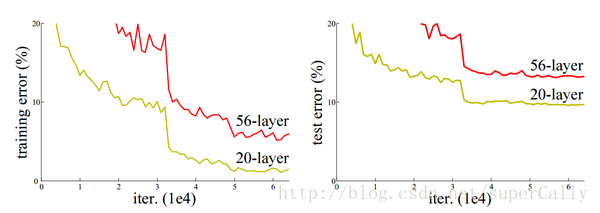
# 深度残差神经网络(Deep Residual Network)

# 1. 什么是DRN，为什么需要DRN

DRN的全称是Deep Residual Network，深度残差网络，是对普通的深度学习网络的一种改进。

我们为什么需要深度残差网络呢？因为普通的深度学习网络存在着这样的问题

在层数比较少的时候，我们增加网络的深度，可以获得更好的表达效果。但是当层数已经足够多，比如说超过了三十层，那这个时候我们增加深度，反而会降低识别率



这是因为，当层数加深的时候，梯度在传播过程中会逐渐消失，导致无法对前面几层的权重进行调整。

【我个人认为，梯度消失是因为：

1. 前向传递时，浅层网络对特征提取对最终结果的影响，从第1层 --> + 的方向逐渐减小。

即，越靠前的特征，越接近原始数据，（它们的取值是否合适）对分类结果影响越大。

而在网络深度加深时，浅层网络却一直得不到机会修正自己的错误。由于梯度消失，浅层网络调整量几乎为0。)

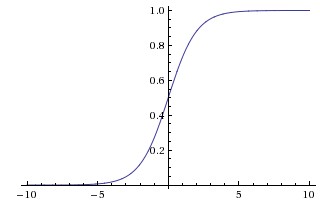
1. 后向传递时，离网络出口越近的网络位置，调整越大。尤其是最后的全连接网络。
2. 结果就是：深度CNN，在初始特征网络参数设置不理想，最终的全连接网络无法组合出需要的分类时。网络绝大多数时间花在了修改(全连接)组合上面，对特征参数的调整，力度不足，(浅层网络)几乎没有。
3. 追问：根源上来说，CNN的原理，本身就是一个，“提取特征值”+“组合特征结果”的过程。它是一个“死”网络。有以上的问题，是不可避免的。】

下面我们就来简单介绍一下梯度消失的问题。

# 2. 梯度消失问题

我们知道，神经网络的优化都是通过梯度下降的方法来优化的。每次我正向计算的时候计算出Loss，然后我需要知道，究竟怎样调整参数矩阵才能使得我的Loss更小，预测和事实更接近，所以我们需要通过back propogation来对传播梯度，但是在传播的过程中，会出现一些我们没有预料到的问题。

## 2.1 Sigmoid当中的梯度消失

   
这是Sigmoid的函数图像，我们可以看到，Sigmoid函数把[-INF,INF]上的值映射到了[-1,1]之间。这个映射可以增加非线性性，但是有个问题，就是当输入值足够大或者足够小的时候，输出值基本上不变，这个时候函数的梯度值基本上为0。这就达不到梯度下降的功能了

## 2.2 用ReLU代替Sigmoid

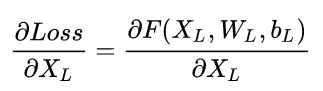
为了解决这个问题，引入了新的非线性单元，ReLU。其实表达式也非常简单。我们可以看到，不管输入值多大，梯度值都是存在的，这就在一定程度上解决了梯度消失的问题，因而ReLU也比Sigmoid在性能上要更好。



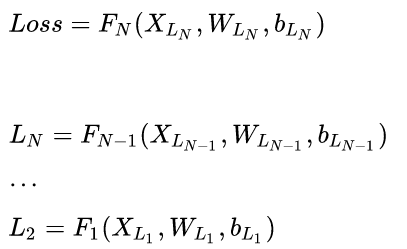
## 2.3 连乘导致的梯度消失问题

但是因为梯度连乘的问题，梯度消失的问题依然存在。我们通过正常的back propagation进行计算，

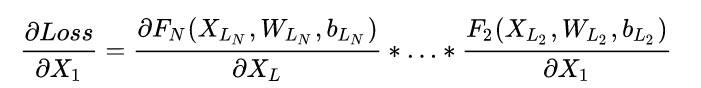




但是这个时候，如果网络很深很深，就会出现这样的情况



这个时候再做back propagation求偏导的话，就是



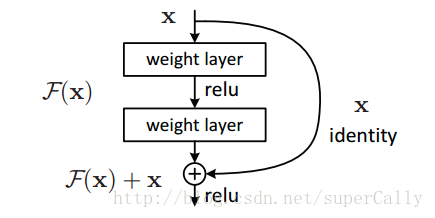
这个偏导就是我们求的梯度（gradient），这个值本来就很小，而且再计算的时候还要再乘stepsize，就更小了。【**虽然说的没有错，但是实际上是问题。理论上，我们是无法仅从公式()就认定会有消失问题的。但是我们知道，由于指数的存在，导致的变化是极不稳定的，只要多一些的值>1，则可能爆炸，如果很多<1，又有可能变成0。工程实践中，为了使网络稳定收敛，我们都是把往小了取，所以必然导致消失。**】

所以通过这里可以看到，梯度在反向传播过程中的计算，如果N很大，那么梯度值传播到前几层的时候就会越来越小，也就是梯度消失的问题。

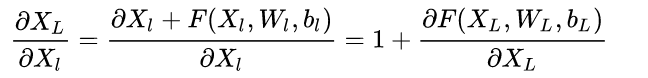
# 3. DRN怎样解决了梯度消失问题？

所以这个时候，DRN就出现了，它在神经网络结构的层面解决了这个问题

它将基本的单元改成了这个样子



其实也很明显，通过求偏导我们就能看到

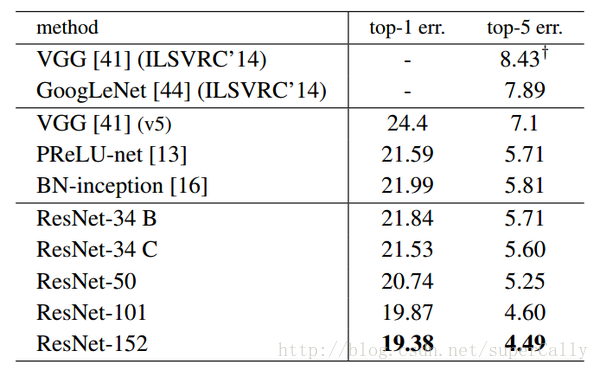


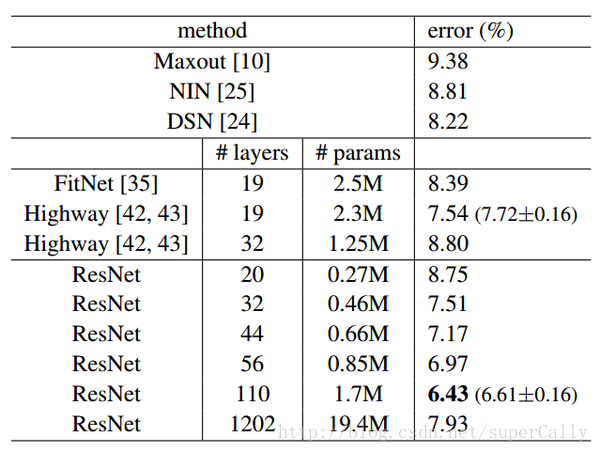
这样就算深度很深，梯度也不会消失了。

也就是说，每一层都会是(1+)的形式，那么梯度当然就不会消失。

# 4. 效果详细说明

zhe yang





# 5. 网友看法

1.

论文中说在深度网络里已经使用了batch normalization，不存在梯度消失的问题。他们猜测深度网络是收敛慢导致结果不好。  
如果导数是这样的话，我们直接给梯度加个一不就解决了？

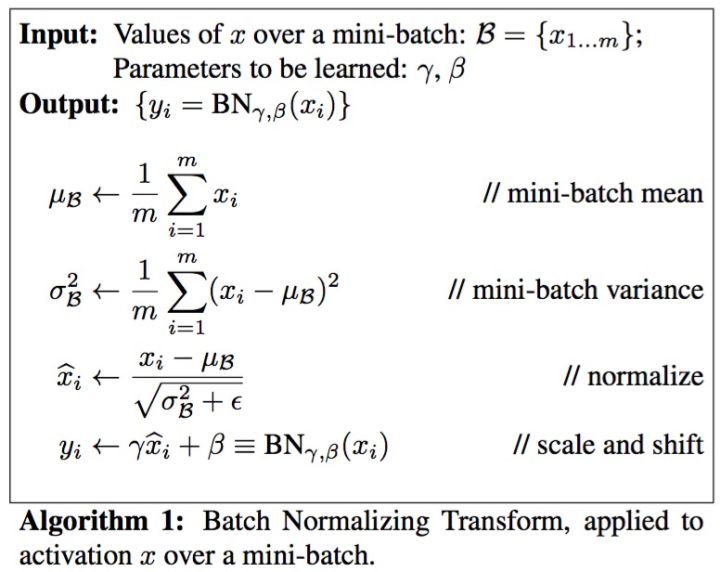
2.

nips16有一篇文章对残余有一个解释，其中也提到了残余网络并没有解决剃度消失的问题。

# 6. Batch Normalization (BN)【洗脸】

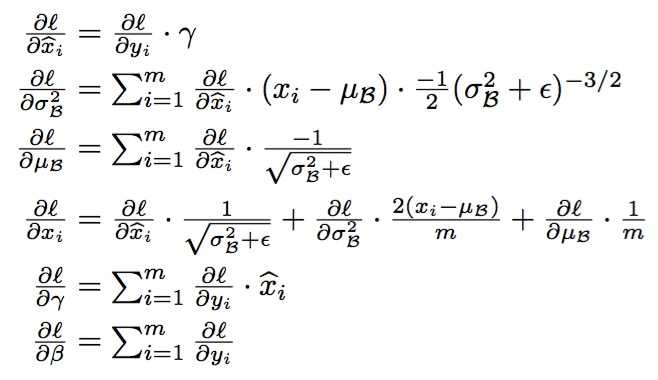
**1. What** is BN?

顾名思义，batch normalization嘛，就是“批量标准化”咯。Google在ICML文中描述的非常清晰，即在每次SGD时，通过mini-batch来对相应的activation做规范化操作，使得结果（输出信号各个维度）的均值为0，方差为1. 而最后的“scale and shift”操作则是为了让因训练所需而“刻意”加入的BN能够有可能还原最初的输入（即当），从而保证整个network的capacity。（有关capacity的解释：实际上BN可以看作是在原模型上加入的“新操作”，这个新操作很大可能会改变某层原来的输入。当然也可能不改变，不改变的时候就是“还原原来输入”。如此一来，既可以改变同时也可以保持原输入，那么模型的容纳能力（capacity）就提升了。）



关于DNN中的normalization，大家都知道白化（whitening），只是在模型训练过程中进行白化操作会带来过高的计算代价和运算时间。因此本文提出两种简化方式：1）直接对输入信号的每个维度做规范化（“normalize each scalar feature independently”）；2）在每个mini-batch中计算得到mini-batch mean和variance（均差和方差）来替代整体训练集的mean和variance. 这便是Algorithm 1.【靠，这个我是刚刚才想到的，在核的输出之间加正则归一化，进行分类的结构，还只是一点想法，没想到人家早想到而且还做好了。】

**2. How** to Batch Normalize?  
怎样学BN的参数在此就不赘述了，就是经典的chain rule：



**3. Where** to use BN?  
BN可以应用于网络中任意的activation set。文中还特别指出在CNN中，BN应作用在非线性映射前，即对做规范化。另外对CNN的“权值共享”策略，BN还有其对应的做法（详见文中3.2节）。

**4. Why** BN?  
好了，现在才是重头戏－－为什么要用BN？BN work的原因是什么？  
说到底，BN的提出还是为了克服深度神经网络难以训练的弊病。其实BN背后的insight非常简单，只是在文章中被Google复杂化了。  
首先来说说“Internal Covariate Shift”。文章的title除了BN这样一个关键词，还有一个便是“ICS”。大家都知道在统计机器学习中的一个经典假设是“源空间（source domain）和目标空间（target domain）的数据分布（distribution）是一致的”。如果不一致，那么就出现了新的机器学习问题，如，transfer learning/domain adaptation(迁移学习/域适应)等。而covariate shift就是分布不一致假设之下的一个分支问题，它是指源空间和目标空间的条件概率是一致的，但是其边缘概率不同，即：对所有，但是。大家细想便会发现，的确，对于神经网络的各层输出，由于它们经过了层内操作作用，其分布显然与各层对应的输入信号分布不同，而且差异会随着网络深度增大而增大，可是它们所能“指示”的样本标记（label）仍然是不变的，这便符合了covariate shift的定义。由于是对层间信号的分析，也即是“internal”的来由。

【总结一下：好像是这个意思，DRN中的每一个RN结构，可以看作是独立的，而且它们都有输入分量x，而且所有RN其实又都有一个共同不变的标签输出。那么，就可以将这些RN结构看作是一个迁移学习(covariate shift)问题，即同一网络对不同训练数据的训练效果发生了偏差——引申到DRN上的意思就是，所有网络应表现出相似的网络性能，这个相似就是x，a的类型的概率分步一致。】

那么好，为什么前面我说Google将其复杂化了。其实如果严格按照解决covariate shift的路子来做的话，大概就是上“importance weight”（[ref](https://link.zhihu.com/?target=http%3A//120.52.72.36/www.jmlr.org/c3pr90ntcsf0/papers/volume8/sugiyama07a/sugiyama07a.pdf)）之类的机器学习方法。可是这里Google仅仅说“通过mini-batch来规范化某些层/所有层的输入，从而可以固定每层输入信号的均值与方差”就可以解决问题。如果covariate shift可以用这么简单的方法解决，那前人对其的研究也真真是白做了。此外，试想，均值方差一致的分布就是同样的分布吗？当然不是。显然，ICS只是这个问题的“包装纸”嘛，仅仅是一种high-level demonstration。  
那BN到底是什么原理呢？说到底还是**为了防止“梯度弥散”**。关于梯度弥散，大家都知道一个简单的栗子：。在BN中，是通过将activation规范为均值和方差一致的手段使得原本会减小的activation的scale变大。可以说是一种更有效的local response normalization方法（见4.2.1节）。

**5. When** to use BN?  
OK，说完BN的优势，自然可以知道什么时候用BN比较好。例如，在神经网络训练时遇到收敛速度很慢，或梯度爆炸等无法训练的状况时可以尝试BN来解决。另外，在一般使用情况下也可以加入BN来加快训练速度，提高模型精度。

诚然，在DL中还有许多除BN之外的“小trick”。**别看是“小trick”，实则是“大杀器”**，正所谓“*The devil is in the details*”。希望了解其它DL trick（特别是CNN）的各位请移步我之前总结的：[**Must Know Tips/Tricks in Deep Neural Networks**](https://link.zhihu.com/?target=http%3A//lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html)   
以上。

# 7. BN的本质

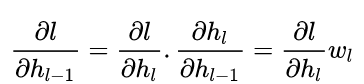
**BN本质上解决的是反向传播过程中的梯度问题。**

详细点说，反向传播时经过该层的梯度是要乘以该层的参数的，即前向有：



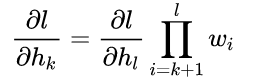
【**对的，相当于，前一层的输出，就是后一层的输入**】

那么反向传播时便有：



【**正确**】

那么考虑从*l*层传到k层的情况，有：



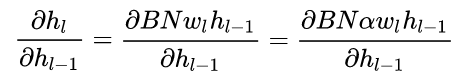
【**叠乘，正确**】

上面这个 便是问题所在。因为网络层很深，如果 大多小于1，那么传到这里的时候梯度会变得很小比如 ；而如果 又大多大于1，那么传到这里的时候又会有梯度爆炸问题 比如 。BN所做的就是解决这个梯度传播的问题，因为**BN作用抹去了w的scale影响**。

具体有：

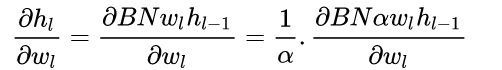


那么反向求导时便有了：



【**的结果本来就是增值为0，方差为1的，而与无关，所以他说传播系数与的尺度无关了，这是对的**】

可以看到此时反向传播乘以的数不再和 的尺度相关，也就是说尽管我们在更新过程中改变了 的值，但是反向传播的梯度却不受影响。更进一步：



即尺度较大的 将获得一个较小的梯度，在同等的学习速率下其获得的更新更少，这样使得整体 的更新更加稳健起来。【这一句没看懂】

**总结起来就是BN解决了反向传播过程中的梯度问题（梯度消失和爆炸），同时使得不同scale的** **整体更新步调更一致。**

更详细的解释可以看我写的一篇BN的文章[Batch Normalization详解](https://blog.csdn.net/qq_34484472/article/details/77982224?locationNum=2&fps=1)

# 8. covariate shift

## 8.1什么是covariate shift？

在论文中经常碰到covariate shift这个词，网上相关的中文解释比较少。你可能会在介绍深度学习Batch Normalization方法的论文到中看到covariate shift这个词，并且所有看过这篇论文的或者实际做过工程的人，都知道BN这种归一化手段很好用，可以避免covariate shift。

**那么covariate shift到底是什么？**

不用想得太复杂，covariate shift最早只是机器学习中的一个问题。同时迁移学习会经常涉及到这个概念。

假设x是属于特征空间的某一样本点，y是标签。covariate这个词，其实就是指这里的x，那么covariate shift可以直接根据字面意思去理解：样本点x的变化。  
这么简单？没错就是这么简单！

我们讲的规范一点：

假设q1（x）是测试集中一个样本点的概率密度，q0（x）是训练集中一个样本点的概率密度。最终我们估计一个条件概率密度p(y|x，θ)，它由x和一组参数θ=｛θ1，θ2......θm｝所决定。对于一组参数来说，对应loss(θ)函数评估性能的好坏。

综上，当我们找出在q0（x）分布上最优的一组θ'时，能否保证q1（x）上测试时也最好呢？

传统机器学习假设训练集和测试集是独立同分布的，即q0（x）=q1（x），所以可以推出最优θ'依然可以保证q1（x）最优。但现实当中这个假设往往不成立，伴随新数据产生，老数据会过时，当q0（x）不再等于q1（x）时，就被称作covariate shift

## 8.2怎么解决covariate shift？

以上已经知道一个样本点分别在训练集和测试集上的概率密度q0（x）和q1（x），实际当中的解决方案是附加一个由x决定的权值

http://img.blog.csdn.net/20170110213604670

使得在训练过程当中对于q1（x）很大或者q0（x）很小的样本视作“重要”样本，这样的样本是有益于测试集预测的，我们应该尽量把它分类正确。而对于q1（x）很小或者q0（x）很大的样本，它只是被时代遗弃的“老数据”，这些样本对于模型训练的意义也是无关紧要了

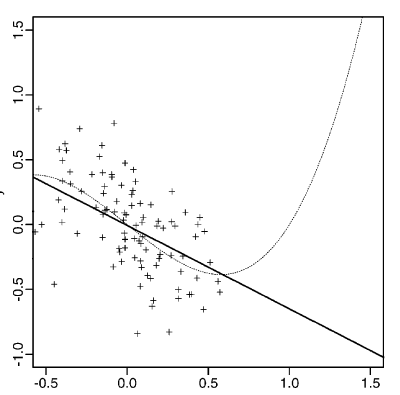
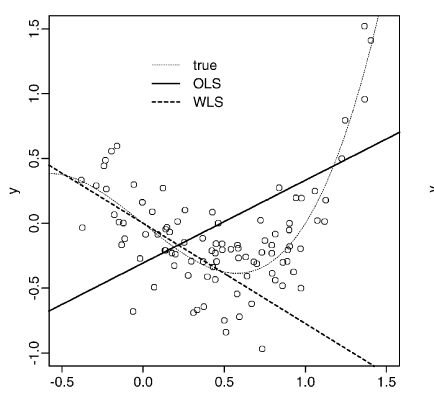
**举一个形象的例子：**

现在我们要通过多项式回归预测某一个函数。数据产生通过下式，并且用正态分布产生噪声加在上面

http://img.blog.csdn.net/20170110211111682

通过一个http://img.blog.csdn.net/20170110211221214其中http://img.blog.csdn.net/20170110211224105产生q0（x）当作训练集，大小为n=100

假设模型的形式是http://img.blog.csdn.net/20170110211638443，最终训练得到一根直线去尽可能拟合这些点。

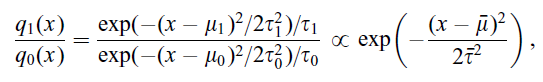


实验结果如左图，最终得到一条线，对应图中OLS

根据http://img.blog.csdn.net/20170110212537796其中http://img.blog.csdn.net/20170110212540525生成测试集q1（x），如右图

最理想的情况，我们直接拟合测试集的点，得到右图中的实线，和左边的线完全不一样，看出covariate shift发生了。

但是我们需要测试集的辅助来解决covariate shift问题（如果直接训练测试集就毫无意义了），求得



通过附加权值的方法，最终训练得到左图中虚线WLS

可以看出最终得到的模型，是可以很好的适应测试集的。

从迁移学习的角度看，这也是一种用source domain的标签数据，结合target domain的无标签数据，指导进行知识的迁移的方法。