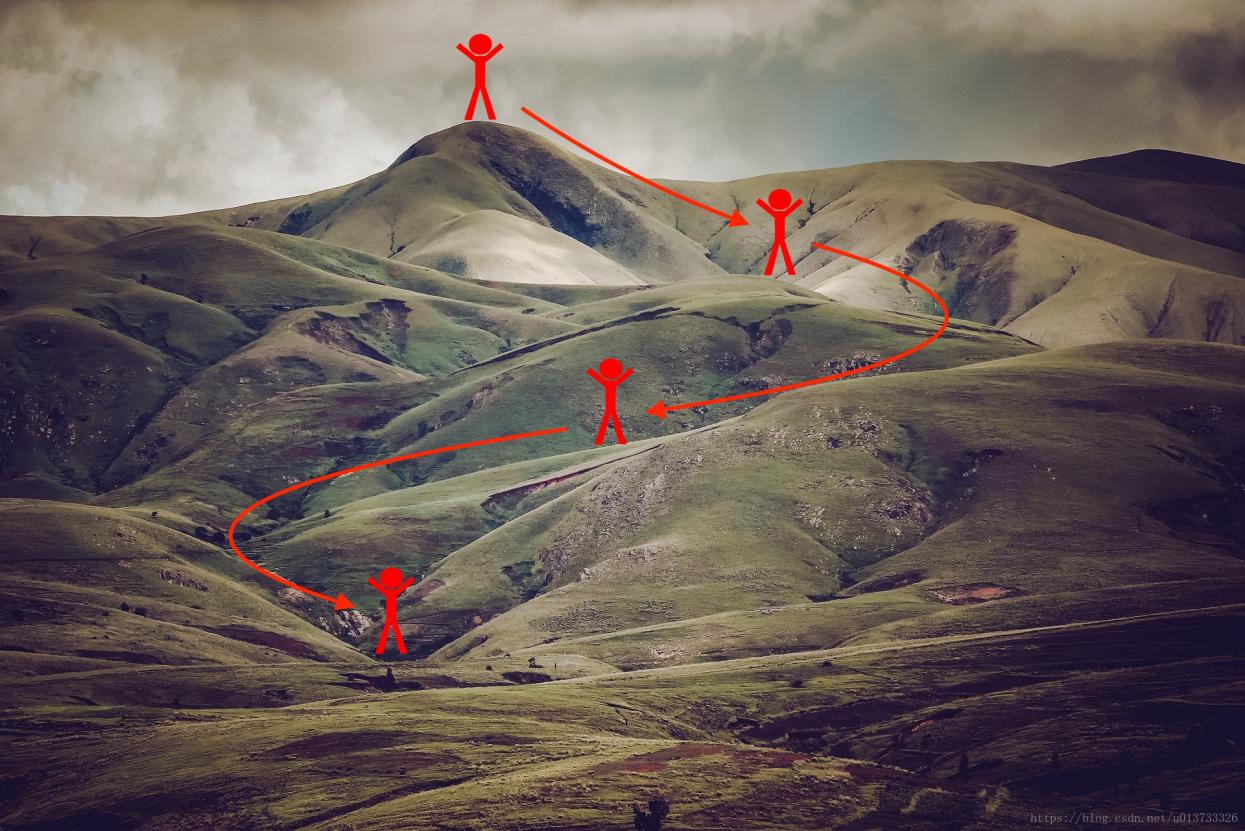
# Optimization Methods

# 1、开始之前

## 在正式开始之前，我们说一下我们要做什么。我们需要做的是分割数据集和优化梯度下降算法，所以我们需要做以下几件事：   1. 分割数据集   2. 优化梯度下降算法：     2.1不使用任何优化算法     2.2 mini-batch梯度下降法     2.3使用具有动量的梯度下降算法     2.4使用Adam算法

到目前为止，我们始终都是在使用梯度下降法学习，本文中，我们将使用一些更加高级的优化算法，利用这些优化算法，通常可以提高我们算法的收敛速度，并在最终得到更好的分离结果。这些方法可以加快学习速度，甚至可以为成本函数提供更好的最终值，在相同的结果下，有一个好的优化算法可以是等待几天和几个小时之间的差异。

   我们想象一下成本函数*J*，最小化成本就像找到丘陵的最低点，在训练的每一步中，都会按照某个方向更新参数，以尽可能达到最低点。它类似于最快的下山的路，如下图：

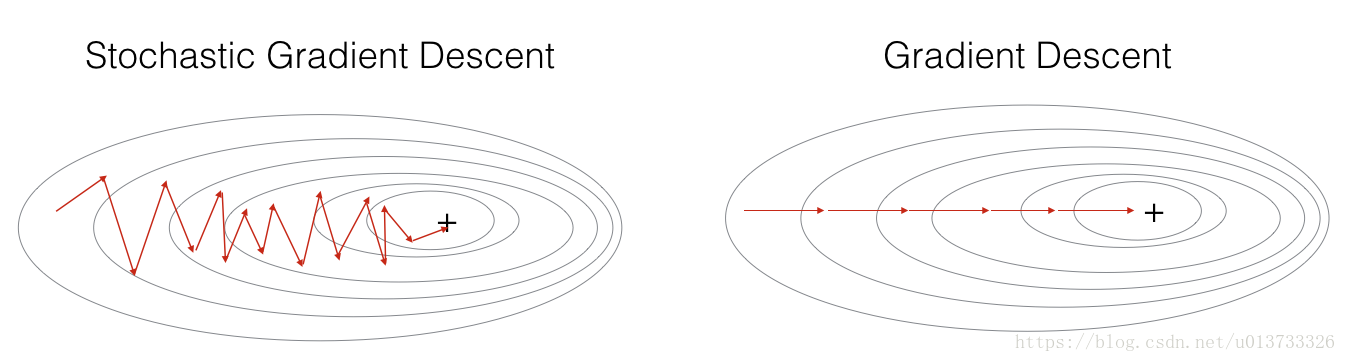


## 1.1梯度下降

在机器学习中，最简单就是没有任何优化的梯度下降(GD,Gradient Descent)，我们每一次循环都是对整个训练集进行学习，这叫做批量梯度下降(Batch Gradient Descent)，我们之前说过了最核心的参数更新的公式，这里我们再来看一下：



由梯度下降算法演变来的还有随机梯度下降（SGD）算法和小批量梯度下降算法，随机梯度下降（SGD）相当于小批量梯度下降，但是和mini-batch不同的是其中每个小批量(mini-batch)仅有1个样本，和梯度下降不同的是你一次只能在一个训练样本上计算梯度，而不是在整个训练集上计算梯度。  
  在随机梯度下降中，在更新梯度之前，只使用1个训练样本。当训练集较大时，随机梯度下降可以更快，但是参数会向最小值摆动，而不是平稳地收敛，我们来看一下比较图:



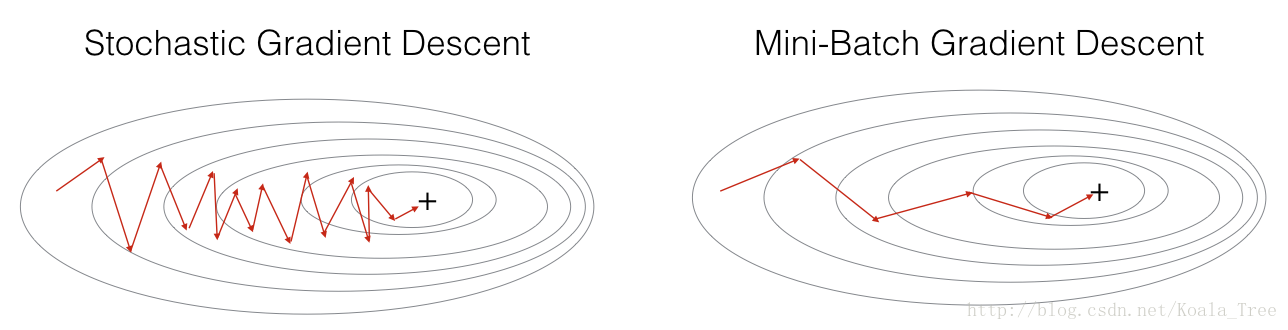
还要注意，实现SGD总共需要3个for循环:

1.所以迭代次数；

2.所有m个训练例子；

3.所有层数(更新所有参数，从(W[1]，b[1])到(W[L]，b[L]))；

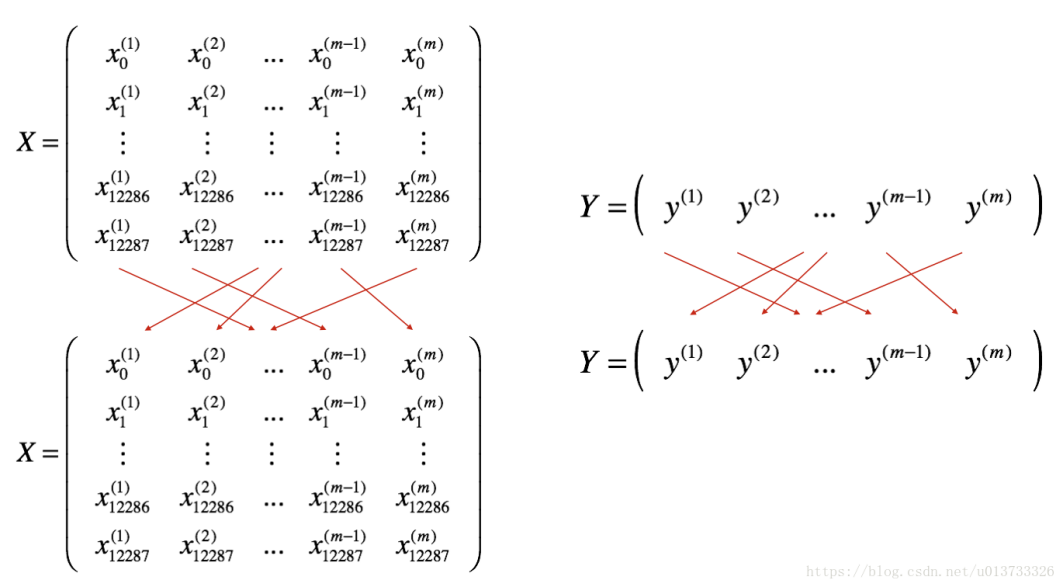
在实际中，更好的方法是使用小批量(mini-batch)梯度下降法，小批量梯度下降法是一种综合了梯度下降法和随机梯度下降法的方法，在它的每次迭代中，既不是选择全部的数据来学习，也不是选择一个样本来学习，而是把所有的数据集分割为一小块一小块的来学习，它会随机选择一小块（mini-batch），块大小一般为2的n次方倍。一方面，充分利用的GPU的并行性，更一方面，不会让计算时间特别长，来看一下比较图：



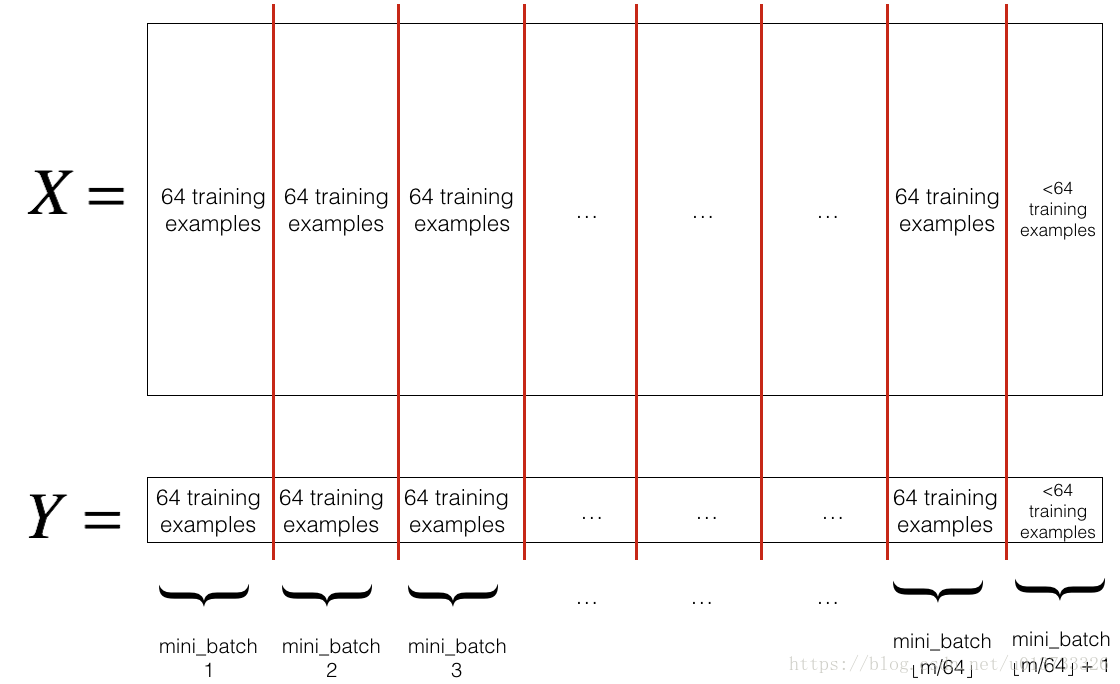
## 1.2 mini-batch梯度下降法

我们使用mini-batch要经过两个步骤：

  1. 把训练集打乱，但是X和Y依旧是一一对应的，之后，X的第i列是与Y中的第i个标签对应的样本。乱序步骤确保将样本被随机分成不同的小批次。如下图，X和Y的每一列代表一个样本

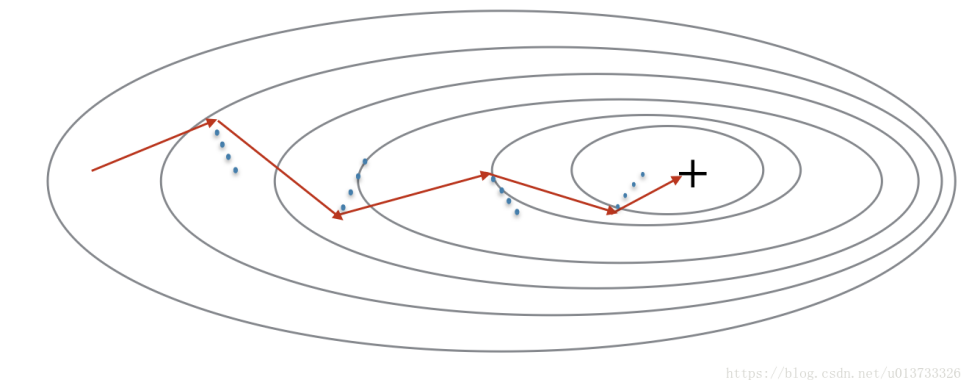


2.切分，我们把训练集打乱之后，我们就可以对它进行切分了。这里切分的大小是64，如下图：



**1.3 包含动量的梯度下降**

由于小批量梯度下降只看到了一个子集的参数更新，更新的方向有一定的差异，所以小批量梯度下降的路径将“振荡地”走向收敛，使用动量可以减少这些振荡，动量考虑了过去的梯度以平滑更新，我们将把以前梯度的方向存储在变量v中，从形式上讲，这将是前面的梯度的指数加权平均值。我们也可以把V看作是滚下坡的速度，根据山坡的坡度建立动量。我们来看一下下面的图：



上图中，红色箭头显示具有动量的小批量梯度下降一步时所采取的方向，蓝色的点显示每个步骤的梯度方向（相对于当前的小批量）。

当然我们不仅要观察梯度，还要让v影响梯度，然后朝v方向前进一步,尽量让前进的方向指向最小值。既然我们要影响梯度的方向，而梯度需要使用到dW和db，那么我们就要建立一个和dW和db相同结构的变量来影响他们。

初始化v完成后，我们就开始影响梯度的方向，我们需要使用以下公式：





其中，*l*是当前神经网络的层数；β是动量，是一个实数；α是学习率。

需要注意的是速度v是用0来初始化的，因此，该算法需要经过几次迭代才能把速度提升上来并开始跨越更大步伐。当beta=0时，该算法相当于是没有使用momentum算法的标准的梯度下降算法。当beta越大的时候，说明平滑的作用越明显。通常0.9是比较合适的值。那如何才能在开始的时候就保持很快的速度向最小误差那里前进呢？我们来看看下面的Adam算法。

如何选择β?

-动量β越大，更新越平滑，因为我们越多地考虑过去的梯度。但是如果β太大了，它也可能使更新过于平滑。

-β的常见值范围从0.8到0.999。如果你不想调整这个，β=0.9通常是合理的默认值。

-对您的模型调整最佳β，可能需要尝试几个值，以查看在降低代价函数J的值方面什么最有效。

动量可以应用于批量梯度下降，小批量梯度下降或随机梯度下降算法中。

**1.4 Adam算法**

Adam算法是训练神经网络中最有效的算法之一，它是RMSProp算法与Momentum算法的结合体。我们来看看它都干了些什么。

1. 计算以前的梯度的指数加权平均值，并将其存储在变量v（偏差校正前）和vcorrected（偏差校正后）中。

2. 计算以前梯度的平方的指数加权平均值，并将其存储在变量s（偏差校正前）和scorrected（偏差校正后）中。

3. 根据1和2更新参数。

我们来看一下它所用的公式：



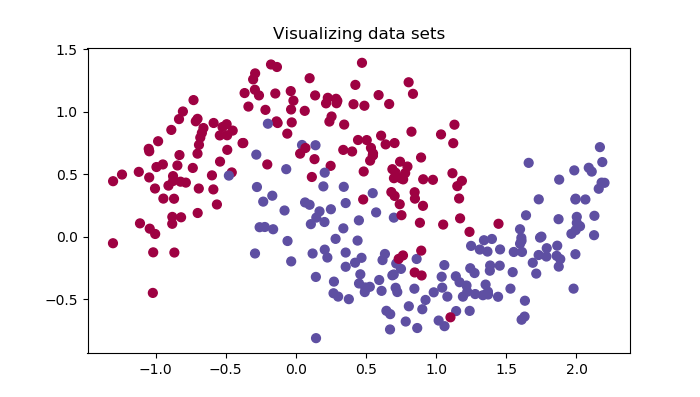
其中，*t* 为当前迭代的次数；*l*为当前神经网络的层数；β1和β2为控制两个指数加权平均值的超参数；α为学习率；ε为一个非常小的数，用于避免除零操作，一般为10−8。

**2、测试**

在测试正式开始之前，我们需要把数据集加载进来。

**2.1加载数据集**

我们使用下面的“月亮（moon）”数据集来测试不同的优化方法。数据集被命名为“月亮”，因为这两个类的数据看起来有点像新月形的月亮。

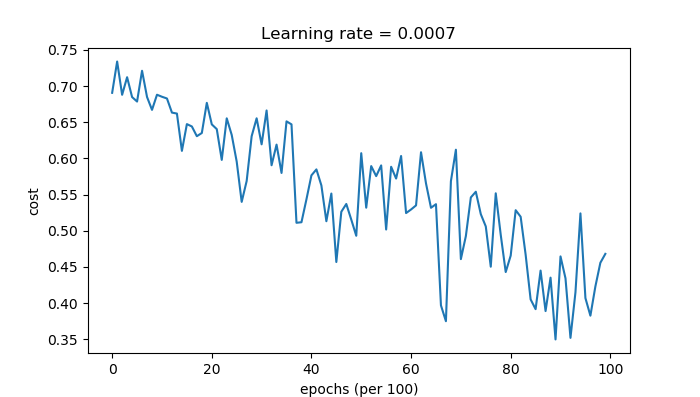


**2.2 测试**

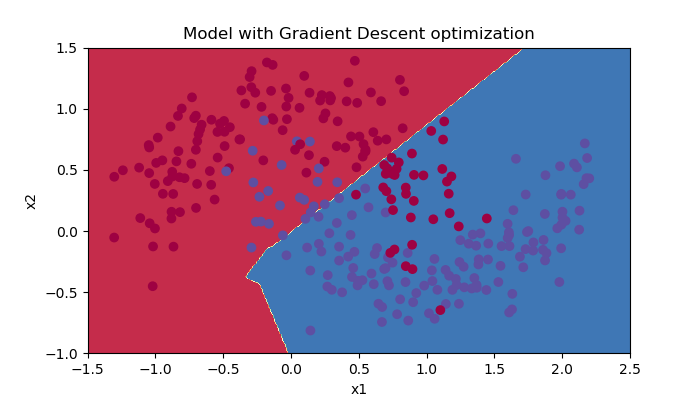
我们之前已经实现过了一个三层的神经网络，我们将分别用它来测试我们的优化器的优化效果。

对各种优化器进行优化：

* 先来测试没有任何优化的梯度下降：梯度下降测试



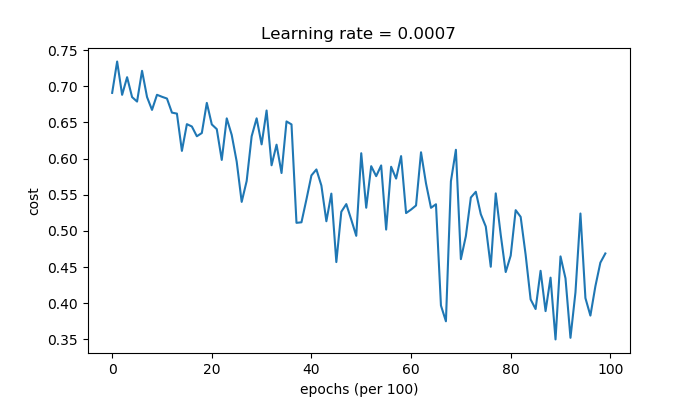
绘制分类的情况：



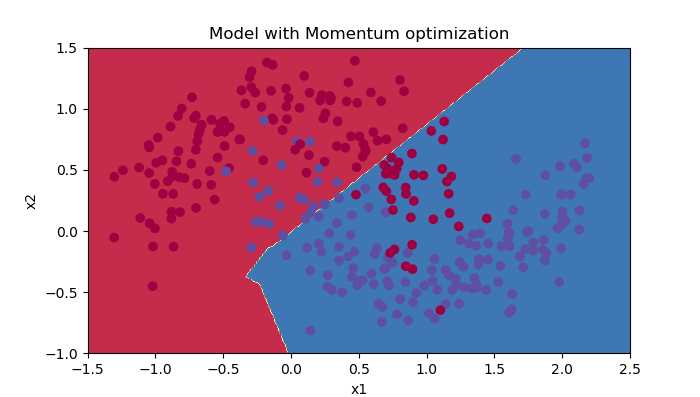
最小的误差：0.46451797221676844

Accuracy: 0.7966666666666666

* 具有动量的梯度下降测试  
   因为这个例子比较简单，使用动量效果很小，但对于更复杂的问题，你可能会看到更好的效果。



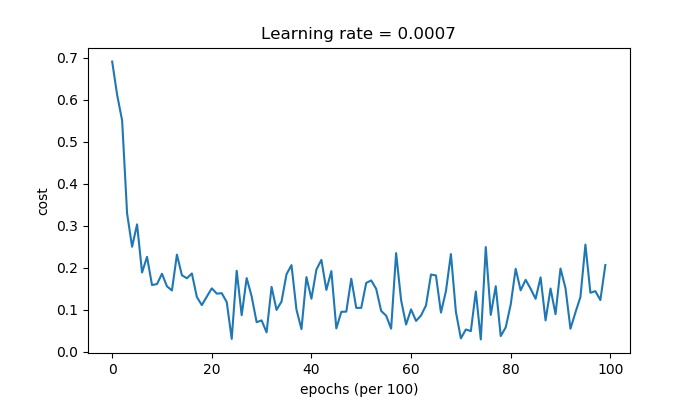
绘制分类的情况：



最小的误差：0.4647395967922748

Accuracy: 0.7966666666666666

* Adam优化后的梯度下降



绘制分类的情况：



最小的误差：0.19794007152465481

Accuracy: 0.94

# 3、总结

|  |  |  |
| --- | --- | --- |
| 优化算法 | 准确度 | 曲线平滑度 |
| 梯度下降 | 79.7% | 震荡 |
| 具有动量的梯度下降算法 | 79.7% | 震荡 |
| Adam优化后的梯度下降 | 94% | 平滑 |

  具有动量的梯度下降通常可以有很好的效果，但由于小的学习速率和简单的数据集所以它的影响几乎是轻微的。另一方面，Adam明显优于小批量梯度下降和具有动量的梯度下降，如果在这个简单的模型上运行更多时间的数据集，这三种方法都会产生非常好的结果，然而，我们已经看到Adam收敛得更快。

  Adam的一些优点包括相对较低的内存要求（虽然比梯度下降和动量下降更高）和通常运作良好，即使对参数进行微调（除了学习率*α*）。