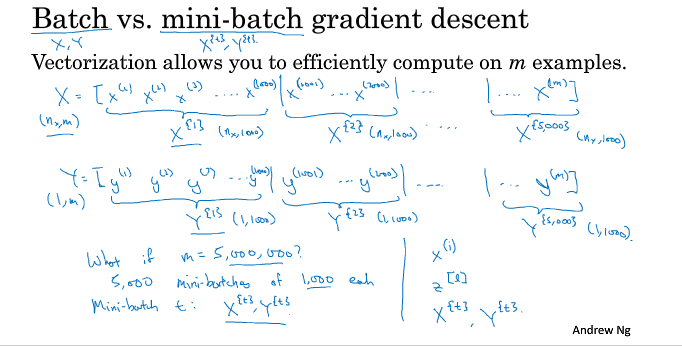
第二章 优化算法

这一章将学习很多优化算法，让你的神经网络运行的更快。机器学习的应用是一个高度依赖经验的过程，伴随着大量迭代的过程，需要训练很多的模型，才能找到最适合的那一个。所以优化算法可以快速训练模型，其中最大一个难点，深度学习没有在大数据领域发挥最大的作用。我们可以利用一个巨大的数据集来训练神经网络，而在巨大数据集基础上进行训练速度很慢，使用好的优化算法大大地提高效率。

2.1 Mini-batch梯度下降法

之前学过向量化能有效的对所有m个例子进行计算，允许你处理整个训练集，而无需某个明确的公式，所以把所有训练样本都用X矩阵来表示，Y也是一样。向量化可以帮助快速处理，但如果m很大的话，处理速度仍然很慢。在对整个训练集执行梯度下降算法时，要做的是必须处理整个训练集才能进行一步梯度下降法，然后再处理m个训练样本，又进行下一步。所以在处理500万个训练样本进行一步梯度下降之前，先让梯度下降法处理一部分，你的算法速度回更快。

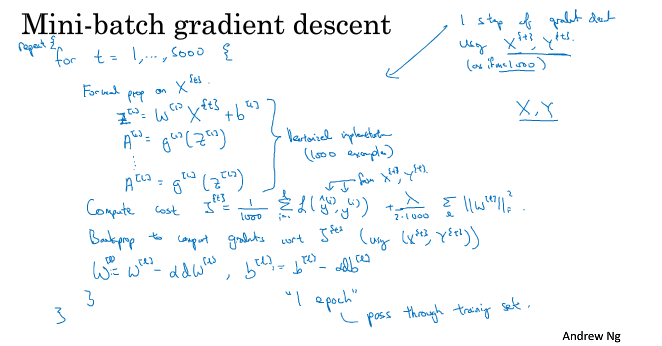


可以把训练集分割成小一点的子训练集，这些子集就叫做Mini-batches。假设每一个子集中只有1000个样本，那就可以对m个样本依次进行划分为一个个子集。将子集依次记作，，，。对Y也行进行相应的处理，，，。Mini-batch的t组成了和，这就是1000个训练样本，包含对应的输入和输出。

Batch梯度下降法指的是之前讲过的梯度下降算法，同时处理整个训练集，并且这个名字来源于能够看到整个batch训练集的样本被处理。准确的讲，Mini-batch是下一节讲到的算法，你每次同时处理的是单个的mini-batch，和，而不是同时处理全部的X和Y训练集。那么mini-batch的原理是什么，为了在训练集上运行mini-batch梯度下降法，运行一下代码：

|  |
| --- |
| (因为有5000个各有1000个样本的mini-batch)  （首先对输入进行前向传播）      …….        (使用和来计算代价的梯度，并且更新权重) |

在for循环里面，对和执行一步梯度下降法，假设你有一个1000个样本的训练集，前面已经很熟悉一次性处理完的方法，那么对于这个子训练集不需要一个明确的for循环去处理1000个样本，而是使用向量化去几乎同时处理1000个样本。以上表格中就是使用mini-batch梯度下降法训练样本的一步（一代epoch的训练）。Epoch这个词意味着只是一次遍历了整个训练集，使用batch梯度下降法，一次遍历训练集只能让你做一个梯度下降。使用mini-batch梯度下降法一次遍历训练集，就是一个epoch能让你做5000个梯度下降。（个人理解：batch梯度下降就是做普通的一次遍历数据集做一次梯度下降，每次执行梯度下降都要遍历一次数据集，而mini-btach就是将整个数据集子集化，遍历一次数据集，则对里面的5000个训练子集都分别进行过一次梯度下降。对于巨大的数据集来说，想要找到梯度值，必须把所有的数据集都遍历完全一遍，这样的时间等代价太大，尽管这样可以得到全局最优解。还是用mini-batch，用到的思想就是利用部分样本代替所有的样本）正常的来说你是希望多几次遍历训练集，你还需要为另一个while循环设置另一个for循环，所以你可以一直处理训练集，指导最后收敛到一个合适的精度。

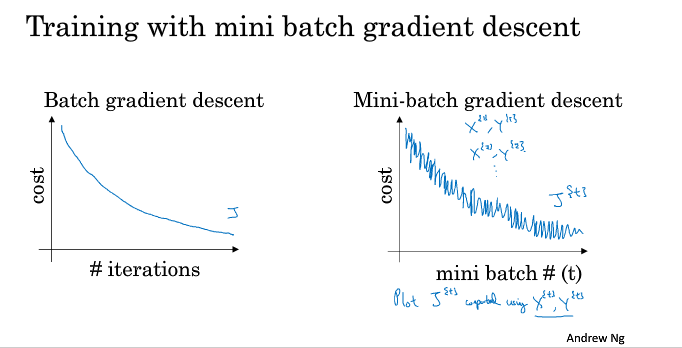


如果你有一个丢失的训练集，mini-batch梯度下降法比batch运行地更快，所以几乎在训练巨大的数据集的时候都会用到mini-batch。

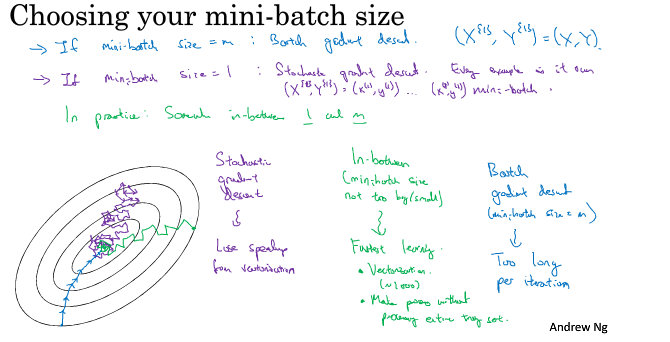
2.2 理解mini-batch梯度下降法

上节学习如何利用mini-batch来开始处理训练集和开始梯度下降，即使你只处理了部分训练集，继续学习如何使用，更好的理解其作用和原理。

使用mini-batch梯度下降法，如果你做出代价函数关于迭代次数的图，并不是每次迭代都是下降的，特别是在每次迭代中你要处理的是和，如果要做出成本函数的图，而只和和有关，也就是每次迭代下你都在训练不同的样本集或者说训练不同的mini-batch，可能会看下如图的结果，整体走向朝下，但有很多的噪声。所以你作的图，因为在训练mini-batch梯度下降法时，会经过多代，所以会看到这样的曲线。没有每次迭代都下降是不要紧的，但是走势应该向下，有很多的噪声的原因也许是和是比较容易计算的mini-batch，因此成本会低一些，不过也许处于偶然，和比较难计算一点，或许你需要一些错误标签的样本，这样一来，成本会更高一些，所以才会出现这些摆动。你需要决定的变量之一是mini-batch的大小，极端情况下，如果mini-batch的大小=m，其实就是batch gradient desend，就只存在,所以把mini-batch设为m，则可以为batch gradient desend。另一种极端情况下，如果mini-batch的大小=1，那么就是一种新的算法，叫做随机梯度下降法，每个样本都是独立的mini-batch，,这就是第一个训练样本。



接下来看一下在两种极端情况下，成本函数的优化情况，如果左下角图是你想要最小化的成本函数的轮廓，最小值大概在圆心处，batch法从某处开始，如蓝色路线所示，相对噪声较低，幅度也大一些，一直在继续找最小值，相反在随机梯度下降中，从某点开始，只对一个样本进行梯度下降，大部分时候向着全局最小值靠近，有时候又会原理，因为刚好那个样本指的方向不对，因此随机梯度下降算法是有很多噪声的，平均来看，最终会靠近最小值，有时候也会方向错误，因为随机梯度下降法永远不会收敛，而是会一直在最小值附近波动，但是不会到达最小值并且在这停留，如紫色路线。实际上你选择的mini-batch大小应该在1到m之间，原因在于，如果你使用batch梯度下降，每个迭代需要处理大量训练样本，特别是在数据量大的时候，单次迭代耗时太长，如果训练样本不大，batch训练效果会很好。相反，如果使用随机法，如果你只要处理一个样本，通过减小学习率，噪声会被改善减少很多，但随机梯度下降的一大缺点是你会失去所有向量化带给你的加速，因为一次只处理一个样本，这样效率太低了。所以应该选择不大不小的mini-batch值，学习率达到最快，两个好处，得到了大量向量化，处理样本快的多，另一方面，不需要等所有样本处理完，就可以开始后续工作。上一节例子中，每代训练集允许我们采取5000个梯度下降步骤。

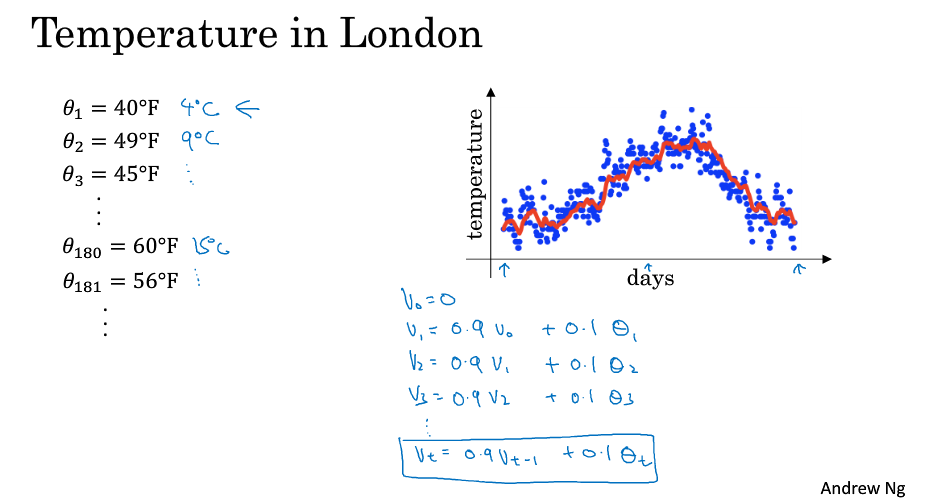


所以使用合适mini-batch值，代价函数不会总朝向最小值靠近，但它比随机下降要更持续的靠近最小值的方向，它也不一定在很小的范围内收敛或者波动，如果这样，我们可以通过缩减学习率来解决这个问题。

那么mini-batch中间值怎么选择，指导原则：首先如果训练集太小（小于2000个样本），直接使用batch，可以快速处理整个训练集。训练集较大时候，mini-batch大小一般为64-512，考虑到电脑内存设置和使用的方式，如果mini-batch是2的次方，代码会运行的快一点。最后要注意的是，你的mini-batch中，和要符合CPU/GPU内存，取决于你的应用方向和数据集大小。如果你处理的mini-batch和CPU/GPU内存不相符，不管你用什么方法处理数据，算法的表现急转下，惨不忍睹。Mini-batch值也是一个重要的变量，需要尝试才能找到最合适的有效的代价函数表现最好的值。

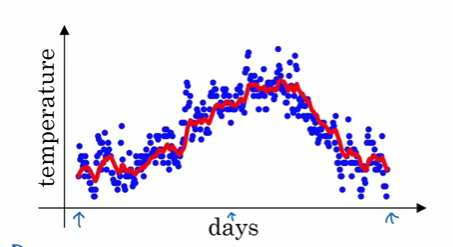
2.3 指数加权平均

介绍几个优化算法，比梯度下降法快，要理解这些算法，需要用到指数加权平均。举个例子，伦敦气温变化随着日期一年变化如图。看起来有些杂乱，如果要计算趋势的话，也就是温度的局部平均值，或者移动平均值，要做的是：



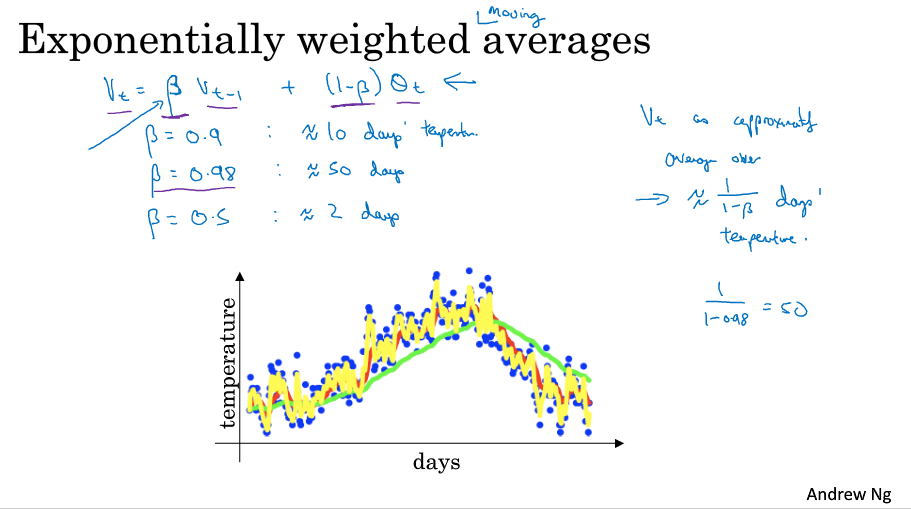
计算下面式子：

使用0.9乘上乘以前一天计算出来的加权数，最后加上0.1乘以当天的温度。那么得到一个新的加权数，从而计算下一个加权数。将以上计算出来的，，作图，可以得到红色曲线，得到了移动平均值，每日温度的指数加权平均值。



将0.9这个常数记为，。由于我们以后要考虑的原因，在计算时可以看成近似平均值，大概是天的每日温度。

比如说，如果代入计算，那么计算的就是十天的平均值，作图为红色部分；如果，那么就是初略的计算了一下平均50天的温度，作图得到绿色曲线。高一点的值结果注意，得到的曲线要平坦一些，原因是多平均了几天的温度，考虑的天数多一些，曲线波动更小，更加平坦。缺点就是曲线进一步右移，因为平均的值更多一些，指数加权平均公式在温度变化时，适应地更缓慢一些，出现一定的延迟。并且，使用，给前面天数的权重比较大，只有0.02权重给了当天的气温值，所以较大，温度变化时指数加权平均值适应地更慢一些。如果，，那么作图得到黄色曲线，因为仅平均了这两天的温度，平均的数据太少，得到的曲线有更多的噪声，所以异常值会多一些，但是这个曲线能够更快的适应温度变化。



指数加权平均，在统计学里又称为指数加权移动平均值，简称指数加权平均数。通过调整这个参数，在后面的算法学习中可以发现这是一个很重要的参数，可以取得稍微不同的效果，往往中间每个值最好。比如红色曲线就更好的显示了温度的变化趋势。

以上就是计算指数加权平均数的基本原理。

2.4 理解指数加权平均

指数加权平均数，是几个能够帮助你训练神经网络的优化算法的关键一环，这一节探讨算法的本质作用。指数加权平均数计算的关键公式：，来理解一下如何使用这个公式计算出每次温度的平均值。使，那么：

……

分析的时候倒过来分析，

关于计算的，看下组成有前100天的数据，并且是加和，关于画图的方法，首先我们有关于一些日期的温度，关于t的一个表示的每日温度的图像，然后我们构建一个指数衰减函数，从图像右边为0.1 开始，往左一次是，…..一次类推，得到的这个指数衰减函数 ，那么我们要计算的就是讲两个函数对应的相乘并且加和。

累加项的所有系数之和相加起来为1，或者逼近1，称之为修正偏差，因为有修正偏差，这才是指数加权平均数。

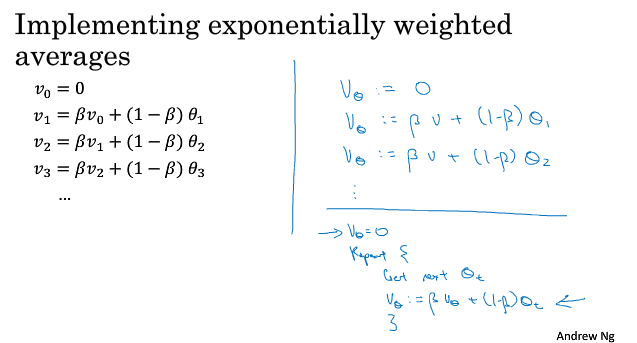
问题：到底多少天的温度才能更好的表示趋势呢。

大体来说，如果，这里，，，换句话说10天后，曲线的高度下降 相当于峰值的，因此当，加入你在计算加权平均数，即相当于，所以只关注了过去10的温度。因为10天后，权重下降不到当日权重的三分之一。相反，如果 ，那么大概多少次方结果会等于0.35左右，结果表示，所以前50天这个数值比大，数值会快速衰减，实质上这是一个下降幅度很大的函数，可以看做 了50的温度。将这个例子中代入，可以发现，，所以指数加权平均可以计算大约天的温度。也就是说，根据一些常数，你就能大概知道能够平均多少天的温度。不过这是思考方向，不是严格的数学证明。

实际中如何执行指数加权平均计算，在实际代码中，可以只用一个变量，可以每次更新成新值。

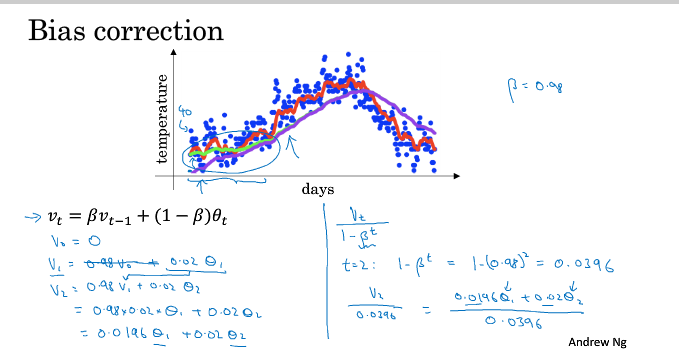
换种写法，那么就是：

指数加权平均的好处是占用极少的内存，电脑内存中只占用一行数据，然后把最新数据代入公式中，不断覆盖就行了。因为这个原因，计算加权指数平均数也只占单行数字的存储和内存，当然也许不是最好的，也不是最精确的计算最近平均的，但是你可以使用计算窗，直接计算过去10天的总和，再计算平均值，如此往往可以最好的估测，但是需要存的就是所有的温度数据，必须占用更多的内存，之行更加复杂，计算成本高。计算多个变量的平均值，从计算和内存效率来说，是个好办法。



2.5 指数加权平均的偏差修正

偏差修正（bias correction），可以让计算平均数运算更加准确。在之前的例子中，当时，对应曲线是绿线，但如果执行计算加权平均的公式，得到的并不是绿线，而是紫线，具有一定的偏差，注意到紫线的起点比较低，看看如何处理。我们将等带进去进行真正的运算发现，，得到的值会小的多，所以第一天估测不准，，那么要远小于数据，所以也不能很好估测这一年前两天的温度。有个办法可以修正，让估测更准确一点，特别是初期，不用，而使用,t就是现在的天数。例如，当t=2时，表示前两天的平均温度。当t越大的时候，就越趋近于1，所以当天数很大的时候，可以适当估计，当t很小时，会出现很大的误差。当t很大时，紫线和绿线基本重合。



不过在开始阶段，修正偏差可以帮助你更好的预测，在计算指数加权平均数的大多时候，很多人几乎不在乎初期，熬过初期阶段，如果你关心初始时期的偏差，在刚计算指数加权平均数的时候偏差修正可以在早期得到更好的估测。

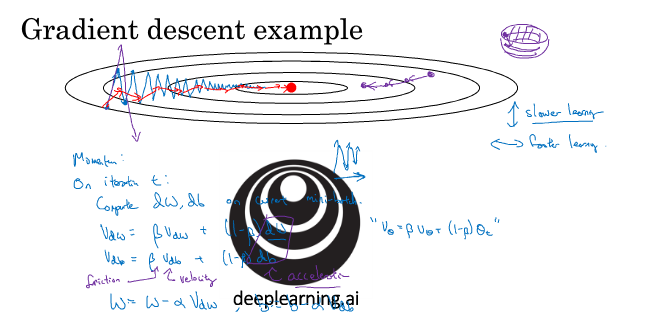
2.6 动量梯度下降法Momentum

Momentum梯度下降法，运行速度几乎总是快于标准的梯度下降算法，基本想法就是计算梯度的指数加权平均数，并利用该梯度更新你的权重，也就是说，计算梯度的方式发生了改变。

例如如果要优化一个成本函数，轮廓如图，红色代表最小值的位置，假设从蓝色点开始进行梯度下降法的一次迭代，无论是batch还是mini-batch，也许会指向蓝色箭头方向，再继续进行下一步，最后下降的结果如蓝色折线，梯度下降法要计算很多很多的步骤，最后慢慢摆动到最小值。这种上下摆动减少了梯度下降的效率，无法使用更大的学习率，如果使用较大的学习率，可能步长过大，结果偏离函数的范围，为了避免波动过大，所以使用一个较小的学习率。另一个看待问题的角度，在梯度下降学习中，我们希望横轴的参数跨度大，希望能够快速的移向最小值，而纵轴上希望学习慢一点，因为不想要这些摆动，所以使用Momentum需要做的就是，在每次迭代中，确切的说在第t次迭代的过程中，用现有的mini-batch来计算微分dW，db，再接下来就是使用指数加权平均数的方式来得到我们更改计算方式后的梯度，步骤如下：

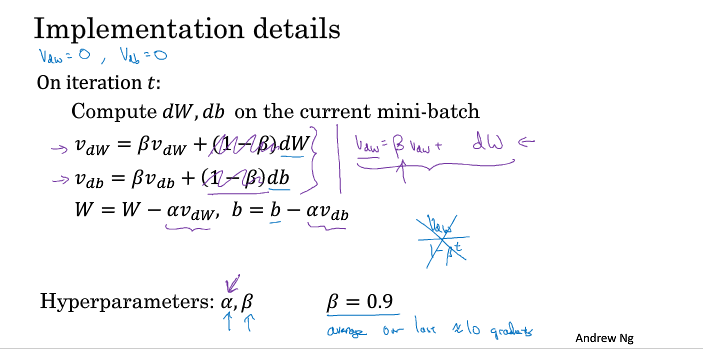
|  |
| --- |
| (计算dW 的移动平均数)    (更新参数) |

这样就可以减缓梯度下降的幅度，你可能看到很多幅度较小的折线向最小值靠近，会发现这些纵轴上的摆动平均值接近于0，所以在纵轴方向，速度慢一点，平均过程中，正负数相互抵消，而在横轴方向，所有的微分都指向横轴方向，因此横轴方向的平均值仍然较大，因此用算法几次迭代后，会发现Momentum梯度下降最终纵轴方向的摆动小了，横轴方向运动更快，因此你的梯度下降走了一条更接近的路，在抵达最小值的路上减少了摆动。



Momentum的一个本质，就是如果你要最小化碗状函数，能够最小化碗状函数，你可以想象从山上往下滚的一个球，提供了加速度，这些项就相当于速度，微分项给了这个球加速度，此时球正向山下滚，球因为加速度越滚越快，而因为稍小于1，表现出一些摩擦力，所以球不会无限加速下去，所以不像梯度下降法，每一步都独立于前面的步骤，你的球可以向下滚，获得动量。

算法如下图，所以你有两个超参数，学习率以及参数，控制指数加权平均数，最常用的值使0.9，所以平均了前十次的迭代的梯度，为0.9时效果不错，也可以尝试不同的值。那么关于修正偏差，使用，实际上人们不这么做，因为梯度下降经过10次之后，已经过了初始阶段，不再是一个具有偏差的预测，实际上，在使用梯度下降法以及Momentum时，人们不会受到修正偏差的困扰。初始参数，注意是一个与dW有相同维度的参数，同理也是。



最后一点，如果看到相关Momentum梯度下降法相关资料，通常看到的公式可能省去了这个项，最后得到

用这组计算公式可以看出，缩小了倍，相当于乘以，如果使用这种梯度更新值的话，α要根据相应变化，实际上，二者效果都不错，只是会影响到学习率α的最佳值，但是后面这个没有那么自然，因为你调整这个参数，就会影响到，也许还要修改学习率，所以一般用之前的公式，但是两个公式里的都是使用0.9最好。只是两种公式的学习率α有所不同。

Momentum梯度下降肯定要好于之前的没有Momentum的梯度下降算法，还可以使用其他办法来加速梯度下降过程。

2.7 RMSprop均方根梯度下降法

RMSprop全称root mean square均方根，也可以加速梯度下降，仍然是前面的例子，关于一个损失函数进行梯度下降时，横轴在不断的推进，纵轴上有波动。上一节关于Momentum中，我们希望损失函数值随着梯度下降步长的进行，值的波动小一点，所以在参数更新时对参数进行求移动平均值，使得更新参数时参数的步长变大。在RMSprop中，关于神经网络梯度下降过程参数有很多，假设纵轴影响幅度的参数是b，横轴为W，很明显参数很多，多维度，这是为了便于理解只使用了W和b，可能是W1 W2或其他重要的参数。

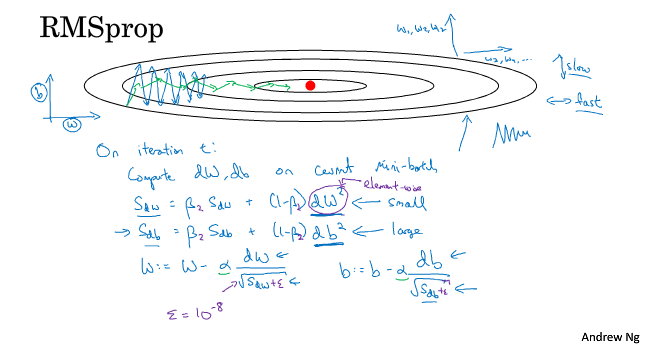
同理这种情况下，希望减缓b方向的学习，即纵轴方向，同时加快或者至少不是减缓横轴的方向。减缓一个方向的学习，即是使该方向步长小一点。RMSprop算法可以实现这一点。在第t次迭代，该算法会照常计算，整个计算过程步骤如下：

|  |
| --- |
| (保留微分平方的加权平均数)    （1）  （2） |

理解一下原理，在横轴方向即W方向，希望学习速度快，而在垂直方向也就是b方向学习慢些，减缓纵轴上的摆动，所以有了和，我们希望会相对较小，所以我们在更新W时给下面除一个较小的数，同时又希望比较大，所以除以一个较大的数。根据上图中折线的变化，可以看出垂直方向梯度要比水平方向大的多，所以斜率在b方向特别大，所以这些微分中，db较大，dW较小。因为函数的倾斜程度在b方向上要大于横轴方向，所以比较大，比较小。结果就是纵轴上的更新要被一个较大的数除，就能消除摆动，而水平方向上就被一个较小的数除。（个人理解，将根号放到学习率的分母下，理解更好理解，即是两个参数的更新都修改了学习率，在（1）（2）两个公式中进行理解，将以及分别理解成W和b参数的更新加速度，前者分母较小，所以加速度比较大，那么更新速度就会更快，而后者速率减缓，即每次迭代变化的量比较小。）

PMSprop最后的影响就是更新后的梯度下降过程如绿色折线，即在W方向的上的变化较大，b方向的变化比较缓慢摆动较小。还有一个影响就是你可以使用更大的学习率，,从而加快你的算法速率，而不会在纵轴上偏离。

这里我们讲解使用的是W和b参数，实际上处于一个高纬度的参数环境中，所以需要消除摆动的垂直方向的摆动实际上也许是很多参数的合集，例如W2,W3,W17，水平维度可能是W3，W4等。实际上dW和db都是一个高纬度向量。直觉则应该是在你要消除摆动的维度中最终你要计算一个更大的和值，即是平方和微分的加权平均值，最后去掉那些有摆动的方向。



均方根，将微分进行平方，最后使用到平方根，最后关于一些细节，下一节将Momentum和RMSprop结合起来，两种算法里面都有超参数，避免混淆，这里使用，保证后面结合的时候我们采用同一个超参数。另一点，要确保你的算法不会除以0，如果平方根趋近于0怎么办，所以通常在分母加上一个比较小的，确保数值稳定，是多少没有关系，无论什么原因，都不会除以一个很小的数。

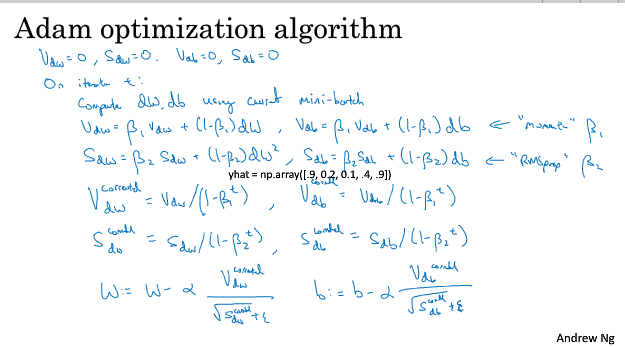
所以Momentum和RMSprop有的共同点就是可以消除梯度下降中的摆动，并且允许你使用一个更大的学习率来加快下降的过程，加快算法的学习速度。

2.8 Adam优化算法

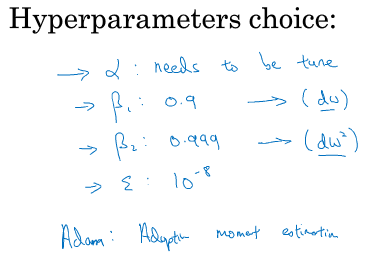
在深度学习历史上，提出了很多优化算法，并且很好的解决了一些问题，但后来这些优化算法被指出并不能一般化，并不适用于多种神经网络。关于优化算法就有很多质疑，有人认为Momentum梯度下降法很好用，很难想出更好的办法。所以RMSprop和Adam就是少有的两种经得住考验的算法，已被证明适用于不同的深度学习结构。

Adam算法基本上就是将Momentum和RMSprop算法结合在一起，使用Adam算法，首先要初始化：

|  |
| --- |
| (明显这里使用mini-batch梯度下降法进行计算)  (与后面RMSprop中用到的β超参数进行区分)  (使用RMSprop再对参数进行更新) |



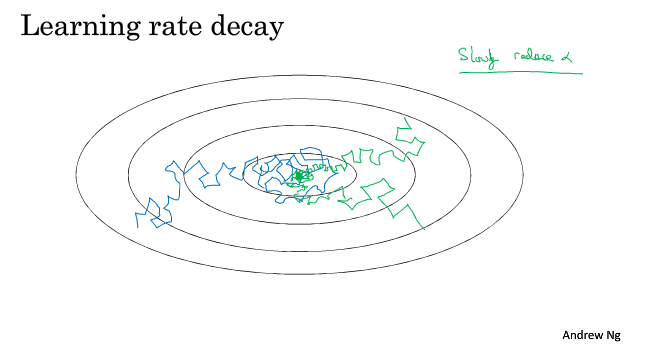
相当于Momentum更新了超参数，RMSprop更新了超参数，一般使用Adam算法时候，要计算偏差修正，对以及都要进行修正，最后更新超参数W以及b，使用的方法则是同时用到了两种算法的计算结果，结合在一起。Adam是一种很有效的优化算法，并且被证明能有效的适用于不同神经网络，适用于广泛的结构。



本算法中有很多超参数，超参数学习率α很重要，也经常需要调试，的缺省值是0.9，Adam的推荐者建议值是0.999，关于的选择没那么重要，建议使用，并不需要设置它，因为它并不会影响算法表现。在使用Adam时，使用缺省值即可。Adam，Adaptive Moment Estimation，用来计算dW,db的指数加权平均数，叫做第一炬，叫第二炬。

2.9 学习率衰减

加快学习算法的一个办法就是随时间慢慢减少学习率，称为学习率衰减。为什么要计算学习率衰减，假设你要使用mini-batch梯度下降法，mini-batch数量不多，大概64或者128个样本，下降朝向最小值时迭代过程中会有噪音，但是不会精确的收敛，算法最后在附近摆动，并不会真正的收敛，原因就是使用了固定的学习率。不同的mini-batch中有噪音，如果要慢慢减少学习率α，初期学习率还比较大，步长比较大，学习相对比较快，后来随着α变小，步伐变小，最后在最小值附近的一块区域里摆动，而不是在训练过程中，大幅度的在最小值附近摆动。所以慢慢减小α的本质在于在学习初期能承受较大的步伐，开始收敛时，小的学习率可以步伐小一些。



学习率衰减方法：

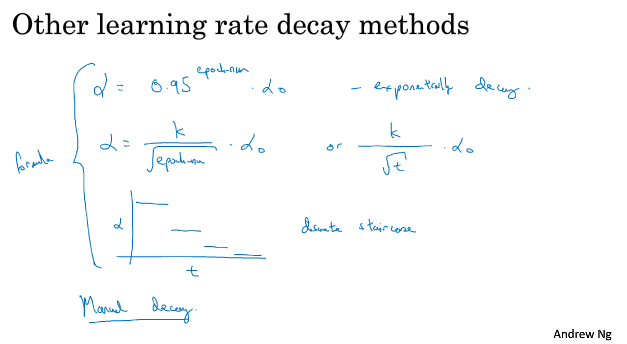
decay\_rate表示衰减率，表示初始学习率。

衰减率是另一个需要调整的超参数，根据上面公式，学习率呈递减趋势，如果要使用衰减学习率，需要尝试不同的值包括超参数以及超参数decay\_rate。

记得一代epoch要遍历一次数据，有一个数据集时先拆分成不同的mini-batch，第一次遍历训练集叫做第一代，第二次则为第二代，以此类推。

关于学习率的衰减方法还有其他公式，比如：

指数衰减，相当于一个小于的值，学习率呈指数下降。



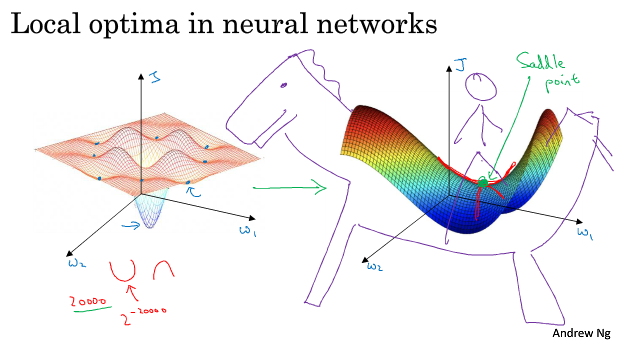
以及还有离散学习的学习率衰减法，就是某个段用某个学习率值。

除了使用公式以外，也可以手动衰减，手动调整只有模型数量小的时候有用，但是也有人闲着没事这样做。现在有多个选择关于衰减方式来控制学习率，关于如此多的超参数怎么系统选择，下一章内容讲。

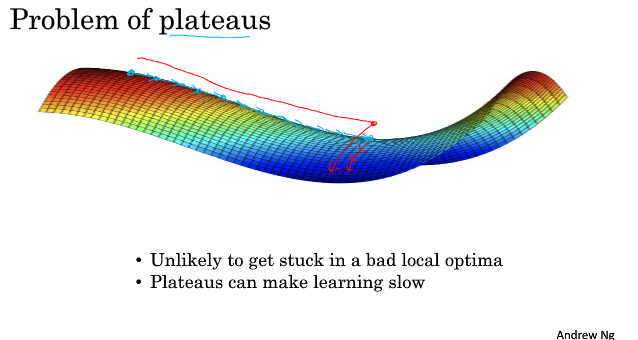
实际上学习率衰减也不是必须的工作，仅仅设定一个固定的，然后去调整一下，学习率衰减的确大有好处，有时候可以加快训练，但这不是率先尝试的内容。

2.10 局部最优的问题

早期深度学习，人们总是担心优化算法会困在极差的局部最优，实际上并没有这种问题。首先说到局部最优，想到下面这幅图，像一个帽子，分别参数为W1，W2，纵轴为平面的高度即损失函数，在图中我们可以看到，到处都分布着局部最优，梯度下降法或者其他的算法可能会困在一个局部最优中，而不会到达全局最优，也就是帽子顶尖。然而这只是一个低维图，影响了我们的理解。事实上如果你要创建神经网络，通常梯度为0的点并不是左图的局部最优点，实际上损失函数的0梯度点通常是鞍点，右图蓝点。



一个具有高维空间的函数，如果梯度为0那么在每个方向，它可能是凸函数或者凹函数，如果你在两万维空间中，那么想要得到局部最优，就得这个点的两万个方向都聚集在这个点，发生的几率很小，更有可能遇到在这个点，有的方向曲线向上弯曲，有的向下弯曲，因此在高维空间中，更有可能碰到鞍点saddle point，而不太可能碰到局部最优。所以我们对低维空间的大部分直觉并不能应用到高维空间中，适用于其他算法。



如果局部最优不是问题，那问题是什么？平稳段会减缓学习，平稳段是一块区域，导数长时间接近于0，蓝色线路，慢慢抵达平稳段的最低点，因为左边和右边都有随机干扰，之后算法才走出平稳段，向下继续行走。

1. 首先不太可能困在局部最优中，条件是你在训练较大的神经网络，存在大量参数，并且成本函数J被定义在较高的维度空间中。
2. 平稳段使得学习十分缓慢，这也是像Momentum和RMSprop这样的算法能够加速学习算法的地方，在这些情况下，更成熟的方向，如Adam能够加快学习速度，让你今早往下走出平稳段。

因为你的网络要解决优化问题，说实话面临如此高维度空间，没有人能直观感觉到是什么样子，理解还在不断发展。