## P61 小批量梯度下降

如果m很大，例如达到百万数量级，训练速度往往会很慢，因为每次迭代都要对所有样本进行进行求和运算和矩阵运算。我们将这种梯度下降算法称为Batch Gradient Descent。

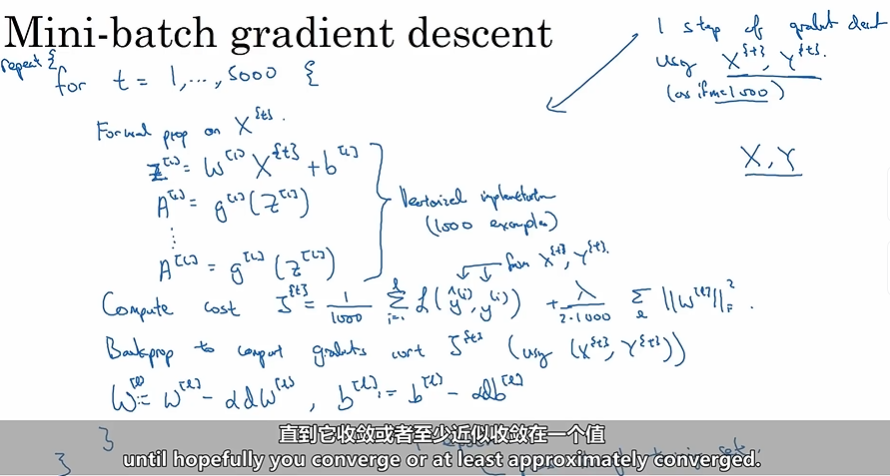
把m个训练样本分成若干个子集，称为mini-batches，这样每个子集包含的数据量就小了，例如只有1000，然后每次在单一子集上进行神经网络训练，速度就会大大提高。这种梯度下降算法叫做Mini-batch Gradient Descent。

假设总的训练样本个数m=5000000，其维度为(nx,m)。将其分成5000个子集，每个mini-batch含有1000个样本。我们将每个mini-batch记为X{t}，其维度为(nx,1000)。相应的每个mini-batch的输出记为Y{t}（**使用大括号{}表示批次**），其维度为(1,1000)，且t=1,2,⋯,5000t=1,2,⋯,5000。（每五千个就更新一次w，b而不是每五百万个数据才更新一次wb）

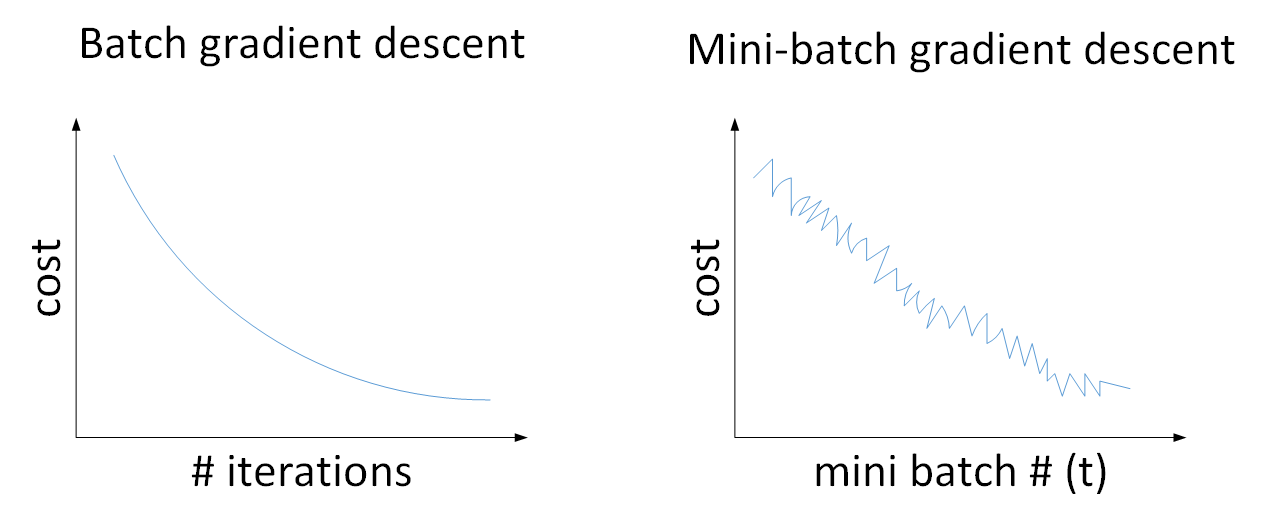
Mini-batches Gradient Descent的实现过程是先将总的训练样本分成T个子集（mini-batches），然后对每个mini-batch进行神经网络训练，包括Forward Propagation，Compute Cost Function，Backward Propagation，循环至T个mini-batch都训练完毕。只有在所有m个（T组）训练样本计算完才算是经历了一个epoch。

对于Batch Gradient Descent而言，一个epoch只进行一次梯度下降算法；而Mini-Batches Gradient Descent，一个epoch会进行T次梯度下降算法。

Ps：对于Mini-Batches Gradient Descent，可以进行多次epoch训练。而且，每次epoch，最好是将总体训练数据重新打乱、重新分成T组mini-batches，这样有利于训练出最佳的神经网络模型。



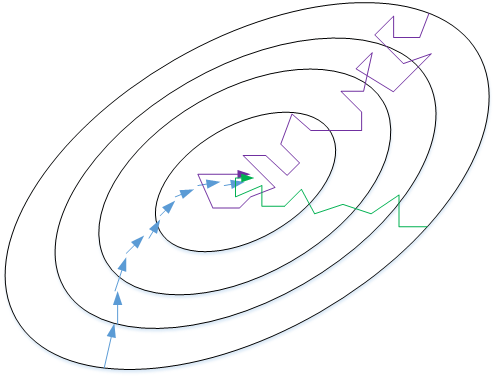
## P62 理解小批量梯度下降



对于一般的神经网络模型，使用Batch gradient descent，随着迭代次数增加，cost是不断减小的。然而，使用Mini-batch gradient descent，随着在不同的mini-batch上迭代训练，其cost不是单调下降，而是受类似noise的影响，出现振荡。但整体的趋势是下降的，最终也能得到较低的cost值。原因：不同的mini-batch之间存在差异，可能某一子集是好的，而某一子集是差的，包含了噪声。

**如何选择每个mini-batch的大小：**

1. 如果mini-batch size=m即为Batch gradient descent。
2. 如果mini-batch size=1，即为Stochastic gradient descent。共有m个子集。



如上图所示：蓝色的线代表Batch gradient descent，紫色的线代表Stochastic gradient descent，绿色的线代表mini-batch gradient descent。

Batch gradient descent会比较平稳地接近全局最小值，但是因为使用了所有m个样本，每次前进的速度有些慢。

Stochastic gradient descent每次前进速度很快，但是路线曲折，有较大的振荡，最终会在最小值附近来回波动，难以真正达到最小值处。且一个子集只有一个数据不能采用向量化来提高运算速度。

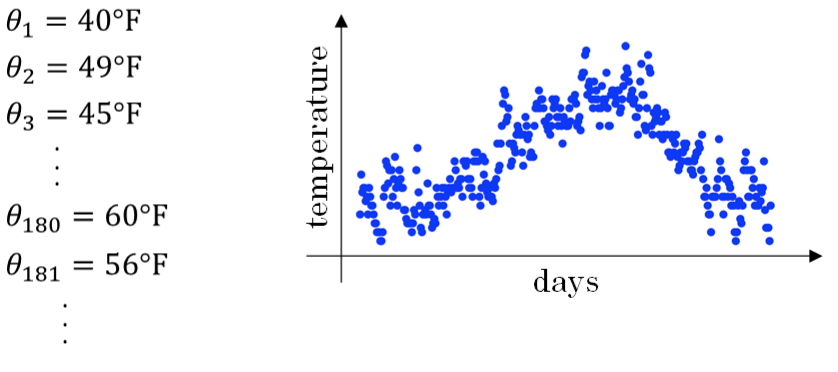
mini-batch gradient descent每次前进速度较快，且振荡较小，基本能接近全局最小值。

实际使用中，mini-batch size不能设置得太大（Batch gradient descent），也不能设置得太小。

Ps：若样本数量不大如m<=2000则直接使用batch gradient descent，如果总体样本数量m很大时，建议将样本分成许多mini-batches。mini-batch size也是一个超参数，推荐常用的mini-batch size为64,128,256,512。这些都是2的幂。之所以这样设置的原因是计算机存储数据一般是2的幂，这样设置可以提高运算速度。

## P63 指数加权平均

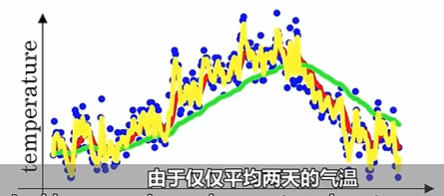
接下来要介绍一些比梯度下降更快的算法。首先要用到指数加权平均也叫做指数加权滑动平均。



**V0=0，当成第0天的气温值。**

**第一天的气温与第0天的气温有关：**

**V1=0.9V0+0.1θ1**



ββ值决定了指数加权平均的天数，近似表示为：

1/1−β

**黄线 每两天气温进行平均β=0.5，红每十天β=0.9，绿每五十天β=0.98**

**一般形式：** **β是一个超参数**

滑动平均为什么在测试过程中被使用

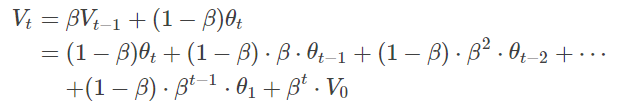
滑动平均可以使模型在测试数据上更健壮（robust）。采用随机梯度下降算法训练神经网络时，使用滑动平均在很多应用中都可以在一定程度上提高最终模型在测试数据上的表现。

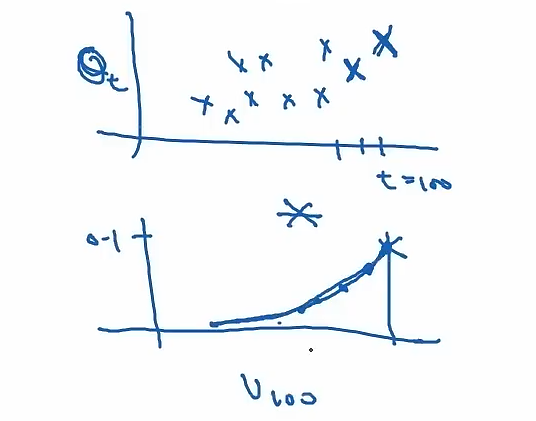
对神经网络边的权重 weights 使用滑动平均，得到对应的影子变量 shadow\_weights。在训练过程仍然使用原来不带滑动平均的权重 weights，不然无法得到 weights 下一步更新的值，又怎么求下一步 weights 的影子变量 shadow\_weights。之后在**测试过程**中使用 shadow\_weights 来代替 weights 作为神经网络边的权重，这样在测试数据上效果更好。因为 shadow\_weights 的更新更加平滑，**对于随机梯度下降而言，更平滑的更新说明不会偏离最优点很远**；**对于梯度下降 batch gradient decent，我感觉影子变量作用不大，因为梯度下降的方向已经是最优的了**，loss 一定减小；**对于 mini-batch gradient decent，可以尝试滑动平均，毕竟 mini-batch gradient decent 对参数的更新也存在抖动**。

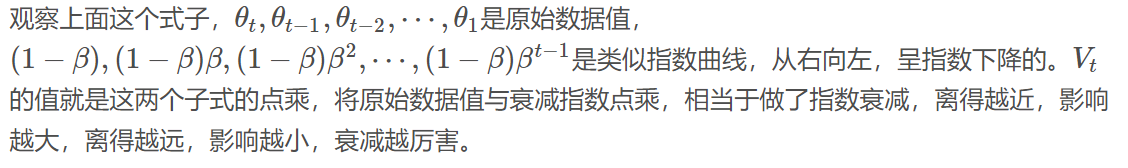
设 decay=0.999, 一个更直观的理解，在最后的 1000 次训练过程中，模型早已经训练完成，正处于抖动阶段，而滑动平均相当于将最后的 1000 次抖动进行了平均，这样得到的权重会更加 robust。

## P64 理解指数加权平均

指数加权平均一般公式：

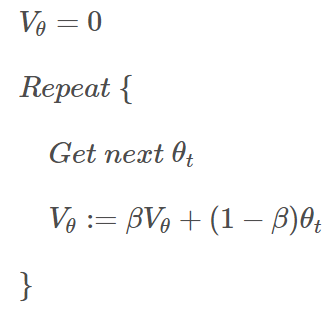


上下做点乘然后相加

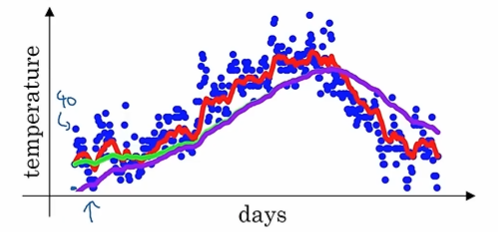


若β=0.9 0.910 ≈ 0.35 ≈1/e （β）1/（1-β） = 1/e

实际应用中，为了减少内存的使用，我们可以使用这样的语句来实现指数加权平均算法：

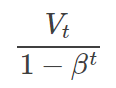


## P65 指数加权平均的偏差修正



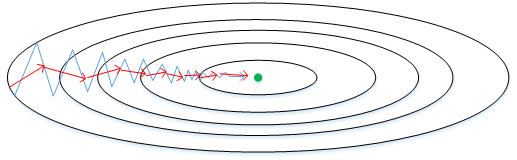
未进行偏差修正的曲线实际是紫色的。由于初始化v0 = 0，起步会很慢。

偏移校正（bias correction），即在每次计算完Vt后，对Vt进行下式处理：



Ps：机器学习中，**偏移校正并不是必须的**。因为，在迭代一定次数后（t较大），Vt受初始值影响微乎其微，紫色曲线与绿色曲线基本重合。所以，一般可以忽略初始迭代过程，等到一定迭代之后再取值，这样就不需要进行偏移校正了。

## P66 动量梯度下降

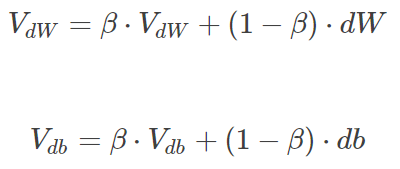


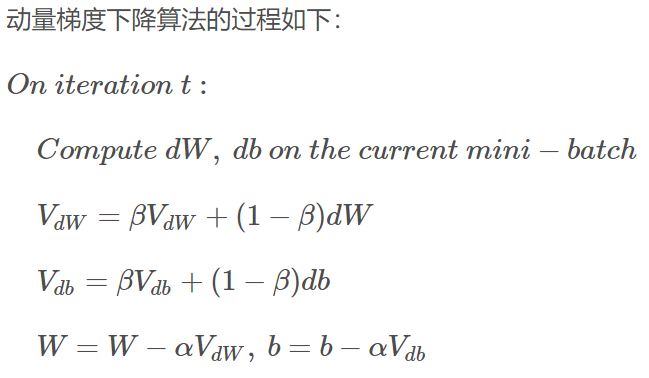
上图**红线为动量梯度下降**，蓝线为**普通梯度下降**。

我们想要在横轴上加快下降速度，纵轴上减小波动幅度。

动量梯度下降算法，其速度要比传统的梯度下降算法快很多。

做法是：**在每次训练时，对梯度进行指数加权平均处理**，然后用得到的梯度值更新权重W和常数项b。

具体的实现过程：

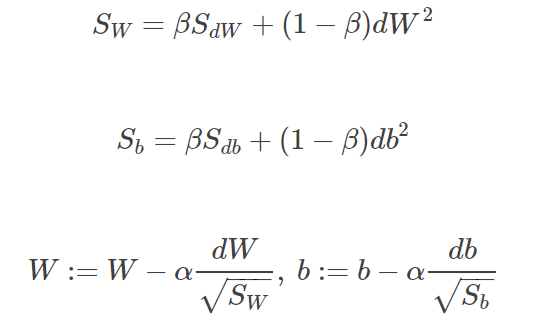


一般设置β=0.9 即指数加权平均前10次迭代的数据，实际应用效果较好。

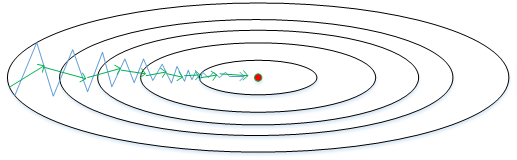
另外，关于偏移校正，可以不使用。因为经过10次迭代后，随着滑动平均的过程，偏移情况会逐渐消失。

## P67 RMSprop（root mean square）

RMSprop是另外一种优化梯度下降速度的算法。每次迭代训练过程中，其权重W和常数项b的更新表达式为：



**理解：**

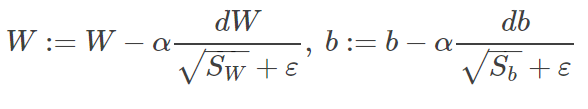


**假设**上图中水平方向为W的方向，垂直方向为b的方向。（实际中不一定，都有可能。比如震荡的是一部分w和一部分b，水平的是一部分w和一部分b）

垂直方向（b）上振荡较大。水平方向w更新较慢。梯度dW较小，而db较大。为了解决这个问题，在更新W和b的表达式中，变化值dW/根号Sw变大。而db/√Sb变小。总得来说，就是如果哪个方向振荡大，就减小该方向的更新速度，从而减小振荡。如果更新速度过小，又会加快更新。

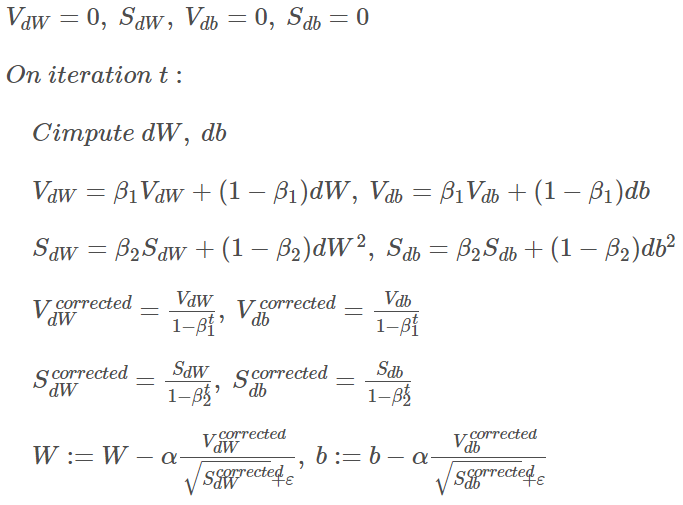
**总之RMSprop能够降低梯度下降和小批量梯度下降的震荡，并且可以使用更大的学习率来加快学习速度。**

**PS：为了避免除以零，会给分母加上一个极小的值，其中，ε=10−8，或者其它较小值。**



## P68 Adam优化算法（adaptive moment estimation自适应矩估计）

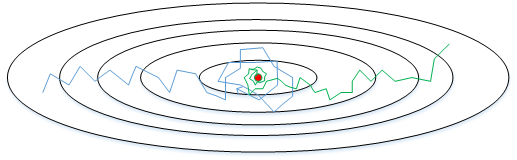
本质上是将**动量梯度下降算法**和**RMSprop**各自的优点结合起来，使得神经网络训练速度大大提高。



**注意事项：**Adam算法包含了几个超参数，分别是：α,β1,β2,ε。其中，β1通常设置为0.9，β2通常设置为0.999，ε通常设置为10-8。且通常只修改学习率α，其他参数都用默认值。

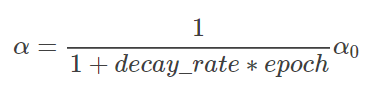
## P69 学习率衰减（learning rate decay）

减小学习率α也能有效提高神经网络训练速度，这种方法被称为learning rate decay。



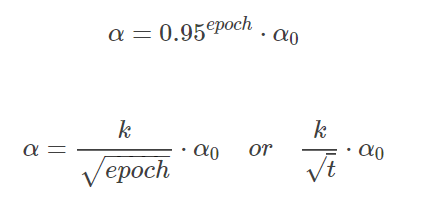
Ps：蓝色折线表示使用恒定的学习因子α，绿色折线表示使用不断减小的α

Learning rate decay中α可由下列公式得到：



Ps：decay\_rate是另一超参数，epoch是训练完所有样本的次数

除了上面计算α的公式之外，还有其它可供选择的计算公式：

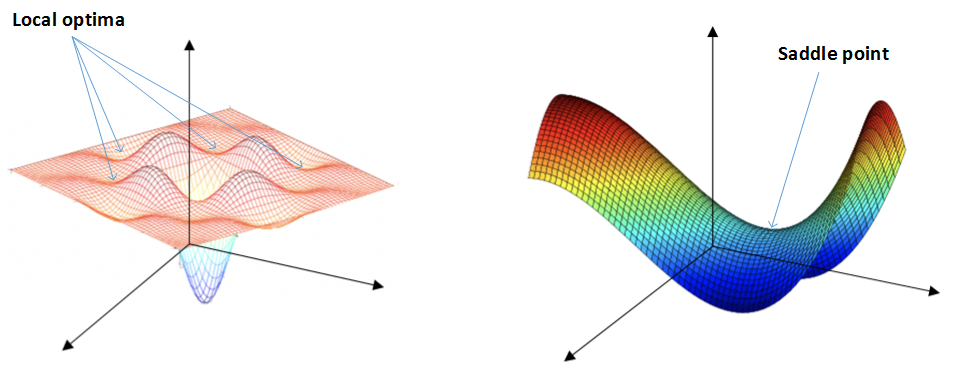


Ps：其中，k为可调参数，t为mini-bach number。

除此之外，还可以设置αα为关于t的离散值，随着t增加，α呈阶梯式减小。当然，也可以根据训练情况灵活调整当前的αα值，但会比较耗时间。

## P70 局部最优解问题（local optima）

在使用梯度下降算法不断减小cost function时，可能会得到局部最优解（local optima）而不是全局最优解（global optima）。之前我们对局部最优解的理解是形如碗状的凹槽，如下图左边所示。但是在神经网络中，local optima的概念发生了变化。准确地来说，大部分梯度为零的“最优点”并不是这些凹槽处，而是形如右边所示的马鞍状，称为saddle point。也就是说，**梯度为零并不能保证都是convex（极小值），也有可能是concave（极大值）**。特别是在神经网络中参数很多的情况下，所有参数梯度为零的点很可能都是右边所示的马鞍状的saddle point，而不是左边那样的local optimum。



类似马鞍状的plateaus会降低神经网络学习速度。Plateaus是梯度接近于零的平缓区域，如下图所示。在plateaus上梯度很小，前进缓慢，到达saddle point需要很长时间。到达saddle point后，由于随机扰动，梯度一般能够沿着图中绿色箭头，离开saddle point，继续前进，只是在plateaus上花费了太多时间。

**总的来说，关于local optima，有两点总结：**

* **只要选择合理的强大的神经网络，一般不太可能陷入local optima**
* **Plateaus可能会使梯度下降变慢，降低学习速度**

值得一提的是，上文介绍的动量梯度下降，RMSprop，Adam算法都能有效解决plateaus下降过慢的问题，大大提高神经网络的学习速度。

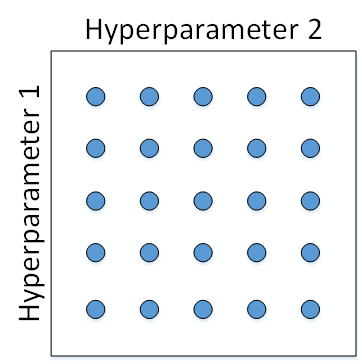
## P71 参数调整过程

学习率α；动量梯度下降的β（一般0.9）；adam优化算法的β1（0.9），β2（0.999），ε（10-8）；

网络层数layers，每层的隐藏单元数目hidden units；

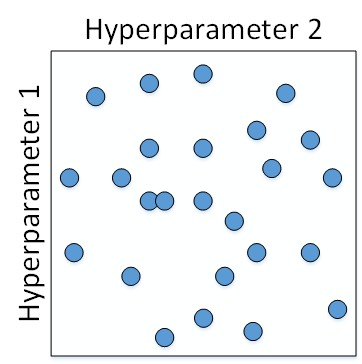
学习率衰减 learning rate decay；mini-batch size；

Ps：调参数时，**优先级：红色最高，**其次**黄色**，再就**绿色.** 超参数重要性的排名并不是绝对的，具体情况，具体分析。



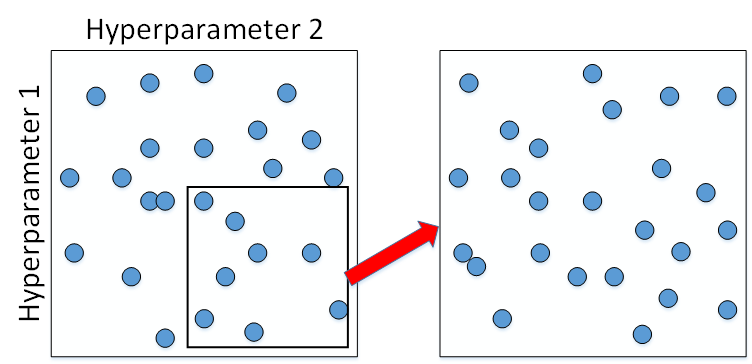
**传统取样**

这种做法在参数比较少的时候效果较好。但是在深度神经网络模型中，我们一般不采用这种均匀间隔取点的方法，比较好的做法是使用随机选择。也就是说，对于上面这个例子，我们随机选择25个点，作为待调试的超参数，如下图所示：



在实际应用中完全不知道哪个参数更加重要的情况下，随机采样的方式能有效解决这一问题，但是均匀采样做不到这点。

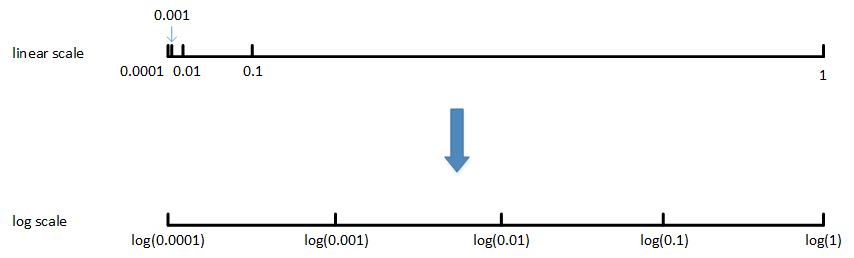
在经过随机采样之后，我们可能得到某些区域模型的表现较好。然而，为了得到更精确的最佳参数，我们应该继续对选定的区域进行由粗到细的采样（coarse to fine sampling scheme）。也就是放大表现较好的区域，再对此区域做更密集的随机采样。例如，对下图中右下角的方形区域再做25点的随机采样，以获得最佳参数。



## P72 如何正确取样超参数

部分超参数如**#layers**和**#hidden units**是可以进行尺度均匀随机采样的，因为它们每次的变化尺度都是一致的（比如每次变化1，刻度是均匀的）。但是某些超参数需要选择不同的合适尺度进行随机采样。

对另外一部分超参数，若它的待调范围是[0.0001，1].若使用均匀随机采样。则有百分之90的采样点分布在[0.1,1]之间，只有10%分步在[0.0001, 0.1]。而实际应用中最佳的值可能主要在[0.0001, 0.1]内。而[0.1, 1]范围内值效果并不好。应该在[0.0001, 0.1]细分更多。



r = -4 \* np.random.rand() r属于[-4,0]. A = 10r A属于[10-4,1]

一般解法是，如果线性区间为[a, b]，令m=log(a)，n=log(b)，则对应的log区间为[m,n]。对log区间的[m,n]进行随机均匀采样，然后得到的采样值r，最后反推到线性区间，即。就是最终采样的超参数。相应的Python语句为：

m = np.log10(a)

n = np.log10(b)

r = (m-n)\* np.random.rand()

r = np.power(10,r)

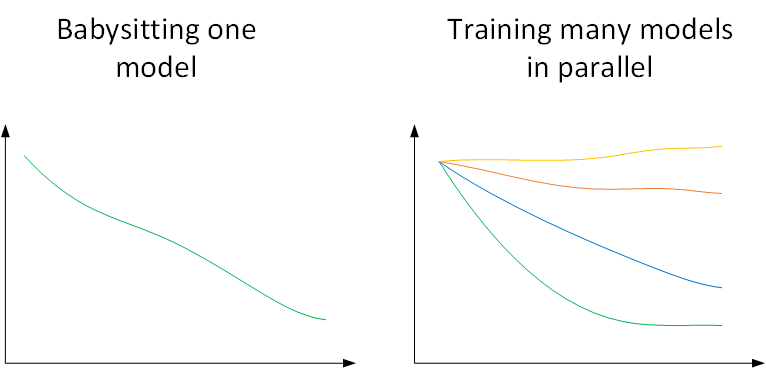
另外，**动量梯度因子β**也是一样，在超参数调试的时候也需要进行非均匀采样。一般的取值范围在[0.9, 0.999]之间，那么的1-β取值范围就在[0.1，0.001]之间。那么直接对在[0.001, 0.1]区间内进行log变换即可。

这里解释下为什么也需要向那样做非均匀采样。假设从0.9000变化为0.9005，那么1/（1-β）基本没有变化，近似指数加权平均的个数还是10次迭代的数据。但假设从0.9990变化为0.9995，那么前后差别1000。越接近1，指数加权平均的个数越多，变化越大。所以对接近1的区间，应该采集得更密集一些。

## P73 实践中超参数的调整（熊猫和鱼子酱模式）

经过调试选择完最佳的超参数并不是一成不变的，一段时间之后（例如一个月），需要根据新的数据和实际情况，再次调试超参数，以获得实时的最佳模型。

在训练深度神经网络时，一种情况是受计算能力所限，我们只能对一个模型进行训练，调试不同的超参数，使得这个模型有最佳的表现。我们称之为Babysitting one model。另外一种情况是可以对多个模型同时进行训练，每个模型上调试不同的超参数，根据表现情况，选择最佳的模型。我们称之为Training many models in parallel。



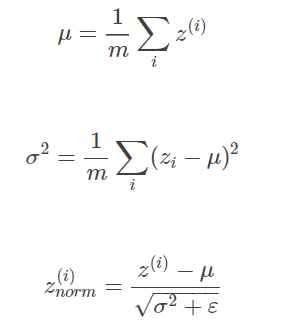
因为第一种情况只使用一个模型，所以类比做Panda approach；第二种情况同时训练多个模型，类比做Caviar approach。使用哪种模型是由计算资源、计算能力所决定的。一般来说，对于非常复杂或者数据量很大的模型，使用Panda approach更多一些。

## P74 网络中的正则化激活

Batch Normalization，对隐藏层的各个输入进行标准化处理。Batch Normalization不仅可以让调试超参数更加简单，而且可以让神经网络模型更加“健壮”。也就是说较好模型可接受的超参数范围更大一些，包容性更强，使得更容易去训练一个深度神经网络。

**对a还是z做正则化一直存在争议，普遍对z进行归一化。**

Batch Normalization对第层隐藏层的输入做如下标准化处理，忽略层数上标

****

这样，使得该隐藏层的所有输入均值为0，方差为1。但是，大部分情况下并不希望所有的均值都为0，方差都为1，也不太合理。通常需要对进行进一步处理：

C:\Users\xingxinda\AppData\Roaming\Tencent\Users\1756775636\QQ\WinTemp\RichOle\_[WQA{]@H9`NSVC2})_HR1H.png

γ和β可以让z～的均值是任何想要的数。例如，令：

C:\Users\xingxinda\AppData\Roaming\Tencent\Users\1756775636\QQ\WinTemp\RichOle\]%{D4BLU{Q0QQ0C_F)%6K5N.png

则z～ = z。可见，设置γ和β为不同的值，可以得到任意的均值和方差。

**PS：**输入的标准化处理Normalizing inputs和隐藏层的标准化处理Batch Normalization是有区别的。Normalizing inputs使所有输入的均值为0，方差为1。

而Batch Normalization可使各隐藏层输入的均值和方差为任意值。

实际上，从激活函数的角度来说，如果各隐藏层的输入均值在靠近0的区域即处于激活函数的线性区域，这样不利于训练好的非线性神经网络，得到的模型效果也不会太好。

## P75 将Batch Norm拟合到神经网络中

Batch Norm的β和adam的β1，β2不一样

接下来将研究如何把Bath Norm应用到整个神经网络中。

对于L层神经网络，经过Batch Norm的作用，整体流程如下：



实际上，Batch Norm经常使用在mini-batch上，这也是其名称的由来。

值得注意的是，因为Batch Norm对各隐藏层有去均值的操作，所以常数项b可以消去，其数值效果完全可以由Batch Norm来实现。因此，我们在使用Batch Norm的时候，可以忽略各隐藏层的常数项b。在使用梯度下降算法时，分别对，和进行迭代更新。

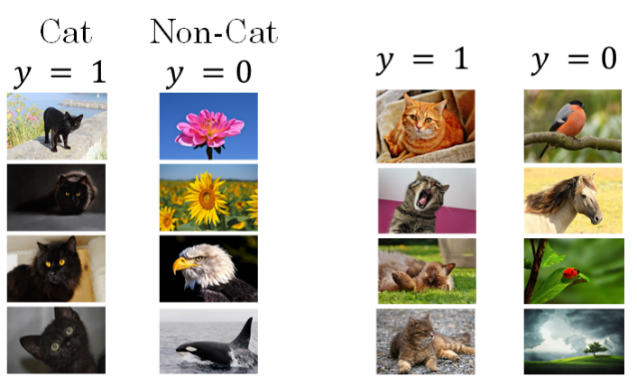
除了传统的梯度下降算法之外，还可以用动量梯度下降、RMSprop或者Adam等优化算法。

## P76 Batch Norm为什么有效

我们可以把输入特征做均值为0，方差为1的规范化处理，来加快学习速度。而Batch Norm也是对隐藏层各神经元的输入做类似的规范化处理。总的来说，**Batch Norm不仅能够提高神经网络训练速度，而且能让神经网络的权重W的更新更加“稳健”**，尤其在深层神经网络中更加明显。比如**神经网络很后面的W对前面的W包容性更强，即前面的W的变化对后面W造成的影响很小**，整体网络更加健壮。

举个例子来说明，假如用一个浅层神经网络（类似逻辑回归）来训练识别猫的模型。如下图所示，提供的所有猫的训练样本都是黑猫。然后，用这个训练得到的模型来对各种颜色的猫样本进行测试，测试的结果可能并不好。

其原因是训练样本不具有一般性（即不是所有的猫都是黑猫），这种训练样本（黑猫）和测试样本（猫）分布的变化称之为**covariate shift协变量偏移**。



对于这种情况，如果实际应用的样本与训练样本分布不同，即发生了covariate shift，则一般是要对模型重新进行训练的。

在神经网络，尤其是深度神经网络中，covariate shift会导致模型预测效果变差，重新训练的模型各隐藏层的和均产生偏移、变化。而Batch Norm的作用恰恰是减小covariate shift的影响，让模型变得更加健壮，鲁棒性更强。

Batch Norm减少了各层、之间的耦合性，让各层更加独立，实现自我训练学习的效果。也就是说，如果输入发生covariate shift，那么因为Batch Norm的作用，对个隐藏层输出进行均值和方差的归一化处理，和更加稳定，使得原来的模型也有不错的表现。针对上面这个黑猫的例子，如果我们使用深层神经网络，使用Batch Norm，那么该模型对花猫的识别能力应该也是不错的。

从另一个方面来说，**Batch Norm也起到轻微的正则化**（regularization）效果。具体表现在：

**每个mini-batch都进行均值为0，方差为1的归一化操作**

**每个mini-batch中，对各个隐藏层的添加了随机噪声，效果类似于Dropout**

**mini-batch越小，正则化效果越明显**

**不要把BN看作正则化方法，而是看作归一化方法加速学习。**

## P77 Batch Norm在测试时的注意事项

训练过程中，Batch Norm是对单个mini-batch进行操作的，但在测试过程中，如果是单个样本，该如何使用Batch Norm进行处理呢？

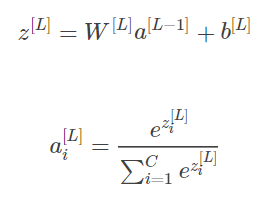
在**测试过程中，如果只有一个样本，求其均值和方差是没有意义**的，就需要对和进行估计。估计的方法有很多，理论上我们可以将所有训练集放入最终的神经网络模型中，然后将每个隐藏层计算得到的和直接作为测试过程的和来使用。但是，实际应用中一般不使用这种方法，而是使用我们之前介绍过的指数加权平均（exponentially weighted average）的方法来预测测试过程单个样本的和。

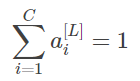
指数加权平均的做法很简单，**对于第L层隐藏层，考虑所有mini-batch的μ和σ2在该隐藏层L下的和，然后用指数加权平均的方式得到新的μ和σ2来对当前样本进行预测**。这样就实现了对测试过程单个样本的均值和方差估计。最后，再利用训练过程得到的和值计算出各层的值。

## P78 Softmax Regression回归

目前我们介绍的都是二分类问题，神经网络输出层只有一个神经元，表示预测输出是正类的概率，则判断为正类，则判断为负类。

对于多分类问题，用C表示种类个数，神经网络中输出层就有C个神经元，即。其中，每个神经元的输出依次对应属于该类的概率，即。为了处理多分类问题，我们一般使用Softmax回归模型。Softmax回归模型输出层的激活函数如下所示：



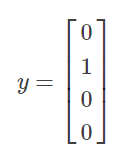
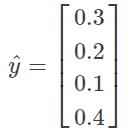
输出层每个神经元的输出对应属于该类的概率，满足

a[L]维度为(C,1)

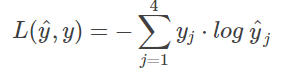
ps：与之对应的时hardmax 只保留概率最大者为1，其余都为零，当C为2时则退化成了logistic回归。

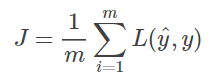
## P79 训练一个softmax 分类器

假如C=4，某个样本的预测输出和真实输出为：

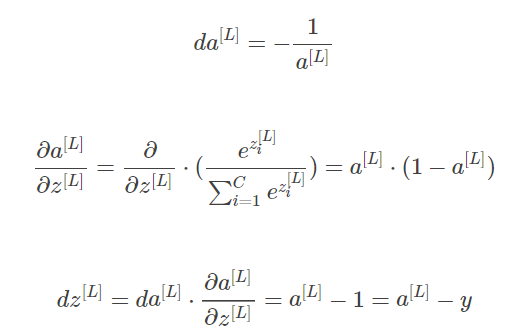


预测结果是第一类概率最大，而真实结果是第二类。

定义loss function：

m个样本的cost function：，其预测输出向量的维度为(4, m)。

softmax回归的反向传播过程和逻辑回归相比只有输出层不一样。因此先推到输出层：



结果发现和逻辑回归一样。可见的表达式与二元分类结果是一致的，虽然推导过程不太一样。然后就可以继续进行反向传播过程的梯度下降算法了，推导过程与二元分类神经网络完全一致。

## P80 深度学习框架

深度学习框架有很多，例如：

Caffe/Caffe2，CNTK，DL4J，Keras，Lasagne，mxnet，Paddle，TensorFlow，Theano，Torch。

一般选择深度学习框架的基本准则是：

**Ease of programming(development and deployment)**

**Running speed**

**Truly open(open source with good governance)**

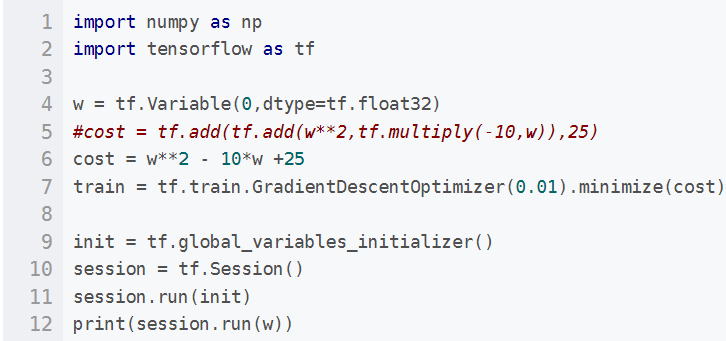
实际应用中，我们应该根据自己的需求选择最合适的深度学习框架。

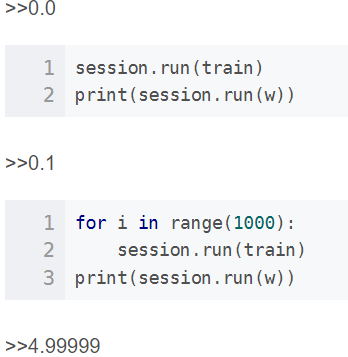
## P81 tensorflow

举个例子来说明，例如cost function是参数w的函数：

C:\Users\xingxinda\AppData\Roaming\Tencent\Users\1756775636\QQ\WinTemp\RichOle\K4D%WLG`UW3DFW`X]]BCK4T.png

使用TensorFlow对cost function进行优化，求出最小值对应的w，程序如下：





针对上面这个例子，如果对w前的系数用变量x来代替，程序如下：



结果跟之前是一样的。除此之外，我们还可以更改x即cofficients的值，而得到不同的优化结果w

