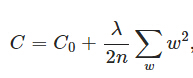
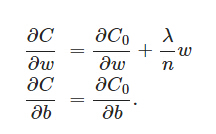
## L2 regularization(ridge regression)

L2正则化就是在代价函数后面再加上一个正则化项：

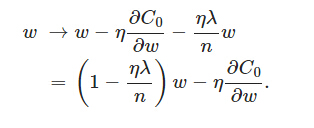


C0代表原始的代价函数，后面那一项就是L2正则化项，它是这样来的：所有参数w的平方的和，除以训练集的样本大小n。λ就是正则项系数，权衡正则项与C0项的比重。另外还有一个系数1/2，1/2经常会看到，主要是为了后面求导的结果方便，后面那一项求导会产生一个2，与1/2相乘刚好凑整。

L2正则化项是怎么避免overfitting的呢？我们推导一下看看，先求导：

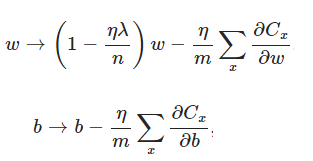


可以发现L2正则化项对b的更新没有影响，但是对于w的更新有影响:



在不使用L2正则化时，求导结果中w前系数为1，现在w前面系数为 1−ηλ/n ，因为η、λ、n都是正的，所以 1−ηλ/n小于1，它的效果是减小w，这也就是权重衰减（weight decay）的由来。当然考虑到后面的导数项，w最终的值可能增大也可能减小。

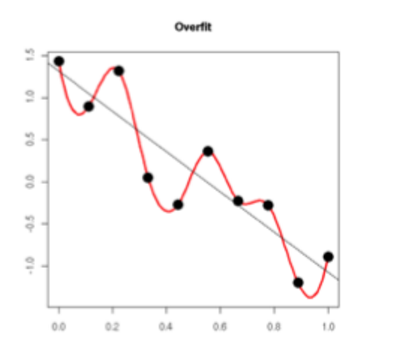
另外，需要提一下，对于基于mini-batch的随机梯度下降，w和b更新的公式跟上面给出的有点不同：



对比上面w的更新公式，可以发现后面那一项变了，变成所有导数加和，乘以η再除以m，m是一个mini-batch中样本的个数。

到目前为止，我们只是解释了L2正则化项有让w“变小”的效果，但是还没解释为什么w“变小”可以防止overfitting？一个所谓“显而易见”的解释就是：更小的权值w，从某种意义上说，表示网络的复杂度更低，对数据的拟合刚刚好（这个法则也叫做奥卡姆剃刀），而在实际应用中，也验证了这一点，L2正则化的效果往往好于未经正则化的效果。当然，对于很多人（包括我）来说，这个解释似乎不那么显而易见，所以这里添加一个稍微数学一点的解释（引自知乎）：

过拟合的时候，拟合函数的系数往往非常大，为什么？如下图所示，过拟合，就是拟合函数需要顾忌每一个点，最终形成的拟合函数波动很大。在某些很小的区间里，函数值的变化很剧烈。这就意味着函数在某些小区间里的导数值（绝对值）非常大，由于自变量值可大可小，所以只有系数足够大，才能保证导数值很大。



而正则化是通过约束参数的范数使其不要太大，所以可以在一定程度上减少过拟合情况。

## ****L1 regularization（LASSO）****

Least Absolute Shrinkage and Selection Operator

在原始的代价函数后面加上一个L1正则化项，即所有权重w的绝对值的和，乘以λ/n（这里不像L2正则化项那样，需要再乘以1/2，具体原因上面已经说过。）





上式中sgn(w)表示w的符号。那么权重w的更新规则为：

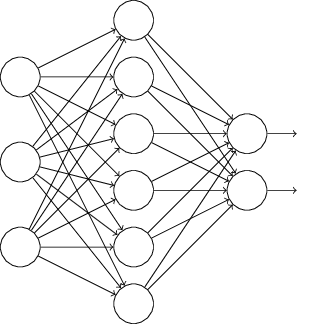


比原始的更新规则多出了η \* λ \* sgn(w)/n这一项。当w为正时，更新后的w变小。当w为负时，更新后的w变大——因此它的效果就是让w往0靠，使网络中的权重尽可能为0，也就相当于减小了网络复杂度，防止过拟合。

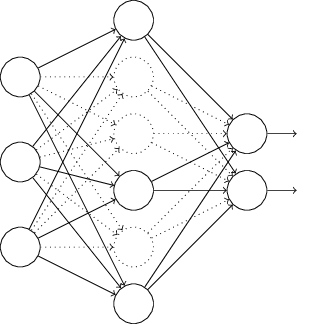
另外，上面没有提到一个问题，当w为0时怎么办？当w等于0时，|W|是不可导的，所以我们只能按照原始的未经正则化的方法去更新w，这就相当于去掉η\*λ\*sgn(w)/n这一项，所以我们可以规定sgn(0)=0，这样就把w=0的情况也统一进来了。（在编程的时候，令sgn(0)=0,sgn(w>0)=1,sgn(w<0)=-1）

## ****Dropout****

L1、L2正则化是通过修改代价函数来实现的，而Dropout则是通过修改神经网络本身来实现的，它是在训练网络时用的一种技巧（trike）。它的流程如下：



假设我们要训练上图这个网络，在训练开始时，我们随机地“删除”一半的隐层单元，视它们为不存在，得到如下的网络：



保持输入输出层不变，按照BP算法更新上图神经网络中的权值（虚线连接的单元不更新，因为它们被“临时删除”了）。

以上就是一次迭代的过程，在第二次迭代中，也用同样的方法，只不过这次删除的那一半隐层单元，跟上一次删除掉的肯定是不一样的，因为我们每一次迭代都是“随机”地去删掉一半。第三次、第四次……都是这样，直至训练结束。

以上就是Dropout，它为什么有助于防止过拟合呢？可以简单地这样解释，运用了dropout的训练过程，相当于训练了很多个只有半数隐层单元的神经网络（后面简称为“半数网络”），每一个这样的半数网络，都可以给出一个分类结果，这些结果有的是正确的，有的是错误的。随着训练的进行，大部分半数网络都可以给出正确的分类结果，那么少数的错误分类结果就不会对最终结果造成大的影响。

更加深入地理解，可以看看Hinton和Alex两牛2012的论文《ImageNet Classification with Deep Convolutional Neural Networks》

## ****数据集扩增（data augmentation）****

“有时候不是因为算法好赢了，而是因为拥有更多的数据才赢了。”

不记得原话是哪位大牛说的了，hinton？从中可见训练数据有多么重要，特别是在深度学习方法中，更多的训练数据，意味着可以用更深的网络，训练出更好的模型。

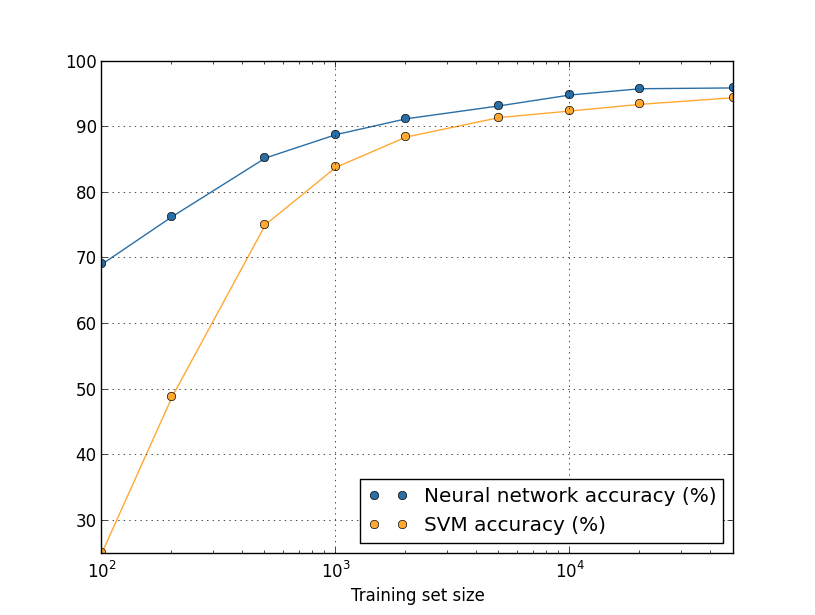
既然这样，收集更多的数据不就行啦？如果能够收集更多可以用的数据，当然好。但是很多时候，收集更多的数据意味着需要耗费更多的人力物力，有弄过人工标注的同学就知道，效率特别低，简直是粗活。

所以，可以在原始数据上做些改动，得到更多的数据，以图片数据集举例，可以做各种变换，如：

* 将原始图片旋转一个小角度
* 添加随机噪声
* 一些有弹性的畸变（elastic distortions），论文《Best practices for convolutional neural networks applied to visual document analysis》对MNIST做了各种变种扩增。
* 截取（crop）原始图片的一部分。比如DeepID中，从一副人脸图中，截取出了100个小patch作为训练数据，极大地增加了数据集。感兴趣的可以看《Deep learning face representation from predicting 10,000 classes》.

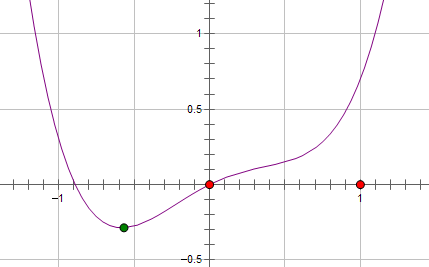
更多数据意味着什么？

用50000个MNIST的样本训练SVM得出的accuracy94.48%，用5000个MNIST的样本训练NN得出accuracy为93.24%，所以更多的数据可以使算法表现得更好。在机器学习中，算法本身并不能决出胜负，不能武断地说这些算法谁优谁劣，因为数据对算法性能的影响很大。



# l1 相比于 l2 为什么容易获得稀疏解？

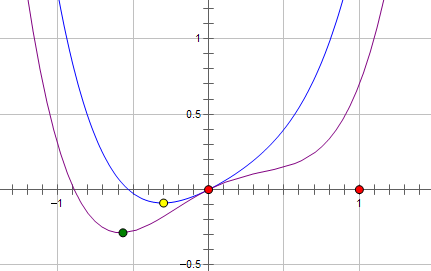
假设费用函数 L 与某个参数 x 的关系如图所示：



则最优的 x 在绿点处，x 非零。

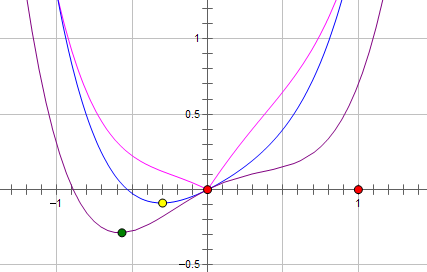
现在施加 L2 regularization，新的费用函数如图中蓝线所示：

最优的 x 在黄点处，x 的绝对值减小了，但依然非零。



最优的 x 在黄点处，x 的绝对值减小了，但依然非零。

而如果施加 L1 regularization，则新的费用函数如图中粉线所示：最优的 x 就变成了 0。这里利用的就是绝对值函数的尖峰。



最优的 x 就变成了 0。这里利用的就是绝对值函数的尖峰。

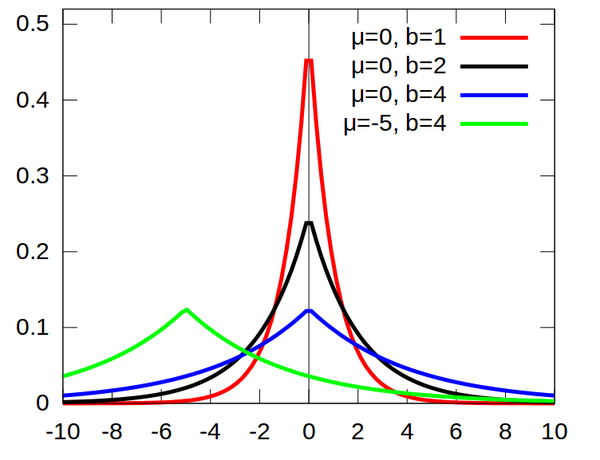
两种 regularization 能不能把最优的 x 变成 0，取决于原先的费用函数在 0 点处的导数。如果本来导数不为 0，那么施加 L2 regularization 后导数依然不为 0，最优的 x 也不会变成 0。而施加 L1 regularization 时，只要 regularization 项的系数 C 大于原先费用函数在 0 点处的导数的绝对值，x = 0 就会变成一个极小值点。

上面只分析了一个参数 x。事实上 L1 regularization 会使得许多参数的最优值变成 0，这样模型就稀疏了。

。

首先你要知道L1范式和L2范式是怎么来的,然后是为什么要把L1或者L2正则项加到代价函数中去.L1,L2范式来自于对数据的先验知识.如果你认为,你现有的数据来自于高斯分布,那么就应该在代价函数中加入数据先验P(x),一般由于推导和计算方便会加入对数似然,也就是log(P(x)),然后再去优化,这样最终的结果是,由于你的模型参数考虑了数据先验,模型效果当然就更好.

哦对了,如果你去看看高斯分布的概率密度函数P(x),你会发现取对数后的log(P(x))就剩下一个平方项了,这就是L2范式的由来--高斯先验.同样,如果你认为你的数据是稀疏的,不妨就认为它来自某种laplace分布.不知你是否见过laplace分布的概率密度函数,我贴出一张维基上的图,



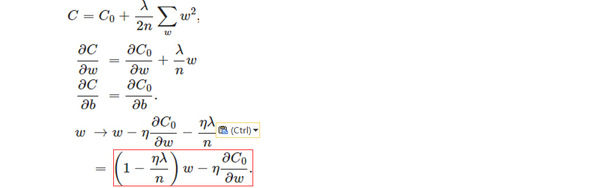
laplace分布是尖尖的分布,是不是很像一个pulse?从这张图上,你应该就能看出,服从laplace分布的数据就是稀疏的了(只有很小的概率有值,大部分概率值都很小或为0).那么,加入了laplace先验作为正则项的代价函数是什么?  
再看看laplace分布的概率密度函数

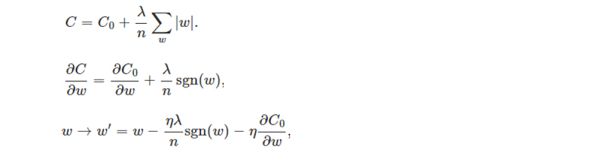
https://pic2.zhimg.com/2cdb491cc9f4814caa4f4d007e543a05_b.png

看到没,如果取对数,剩下的是一个一次项|x-u|,这就是L1范式.所以用L1范式去正则,就假定了你的数据是laplace分布,是稀疏的.

从另外的一个梯度的角度看：

假设原先损失函数是C0，那么在L2和L1正则条件下对参数求导分别是：





可以想象，帝都下降的方向，当w小于1的时候，L2正则项的蒸发效果越来越小，L1正则项惩罚效果依然很大，L1可以惩罚到0，而L2很难。也就是说： l1范数恒定以1的梯度向0靠拢, 容易收敛到0。l2范数越接近0梯度数值越小, 容易收敛到一堆小值。

2. l1对于小值的惩罚，比l2的大。自然求优化的时候，求得小系数的值越小。