1. sigmoid \tanh\telu

sigmoid = 1/1+e(-x)

缺点：容易出现过拟合，不适合训练很深的网络中，最大导数为0.25

函数输出不是zero-centered

幂运算相对来讲比较耗时

函数输出不是zero-centered：Sigmoid函数的输出值恒大于0，这会导致模型训练的收敛速度变慢。举例来讲，对，如果所有均为正数或负数，那么其对的导数总是正数或负数，这会导致如下图红色箭头所示的阶梯式更新，这显然并非一个好的优化路径。深度学习往往需要大量时间来处理大量数据，模型的收敛速度是尤为重要的。所以，总体上来讲，训练深度学习网络尽量使用zero-centered数据 (可以经过数据预处理实现) 和zero-centered输出

Tanh = e(x)-e(-x)/e(x)+e(-x)

最大导数为1，解决了zero-centered的输出问题，经过原点，还是会梯度消失

Relu = x if x >0 else 0 或者 max(0, x)

优点：

解决了gradient vanishing问题 (在正区间)

计算速度非常快，只需要判断输入是否大于0

收敛速度远快于sigmoid和tanh，导数为1（正区间）

缺点：

ReLU的输出不是zero-centered

Dead ReLU Problem，指的是某些神经元可能永远不会被激活，导致相应的参数永远不能被更新。（当神经元的输出为0时，对应的w权重将不会更新）

有两个主要原因可能导致这种情况产生: (1) 非常不幸的参数初始化，这种情况比较少见 (2) learning rate太高导致在训练过程中参数更新太大，不幸使网络进入这种状态。解决方法是可以采用Xavier初始化方法（该方法的作者文章主要的目标就是使得每一层输出的方差应该尽量相等，他认为优秀的初始化应该使得各层的激活值和状态梯度的方差在传播过程中的方差保持一致），以及避免将learning rate设置太大或使用adagrad等自动调节learning rate的算法。

LeakyRelu = max(0.01x, x)

人们为了解决Dead ReLU Problem，提出了将ReLU的前半段设为而非0。另外一种直观的想法是基于参数的方法，即Parametric ReLU:，其中可由back propagation学出来。理论上来讲，Leaky ReLU有ReLU的所有优点，外加不会有Dead ReLU问题，但是在实际操作当中，并没有完全证明Leaky ReLU总是好于ReLU。

1. 常用的优化算法

Stochastic Gradient Descent (type: "SGD"),

SGD 虽然能达到极小值，但是比其它算法用的时间长，而且可能会被困在鞍点。

三种Sgd比较：

随机梯度下降：SGD 因为更新比较频繁，会造成 cost function 有严重的震荡；缺点是SGD的噪音较BGD要多，使得SGD并不是每次迭代都向着整体最优化方向。所以虽然训练速度快，但是准确度下降，并不是全局最优。

BatchSGD：介于两者之间，收敛更稳定。缺点是学习率不好调。

AllSGD: 由于这种方法是在一次更新中，就对整个数据集计算梯度，所以计算起来非常慢，遇到很大量的数据集也会非常棘手，而且不能投入新数据实时更新模型. 对于凸函数可以收敛到全局极小值，对于非凸函数可以收敛到局部极小值。

随机梯度下降，一般指minibatchSGD

AdaDelta (type: "AdaDelta"),

自适应梯度，如果数据是稀疏的，就用自适用方法，即 Adagrad, Adadelta, RMSprop, Adam。

Adaptive Gradient (type: "AdaGrad"),

Adam (type: "Adam"),

Nesterov’s Accelerated Gradient (type: "Nesterov") and

RMSprop (type: "RMSProp")

1. inception之间的升级<https://www.cnblogs.com/bonelee/p/8977912.html>

v1->v4:网络深度不断加深

v1：

增加了网络的宽度，基础结构包括1\*1 3\*3 5\*5

使用了1\*1降维

使用了三个辅助分类

采用了global avg pool代替全连接，保持了空间信息，降低参数

V2:

采用了BN（什么是bn）

使用了两个3\*3代替5\*5，好处是降低参数，增加网络非线性

V3:

使用了卷积分解1\*3和3\*1代替3\*3，好处这样的好处，既可以加速计算

增加网络宽度，网络输入从224\*224变为了299\*299

V4:

结合了残差网络，使得训练加速收敛更快，精度更高

Xinception:

基本思想就是通道分离式卷积

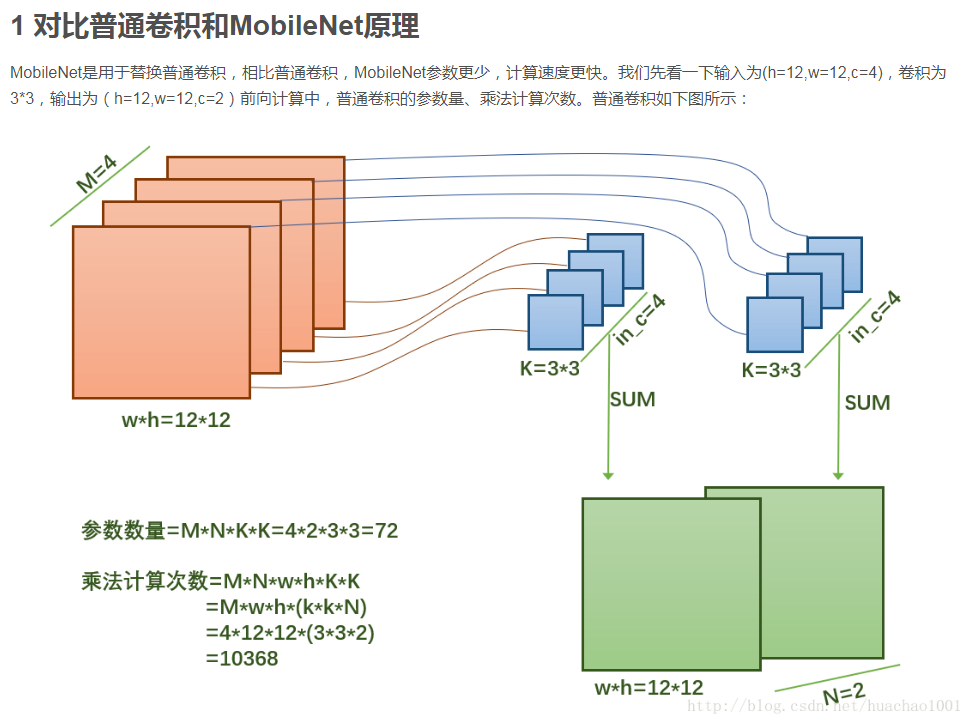
先说，卷积的操作，主要进行2种变换，(1)spatial dimensions，空间变换(2)channel dimension，通道变换

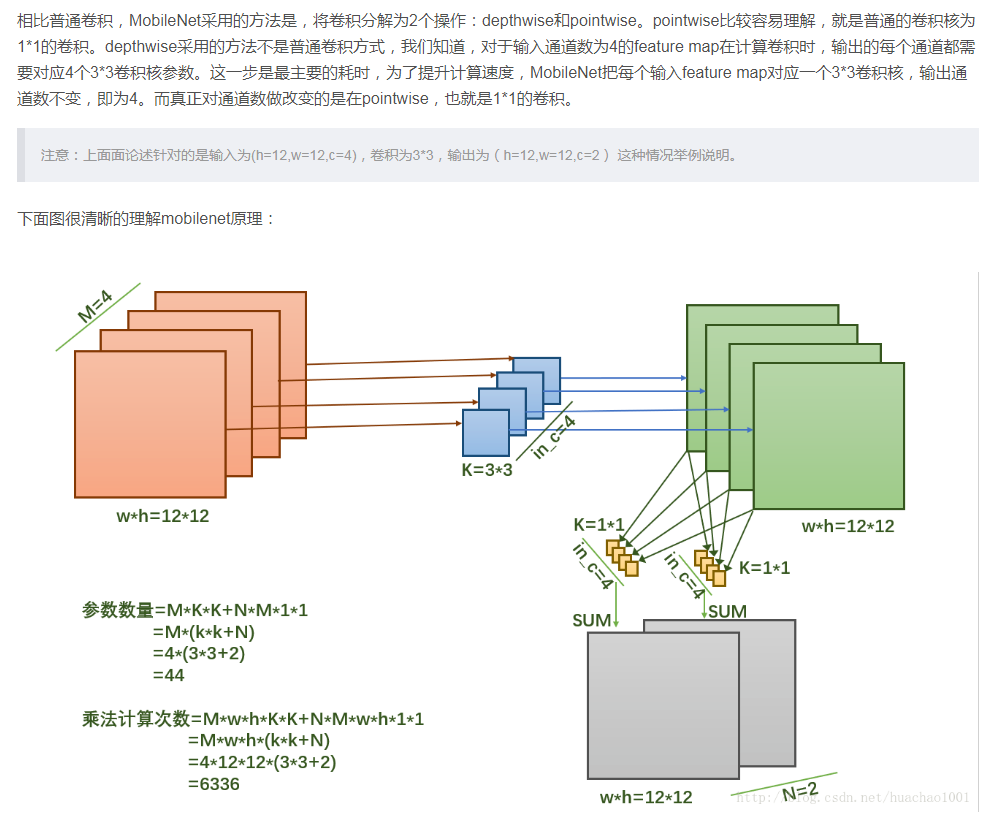
改进一：inception V3是先做1\*1的卷积，再做3\*3的卷积，这样就先将通道进行了合并，即通道卷积，然后再进行空间卷积，而Xception则正好相反，先进行空间的3\*3卷积，再进行通道的1\*1卷积

改进二：Inception V3在每个module中都有RELU操作，而Xception在每个module中是没有RELU操作的

MobileNets其实就是Exception思想的应用。区别就是Exception文章重点在提高精度，而MobileNets重点在压缩模型，同时保证精度。

Mobilenet结构图：



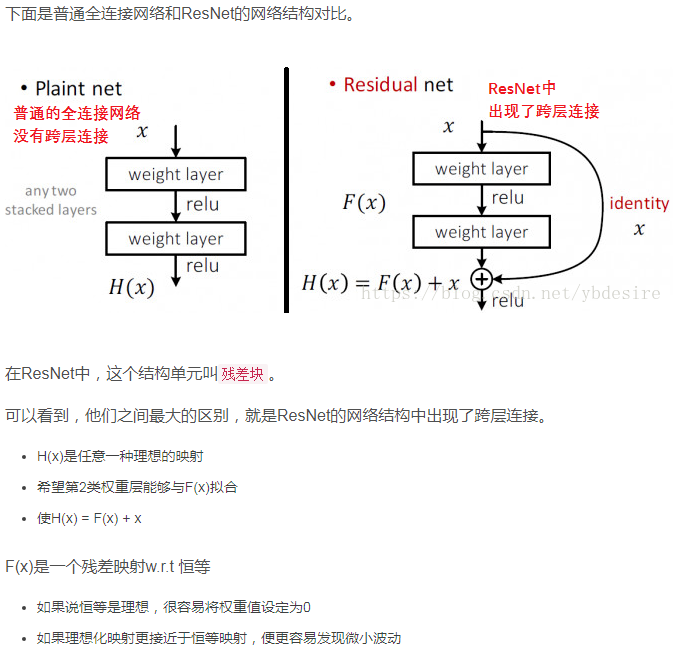


1. Resnet效果好的原因：

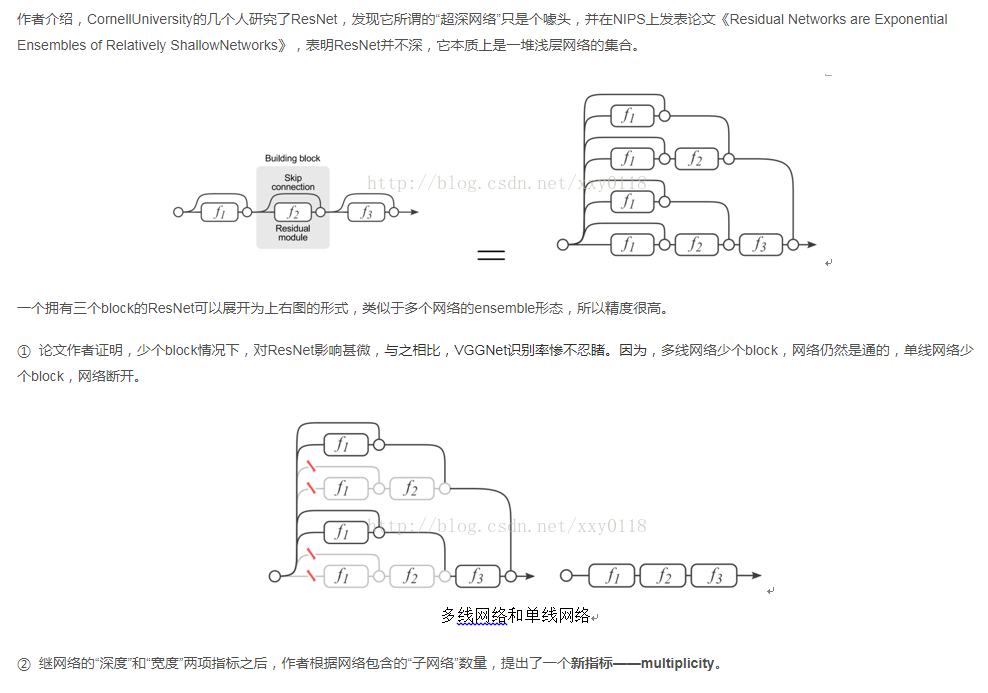
对vgg做实验，重复在加深网络深度，识别效果并没有提升反而会下降一点点，原因：网络越深，容易导致拟合，比较难训练。

Resnet 效果好的原因：

第一种解释：使网络更容易在某些层学到恒等变换（identity mapping）。在某些层执行恒等变换是一种构造性解，使更深的模型的性能至少不低于较浅的模型。这也是作者原始论文指出的动机。H(X) = F(X) + X =>F(X) = H(X) – X



第二种解释：残差网络是很多浅层网络的集成（ensemble），层数的指数级那么多。主要的实验证据是：把 ResNet 中的某些层直接删掉，模型的性能几乎不下降。



第三种解释：残差网络使信息更容易在各层之间流动，包括在前向传播时提供特征重用，在反向传播时缓解梯度信号消失。

1. dropout解读

<https://blog.csdn.net/program_developer/article/details/80737724>

为什么要dropout？

在机器学习中，如果模型参数太多，训练数据太少，训练出来的模型容易导致过拟合现象。具体表现就是：模型在训练数据集上的loss较小，预测准确率较高；但是在测试数据集上loss较大，准确率较低。

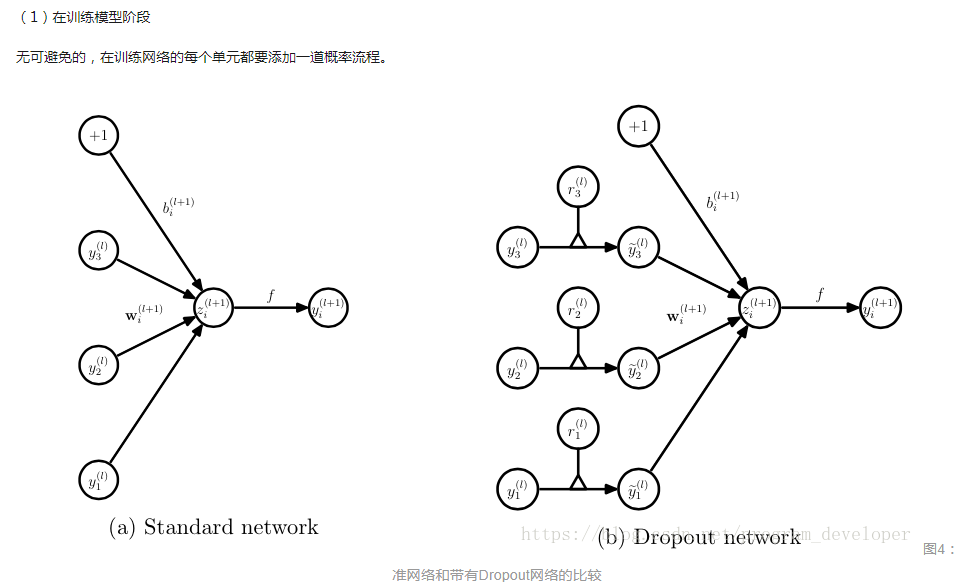
为了解决过拟合问题，一般会采用模型集成的方法，即训练多个模型进行组合，此时，训练多个模型很费时，测试多个模型也很费时。所以训练深度神经网络遇到的两个问题：

* 过拟合
* 费时

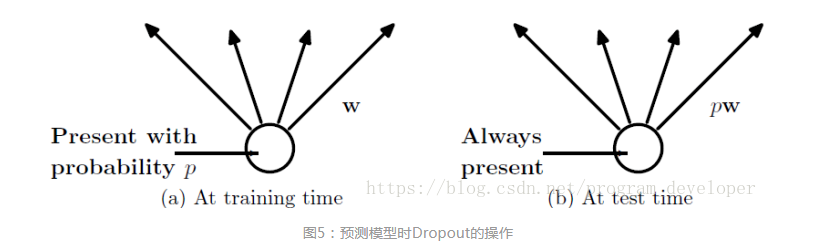
为了缓解这个问题，引入dropout。

Dropout怎么做？

在训练的时候，在吗，每次迭代的过程中，以一定概率使神经元休眠，即设置为0



在测试模型的时候，所有神经元的权重参数都乘以概率p



为什么dropout可以解决过拟合？

从bagging的角度：去平均的作用

假设模型没有用dropout，训练5个不同的神经网络，那就会得到5个输出结果，此时我们可以采用多数取胜或者取平均的策略决定最终的预测结果。比如三个模型判定为数字9，那最终的结果就是9.这种策略可以防止过拟合问题，因为不同的模型产生不同的过拟合，取平均会使得一些相反的拟合相互抵消。Dropout训练的时候随机的休眠一些神经元就是相当于在训练不同的网络，整个dropout过程相当于对很多不同的网络取平均，不同的网络产生的那些反向过拟合会一定的抵消，从而整体上使得网络减少过拟合。

从正则化的角度：减少神经元之间复杂的共适应关系

Dropout导致两个神经元不一定都在同一个dropout网络中出现，这样权值的更新不在依赖固定的关系的隐含节点的共同作用，阻止了某些特征仅仅在其他特定的特征下才有的效果的情况，这就使得网络去学习更加鲁棒性的特征。换句话说，网络在作出某种预测，它不应该对特定的线索片段太过敏感，即使丢掉一些特征，它也能从其他众多的特征中学习到共同的特征。从这个角度看dropout就有点像L1，L2正则，减少权重使得网络对丢失特定神经元连接的鲁棒性提高。

1. batch normlization解读：《Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift》

https://www.cnblogs.com/guoyaohua/p/8724433.html

<https://blog.csdn.net/hjimce/article/details/50866313>

机器学习领域有个很重要的假设：

**IID独立同分布假设**，就是假设训练数据和测试数据是满足相同分布的，这是通过训练数据获得的模型能够在测试集获得好的效果的一个基本保障。

Internal Covariate Shift的解释：

**在训练过程中，隐层的输入分布老是变来变去，这就是所谓的“Internal Covariate Shift”，Internal指的是深层网络的隐层，是发生在网络内部的事情，而不是covariate shift问题只发生在输入层。BatchNorm就是在深度神经网络训练过程中使得每一层神经网络的输入保持相同分布的。**

BN引入：

　然后提出了BatchNorm的基本思想：能不能**让每个隐层节点的激活输入分布固定下来呢**？这样就避免了“Internal Covariate Shift”问题了。

BN不是凭空拍脑袋拍出来的好点子，它是有启发来源的：之前的研究表明如果在图像处理中对输入图像进行白化（Whiten）操作的话——所谓**白化**，**就是对输入数据分布变换到0均值，单位方差的正态分布**——那么神经网络会较快收敛，那么BN作者就开始推论了：图像是深度神经网络的输入层，做白化能加快收敛，那么其实对于深度网络来说，其中某个隐层的神经元是下一层的输入，意思是其实深度神经网络的每一个隐层都是输入层，不过是相对下一层来说而已，那么能不能对每个隐层都做白化呢？这就是启发BN产生的原初想法，而BN也确实就是这么做的，**可以理解为对深层神经网络每个隐层神经元的激活值做简化版本的白化操作。**

BN的基本思想:

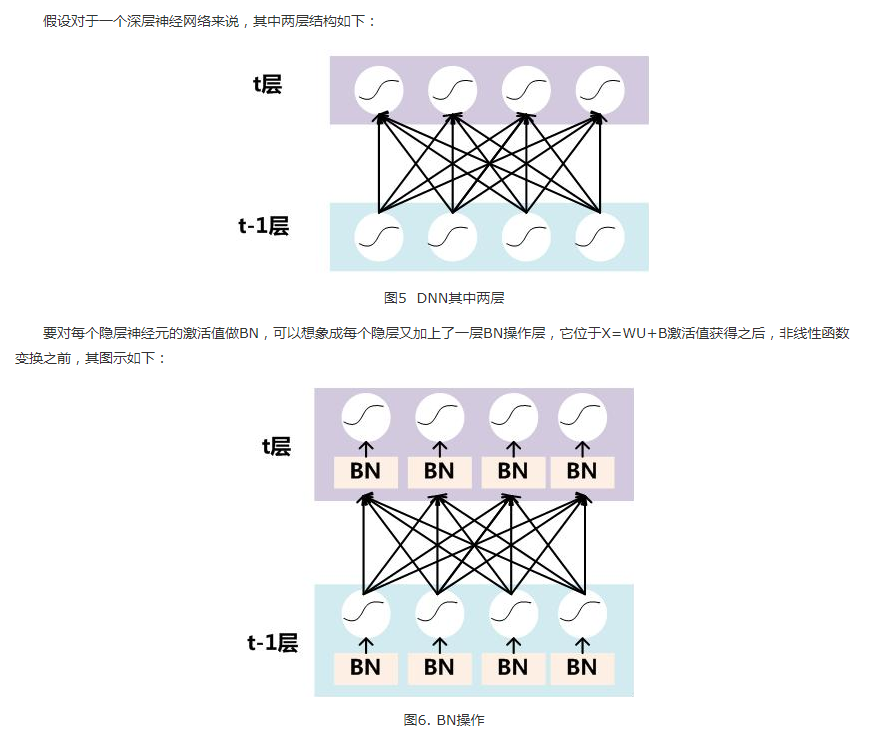
BN的基本思想其实相当直观：因为深层神经网络在做非线性变换前的**激活输入值**（就是那个x=WU+B，U是输入）**随着网络深度加深或者在训练过程中，其分布逐渐发生偏移或者变动，之所以训练收敛慢，一般是整体分布逐渐往非线性函数的取值区间的上下限两端靠近**（对于Sigmoid函数来说，意味着激活输入值WU+B是大的负值或正值），所以这**导致反向传播时低层神经网络的梯度消失**，这是训练深层神经网络收敛越来越慢的**本质原因**，**而BN就是通过一定的规范化手段，把每层神经网络任意神经元这个输入值的分布强行拉回到均值为0方差为1的标准正态分布**，其实就是把越来越偏的分布强制拉回比较标准的分布，这样使得激活输入值落在非线性函数对输入比较敏感的区域，这样输入的小变化就会导致损失函数较大的变化，意思是**这样让梯度变大，避免梯度消失问题产生，而且梯度变大意味着学习收敛速度快，能大大加快训练速度。**

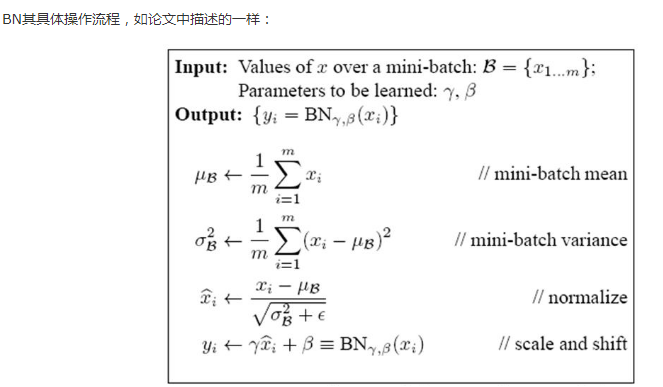
**到这里会产生一个问题？强行变换会导致网络的非线性下降，也就意味着网络的表达能力下降了？**

**解决办法：**

BN为了保证非线性的获得，对变换后的满足均值为0方差为1的x又进行了scale加上shift操作(y=scale\*x+shift)，每个神经元增加了两个参数scale和shift参数，这两个参数是通过训练学习到的，意思是通过scale和shift把这个值从标准正态分布左移或者右移一点并长胖一点或者变瘦一点，每个实例挪动的程度不一样，这样等价于非线性函数的值从正中心周围的线性区往非线性区动了动。核心思想应该是想找到一个线性和非线性的较好平衡点，既能享受非线性的较强表达能力的好处，又避免太靠非线性区两头使得网络收敛速度太慢。

BN的具体训练流程：





第一步，求出某个batch中的均值和方差，然后归一化每个输出，变换后某个神经元的激活x形成了均值为0，方差为1的正态分布；

第二部，第一步实现了把值往后续要进行的非线性变换的线性区拉动，增大导数值，增强反向传播信息流动性，加快训练收敛速度。但是这样会导致网络表达能力下降，为了防止这一点，每个神经元增加两个调节参数（scale和shift），这两个参数是通过训练来学习到的，用来对变换后的激活反变换，使得网络表达能力增强，即对变换后的激活进行如下的scale和shift操作，这其实是变换的反操作，上图公式中最后一步。

源码中的实现：

过上面的学习，我们知道BN层是对于每个神经元做归一化处理，甚至只需要对某一个神经元进行归一化，而不是对一整层网络的神经元进行归一化。既然BN是对单个神经元的运算，那么在CNN中卷积层上要怎么搞？假如某一层卷积层有6个特征图，每个特征图的大小是100\*100，这样就相当于这一层网络有6\*100\*100个神经元，如果采用BN，就会有6\*100\*100个参数γ、β，这样岂不是太恐怖了。因此卷积层上的BN使用，其实也是使用了类似权值共享的策略，把一整张特征图当做一个神经元进行处理。

卷积神经网络经过卷积后得到的是一系列的特征图，如果min-batch sizes为m，那么网络某一层输入数据可以表示为四维矩阵(m,f,p,q)，m为min-batch sizes，f为特征图个数，p、q分别为特征图的宽高。在cnn中我们可以把每个特征图看成是一个特征处理（一个神经元），因此在使用Batch Normalization，mini-batch size 的大小就是：m\*p\*q，于是对于每个特征图都只有一对可学习参数：γ、β。说白了吧，这就是相当于求取所有样本所对应的一个特征图的所有神经元的平均值、方差，然后对这个特征图神经元做归一化。

BN的单个图片前向传播过程：

单张图片没有minibatch的定义，所以归一化均值和方差使用训练过程中的所有batch的平均值，r和b则是训练的好的参数，其训练方法和权值w\b训练方法一样。

BN的好处：

。①不仅仅极大提升了训练速度，收敛过程大大加快；②还能增加分类效果，一种解释是这是类似于Dropout的一种防止过拟合的正则化表达方式，所以不用Dropout也能达到相当的效果；③另外调参过程也简单多了，对于初始化要求没那么高，而且可以使用大的学习率等。