## P47 训练集（training），验证集（development），测试集（test）

Train sets用来**训练你的算法模型**；Dev sets用来**验证不同算法**的表现情况，从中选择最好的算法模型；Test sets用来**测试最好算法**的实际表现，作为该算法的无偏估计。

通常Train sets和Test sets比例为70%和30%。如果有Dev sets，比例为60%、20%、20%，分别对应Train/Dev/Test sets。这种比例分配在样本数量**不大**的情况下，是合理的。

但是如果**数据量很大**的时候，对于大数据样本，Train/Dev/Test sets的比例通常可以设置为98%/1%/1%，或者99%/0.5%/0.5%。样本数据量越大，相应的Dev/Test sets的比例可以设置的越低一些。

**Make sure train and test come from same distribution**。

现代深度学习还有个重要的问题就是训练样本和测试样本分布上不匹配，网络下载的数据和自己拍照上传的数据不是同一分布。尽量保证Dev sets和Test sets来自于同一分布。训练样本非常重要，对样本进行预处理。例如图片的翻转、加入随机噪声等，来扩大训练样本的数量，增加模型的鲁棒性。即使Train sets和Dev/Test sets不来自同一分布，使用这些技巧也能提高模型性能

如果没有Test sets也是没有问题的。Test sets的目标主要是进行无偏估计。我们可以通过Train sets训练不同的算法模型，然后分别在Dev sets上进行验证，根据结果选择最好的算法模型。这样也是可以的，不需要再进行无偏估计了。如果只有Train sets和Dev sets，通常也有人把这里的Dev sets称为Test sets，要注意加以区别。

## P48 Bias（偏差）/Variance（方差）

传统的机器学习算法中，Bias和Variance是对立的，分别对应着欠拟合和过拟合，需要在Bias和Variance间进行权衡。而深度学习中可以同时降低二者，构建最佳神经网络模型。

假设Train set error为1%，而Dev set error为11%，即该算法模型对训练样本的识别很好，但是对验证集的识别却不太好。这说明了该模型对训练样本可能存在过拟合，模型泛化能力不强，导致验证集识别率低。这恰恰是high variance的表现。

假设Train set error为15%，而Dev set error为16%，虽然二者error接近，即该算法模型对训练样本和验证集的识别都不是太好。这说明了该模型对训练样本存在欠拟合。这恰恰是high bias的表现。

假设Train set error为15%，而Dev set error为30%，说明了该模型既存在high bias也存在high variance（深度学习中最坏的情况）。

再假设Train set error为0.5%，而Dev set error为1%，即low bias和low variance，是最好的情况。

值得一提的是，以上的这些假设都是建立在base error是0的基础上，即人类都能正确识别所有猫类图片（手工给数据打的标签）。base error不同，相应的Train set error和Dev set error会有所变化，但没有相对变化。

## P49 机器学习的基本诀窍

**解决high bais：**增加神经网络的隐藏层个数、神经元个数，训练时间延长，选择其它更复杂的NN模型等。

在bais降下来后，即解决欠拟合问题后，再考虑variance的问题。

**解决high variance：**more data ， Regularization正则化，选择其他更复杂的NN模型等。

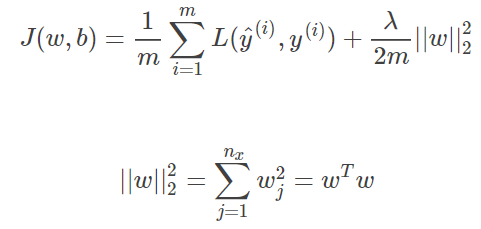
在深度学习之前，没有一个好的方法单独降低偏差或方差，深度学习诞生后，通过使用更复杂的神经网络和海量的训练样本，一般能够同时有效减小Bias和Variance。

## P50 正则化（regularization）

正则化用于减少方差。虽然增加数据量也可以减少方差，但是搜集数据的成本太高。

### 在逻辑回归中：

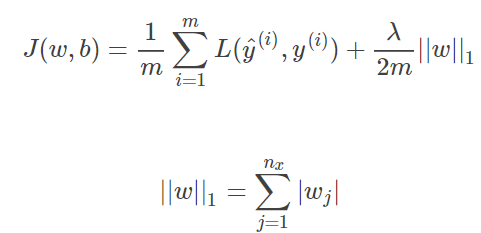
**L2 regularization：**



使用的是参数w的欧几里得范数也称L2范数。

**为什么 b 不正则化**：一般w的维度很大，而b只是一个常数。相比较来说，参数很大程度上由w决定，改变b值对整体模型影响较小。可以但没必要。

**L1 regularization：**

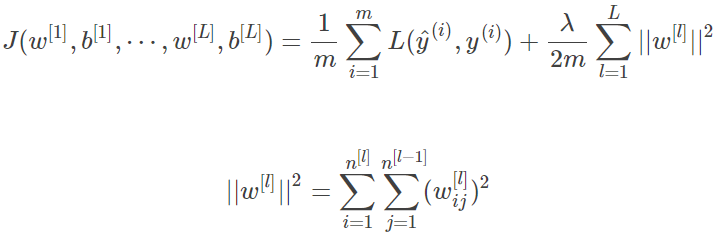


与L2 regularization相比，L1 regularization得到的w更加稀疏，即很多w为零值。其优点是节约存储空间，因为大部分w为0。然而，实际上L1 regularization在解决high variance方面比L2 regularization并不更具优势。而且，L1的在微分求导方面比较复杂。所以，一般L2 regularization更加常用。

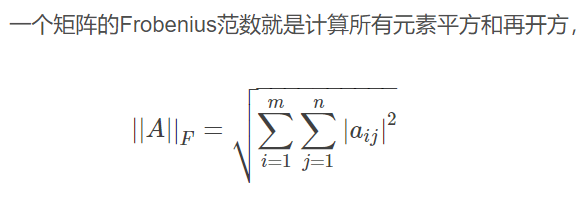
上面的λ（Lambda）称为正则化参数，超参数的一种。在Dev set中进行验证，选择最佳的λ。在python中，由于lambda是保留字，所以为了避免冲突，我们使用lambd来表示。

### 在深度学习中：

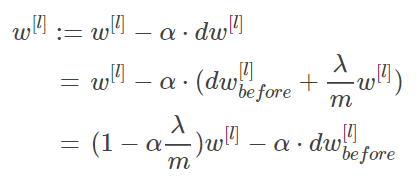
**L2 regularization：**



第二行对w[l] 求L2范数，矩阵的每个元素平方再求和。也称为Frobenius范数。

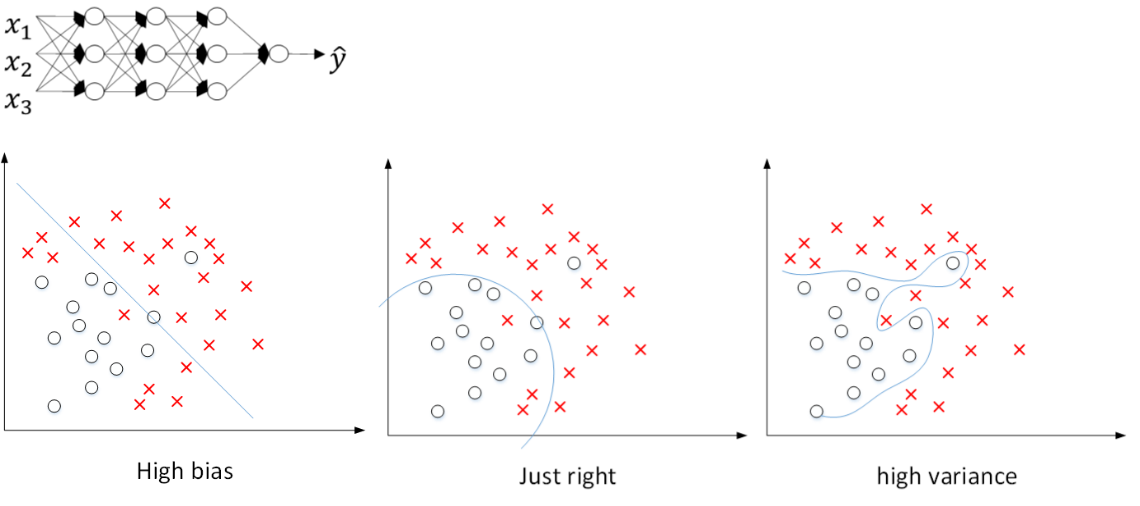


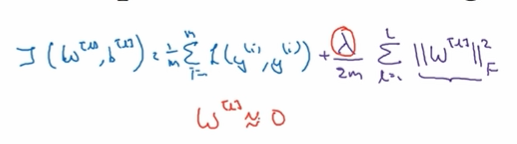
L2 regularization也被称做weight decay。这是因为，由于加上了正则项，有个增量，在更新的时候，会多减去这个增量（1-α\*λ/m），使得比没有正则项的值要小一些。



## P51 为什么正则化可以减少过拟合

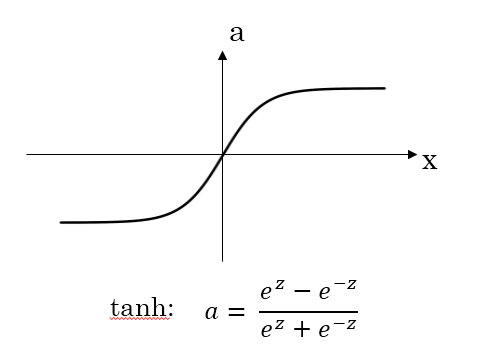
Eg1：





直观理解：若λ非常大，则部分w接近于0。即部分神经元结点不发挥影响。解决过拟合问题。λ特别小就过拟合。因此存在一个适中值使得just right。

Eg2：



在z的[-1,1]等其他小区间内呈线性，在两端为非线性。若λ增大，则w减小。Z = wa+b 则z也会变小。则每一层几乎都是线性的。若每层都是线性的，整个网络也是线性的。因此就不能拟合那些非线性的过拟合函数。

## P52 dropout正则化

随机失活技术。每层的神经元都有概率失活，从而得到更简单的神经网络。

**Inverted dropout 反向随机失活：**

假设有三层的网络，l=3，keep\_prob = 0.8(存活保留的 概率)

则d3 = np.random.randn(a3.shape[0],a3.shape[1])<keep\_prob。d3矩阵元素有0.8的概率为1.

a3为计算的激活后矩阵，a3 = np.multply(a3,d3).这里的multiply是**逐元素相乘**。即a3\*d3.将a3中对应d3位置的元素置0.

最后放大a3，a3/=keep\_prob即除以0.8. 原因：假如第三层有50个单元，则将有10个单元失活。则对于z4 = w4 a3 + b4 ，a3的期望值会减少20%，为了不减少z4的期望值，因此a3要先除以0.8。

这一步也简化了测试部分，它减少了可能引入的缩放问题。在早期除以keep\_prob使得在测试过程中求取平均值变得越来越复杂。

在第一次迭代中，随机清零部分单元，第二次时再次随机清零部分单元。由dl随机化决定。

**通过dropout完成训练后在测试阶段的做法：**

Not use dropout：训练阶段随机就算了，我们不想让我们的测试阶段的输出也是随机的。测试阶段使用只会为预测增加噪声。

另外一个好处就是在对该dropout后的神经网络进行测试时能够减少scaling问题。因为在训练时，使用scale up（除以keep\_prob）保证的期望值没有大的变化，测试时就不需要再对样本数据进行类似的尺度伸缩操作了。

## P53 为什么dropout有效

Dropout通过每次迭代训练时，随机选择不同的神经元，相当于每次都在不同的神经网络上进行训练，类似机器学习中Bagging的方法（三个臭皮匠，赛过诸葛亮），能够防止过拟合。

除此之外，还可以从权重w的角度来解释为什么dropout能够有效防止过拟合。对于某个神经元来说，某次训练时，它的某些输入在dropout的作用被过滤了。而在下一次训练时，又有不同的某些输入被过滤。经过多次训练后，某些输入被过滤，某些输入被保留。这样，该神经元就不会受某个输入非常大的影响，影响被均匀化了。也就是说，对应的权重w不会很大。这从从效果上来说，与L2 regularization是类似的，都是**对权重w进行“惩罚”**，减小了w的值。

对于同一组训练数据，利用不同的神经网络训练之后，求其输出的平均值可以减少overfitting。

**使用dropout注意事项：**

不同隐藏层的dropout系数keep\_prob可以不同。神经元越多的隐藏层，设置的小一些如0.5，反之亦然如0.8。

另外，实际应用中，不建议对输入层进行dropout，如果输入层维度很大，例如图片，那么可以设置dropout，但keep\_out应设置的大一些，例如0.8，0.9。**Dropout在电脑视觉CV领域应用比较广泛，因为输入层维度较大，而且没有足够多的样本数量**。

**使用dropout的时候，就不能使用绘图代价函数的方式调试错误**了。一般做法是，关闭dropout，将所有层的keep\_prob全设置为1，再绘制cost function，即涵盖所有神经元，看J是否单调下降。下一次迭代训练时，再将keep\_prob设置为其它值。

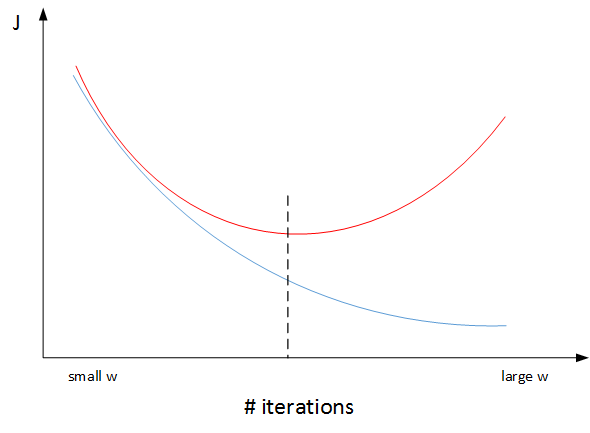
## P54 其他正则化方法

一种方法是增加训练样本数量。但是通常成本较高。我们可以对已有的训练样本进行一些处理来“制造”出更多的样本，称为data augmentation。例如图片识别问题中，可以对已有的图片进行**水平翻转**、垂直翻转（不推荐）、**任意角度旋转、缩放或扩大**等等。不需要增加额外成本，也能起到防止过拟合的效果。



在数字识别中，也可以将原有的数字图片进行任意旋转或者扭曲，或者增加一些noise。

另外一种防止过拟合的方法：early stopping。一个神经网络模型随着迭代训练次数增加，**train set error一般是单调减小**的，而**dev set error 先减小，之后又增大。**也就是说训练次数过多时，模型会对训练样本拟合的越来越好，但是对验证集拟合效果逐渐变差，即发生了过拟合。因此，**迭代训练次数不是越多越好**，可以通过train set error和dev set error随着迭代次数的变化趋势，选择合适的迭代次数，即early stopping。如下图



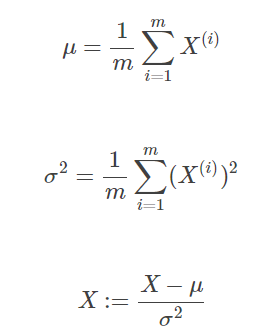
Early stopping缺点：**通过减少迭代次数同时优化J和防止过拟合，导致二者优化效果都不好**

机器学习训练模型有两个目标：一是优化cost function，尽量减小J；二是防止过拟合。这两个目标彼此对立的，即减小J的同时可能会造成过拟合，反之亦然。我们把这二者之间的关系称为正交化orthogonalization。在深度学习中，我们可以同时减小Bias和Variance，构建最佳神经网络模型。但是，Early stopping的做法通过减少迭代训练次数来防止过拟合，这样J就不会足够小。也就是说，early stopping将上述两个目标融合在一起，同时优化，但可能没有“分而治之”的效果好。

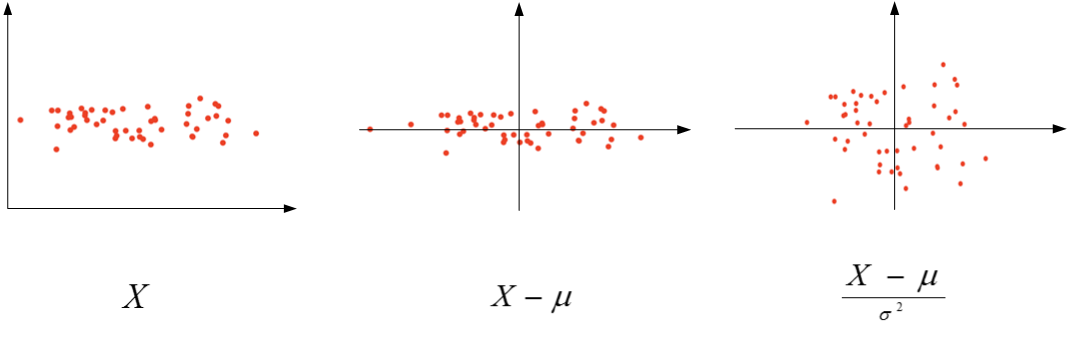
与early stopping相比，**L2 regularization可以实现“分而治之”**的效果：迭代训练足够多，减小J，而且也能有效防止过拟合。而**L2 regularization的缺点之一是最优的正则化参数的选择比较复杂**。对这一点来说，early stopping比较简单。总的来说，L2 regularization更加常用一些。

## P55 归一化输入（normalizing inputs标准化输入）

在训练神经网络时，标准化输入可以提高训练的速度。标准化输入就是对训练数据集进行归一化的操作，即将原始数据减去其均值后，再除以其方差：下图第二行的xi是已经减去均值后的xi



以二维平面为例，下图展示了其归一化过程：



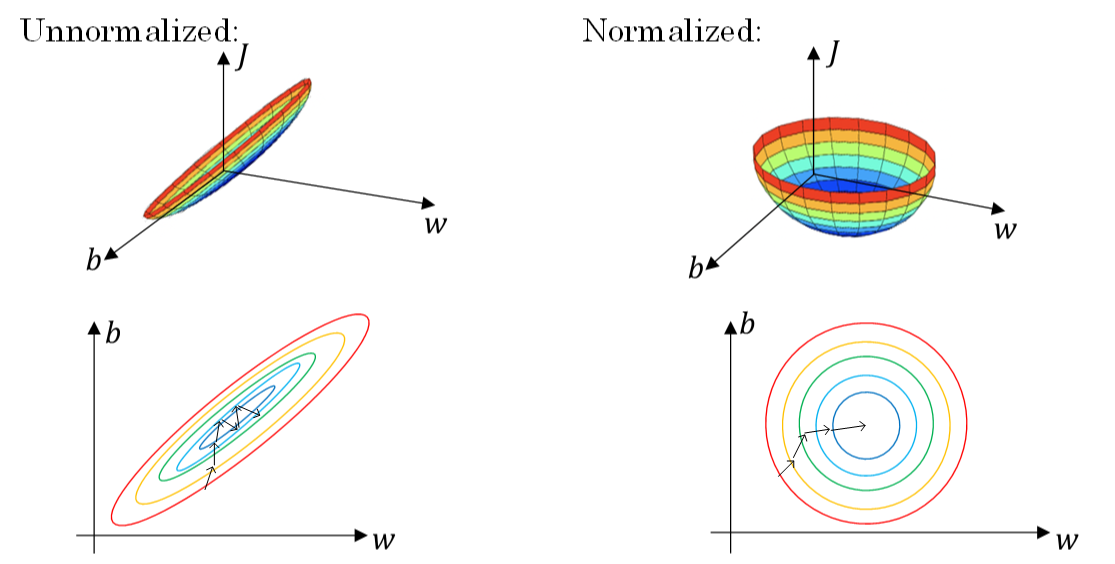
Ps：**训练集用的μ和σ2**进行标准化处理，则**测试集以及之后的应用**也应该**用相同的μ和σ2**进行标准化处理。

让所有输入归一化同样的尺度上，方便进行梯度下降算法时能够更快更准确地找到全局最优解。

假如输入特征是二维的，且x1的范围是[1,1000]，x2的范围是[0,1]。

**如果不进行标准化处理**，x1与x2之间分布极不平衡，训练得到的w1和w2也会在数量级上差别很大。这样导致的结果是cost function与w和b的关系可能是一个非常细长的椭圆形碗。对其进行梯度下降算法时，由于w1和w2数值差异很大，只能选择很小的学习因子，来避免J发生振荡。一旦较大，必然发生振荡，J不再单调下降。如下左图所示。

**进行标准化操作后**，x1与x2分布均匀，w1和w2数值差别不大，得到的cost function与w和b的关系是类似圆形碗。对其进行梯度下降算法时，可以选择相对大一些，且J一般不会发生振荡，保证了J是单调下降的。如下右图所示

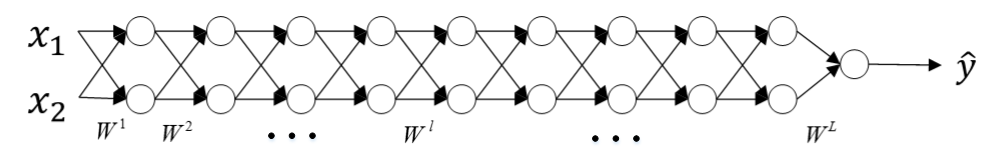


Ps：另外一种情况，如果输入特征之间的范围本来就比较接近，那么不进行标准化操作也是没有太大影响的。但是，标准化处理在大多数场合下还是值得推荐的。

## P56 梯度消失与梯度爆炸

在神经网络尤其是深度神经网络中存在可能存在这样一个问题：梯度消失和梯度爆炸。意思是当训练一个 **层数非常多的神经网络**时，计算得到的梯度可能非常小或非常大，甚至是指数级别的减小或增大。这样会让训练过程变得非常困难。

举个例子来说明，假设一个多层的每层只包含两个神经元的深度神经网络模型，如下图所示：



为了简化复杂度，便于分析，我们令各层的激活函数为线性函数，即。且忽略各层常数项b的影响，令b全部为零。那么，该网络的预测输出为：

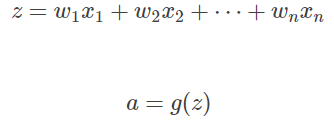


如果各层权重的元素都稍大于1，例如1.5，则预测输出将正比于网络层数L。L越大，越大，且呈指数型增长。我们称之为数值爆炸。相反，如果各层权重的元素都稍小于1，例如0.5，则预测输出将正比于。网络层数L越多，呈指数型减小。我们称之为数值消失。

同样，这种情况也会引起**梯度**呈现同样的指数型增大或减小的变化。L非常大时，例如L=150，则梯度会非常大或非常小，引起每次更新的步进长度过大或者过小，这让训练过程十分困难。

## P57 深度网络权重初始化

下面介绍如何改善Vanishing and Exploding gradients这类问题，方法是对权重w进行一些初始化处理。

初始化单个神经元：

忽略常数b=0。. 为了让z不会过大或者过小，思路是让w与n（**该层输入特征的数目**即n[l-1]）有关，且n越大，w权重应该越小。,虽然不能解决问题，但降低了梯度消失和爆炸的程度，使得w不会比1大很多，也不会比1小很多。

如果激活函数是tanh,：其方差为1/n[l-1]：

w[l] = np.random.randn(n[l],n[l-1])\*np.sqrt(1/n[l-1]);

如果激活函数是ReLU，权重w的初始化一般令其方差为2/n[l-1]：

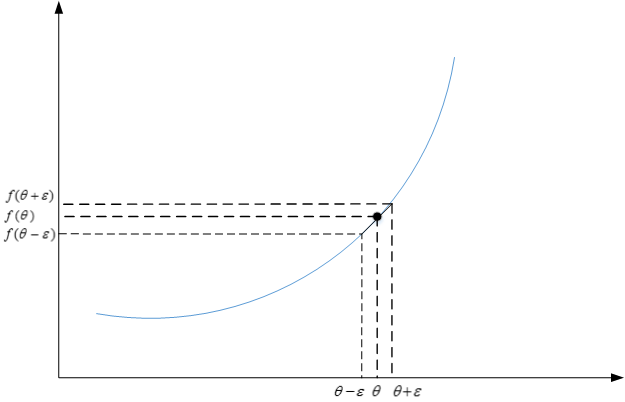
除此之外，Yoshua Bengio提出了另外一种初始化w的方法，令其方差为2/(n[l-1]+n[l]).

可以根据激活函数选择不同初始化方法。另外可以对这些初始化方法设置某些参数，如**此处方差**作为超参数但是用处通常不大，通过验证集进行验证，得到最优参数，来优化神经网络。

## P58 梯度的数值近似

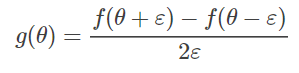
Back Propagation神经网络有一项重要的测试是**梯度检查（gradient checking）**。其目的是检查验证反向传播过程中梯度下降算法是否正确。

本小节将先介绍如何**近似求出梯度值**。



在该点两侧进行计算梯度。若只取一侧误差大很多。

函数f在点θ处的梯度可以表示成：



## P59 梯度检查（gradient checking）

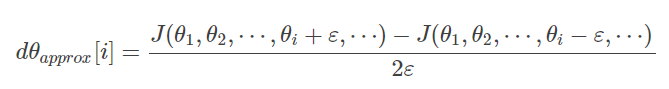
介绍完如何近似求出梯度值后，我们将介绍如何进行梯度检查，来验证训练过程中是否出现bugs。

**1.将wb构造一维向量θ**：梯度检查首先要做的是分别将W[1],b[1],⋯,W[L],b[L]这些矩阵（首尾相接）构造成一维向量，然后将这些一维向量组合起来构成一个更大的一维向量θ。这样cost function J(W[1],b[1],⋯,W[L],b[L])就可以表示成J(θ)。

**2.dwdb构造一维向量dθ：**然后将反向传播过程通过梯度下降算法得到的dW[1],db[1],⋯,dW[L],db[L]按照一样的顺序构造成一个一维向量dθ。dθ的维度与θ一致。

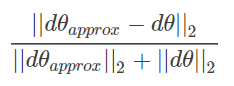
**3.**根据J(θ)对每个**θi**计算近似梯度，与反向传播时得到的**dθi**进行比较，检查是否一致。

近似梯度计算方法为：



**4.比较dθapprox与dθ的相似度。**

通过欧氏距离公式如下：



若欧氏距离越小，例如10−7，甚至更小，则表明**dθapprox**与**dθ**越接近，即反向梯度计算是正确的，没有bugs。如果欧氏距离较大，例如10−5，则表明梯度计算可能出现问题，需要再次检查是否有bugs存在。如果欧氏距离很大，例如10−3，甚至更大，则表明**dθapprox**与**dθ**差别很大，梯度下降计算过程有bugs，需要仔细检查。

## P60 梯度检查的实现时的注意事项

* 不要在整个训练过程中都进行梯度检查，**仅仅作为debug使用**。
* 如果梯度检查出现错误，找到对应出错的梯度，**检查其推导是否出现错误。**
* 注意**不要忽略正则化项**，计算近似梯度的时候要包括进去。
* 梯度检查时**关闭dropout**，检查完毕后再打开dropout。
* 随机初始化时运行梯度检查，经过一些训练后再进行梯度检查（不常用）。