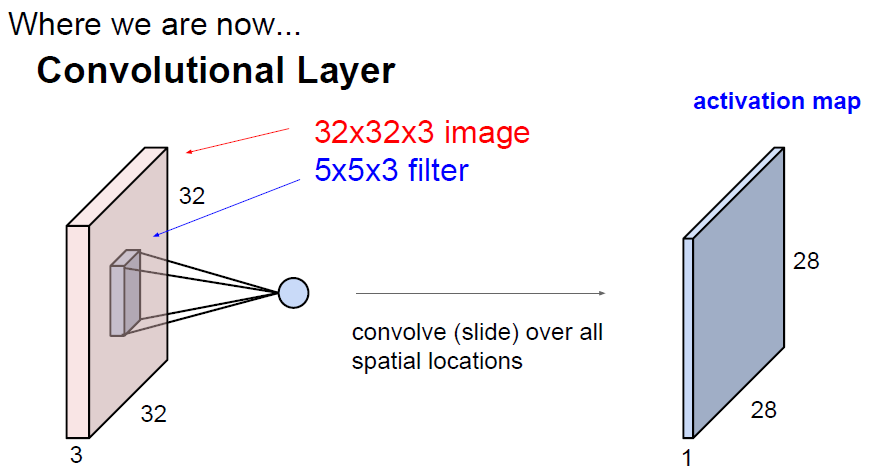
Lecture 6:

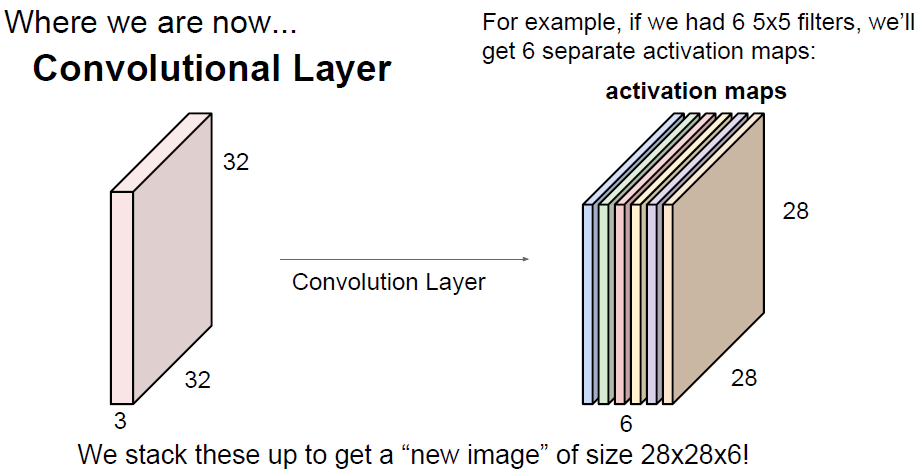
Training Neural Networks,

Part I

Notes on backprop for linear layer & vector tensor derivative linked to lecture 4 on syllabus

目前的进度：





Mini-batch SGD

Loop:

1. Smple a batch of data
2. Forward prop it through the graph(network) get loss
3. Backprop to calculate the gradients
4. Update the parameters using the gradients

下一步：

Training Neural Networks

1. **One time Setup:**

Activation functions

Pre-Processing,

Weight initial

Regularization,Gradient check

1. **Training dynamics:**

Babysitting the learning process 监控过程

Parameter updates

Hyperparameter optimization

1. **Evaluation**

Model ensembles

# Part 1 :

Activation function

Data pre-process

Weight initial

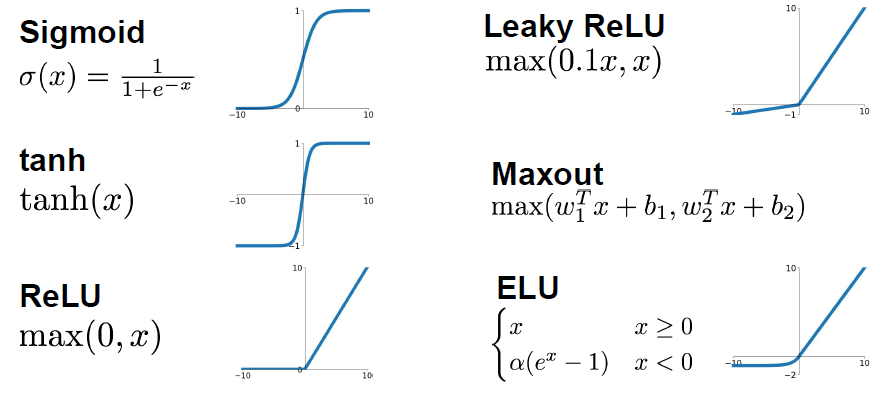
Batch normalization

Babysitting the training process

Hyper-parameter optimization

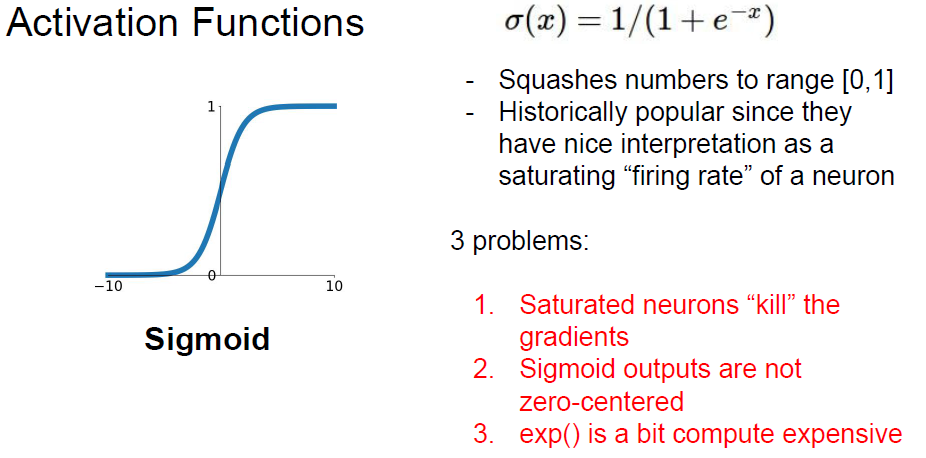
Activation Functions激活函数





## sigmoid:

把输出可以压缩到[0,1]之间，很大的值趋于1，很小的值趋于0，类似于神经元的饱和放电率

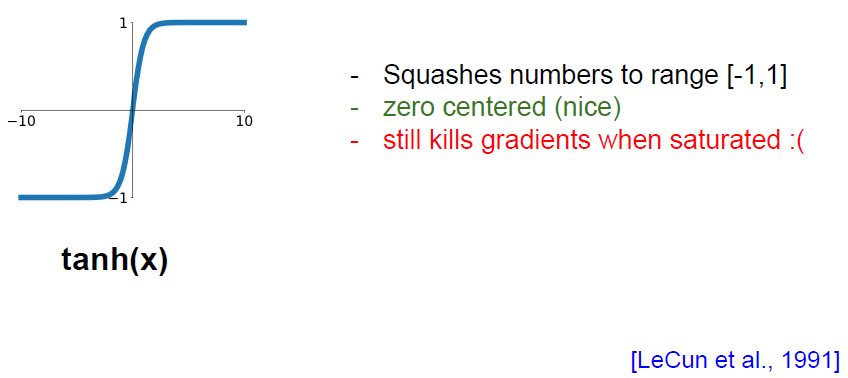




，其中,如果X都是正的，那么就是正的，而要么是全正，要么是全负，也就是说梯度更新所有权重参数w时，要么全加上正数，要么全加上负数，这样的更新效率是很低的

exp运算有一点压力，不过卷积神经网络用到矩阵的点乘，这比exp的消耗还要大…

## tanh(x)函数



值域[-1,+1]

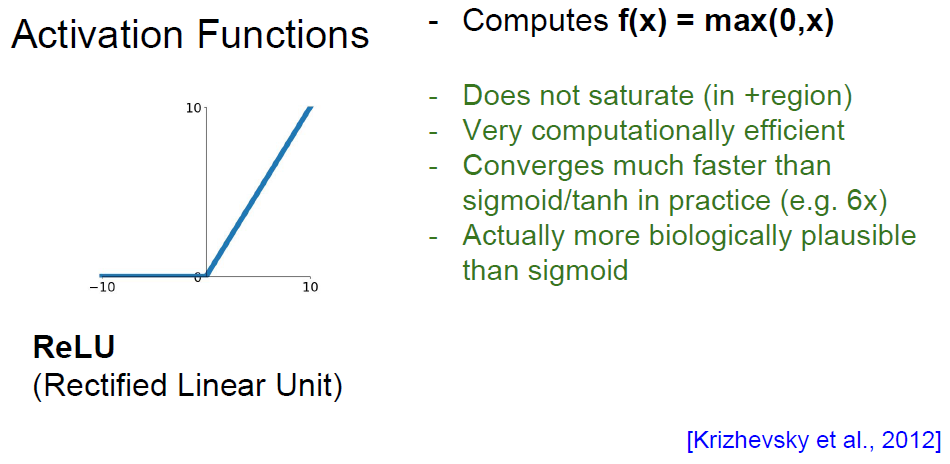
零中心，不像sigmoid那样如果X是正的，W求导出来也都是正的，更新慢

但仍然有梯度消失的问题

## ReLU函数(rectified linear unit)

[Krizhevsky et al., 2012]

**f(x)=max(0,x)**



不会在正区域出现饱和（saturate）现象,

计算成本也低，没有exp等运算，

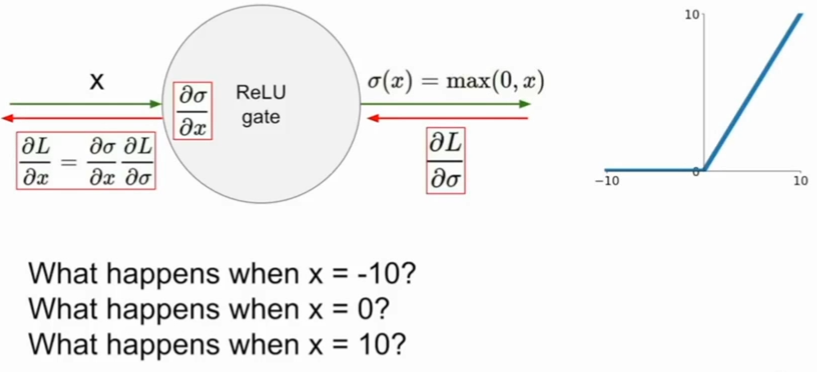
大约比sigmoid, tanh 收敛快6倍

Alexnet出现以后被大量应用

The gradient when x<0 : **0**

而ReLU也有非0是中心的问题，就是说如果X全是正的，则dw也一样是全正或全负

考虑如下例子：



X=-10, =0

X=10, =10

X=0, =0

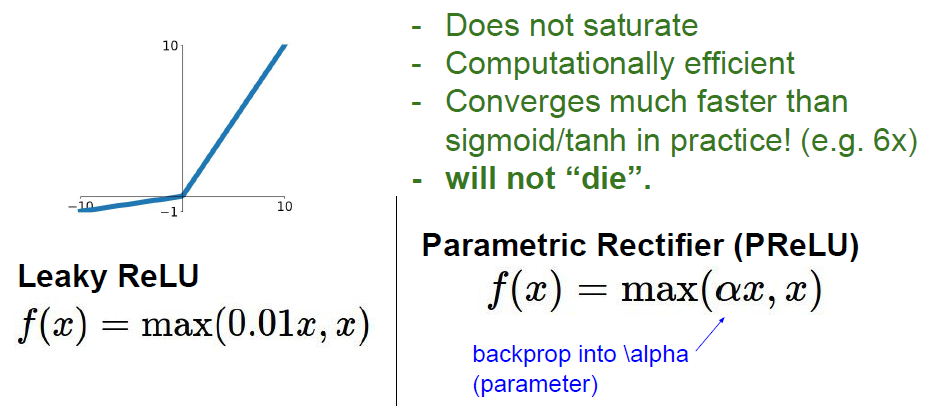
也就是说对于小于0的X，梯度消失了，这时称为**dead ReLU,**

有时在初始化ReLU时，为了避免这种情况，初始化时附一个很小的正数(0.01),以增加它在初始化时被激活的可能性，获得更新（有时候用，有时候不用）

## Leaky ReLU函数

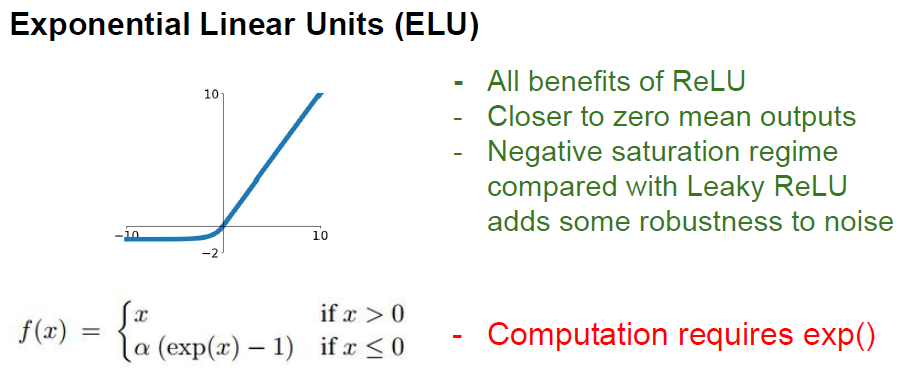
[Mass et al., 2013]

[He et al., 2015]



## Exponential Linear Units (ELU)

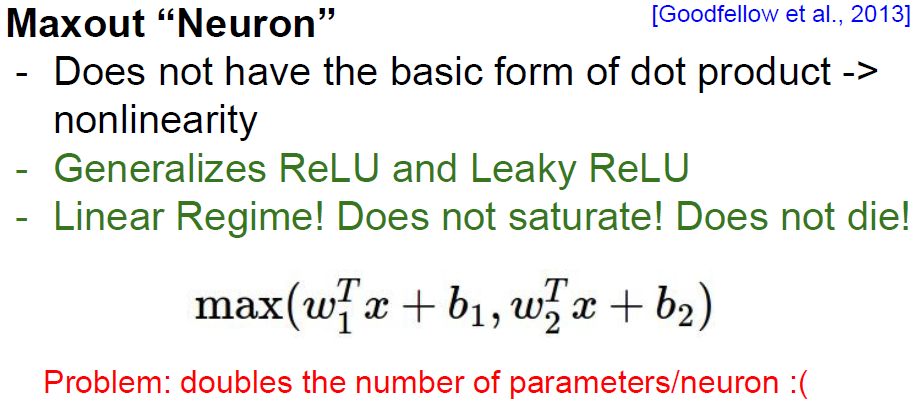
[Clevert et al., 2015]

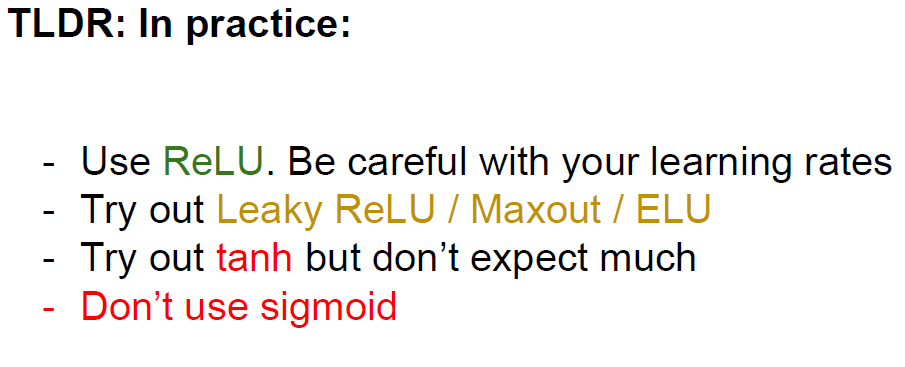


其实在负区间会存在饱和

## Maxout

[Goodfellow et al., 2013]

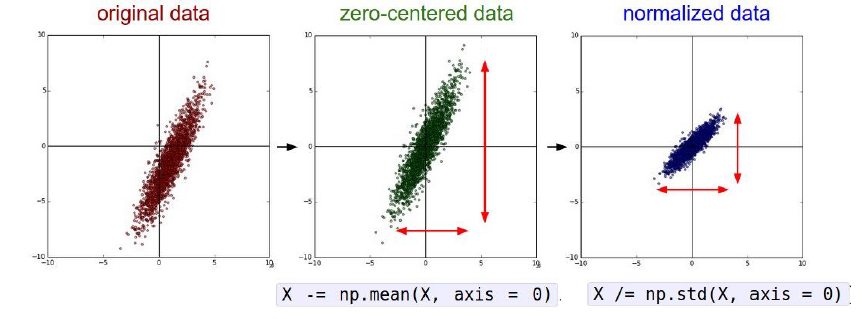




# Data Preprocessing

# 数据预处理

假设INPUT: X(N,D), 每一条数据是一行(1,D)



主要是0中心化：避免数据是全正或全负

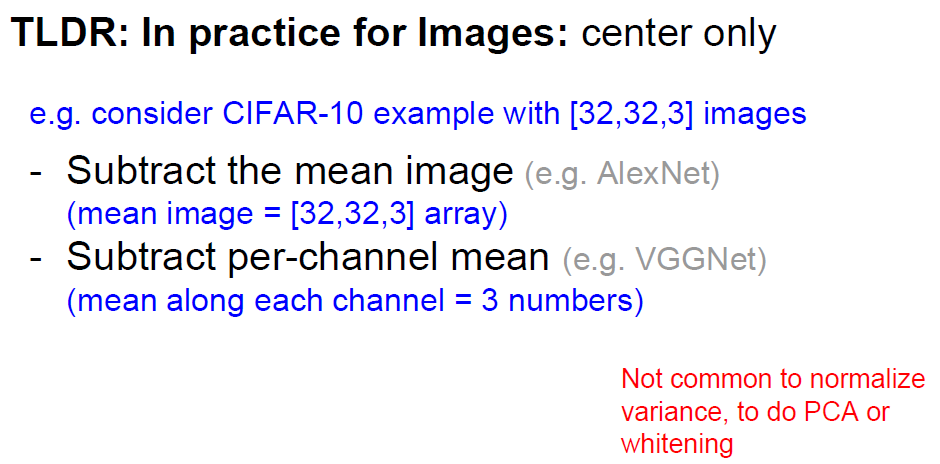
对于深度学习，一般不会过多的做归一化像素，因为一般的图像都是0-255固定区间的

另外还有PCA, whiting，对于图像识别，就坚持做0中心，其他不怎么做，因为不会需要保留过多细节，而是用卷积层去映射到新的维度上去。

另外我们除了在预处理阶段做这样的操作，我们也在training process中做：

以CIFAR-10为例 每条数据都是 [32,32,3]

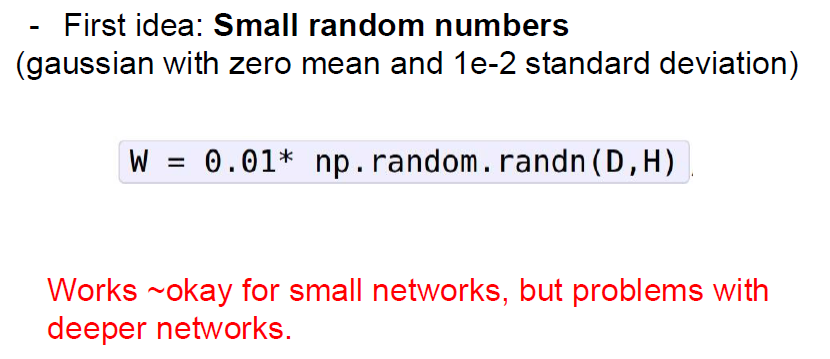
1. 减去均值（mean image）(e.g. AlexNet)
2. 按通道减去一每个通道对应的均值（这里是有3个通道,RGB），3个数字(e.g. VGGNet)



# Weight Initialization

# 权重W初始化

假设 W初始化是全0值？ 那么后面的所有输出都是0+b

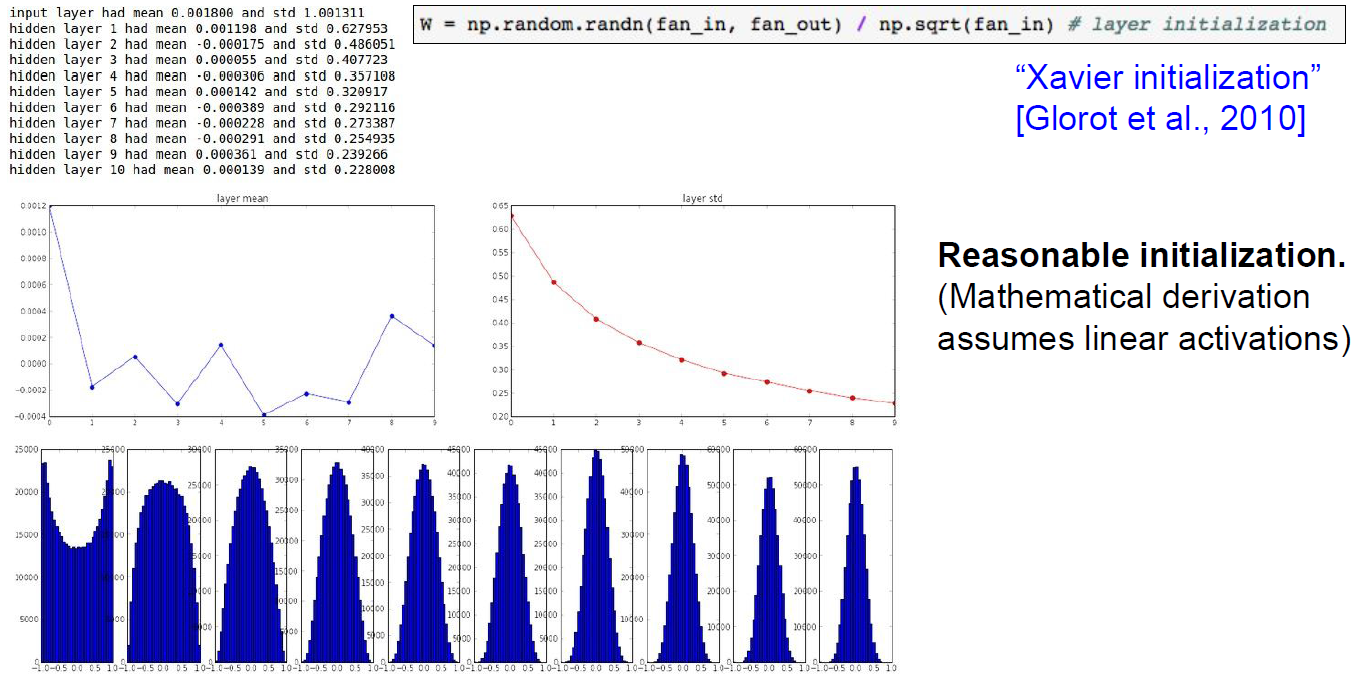


“Xavier initialization”

[Glorot et al., 2010]

少量的input => 较大的weights

大量的input => 较小的weights

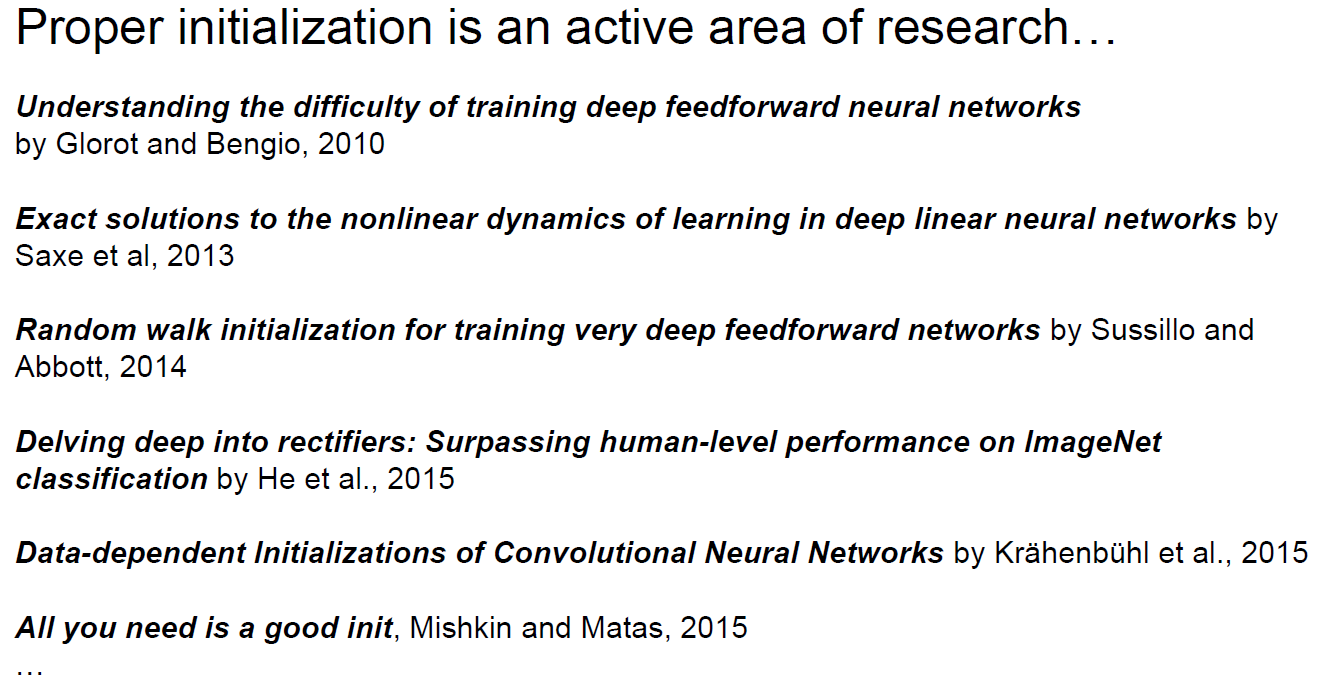


由于ReLU层， 使得一半的节点没有激活， 有个方法是对输出除以2以抵消这种现象

具体看讲义

<http://cs231n.github.io/neural-networks-2/>

相关文档：



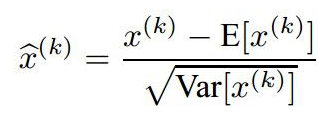
W的初始化目前还是一个很热闹的领域，每年有很多论文

目前比较好的方法是用Xavier方法

# Batch Normalization 批量归一化[Ioffe and Szegedy, 2015]

在我们想要的高斯范围内保持激活

想像在某一层有一批激活，为了让每个维度上都是0-mean(0中心)的单位方差，进行如下变换：

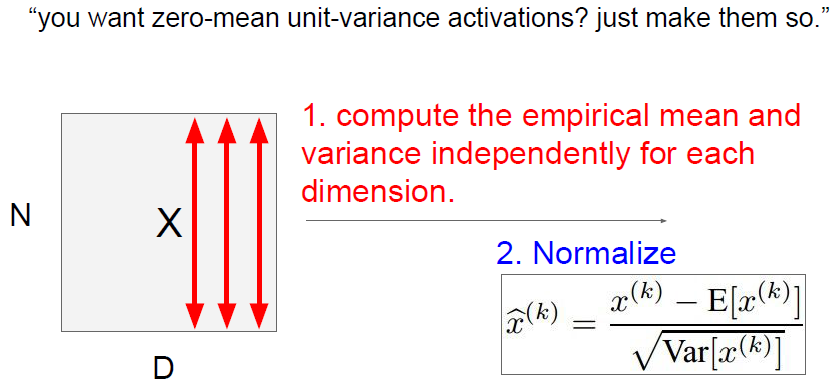


：批量输入的均值，而除以就是进行Normalize归一化,

此动作是在training阶段做，而不是weight initial阶段，因为希望训练中都实现激活

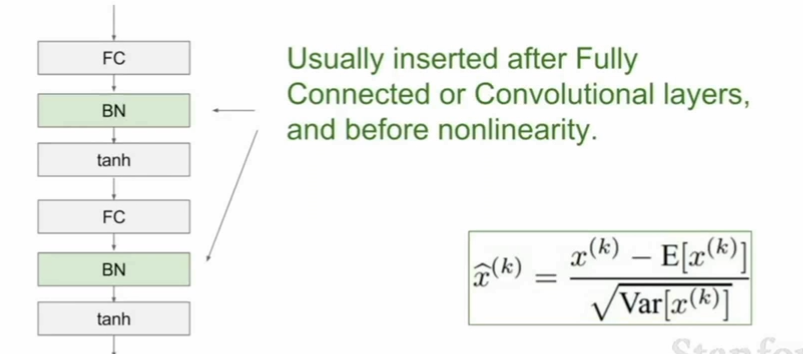
而这个函数是可微的

假设输入为(N,D),每一行是一条数据，



一般这一层放在FC层(fully connected)或卷积层(Convolution layer)之后，非线性层(激活函数)之前(比如relu, tanh),唯一不同的是卷积层的情况下，不只是对所有training examples，和每一个独立特征维度 进行归一化，而且还对所有的特征维度，所有激活空间的维度 进行归一化，这么做是因为我们要服从卷积性质，我们希望附近区域也被进行归一化，所有对于卷积层，每一个激活映射，我们有一个均值和一个标准差（方差的算术平方根），然后我们会用这两个数据，对batch中的所有样本进行归一化 。

作业中会有这方面的内容，细节见2015年的论文 **[****Ioffe and Szegedy, 2015]**



### 问题：我们是否真的需要在FC和tanh之间有这么一个Batch Normalization的，高斯单位的层？因为这么做是限制输入为非线性函数的线性区域。完全去饱和的情况 .

实际上有一定的饱和可能也是有用的

在完成归一化之后，需要进行额外的缩放操作，对缩放，再平移

，就学习得到的

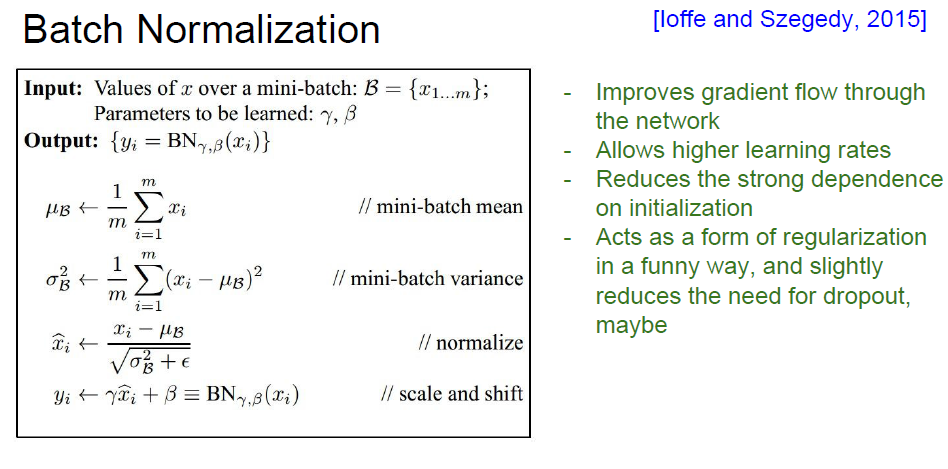
如果：



那么就相当于不做归一化

用来恢复恒等映射函数 identity function,就像没有进行批量归一化一样，

通过学习，,控制让tanh具有更高或更低饱和度的能力，to have good training



所以一但使用批量归一化，训练就变得更加容易

另外也可以看成是regularization 的一种方式 ，相当于给x一定程度的抖动

一般是用在标准的卷积神经网络中的，强化学习 和递归神经网络中 用到的比较少

### 问题：如果我们强制要求输入是符合Gaussian distribution,是否会损失一些structure information?

实际上只是平移和缩放来方便学习，在卷积层，确实有一些空间结构希望保留，这时，我们会对整个激活映射找一个均值，所以我们只找训练样本的先验均值和方差，（作业中也有这方面的内容 。）

实际中不一定要完成符合Gaussian , 差不多就行了

### 测试阶段的批量归一有一些不同：

不会学习，用training-time 的。

Babysitting

the Learning Process

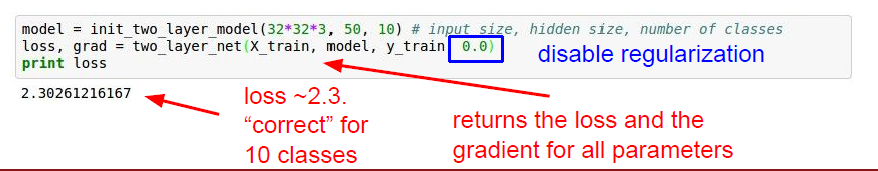
and Adjust those Hyperparameters

pre process =>

=> choose architecture

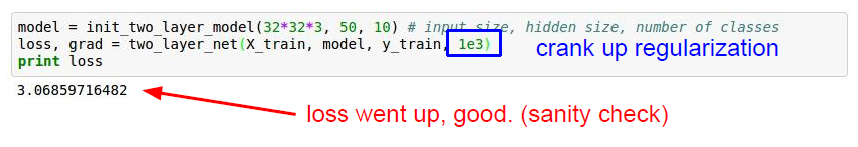
=> initial network

=> forward pass : get loss and check



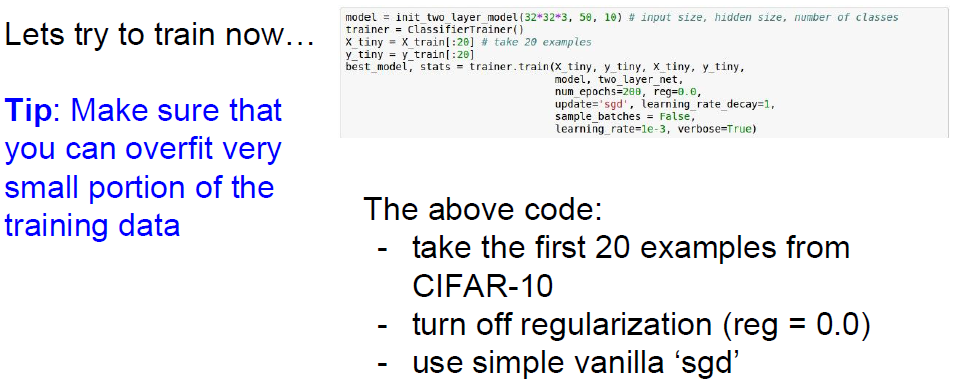
=>add regularization

这时loss应该变大了

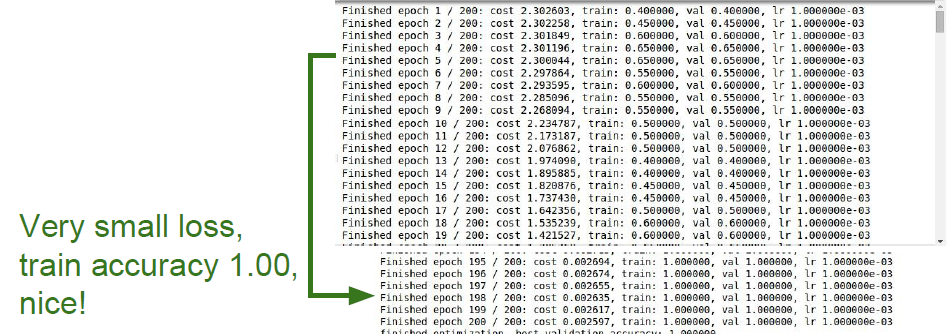


=> training （检查阶段）

最好是从小数据集开始，因为如果有个小数据集，可以拟合的很好，loss很小，这时把regularization 关闭，这时再看看loss能否变成0



这时看到准确率上升到1

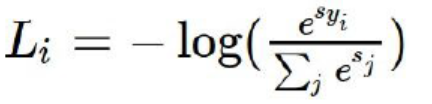


=>training (真正的训练阶段)

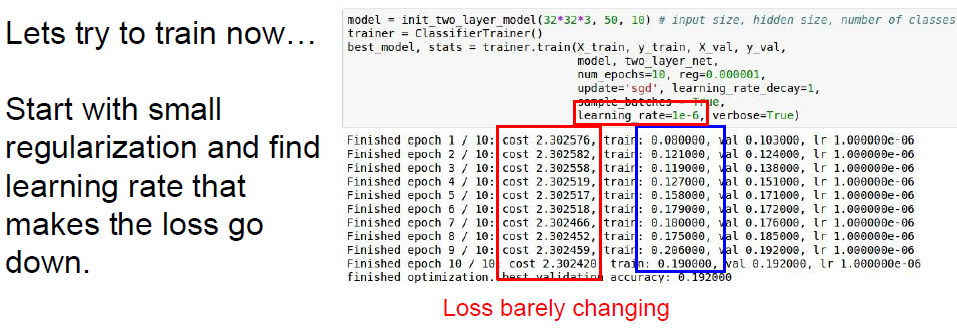
掏出真正的训练数据，加上一个小的regularization

1. learning rate 是最重要的参数，首先来调整它，首先这里用的是1e-06,这时loss基本不变，cost也一样，说明太小了，

注意这里loss虽然没怎么变，但acc很快的升到了20%， 这里用的是softmax:

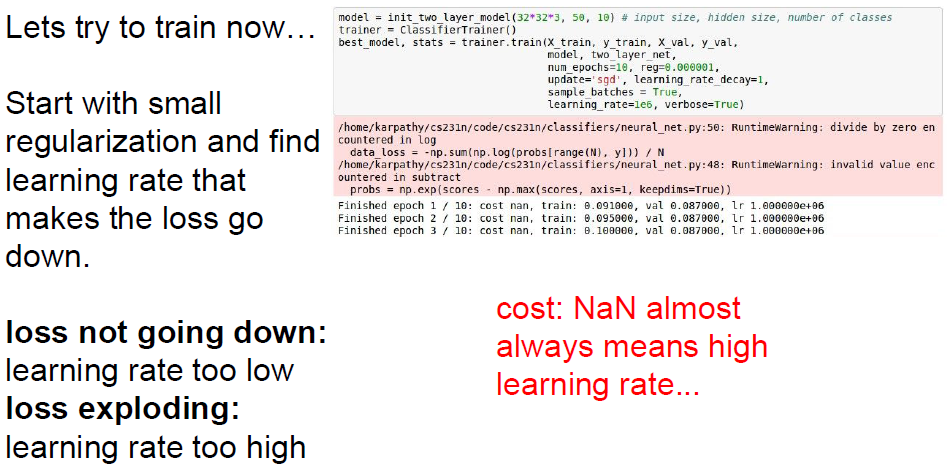


说明这时分布依然很分期，所以loss很接近，但我们把这些所有的分布都朝着正确的方向在轻微的移动，weight在朝正确的方向改变，现在acc可能发生突变jump



这时如果我用一个很大的learning rate 1e6:

Cost 变成NaN, 基本都预示着learning rate 太大



一般的learning rate 在[1e-3,1e-5]之间

=> backward pass => update weights

**Hyperparameter Optimization**

**超参数的优化**

**-交叉验证**

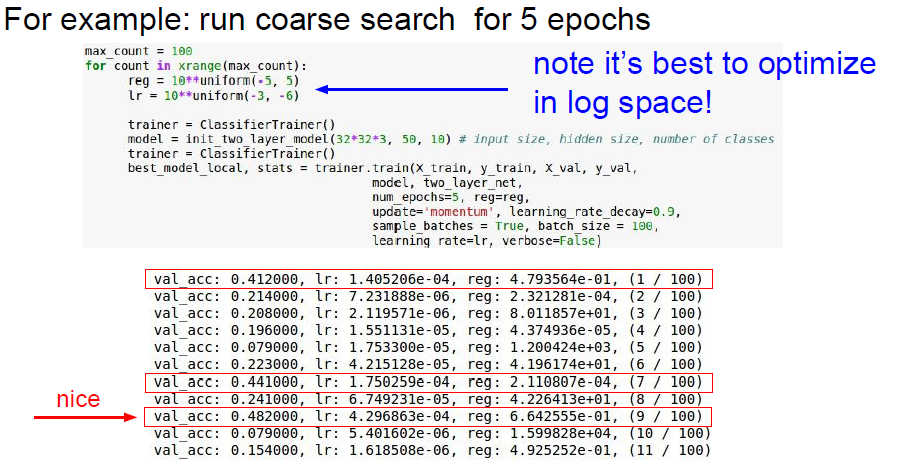
先确定那些参数大概的情况 和 区间， 重要程度 ，再去掉出具体的值。

\*\*\*如果cost 大于3了， 说明loss太大，可以终止

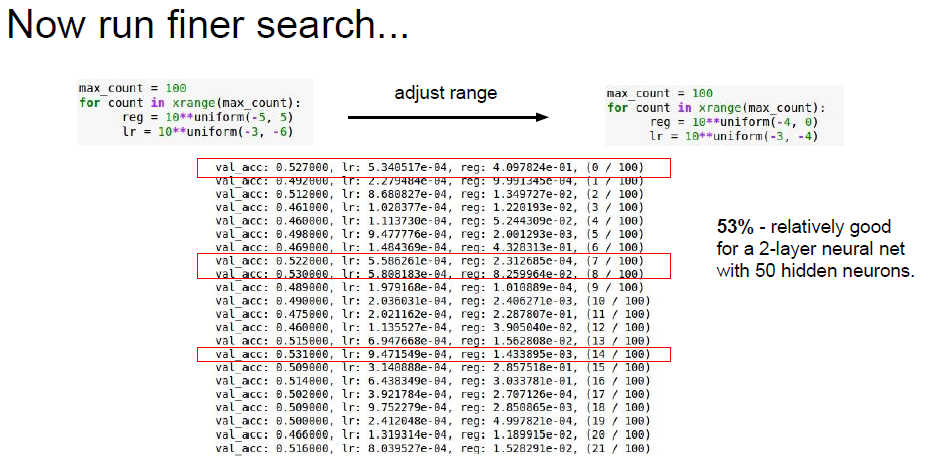
下面我们用5个epoch来进度初始搜索：

找出acc比较高的几个。

通常使用对数来优化，



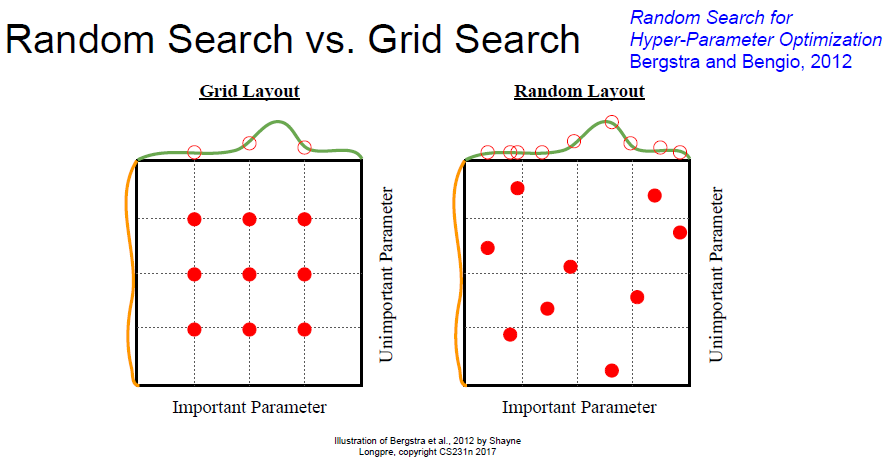
下面进行具体确认搜索



## 另一个方法是网格法。

*[Random Search for Hyper-Parameter Optimization*

Bergstra and Bengio, 2012]

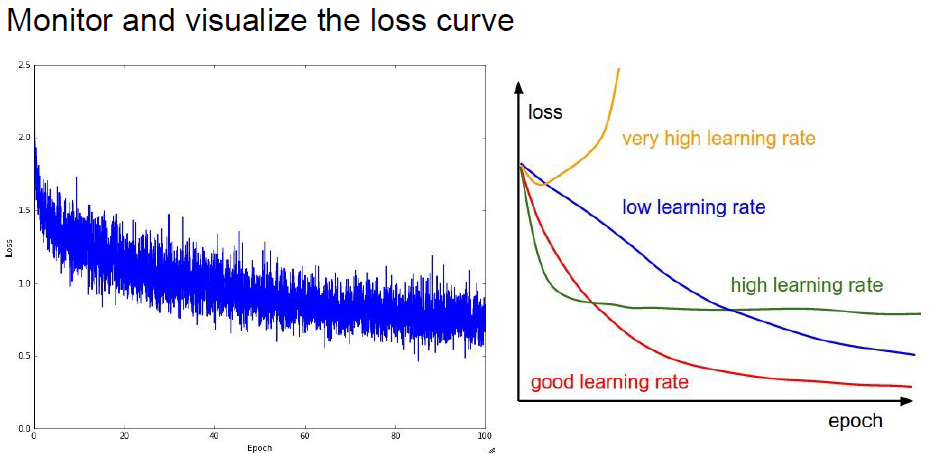


超参数优化主要处理：

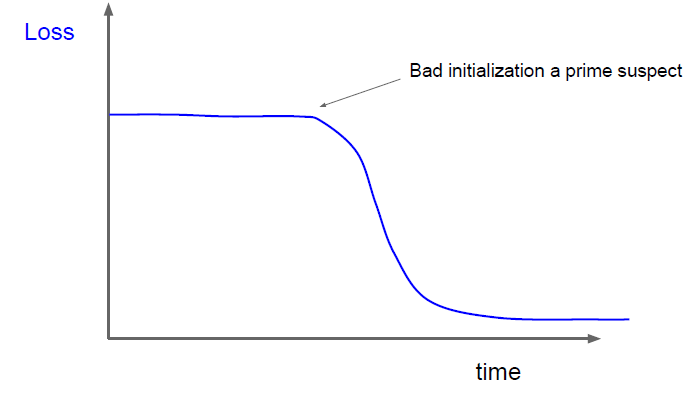


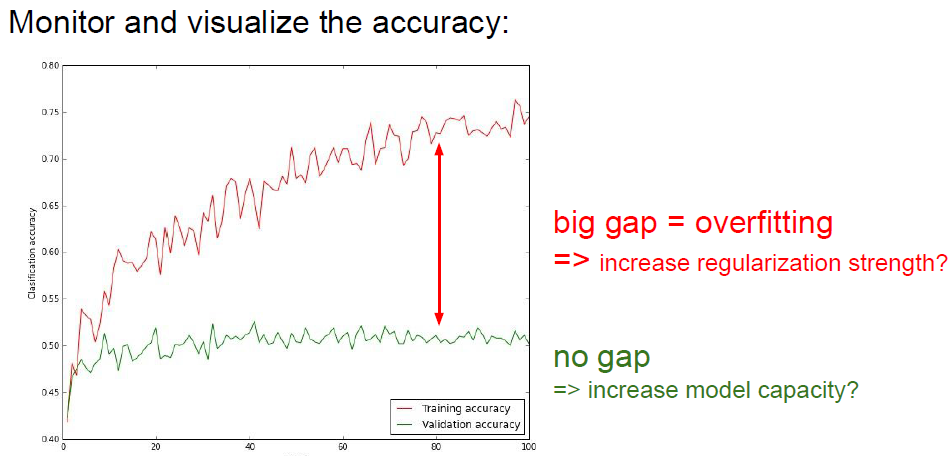
实际中会有大量的交叉验证：



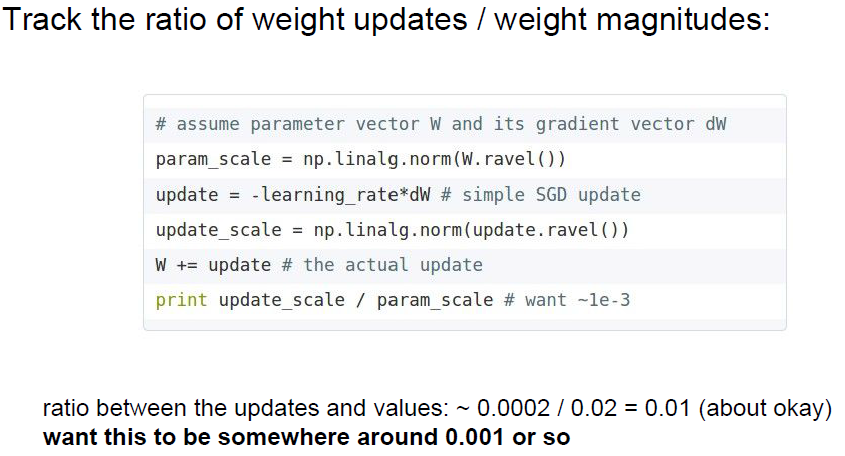


还会有这种情况：





### 更新步幅：



Summary小节:

We looked in detail at:

- Activation Functions (use ReLU)

- Data Preprocessing (images: subtract mean)

- Weight Initialization (use Xavier/He init)

- Batch Normalization (use)

- Babysitting the Learning process

- Hyperparameter Optimization

(random sample hyperparams, in log space when appropriate)