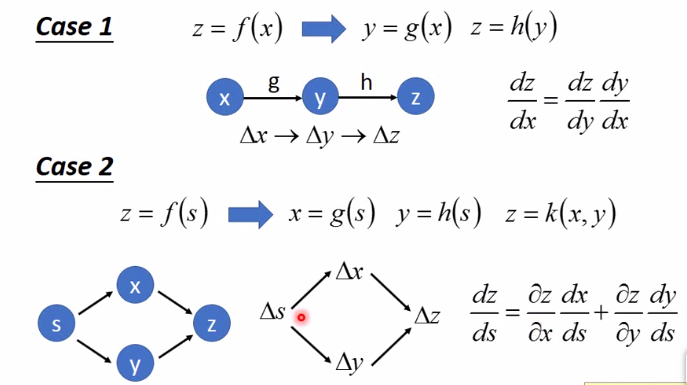
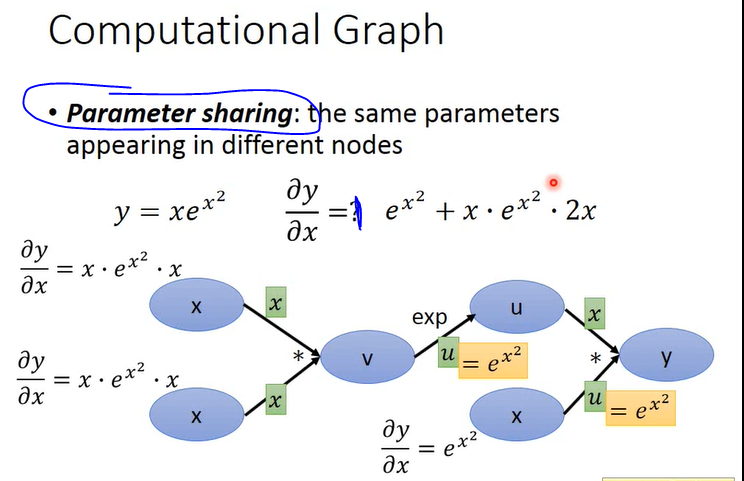
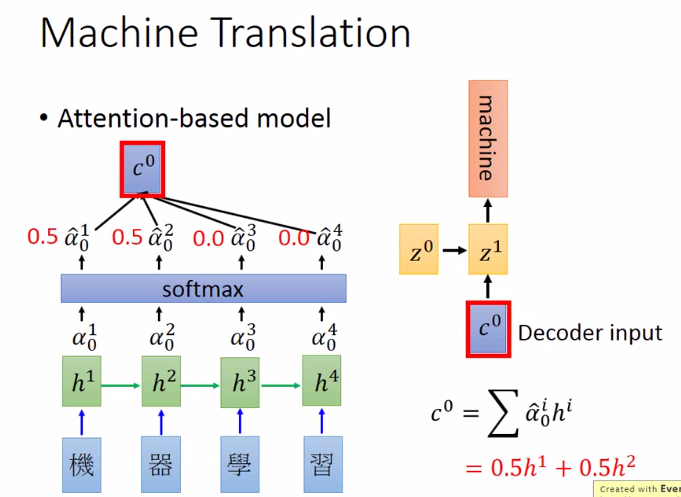
# 1.反向传播



参数共享时的情况:



# 2.Attention注意力模型



h1是encode阶段某一时刻的输出

z0是一个初始化的向量，也可以是可训练的

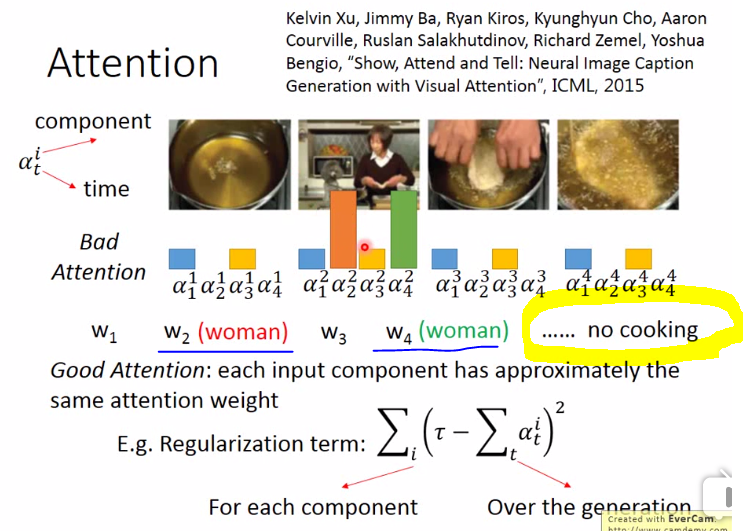
α是h1与z0之间计算匹配度的匹配结果，这个计算匹配度的方法可以有很多种，自己设计

然后进过softmax，将结果与h1相乘，然后累加得到c0，将z0和c0作为decoder阶段的输入。然后重复循环。

优点:计算匹配度的时候，比如考虑机器的时候，会忽略掉学习，将注意力集中在机器上，而计算学习时可以忘掉机器。因为当h1是机的时候，机与器的匹配度α是很高的，而与学习无关，这样会反应在c上。

# 3.训练中存在的一些技巧

## 1.注意力模型技巧：



例子1：生成的句子里出现好几个woman这个单词，是因为attention的注意力过多的放在了女人上面，因此一个好的注意力模型应该在不同的图片里给予相同的注意力。

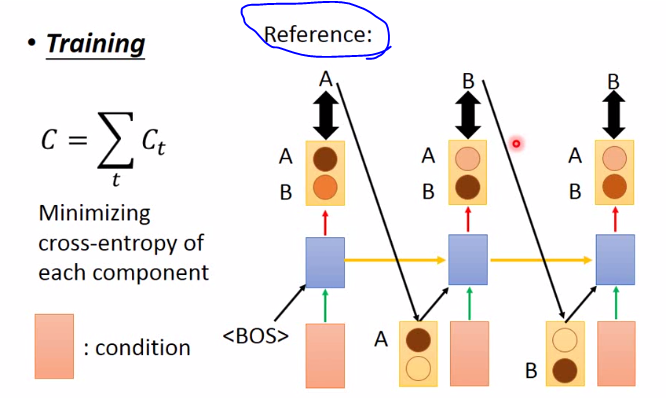
解决办法举例:添加正则化项，τ是每张图片注意力(α匹配度)的累加，用τ减去本章图片的注意力(α匹配度),注意力小的，得到的值就大，损失就越大，这样就会对没有注意力的更加注意

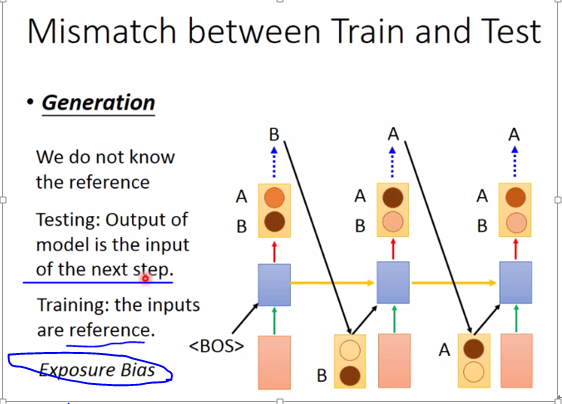
## 2.RNN技巧：

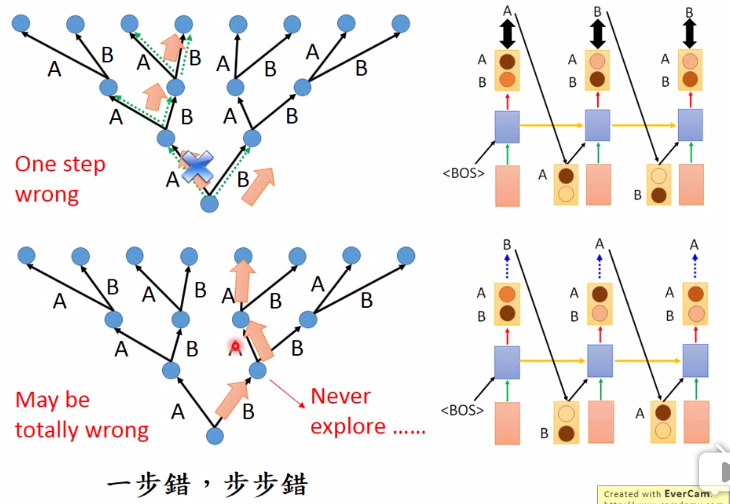
在训练的时候，我们是拿正确的结果A作为下一步的输入，而不是预测值。

而在测试的时候，我们是拿预测值作为下一步的输入，这样存在偏差，称为Exposure Bias(曝光偏差)。

这样导致的后果就是图3中，如果训练时存在错误预测的是B，但是我们下一步的输入仍然是正确值A，但是假如用一样的句子来测试时就会将错误的值B用做下一步的输入，这样以来，往B走的路线是从来没有学习过的，就会出现完全错误的句子。

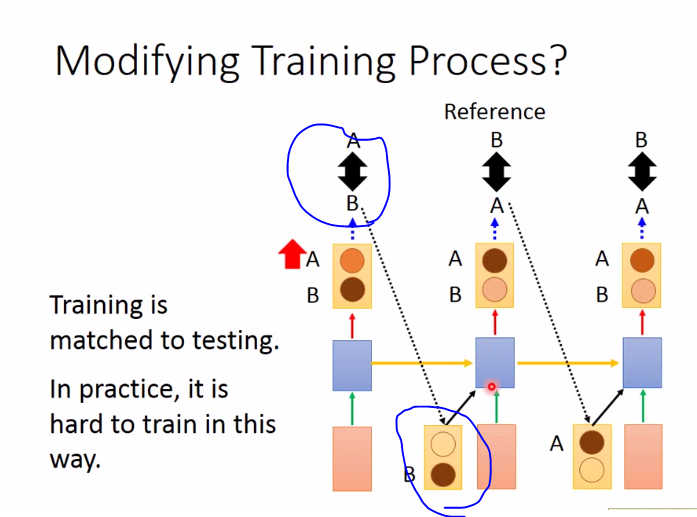






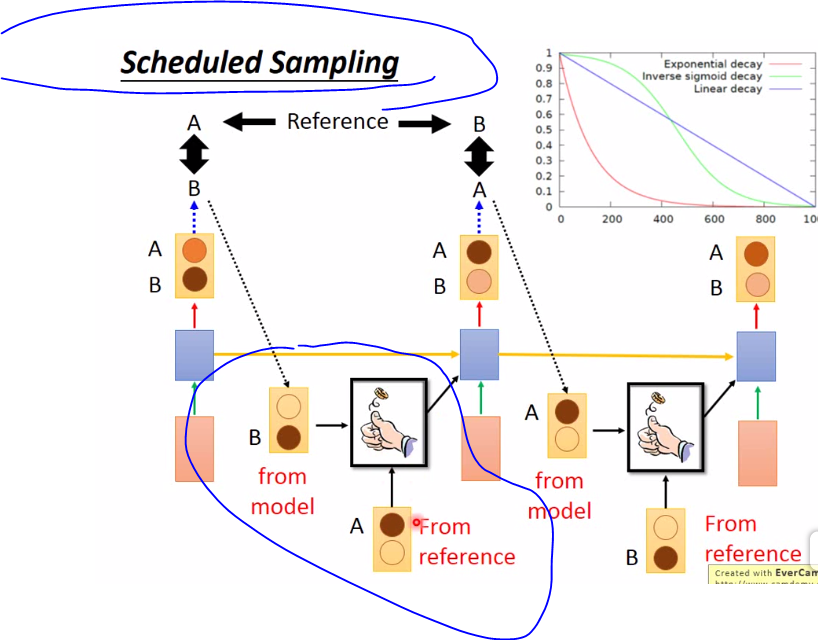
对于以上问题，那么假如我们修改训练过程，使用预测值作为下一步的输入的话，你会发现训练时很难loss收敛。

假如我们预测值是B，真实值是A，那么由于存在偏差，我们会调高A的权重。这个时候，我们把错误的B作为下一步的输入，但是当我们完全重新开始训练第二回的时候，由于前一次训练，将A的权重提高了，那么这个时候输出的结果变成A了，上一次是B，当把A作为下一步的输入的时候，你会发现上一次B作为输入的训练结果全部作废了甚至可能产生更加奇怪的结果。



解决办法：scheduled sampling(定期取样)，以一定的概率选择预测值或者是真实值。

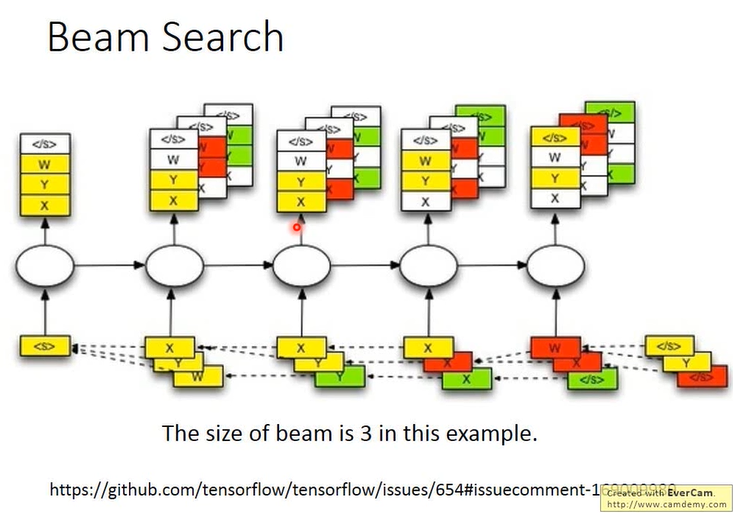
训练刚开始的时候较大概率选择真实值，然后慢慢下降，逐渐选择预测值



Grid Search：暴力搜索，计算所有情况，找到最大的情况

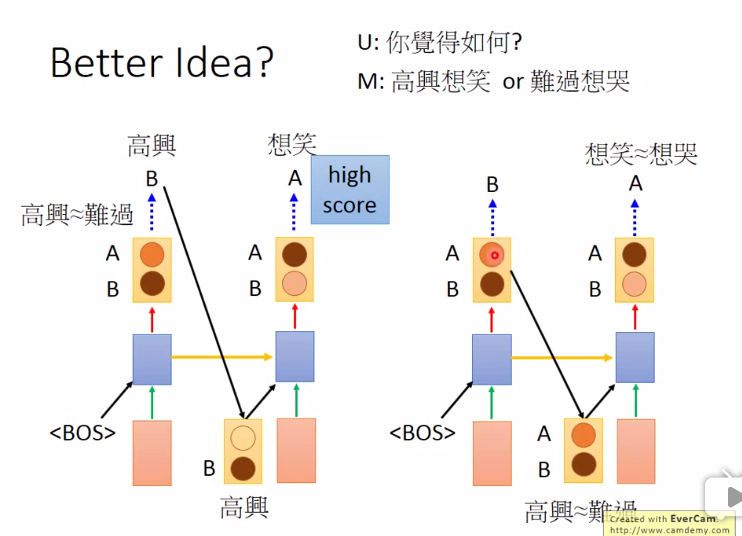
Beam Search：每次只选择当前几条概率最高的

其它 Search：每次只选择当前概率最高的，依次找下去.这种简单粗暴，但是问题也明显，0.6->0.6->0.6,明显没有0.4->0.9->0.9的组合概率高。

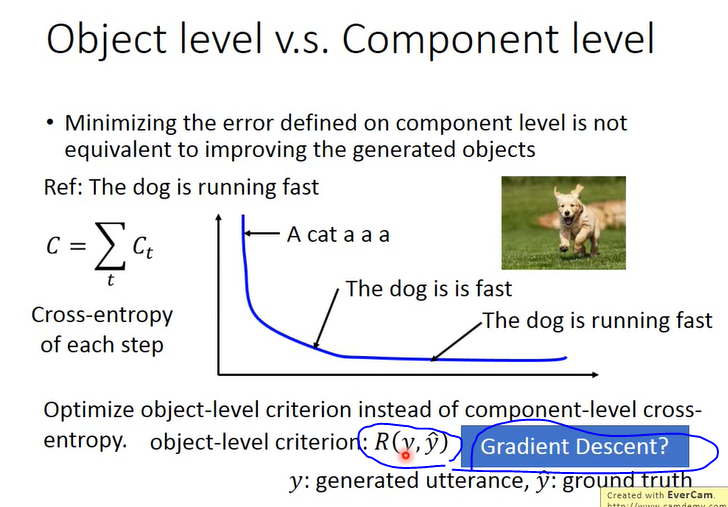


还有一种思路：是将分布的权重W直接作为下一步的输入，不使用预测值，也不使用实际值。这样做的好处是，第一，不用做scheduled sampling，所有的信息都传递给下一步。第二，维度没有变化，不用做处理，直接就可以拿来用。但是同时也存在问题，比如，高兴和难过的概率差不多，你都作为下一步的输入，那么输出想笑和想哭的概率也差不多，最后得到的结果有可能是高兴想哭。而之前的直接输入高兴，下一步就只会输出想笑。

因此，scheduled sampling效果更好

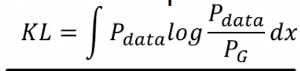


如果是按照一个个单词是否匹配作为损失函数，那么the dog is is fast其实损失值已经很低了，很难再近一步。但是它的语法其实是完全不可用的。如果存在一种损失函数R，是基于整个句子的，那么the dog is is fast其实损失较大让。但是问题是，这样的损失函数是不可以微分的，梯度无法下降，因为当你调参时，不同单词的分布会有变化，但是概率最大的仍然是那个词，导致最后的句子y没有变化，那么R的结果也不会有变化。



# 4.基础概念理解:

KL:又称为相对熵，衡量两个分布之间的相似度，越小越好，KL越小，最大似然估计越大，Pdata是真实分布，Pg是模拟生成的



[信息熵](https://baike.baidu.com/item/%E4%BF%A1%E6%81%AF%E7%86%B5)，是随机变量或整个系统的不确定性。熵越大，随机变量或系统的不确定性就越大。

相对熵，用来衡量两个取值为正的函数或概率分布之间的差异。

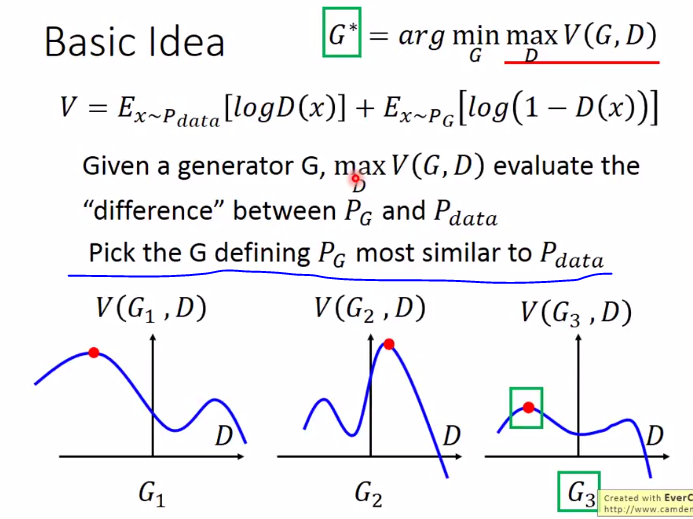
[交叉熵](https://baike.baidu.com/item/%E4%BA%A4%E5%8F%89%E7%86%B5)，用来衡量在给定的真实分布下，使用非真实分布所指定的策略消除系统的不确定性所需要付出的努力的大小。

相对熵=交叉熵-信息熵：

https://gss3.bdstatic.com/-Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D192/sign=18d33eaba064034f0bcdc60f9dc37980/1c950a7b02087bf459f9a79ff9d3572c11dfcfab.jpg

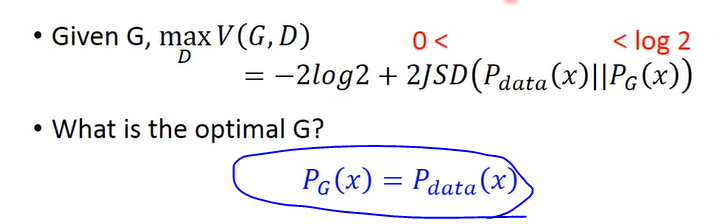
# 5.GAN

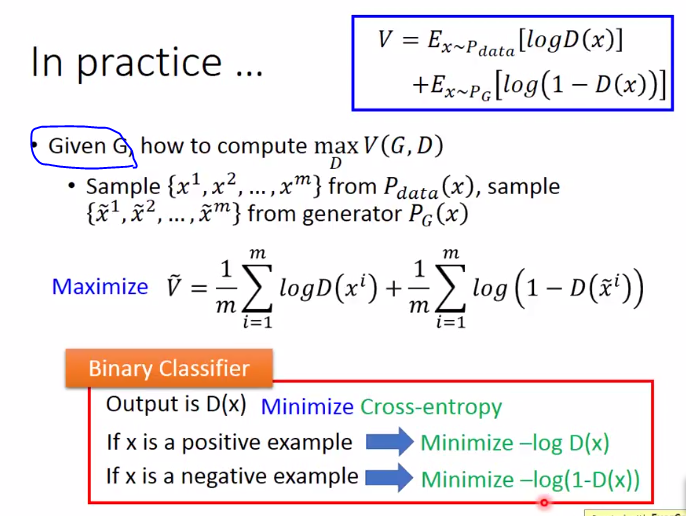
在G给定的情况下，找到一个D，最大化V(G,D),，这个V(G,D) 最大时的值就是生成的数据与真实数据之间的差异值，越小越好。然后接下来找到一个G，让差异最小



那么什么样的G能够让V(G,D) 最大时的值最小呢？

我们可以证明:在G给定的情况下，D的loss与JSD是成正比的，D的loss越大，说明最终生成的与真实的差别越小，难以区分，此时V(G,D)也越大。

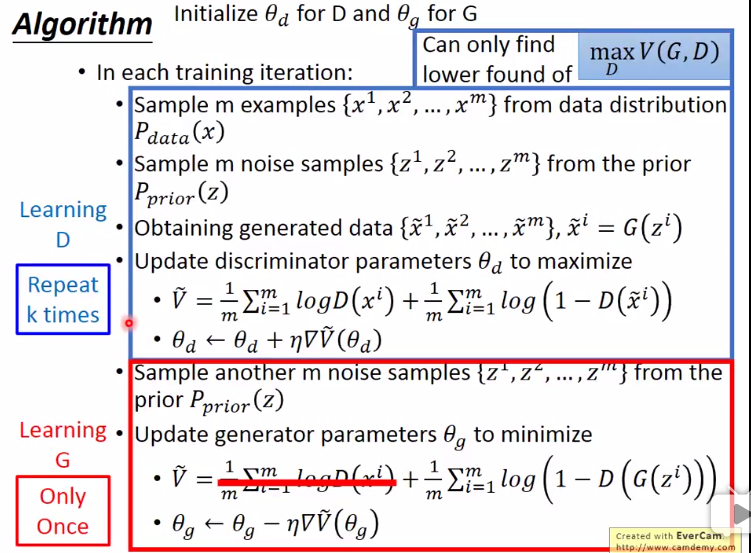




算法的真正实现:

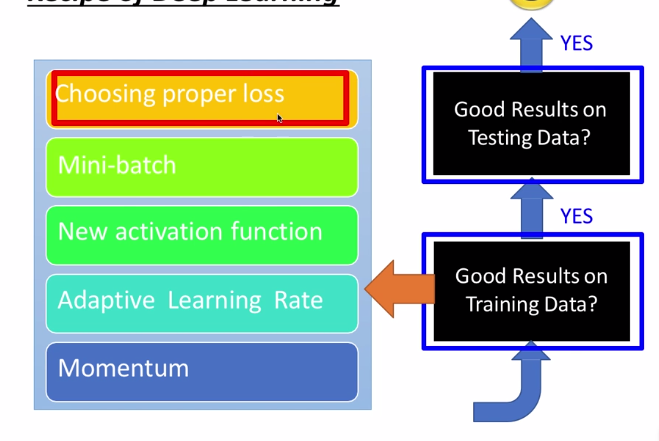
第一步:学习D，在给定的G的情况下，最大化V

第二步:在上一步确定的D不变的情况下，最小化V。由于此时V的前半部和G没有关系，所以图中用红线删除。



# 6.深度学习训练技巧

## 1.如何提高准确率：



### 1.损失函数不好：

例如，交叉熵和平方差可能效果差很多。因为不同的损失函数，在不同的w的时候存在不同的梯度，可以利用梯度下降来选择最优解

当我们选择softmax作为输出层的时候，选择交叉熵

### 2.Mini-batch：每次只拿一部分数据去训练

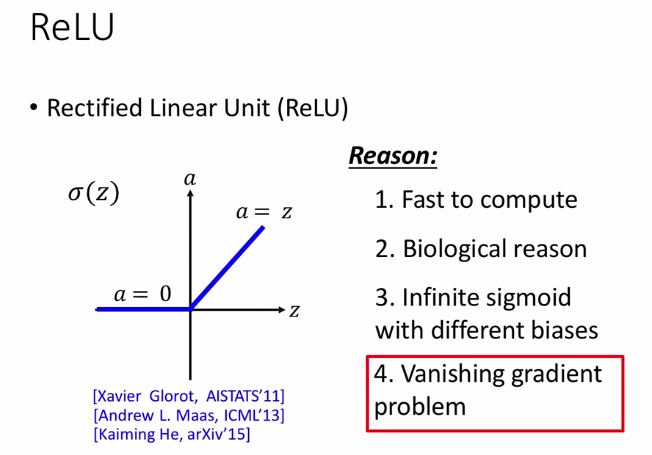
总的数据量除以batch\_size=n,每次拿batch\_size的数据去训练，训练n次称为一个epoch。

1. update的次数更多表现更好，硬件一定时也更快

### 3.选择激活函数

当网络叠的很深的时候，容易梯度消失。例如sigmod时结果都是0到1之间的值，

反向传播时，乘的数越多，值越小，那么参数的更新就会越小，就容易梯度消失。



Relu特点:可以大幅度降低梯度消失的问题。它一直困扰着神经网络。当叠了很多层的时候都会存在这方面的困扰。

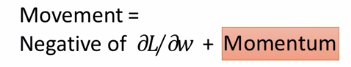
### 4.调整学习率

可以影响步长，步长太大，会越过最小值。步长太小，更新速度很慢。

一般是开始的时候学习率很大，后来越来越小。

### 5.Momentum(惯性参数)

每次移动的距离=梯度+惯性,当在某个局部最优点时，梯度很小，如果我保持之前的惯性，可能会导致loss暂时变大，但是会跳出局部最优点，找到更加的结果。



最常见的就是卡在局部最优点。

Adam的优化器，其实就包含了调整学习率和Momentum

## 如何防止过拟合

过拟合原因:学习目标是训练数据，学习能力太强

### 万能药:收集更多的数据

可以试图创造一些数据，例如图片旋转一下图片。

人为的加入一些噪音

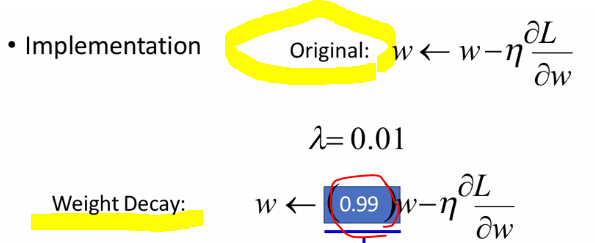
### 2.Early Stopping

当训练到达一定次数的时候，训练loss继续降低，但是测试loss开始升高。

那么继续训练下去，肯定就过拟合了。这个时候就应该停止训练

### 3.Weight Decay

某些权重一直没人用到，没什么梯度，那么它就应该逐步衰减掉



原始的是:原始w减去梯度，得到新的w。

Weight Decay:是先乘以一个系数0.99,这样一来，梯度跟新一直为0的的w会逐步衰减掉。而如果经常用到，那么跟新的梯度会改变w的值，这样不会影响原来的结果。

### 4.Dropout

每次都有一定的概率让一些神经元失效

### 5.Regularization

正则化

### 6.Network Structure

如果以上办法都用了，那么你应该改变你的网络架构