20180412 正则化

总结： L2 正则化， 在代价函数上添加二阶项，使最终的w减小，更小的权值，网络复杂度会更低

L1 正则化， 代价函数加上一阶 绝对值项，使w 向 0 靠近（有正有负）， 更小的w，网络复杂度会更低。

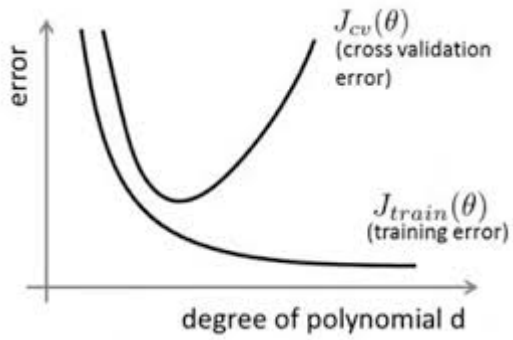
Dropout： 输入层和输出层不变，每次随机扔掉一半的隐藏层，最终所有这些半数的隐藏层决定了最终结果，正确的结果会在最后取到 越来越重要的作用，而非 正确的结果， 对结果的影响是越来越小的。

数据集的扩增： 对图片进行旋转，加入随机噪声， 弹性的畸变， 将图片截取为一个个的小 patch。 可增大数据集。

原文：
http://wepon.me/2015/03/14/%E6%AD%A3%E5%88%99%E5%8C%96%E6%96%B9%E6%B3%95%EF%BC%9AL1%E5%92%8CL2%20regularization%E3%80%81%E6%95%B0%E6%8D%AE%E9%9B%86%E6%89%A9%E5%A2%9E%E3%80%81dropout/

正则化方法：防止过拟合，提高泛化能力

在训练数据不够多时，或者overtraining时，常常会导致overfitting（过拟合）。其直观的表现如下图所示，随着训练过程的进行，模型复杂度增加，**在training data上的error渐渐减小，但是在验证集上的error却反而渐渐增大**——因为训练出来的网络过拟合了训练集，对训练集外的数据却不work。



为了防止overfitting，可以用的方法有很多，下文就将以此展开。有一个概念需要先说明，在机器学习算法中，我们常常将原始数据集分为三部分：

training data、validation data，testing data。

这个validation data是什么？它其实就是用来**避免过拟合的**，在训练过程中，我们通常用它来**确定一些超参数**（比如根据validation data上的accuracy来确定early stopping的**epoch**大小、根据validation data确定**learning rate**等等）。

那为啥不直接在testing data上做这些呢？因为如果在testing data做这些，那么随着训练的进行，我们的网络实际上就是在一点一点地overfitting我们的testing data，导致最后得到的testing accuracy没有任何参考意义。

因此，**training data的作用是计算梯度更新权重**，**validation data（防止过拟合）**如上所述，testing data则给出**一个accuracy以判断网络的好坏。**

避免过拟合的方法有很多：**early stopping、数据集扩增（Data augmentation）、正则化（Regularization）包括L1、L2**（L2 regularization也叫weight decay），dropout。

L2 regularization（权重衰减）

L2正则化就是**在代价函数后面再加上一个正则化项**：

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2,$$

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：

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$$
$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b}.$$

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**:

$$\begin{aligned} w &\rightarrow w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta\lambda}{n} w \\ &= \left(1 - \frac{\eta\lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}. \end{aligned}$$

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：

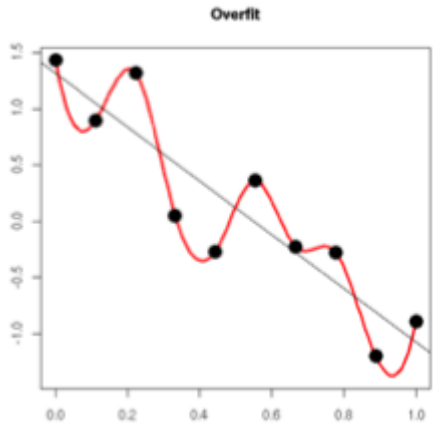
$$w \rightarrow \left(1 - \frac{\eta\lambda}{n}\right) w - \frac{\eta}{m}\sum_x \frac{\partial C_x}{\partial w}$$

$$b \rightarrow b - \frac{\eta}{m}\sum_x \frac{\partial C_x}{\partial b},$$

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

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。）

$$C = C_0 + \frac{\lambda}{n} \sum_w |w|.$$

同样先计算导数：

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} \operatorname{sgn}(w),$$

上式中sgn(w)表示w的符号。那么权重w的更新规则为：

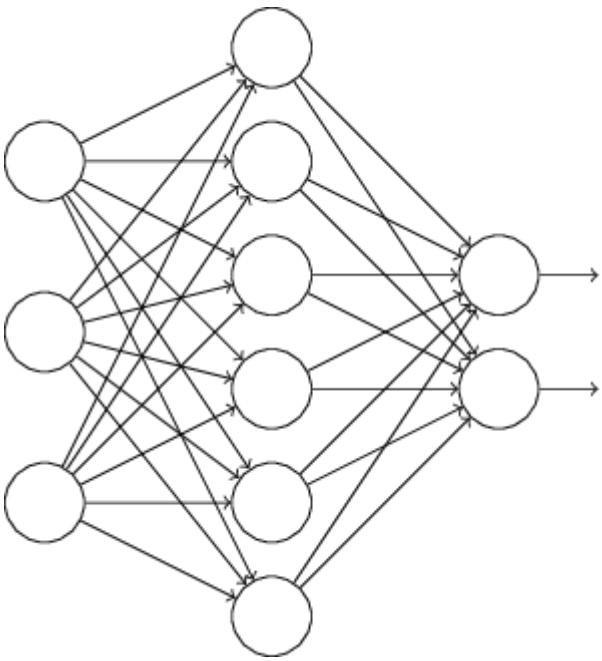
$$w \to w' = w - \frac{\eta\lambda}{n}\mathrm{sgn}(w) - \eta\frac{\partial C_0}{\partial w},$$

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。

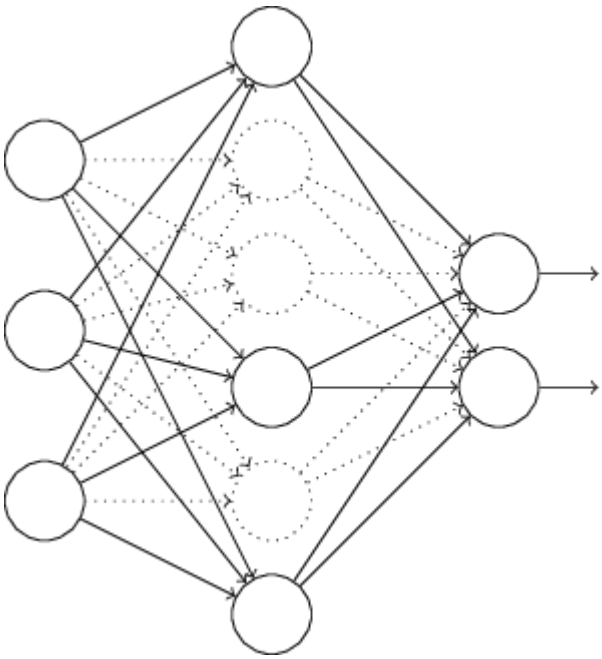
另外，上面没有提到一个问题，当w为0时怎么办？当w等于0时，`|w|`是不可导的，所以我们只能按照原始的未经正则化的方法去更新w，这就相当于去掉`η*λ*sgn(w)/n`这一项，所以我们可以规定`sgn(0)=0`，这样就把w=0的情况也统一进来了。（在编程的时候，令`sgn(0)=0,sgn(w>0)=1,sgn(w<0)=-1`）

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。