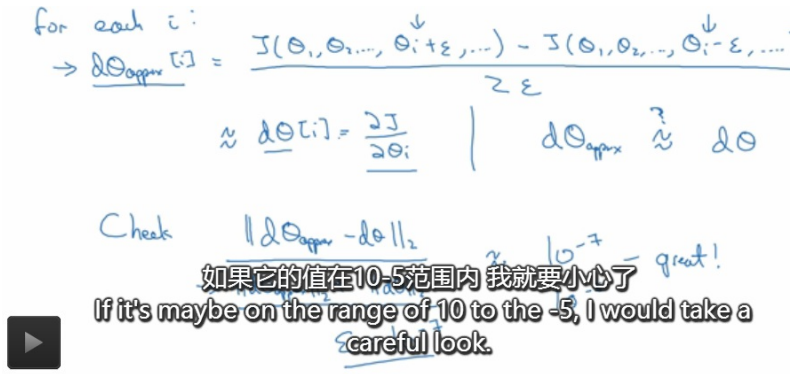


如何定义两个向量是否彼此接近？

计算着两个向量的距离。如下：







如果大于10的负3次方，可能就与有bug

14、关于梯度检测实现的注记

1、不要在训练中使用梯度检测，它只是用于调试。

因为为了实施梯度下降，你必须使用backprop来计算dΘ，这是一个漫长的计算过程。完成之后，你会关闭梯度检测。

2、如果算法的梯度检测失败，要检查所有的项。检查每一项，并试着找出bug，我们要检查看看哪一项的导数dΘapprox[i]与dΘ的值相差这么多。

3、在使用梯度检测时，如果使用正则化，请注意正则项。

4、梯度检测不能与dropout同时进行。