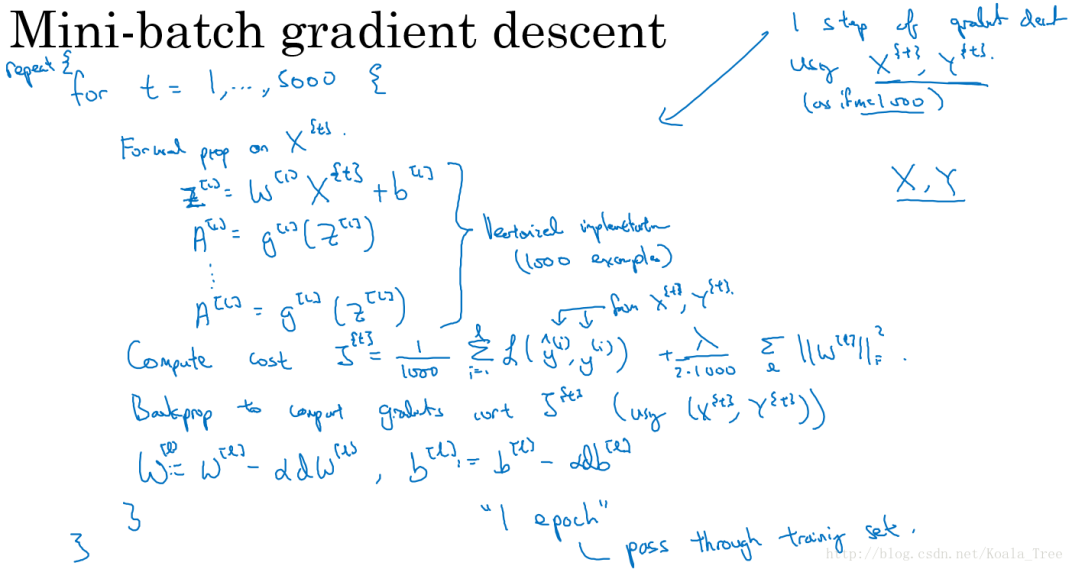
**1. Mini-batch 梯度下降法**

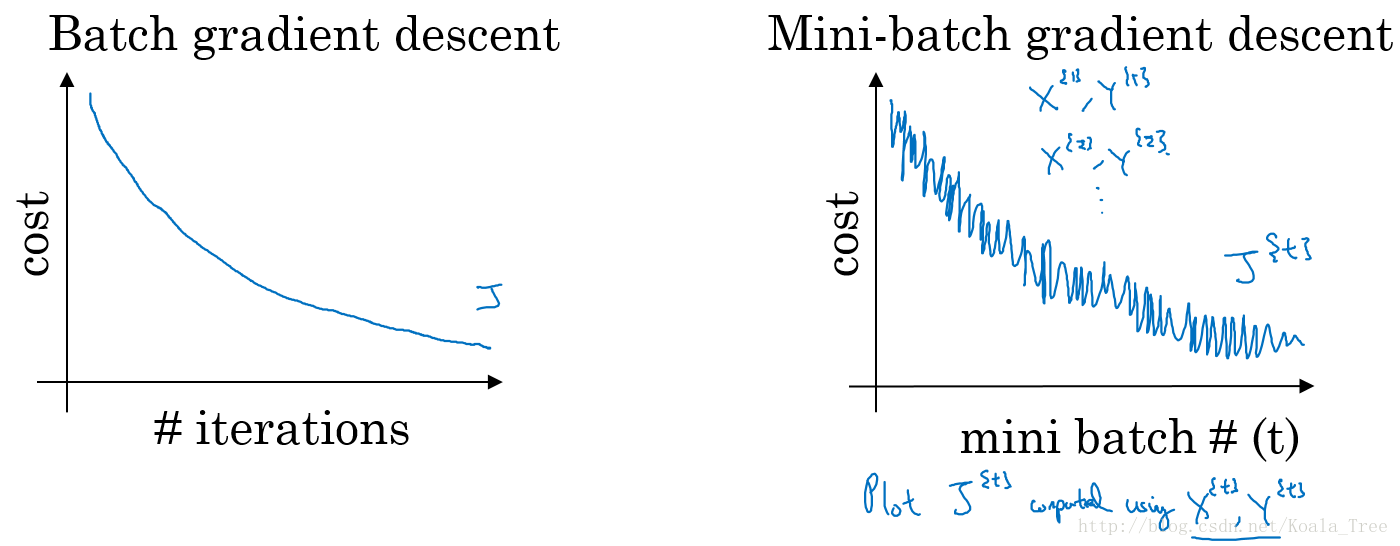
对整个训练集进行梯度下降法的时候，我们必须处理整个训练数据集，然后才能进行一步梯度下降，即每一步梯度下降法需要对整个训练集进行一次处理，如果训练数据集很大的时候，如有500万或5000万的训练数据，处理速度就会比较慢。

但是如果每次处理训练数据的一部分即进行梯度下降法，则我们的算法速度会执行的更快。而处理的这些一小部分训练子集即称为Mini-batch。



**2.理解Mini-batch 梯度下降法**

普通的batch梯度下降法和Mini-batch梯度下降法代价函数的变化趋势：



**batch梯度下降：**

对所有m个训练样本执行一次梯度下降，每一次迭代时间较长；

Cost function 总是向减小的方向下降。

**随机梯度下降：**

对每一个训练样本执行一次梯度下降，但是丢失了向量化带来的计算加速；

Cost function总体的趋势向最小值的方向下降，但是无法到达全局最小值点，呈现波动的形式。

**Mini-batch梯度下降：**

选择一个1<size<m 的合适的size进行Mini-batch梯度下降，可以实现快速学习，也应用了向量化带来的好处。

Cost function的下降处于前两者之间。

**Mini-batch 大小的选择：**

如果训练样本的大小比较小时，如m⩽2000m⩽2000时 —— 选择batch梯度下降法；

如果训练样本的大小比较大时，典型的大小为：

26、27、⋯、21026、27、⋯、210；

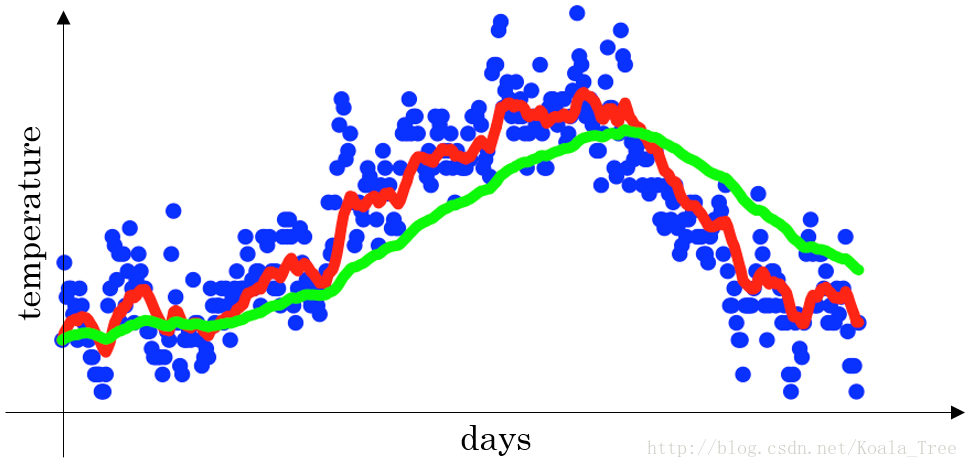
Mini-batch的大小要符合CPU/GPU内存。

**3.指数的加权平均**

**指数加权平均的关键函数：**

**vt=βvt−1+(1−β)θt**

下图是一个关于天数和温度的散点图：



当β=0.9时，指数加权平均最后的结果如图中红色线所示；

当β=0.98时，指数加权平均最后的结果如图中绿色线所示；

当β=0.5时，指数加权平均最后的结果如下图中黄色线所示；

**4.理解指数加权平均**

例子，当β=0.9β=0.9时：

v100=0.9v99+0.1θ100v99=0.9v98+0.1θ99v98=0.9v97+0.1θ98…

v100=0.9v99+0.1θ100v99=0.9v98+0.1θ99v98=0.9v97+0.1θ98…

展开：

v100=0.1θ100+0.9(0.1θ99+0.9(0.1θ98+0.9v97))=0.1θ100+0.1×0.9θ99+0.1×(0.9)2θ98+0.1×(0.9)3θ97+⋯

v100=0.1θ100+0.9(0.1θ99+0.9(0.1θ98+0.9v97))=0.1θ100+0.1×0.9θ99+0.1×(0.9)2θ98+0.1×(0.9)3θ97+⋯

上式中所有θ前面的系数相加起来为1或者接近于1，称之为偏差修正。

**指数加权平均的实现**

v0=0

v1=βv0+(1−β)θ1

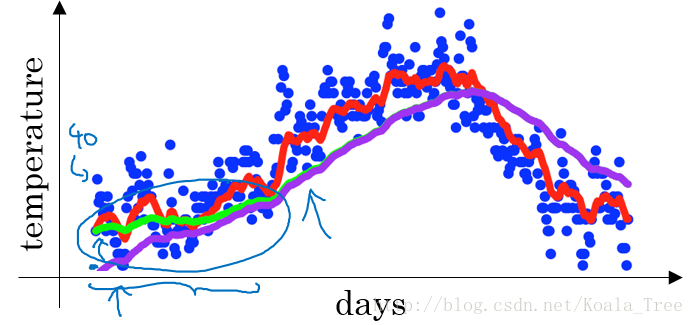
v2=βv1+(1−β)θ2

v3=βv2+(1−β)θ3…

因为，在计算当前时刻的平均值，只需要前一天的平均值和当前时刻的值，所以在数据量非常大的情况下，指数加权平均在节约计算成本的方面是一种非常有效的方式，可以很大程度上减少计算机资源存储和内存的占用。

**5. 指数加权平均的偏差修正**

在我们执行指数加权平均的公式时，当β=0.98时，我们得到的并不是图中的绿色曲线，而是下图中的紫色曲线，其起点比较低。



**偏差修正：**

使用vt/1−βt

当t=2时：

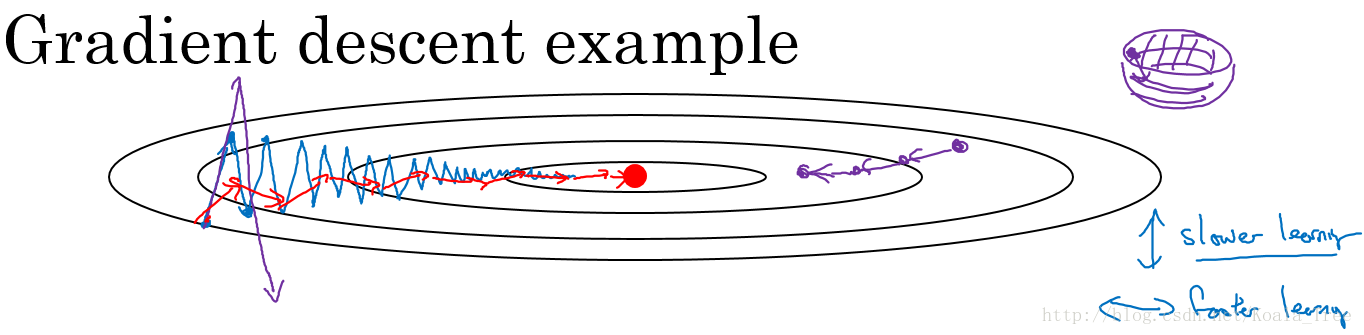
1−βt=1−(0.98)2=0.0396

v2/0.0396=（0.0196θ1+0.02θ2）/0.0396

偏差修正得到了绿色的曲线，在开始的时候，能够得到比紫色曲线更好的计算平均的效果。随着tt逐渐增大，βtβt接近于0，所以后面绿色的曲线和紫色的曲线逐渐重合了。

**6.动量梯度下降法**

动量梯度下降的基本思想就是计算梯度的指数加权平均数，并利用该梯度来更新权重。

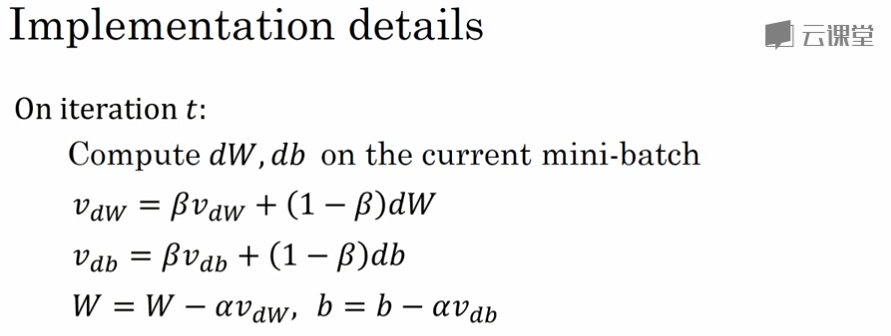


在利用梯度下降法来最小化该函数的时候，每一次迭代所更新的代价函数值如图中蓝色线所示在上下波动，而这种幅度比较大波动，减缓了梯度下降的速度，而且我们只能使用一个较小的学习率来进行迭代。

如果用较大的学习率，结果可能会如紫色线一样偏离函数的范围，所以为了避免这种情况，只能用较小的学习率。

但是我们又希望在如图的纵轴方向梯度下降的缓慢一些，不要有如此大的上下波动，在横轴方向梯度下降的快速一些，使得能够更快的到达最小值点，而这里用动量梯度下降法既可以实现，如红色线所示。

算法：



**算法理解：**

在对应上面的计算公式中，将Cost function想象为一个碗状，想象从顶部往下滚球，其中：

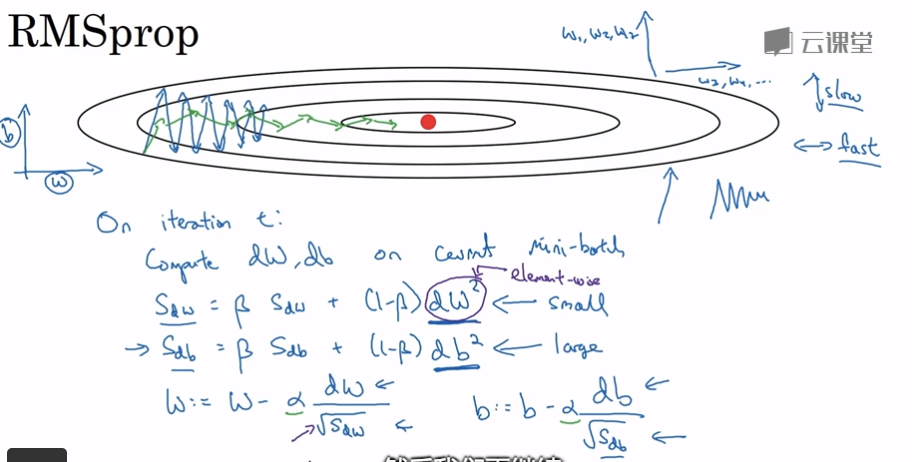
微分项dw,db想象为球提供的加速度；

动量项vdw,vdb相当于速度；

小球在向下滚动的过程中，因为加速度的存在速度会变快，但是由于β的存在，其值小于1，可以认为是摩擦力，所以球不会无限加速下去。

**7. RMSprop**

RMSprop也是一种可以加快梯度下降的算法。



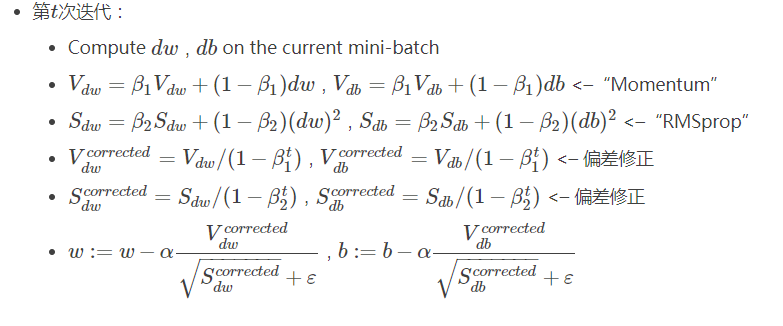
这里假设参数b的梯度处于纵轴方向，参数w的梯度处于横轴方向（当然实际中是处于高维度的情况），利用RMSprop算法，可以减小某些维度梯度更新波动较大的情况，如图中蓝色线所示，使其梯度下降的速度变得更快，如图绿色线所示。

**8.Adam优化算法**

Adam 优化算法的基本思想就是将 Momentum 和 RMSprop 结合起来形成的一种适用于不同深度学习结构的优化算法。

算法实现：

初始化：Vdw=0，Sdw=0，Vdb=0，Sdb=0；



α：需要进行调试；

β1：常用缺省值为0.9，dw的加权平均；

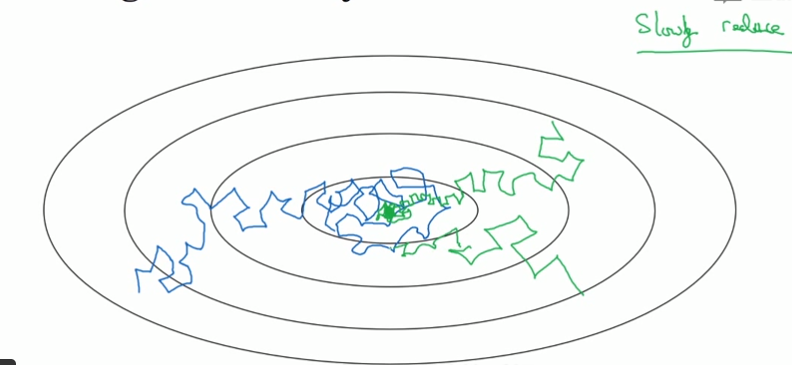
β2：推荐使用0.999，dw2的加权平均值；

ε：推荐使用10−8；

**9.学习率衰减**

在我们利用 mini-batch 梯度下降法来寻找Cost function的最小值的时候，如果我们设置一个固定的学习速率αα，则算法在到达最小值点附近后，由于不同batch中存在一定的噪声，使得不会精确收敛，而一直会在一个最小值点较大的范围内波动，如下图中蓝色线所示。

但是如果我们使用学习率衰减，逐渐减小学习速率αα，在算法开始的时候，学习速率还是相对较快，能够相对快速的向最小值点的方向下降。但随着αα的减小，下降的步伐也会逐渐变小，最终会在最小值附近的一块更小的区域里波动，如图中绿色线所示。



学习率衰减的实现

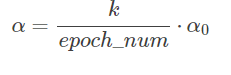
常用：



指数衰减：



其他：



**10.局部最优**

①在高维空间中，如2000维，不太可能出现局部最优的情况，因为局部最优要求这20000个维度的梯度都为0，这是很小概率的时间。

②真正困扰优化问题的并不是局部最优，而是在鞍点附近的停滞区问题。

如下图，鞍点并不是全局最优，但梯度下降会在鞍点附近花费很多时间，先降到鞍点，在鞍点附近受到扰动在降到其他地方。故采用动量，adam，RMSprop等方法来加快停滞区的训练。

