课时6，激活函与批量归一化

Part 1： activation functions

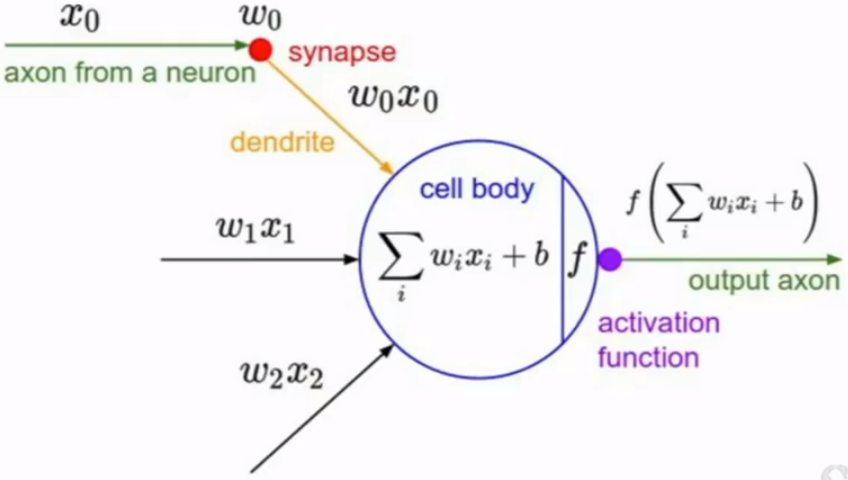


图6.1

在全连接层或者卷积层，我们将输入的数据乘上权重值，然后降结果输入激活函数或者非线性单元，如图6.1。

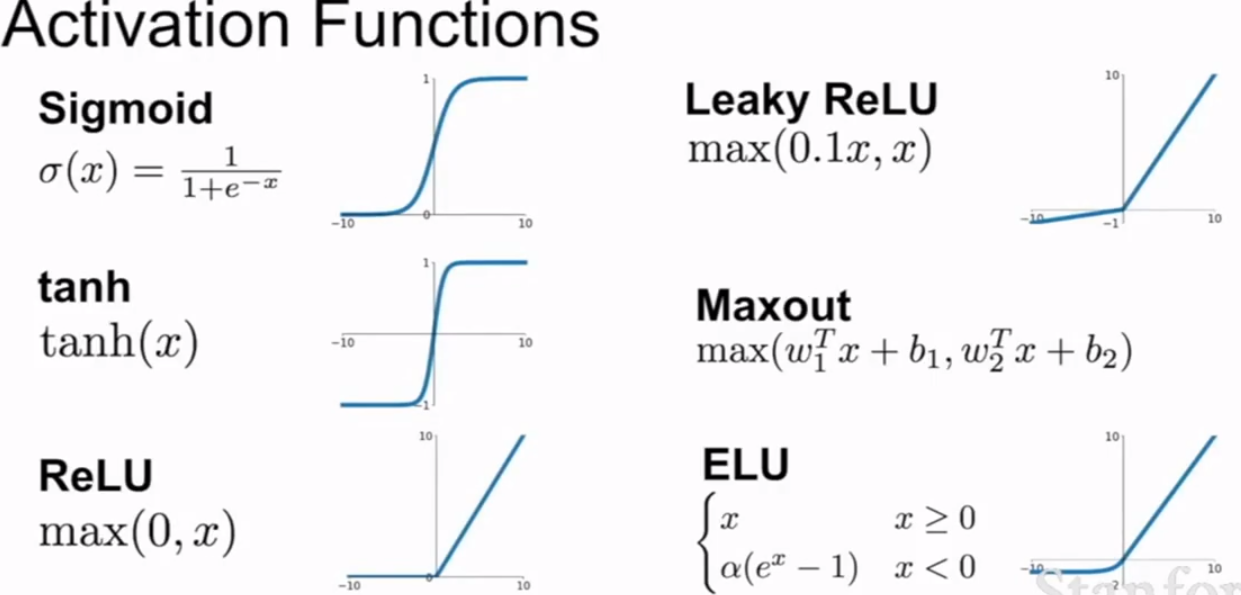


图6.2

图6.3

常用的激活函数如图6.2所示。首先介绍sigmoid函数，如图6.3，表达式如图中，每一个元素输入到sigmoid非线性函数中，输出被压缩到[0,1]范围内，如果有一个非常大的正输入值，那么输出将会十分接近于1，如果有一个绝对值很大的负数输出进去，输出将会十分接近0。在横坐标接近于0的区域中，我们可以将这部分看作是线性区域，他有点像线性函数。Sigmoid在早期比较流行，可以看成是神经元的饱和放电率，值介于0~1之间，然而稍后介绍的ReLU函数在生物学上更加的合理。但是如果我们深入研究sigmoid函数，它存在若干个问题。**首先是饱和神经元将使得梯度消失**，如图6.4所示。

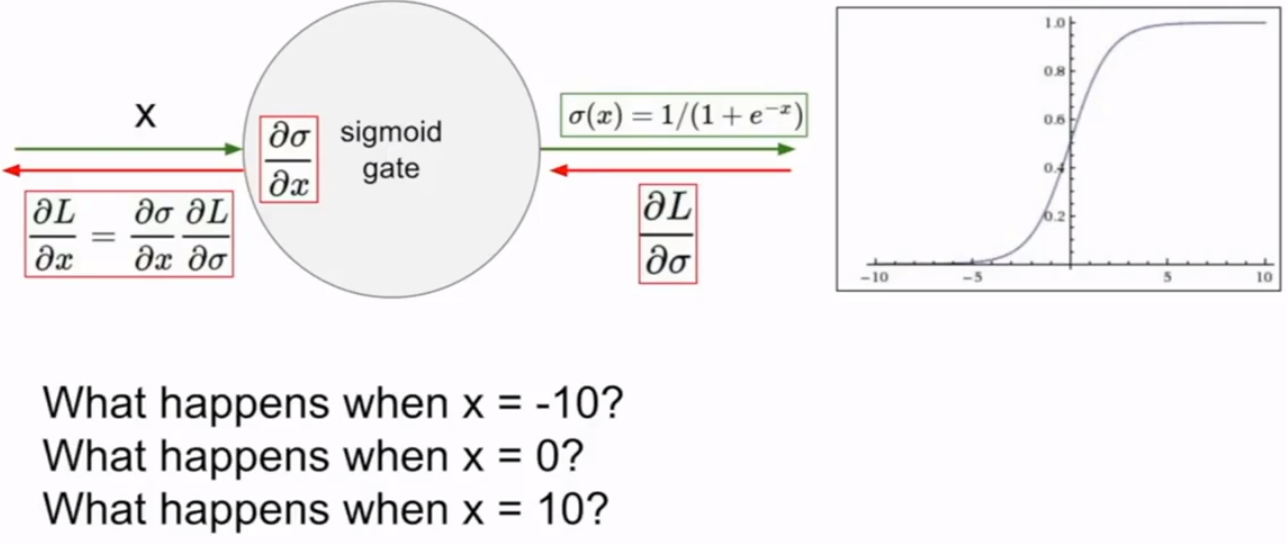


图6.4

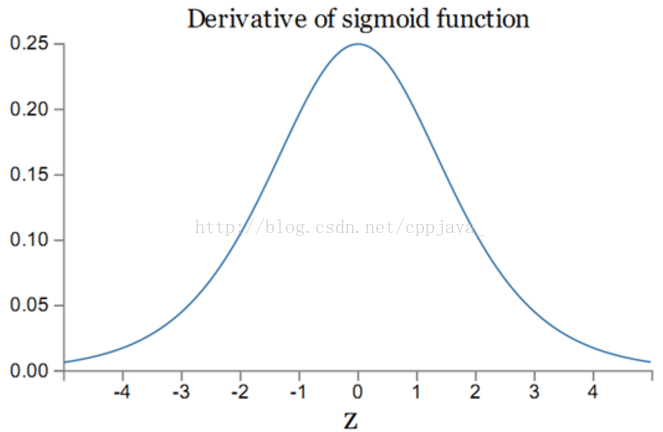


图6.5

我们将X作为输入，然后从sigmoid输出，那它返回的梯度流是怎样的呢？我们让L对求偏导，上游梯度往回传播，然后乘上对x的偏导，这就是一个局部sigmoid函数的梯度，我们把它链接起来作为传递回来的下游梯度。当x= -10时，此时的梯度为0，因为这个负值太过于接近sigmoid的负饱和区域了，这个区域是平的所以梯度变为0，我们将返回的上游梯度乘上一个约等于0的数，所以会得到一个非常小的梯度。某种意义上就是说经过链式法则后会让梯度消失，因此零梯度就会传递到下游的节点。当输入x=0时，我们会得到一个合理的梯度，并能很好的进行反向传播。当x=10时，梯度还是等于0。所以输入是一个很大的正值或者一个很大的负值时，他们会接近sigmoid函数的平滑区域，这些区域会使得梯度消失。Sigmoid函数的导数如图6.5.

第二，sigmoid是一个非零中心的函数，这会导致什么呢？考虑一下输入神经元的数值始终为正的时候会发生什么？输入X全为正，它会乘某个权重W，然后通过激活函数sigmoid，那权重W的梯度将会怎样呢？考虑一下这个线性层的局部梯度是多少？我们用L对任何激活函数求偏导，然后我们的局部梯度就是X本身，所有的x都是正的意味着梯度值要么全正要么全负。我们对上游的梯度即对损失函数L关于f求导时，它的值为正或者为负，用它乘以全为正或者全为负的局部梯度X，如果X的值总是正数，那么对W的梯度就等于将dL/df乘以df/dw，这就相当于把上游梯度的符号传了回来，意味着所有关于W的梯度总是朝着同一个方向移动，当你在做参数更新的时候你可以用同一个正数去增加所有w的值，或者用不同的正数去增加，但是这里出现的问题就是这种方法对于梯度更新来说十分的低效，如图6.6右边部分。

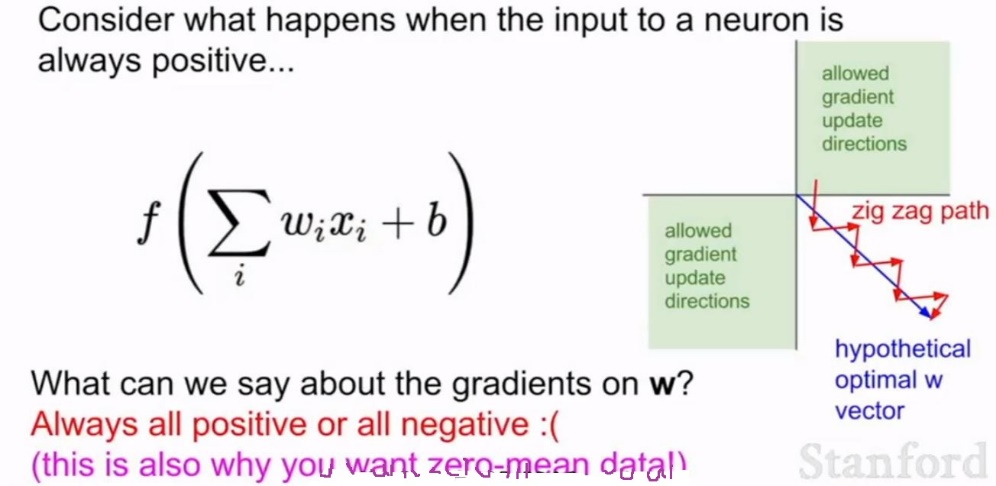


图6.6

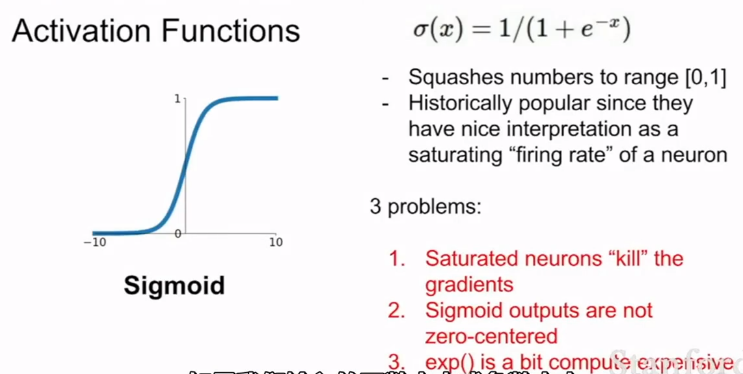


图6.7

我们假设w是二维的，所以可以用平面坐标来表示w，如果我们用全为正或者全为负的数去迭代，能够得到两个象限，即在坐标轴上有两块区域，一块全为正（一象限）一块全为负（三象限），如图中的青色部分。这些是根据梯度更新得到的两个方向，在这种情况下假设最佳的w实际上是图中的蓝色向量，我们从某些已知的点开始，或者从红色箭头的起始端的顶部开始，我们不能沿着w这个最佳方向直接求梯度，因为这个方向并不是允许的两个梯度方向中的一个。所以我们只能这么做：在允许的方向上进行一些列的梯度更新，例如沿着图中这些红色箭头的方向，他们是我们允许进行梯度更新的方向，目标是最终得到最佳的w。这就是为什么一般情况下我们使用均值为0的数据，我们想要我们的输入x的均值为0，这样就能得到正的和负的数值，就不会陷入前面的梯度更新中出现的问题，他们将会沿着同一个方向移动。

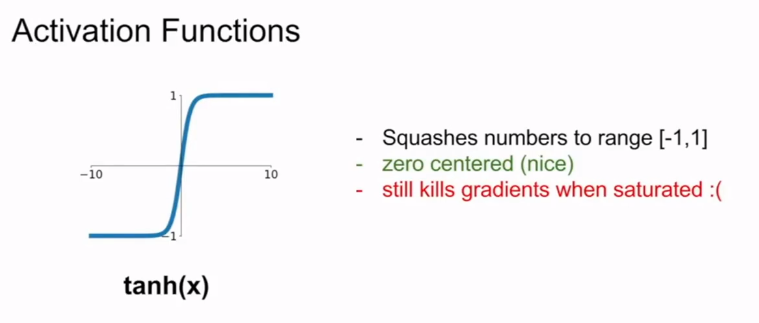


图6.8

现在来看第二个激活函数，tanh。和sigmoid不同之处在于他被压缩到[-1,1]之间，还有就是tanh是以0为中心的，所以不会有sigmoid函数的第二个问题，但是他饱和的时候还是会出现梯度消失问题，在饱和的时候我们看到函数的梯度是扁平的，接近于0，这将会组织梯度的传递。如图6.8。

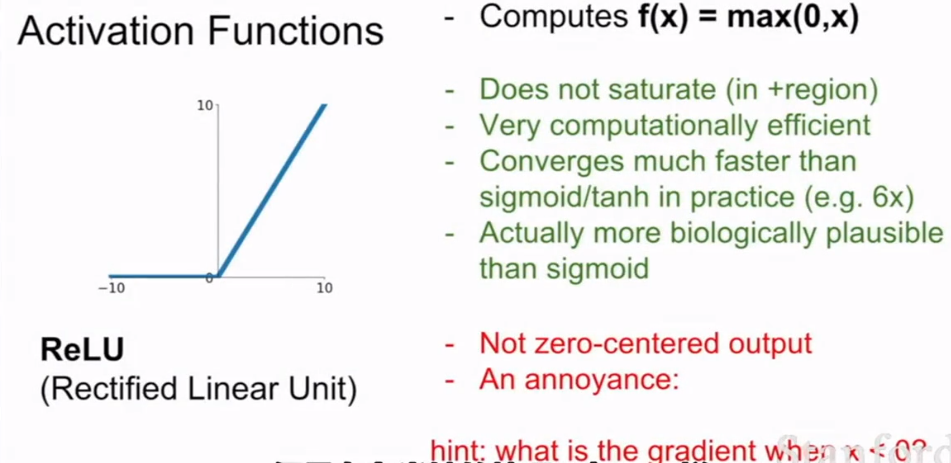


图6.9

现在我们来看ReLU激活函数，如图6.9，ReLU的函数形式为*f*(x)=max(0,x)，根本上来讲，他是在按元素进行操作的，如果输入是负数，那么输出将为0，输入是正数，输出就是原来的数。比较sigmoid与tanh，ReLU不会在正的区域产生饱和现象，因此在我们输入空间的一半上都不会存在饱和，这是一个很大的优势，他的计算成本也不高，比其他的低，因此ReLU在计算上会比较快，实际运用中也比sigmoid和tanh收敛快的多，大约快6倍。但是ReLU不再以0为中心了，同时虽然在正半轴不会饱和，但是负半轴却不一样了。

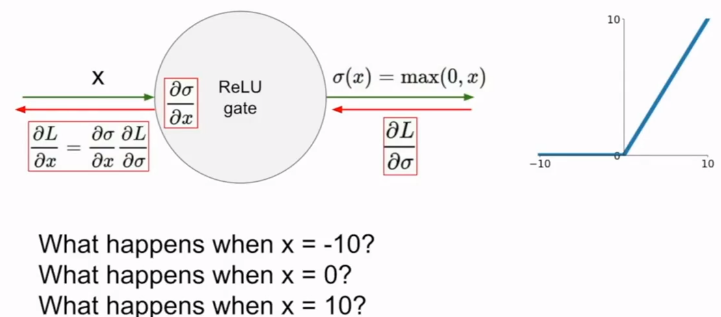


图6.10

当x=-10时，ReLU的梯度为0，当x=10时不会发生问题，因为在线性区域内，当x=0时，梯度不确定，可以取x=0时的梯度为0，因此在一半的区域中会出现梯度消失的问题。

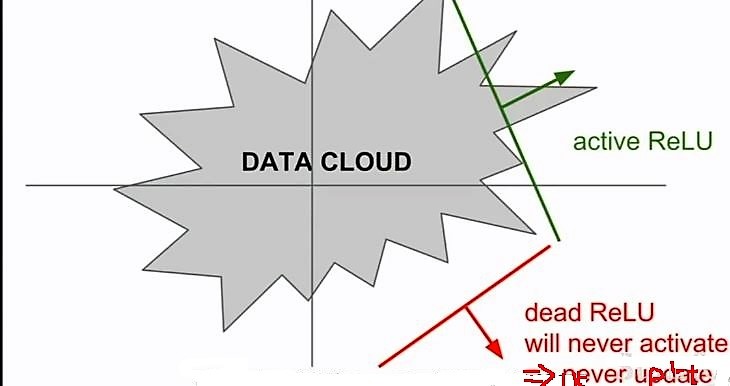


图6.11

我们可以把这种出现梯度消失的现象叫做dead ReLU，这里主要有几个可能的原因，如果观察我们的数据云，来看这些ReLU可能处于的位置，如图6.11。这些ReLU基本上在平面的一半区域能够产生激活，但处于dead ReLU的DATA CLUDE从来不会被激活和更新。主要有以下几个原因，第一是有一个不好的初始化的时候，如果你的权重设置的非常的差，他们恰巧不在数据云里，在这里就会出现dead ReLU的情况，这里应该指的是如果你某个节点设置得不合理，把所有的输入经过线性变换之后都变成了了小于0的数，那么这个节点上的W就没法反向传播更新了，就相当于dead掉了。当你的学习率太高时，在这种情况下你从一个ReLU函数开始，在进行大量更新时权值会不断的波动，然后ReLU单元会被数据的多样性所淘汰，这些会在训练时发生，它在最开始时很很正常，但在某个时间点之后开始变差最后挂掉。在实际中，你冻结一个已经训练好的网络，然后将数据输入进去会发现实际上网络中多达10%到20%的部分是这些挂了的ReLU单元，大多数使用了ReLU的网络都会有这类问题。意思应该是说在网络训练好了之后，某些w会把输入进来的数据变成小于0的数，然后ReLU一筛选就把这个数据晒没了，置为0，相当于没用这个数据一样，dead掉了。在实际运用中，人们喜欢使用较小的正偏置来初始化ReLU，以增加他在初始化时被激活的可能性，并获得一些更新，但实际中有人觉得有用，有人觉得没用，但多数的时候人们只是将偏置初始化为0。

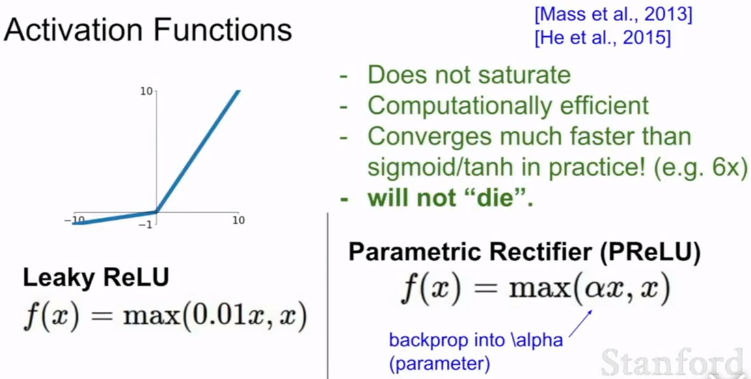


图6.12

接下来看看基于ReLU的一些改进，其中一个例子是Leaky ReLU，如图6.12所示，它和ReLU的唯一区别就是在负区间有一个微小的负斜率，这就解决了前面所说的问题，没有任何的保护机制，即使是在负空间计算仍然是很高效的，比sigmoid和tanh收敛的快，并且没有挂掉的问题。在上图中还有另外一个例子，如图6.12右下角所示，叫做参数整流器，简称PReLU，现在负区间的斜率由参数α确定，我们不需要人为的指定它，也不用硬编码它，而是把它当做一个可以反向传播和学习的参数，具有了更多的灵活性。

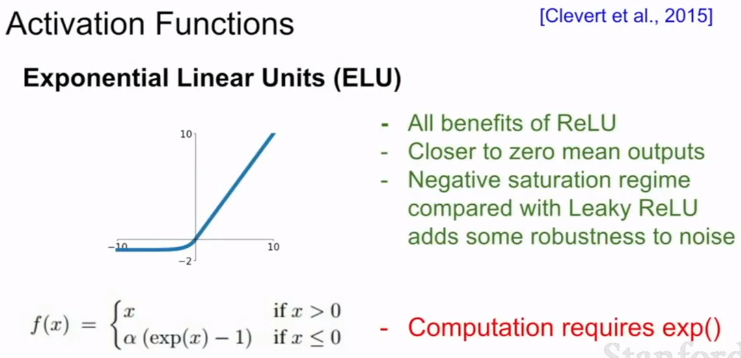


图6.13

还有一些叫做指数线性单元，简称ELU，它具有ReLU的优点，他的输出均值还接近0，图中的论文有关于激活函数的内容。

在实际的操作中，当考察所有的激活函数时，一般最好的经验法则是使用ReLU，是目前比较标准的用法，同时也可以试验一下较为奇特的激活函数，像Leaky ReLU, Maxout ELU，这些方案实验性强一点（实用性弱），也可以试验一下tanh，一般不会用sigmoid。

数据预处理

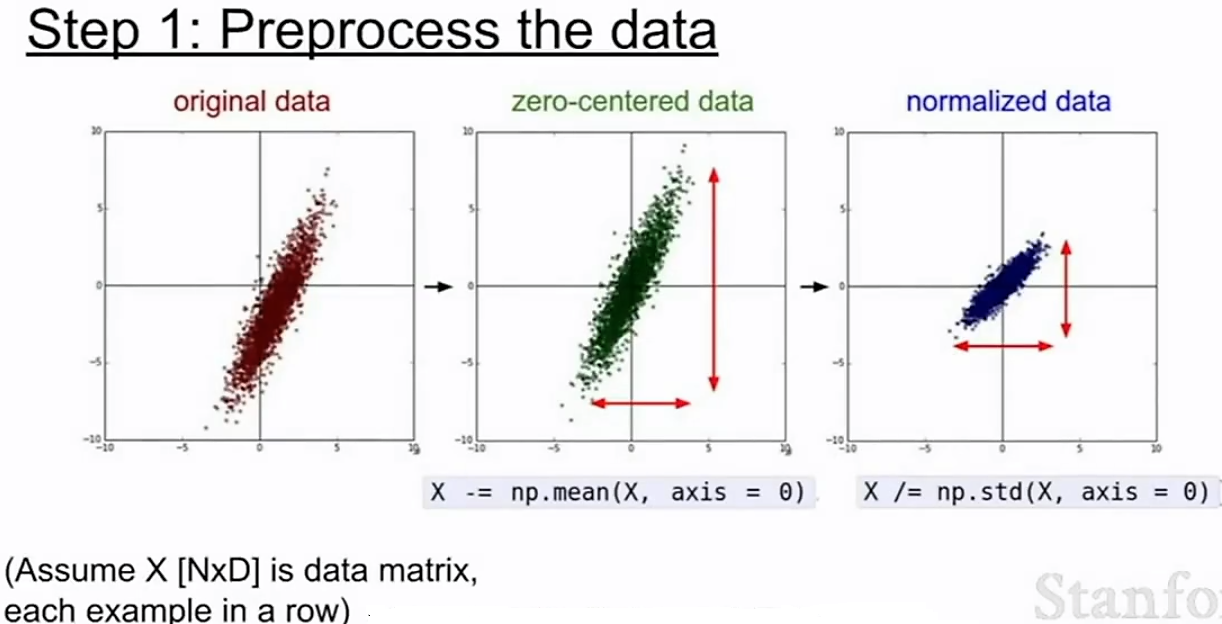


图6.14

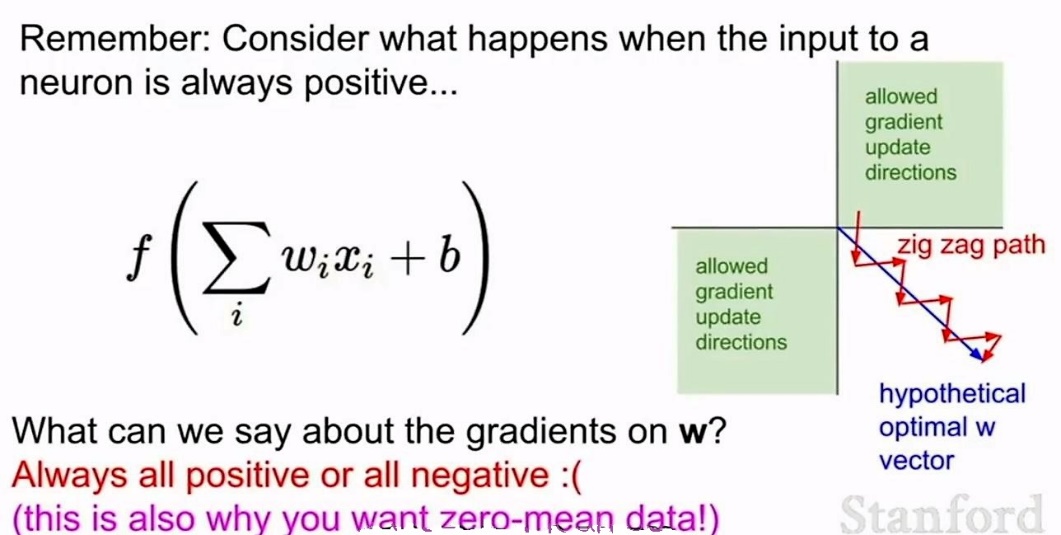


图6.15

一般来说我们总是想预先处理数据，如图6.14所示。一些预处理的标准是，当拿到原始数据，想要进行零均值化，同时还可能想要归一化数据，通过标准差来归一化。但是为什么要进行数据的预处理呢？对于零中心化（zero-centered）,我们应该记得，早些时候有讲若所有的输入数据都是正的，我们得到的权重上的所有梯度也全是正的，然后我们得到的基本是次最优的优化，并且一般来说即使不是非全零或是全负，任意的偏差都会导致这类问题。如图6.15所示。

归一化数据时，所有的特征都在相同的值域内并且这些特征贡献相同。实际上在图像处理问题中，大部分情况下我们会做零中心化处理，但我们不会真的去过多的归一化图像像素值数据，因为一般对图像来说已经在图像的每个位置得到了相对可比较的范围与分布，所以没必要归一化太多。这是因为对比之下，一般的机器学习问题会有很多差异非常大的特征，并且特征所处的范围差别也很大。

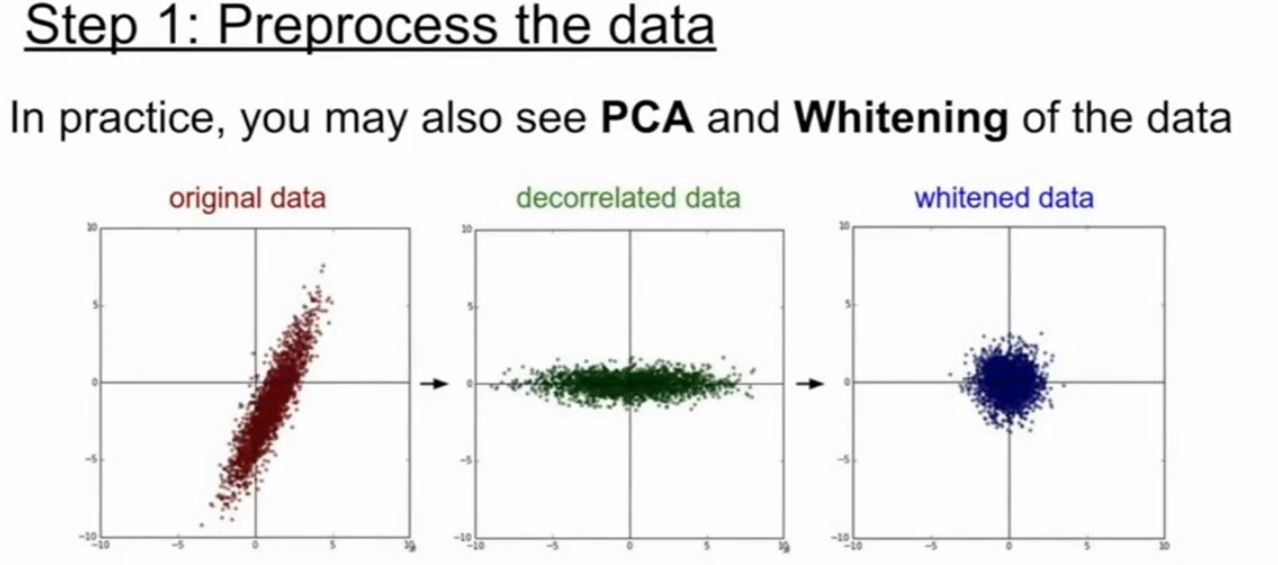
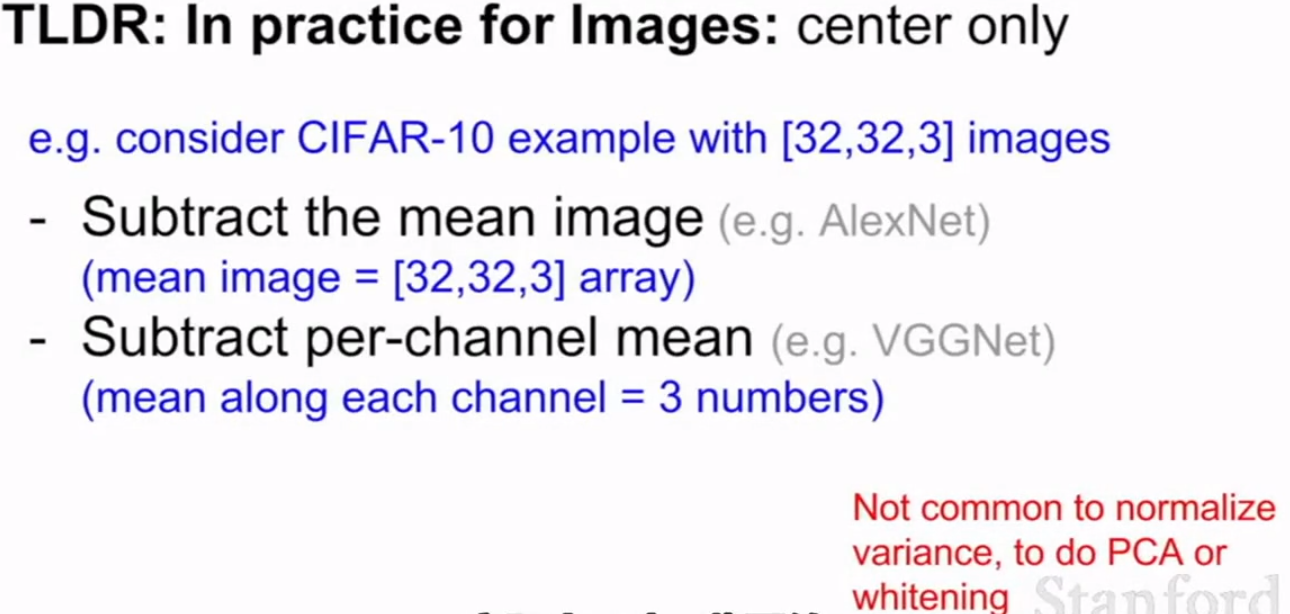


图6.16

在机器学习中还会看到更为复杂的东西，像是PCA或是白化，如图6.16。但是说回图像领域，我们向来坚持零均值化而不做归一化，也不会做更加复杂的预处理。因为我们就想保留图像的空间结构，然后卷积得到原图像的空间结构。

Q：在训练阶段对数据做预处理，那么在测试阶段还要做预处理吗？

Ans: Yes, 一般来说，在训练阶段我们会决定我们的均值，然后我们会将一样的均值运用到测试数据中去，所以我们会用从训练数据中得到的相同的经验均值来归一化。总结来说，对于一般的图像处理，我们就只做零均值化的预处理，并且我们可以减去整张均值图像的值。

从训练数据你可以计算均值图像，其尺寸和你的每张图像都是相同的，举例说，对于32\*32\*3的图像你会得到一组数，然后对你要传到网络中的每张图减去均值图像的值。实际上，对一些网络，我们也通过减去单通道的均值代替用一整张均值图像来将图像集零中心化，我们只是取每个通道的均值，这么做是因为发现纵观整张图片来看，减去均值图像和只是减去单通道值相比没有什么太大的不同，并且更容易传送和处理，会在VGG网络中见到这种操作。

Q:当我们减去均值图像的值时，均值是怎么得到的？

Ans:均值从你所有的训练图像中得到，将所有的训练图像拿来计算均值即可。这里的所有图像没有进行分批次。

Weight initialization（权重初始化）：

一般不用全零去初始化所有的权值，当我们使用全零去初始化时，所有的参数都会获得相同的梯度，用相同的方式更新，就是说最后得到的神经元是完全一样的，这并不是我们所期望的，我们想要不同的神经元学习不同的知识，没有打破参数对称问题。

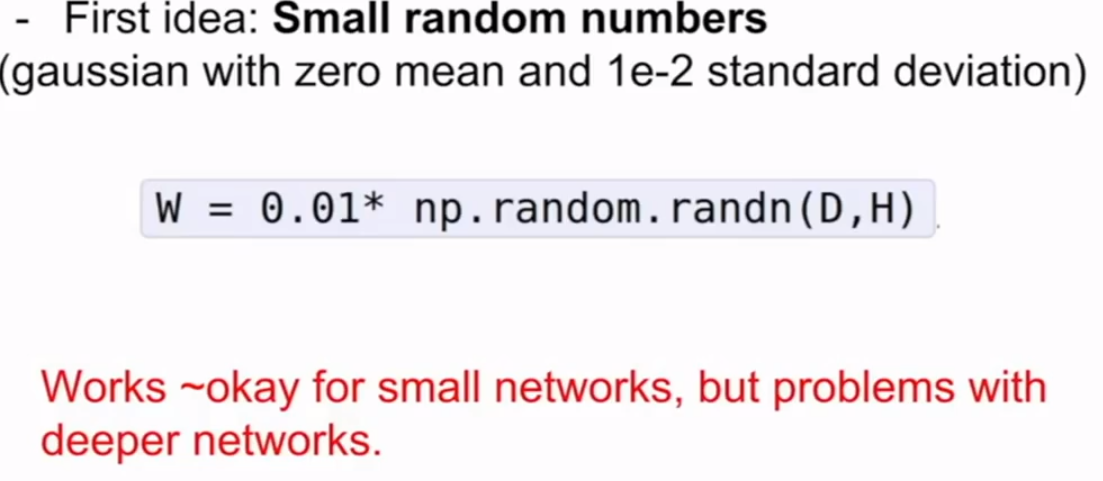


图6.17

我们可以尝试将其改变为所有的权重都是很小的随机数，我们可以从一个概率分布中抽样，比如从标准高斯分布中抽样，对他进行尺度变换，标准差=1e-2，这样给定了很多小的随机参数，这样的参数在小的神经网络中适用，打破了参数对称问题，但在结构深一点的网络中可能会存在问题。怎么解决呢？有一些研究表明初始化权重合适的方式一个很好的经验是Xavier初始化（Xavier initialization），这是Glorot在2010年发表的论文，如图6.18所示，

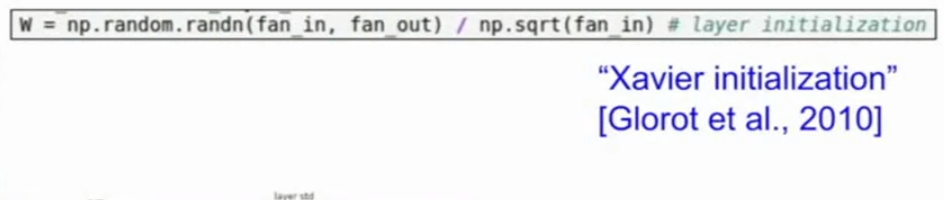


图6.18

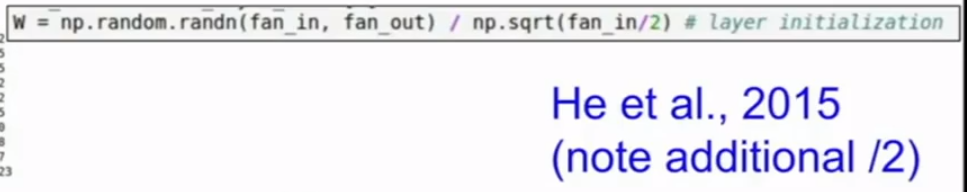


图6.19

关于这个公式的解释是，见论文，这是关于tanh激活函数的权重初始化问题。同时，图6.19有关于Relu激活问题的权重初始化。