**[深度学习与计算机视觉系列(4)\_最优化与随机梯度下降](https://blog.csdn.net/yaoqiang2011/article/details/50178505)**

作者：[寒小阳](http://blog.csdn.net/han_xiaoyang?viewmode=contents" \t "_blank)   
时间：2015年12月。   
出处：[http://blog.csdn.net/han\_xiaoyang/article/details/50178505](http://blog.csdn.net/han_xiaoyang/article/details/50178505" \t "_blank)   
声明：版权所有，转载请联系作者并注明出处

## 1. 引言

上一节[深度学习与计算机视觉系列(3)\_线性SVM与SoftMax分类器](http://blog.csdn.net/han_xiaoyang/article/details/49999583)中提到两个对图像识别至关重要的概念：

1. 用于把原始像素信息映射到不同类别得分的**得分函数/score function**
2. 用于评估参数W效果(评估该参数下每类得分和实际得分的吻合度)的**损失函数/loss function**

其中对于线性SVM，我们有：

1. 得分函数*f*(*xi*,*W*)=*Wxi*

  损失函数*L*=1*N*∑*i*∑*j*≠*yi*[max(0,*f*(*xi*;*W*)*j*−*f*(*xi*;*W*)*yi*+1)]+*αR*(*W*)

在取到合适的参数W的情况下，我们根据原始像素计算得到的预测结果和实际结果吻合度非常高，这时候损失函数得到的值就很小。

这节我们就讲讲，怎么得到这个合适的参数W，使得损失函数取值最小化。也就是**最优化**的过程。

## 2. 损失函数可视化

我们在计算机视觉中看到的损失函数，通常都是定义在非常高维的空间里的(比如CIFAR-10的例子里一个线性分类器的权重矩阵W是10 x 3073维的，总共有30730个参数 -\_-||)，人要直接『看到』它的形状/变化是非常困难的。但是机智的同学们，总是能想出一些办法，把损失函数在某种程度上可视化的。比如说，我们可以把高维投射到一个向量/方向(1维)或者一个面(2维)上，从而能直观地『观察』到一些变化。

举个例子说，**我们可以对一个权重矩阵W(例如CIFAR-10中是30730个参数)，可以找到W维度空间中的一条直线，然后沿着这条线，计算一下损失函数值的变化情况**。具体一点说，就是我们找到一个向量*W*1

(维度要和W一样，这样*W*1才能表示W的维度空间的一个方向)，然后我们给不同的a值，计算*L*(*W*+*aW*1)

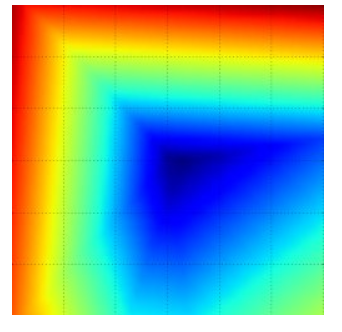
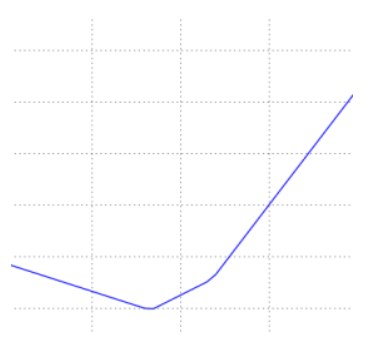
，这样，如果a取得足够密，其实我们就能够在一定程度上描绘出损失函数沿着这个方向的变化了。

同样，如果我们给两个方向*W*1

和*W*2，那么我们可以确定一个平面，我们再取不同值的a和b，计算*L*(*W*+*aW*1+*bW*2)

的值，那么我们就可以大致绘出在这个平面上，损失函数的变化情况了。

根据上面的方法，我们画出了下面3个图。最上面的图是调整a的不同取值，绘出的损失函数变化曲线(越高值越大)；中间和最后一个图是调整a与b的取值，绘出的损失函数变化图(蓝色表示损失小，红色表示损失大)，中间是在一个图片样本上计算的损失结果，最下图为100张图片上计算的损失结果的一个平均。显然沿着直线方向得到的曲线底端为最小的损失值点，而曲面呈现的碗状图形碗底为损失函数取值最小处。



我们从数学的角度，来尝试解释一下，上面的凹曲线是怎么出来的。对于第i个样本，我们知道它的损失函数值为：

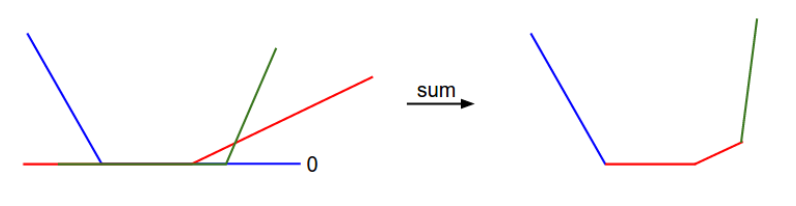
*Li*=∑*j*≠*yi*[max(0,*wTjxi*−*wTyixi*+1)]

所有的样本上的损失函数值，是它们损失函数值（max(0,-)，因此最小值为0）的平均值。为了更好理解，我们假定训练集里面有3个样本，都是1维的，同时总共有3个类别。所以SVM损失(暂时不考虑正则化项)可以表示为如下的式子：

*L*0=*L*1=*L*2=*L*=max(0,*wT*1*x*0−*wT*0*x*0+1)+max(0,*wT*2*x*0−*wT*0*x*0+1)max(0,*wT*0*x*1−*wT*1*x*1+1)+max(0,*wT*2*x*1−*wT*1*x*1+1)max(0,*wT*0*x*2−*wT*2*x*2+1)+max(0,*wT*1*x*2−*wT*2*x*2+1)(*L*0+*L*1+*L*2)/3

因为这个例子里的样本都是1维的，因此其实*xi*

和*wj*都是实数。拿*w*0举例，损失函数里，大于0的值其实都和*w*0是线性关系的，而最小值为0。因此，我们可以想象成，三条折线『合体』得到的最终曲线，如下图所示：



插几句题外话，从之前碗状结构的示意图，你可能会猜到SVM损失函数是一个凸函数，而对于凸函数的最小值求解方法有很多种。但之后当我们把损失函数f扩充到神经网络之后，损失函数将变成一个非凸函数，而如果依旧可视化的话，我们看到的将不再是一个碗状结构，而是凹凸不平的曲面。

## 3. 最优化

在我们现在这个问题中，所谓的『最优化』其实指的就是找到能让损失函数最小的参数*W*。如果大家看过或者了解凸优化的话，我们下面介绍的方法，对你而言可能太简单了，有点原始，但是大家别忘了，我们后期要处理的是神经网络的损失函数，那可不是一个凸函数哦，所以我们还是一步步来一起看看，如果去实现最优化问题。

### 3.1 策略1：随机搜寻(不太实用)

以一个笨方法开始，我们知道，当我们手头上有参数*W*

后，我们是可以计算损失函数，评估参数合适程度的。所以最直接粗暴的方法就是，我们尽量多地去试参数，然后从里面选那个让损失函数最小的，作为最后的*W*。代码当然很简单，如下：

### 3.2 策略2：随机局部搜索

上一个策略完全就是盲搜，要想找到全局最优的那个结果基本是不可能的。它最大的缺点，就在于下一次搜索完全是随机进行的，没有一个指引方向。那我们多想想，就能想出一个在上个策略的基础上，优化的版本，叫做『随机局部搜索』。

这个策略的意思是，我们不每次都随机产生一个参数矩阵*W*

了，而是在现有的参数*W*基础上，搜寻一下周边临近的参数，有没有比现在参数更好的*W*，然后我们用新的*W*替换现在的*W*

，接着在周围继续小范围搜寻。这个过程呢，可以想象成，我们在一座山上，现在要下山，然后我们每次都伸脚探一探周边，找一个比现在的位置下降一些的位置，然后迈一步，接着在新的位置上做同样的操作，一步步直至下山。

从代码实现的角度看，以上的过程，实际上就是对于一个当前*W*

，我们每次实验和添加*δW*′，然后看看损失函数是否比当前要低，如果是，就替换掉当前的*W*，代码如下：

### 3.3 策略3：顺着梯度下滑

刚才的策略，我们说了，最大的缺点是非常耗时，且计算量也很大。我们一直在做的事情，就是在当前的位置基础上，想找到一个最合适的下降方向。我们依旧回到我们假设的那个情境，如果我们在山顶，要以最快的方式下山，我们会怎么做？我们可能会环顾四周，然后找到最陡的方向，迈一小步，然后再找当前位置最陡的下山方向，再迈一小步…

而这里提到的最陡的方向，其实对应的就是数学里『梯度』的概念，也就是说，其实我们无需『伸脚试探』周边的陡峭程度，而是可以通过计算**损失函数的梯度**，直接取得这个方向。

我们知道在1个变量的函数里，某点的斜率/导数代表其变化率最大的方向。而对于多元的情况，梯度是上面情况的一个扩展，只不过这时候的变量不再是一个，而是多个，同时我们计算得到的『梯度方向』也是一个多维的向量。大家都知道数学上计算1维/元函数『梯度/导数』的表达式如下：

*df*(*x*)*dx*=lim*h* →0*f*(*x*+*h*)−*f*(*x*)*h*

对于多元的情况，这个时候我们需要求的东西扩展成每个方向的『偏导数』，然后把它们合在一块组成我们的梯度向量。

我们用几张图来说明这个过程：

## 4. 计算梯度

有两种计算梯度的方法：

1. 慢一些但是简单一些的数值梯度/numerical gradient
2. 速度快但是更容易出错的解析梯度/analytic gradient

### 4.1 数值梯度

根据上面提到的导数求解公式，我们可以得到数值梯度计算法。下面是一段简单的代码，对于一个给定的函数f和一个向量x，求解这个点上的梯度：

#### 4.1.1 实际计算中的提示

我们仔细看看导数求解的公式，会发现数学定义上h是要趋于0的，但实际我们计算的时候我们只要取一个足够小的数(比如1e-5)作为h就行了，所以我们要精准计算偏导的话，要尽量取到不会带来数值计算问题，同时又能很小的h。另外，其实实际计算中，我们用另外一个公式用得更多[*f*(*x*+*h*)−*f*(*x*−*h*)]/2*h*

#### 4.1.2 关于迭代的细节

如果大家仔细看上述代码的话，会发现我们step\_size设的都是负的，确实我们每次update权重W的时候，是用原来的W减掉梯度方向的一个较小的值，这样损失函数才能减小。

#### 4.1.3 关于迭代的步长

我们计算得到梯度之后，就确定了幅度变化最快(负梯度是下降方向)的方向，但是它并没有告诉我们，我朝着这个方向，应该迈进多远啊。之后的章节会提到，选择正确的迭代步长(有时候我们也把它叫做学习速率)是训练过程中最重要(也是最让人头疼)的一个待设定参数。就像我想以最快的速度下山，我们能感知到最陡的方向，却不知道应该迈多大的步子。如果我们小步迈进，那确实每一步都能比上一步下降一些，但是速度太慢了亲！！但是如果我们以非常非常大的步伐迈进(假如腿巨长 -\_-||)，那你猜怎么着，你一不小心可能就迈过山脚迈到另一座山山腰上了…

下图是对以上情况的一个描述和解释：

#### 4.1.4 效率问题

如果你再回过头去看看上面计算数值梯度的程序，你会发现，这个计算方法的复杂度，基本是和我们的参数个数成线性关系的。这意味着什么呢？在我们的CIFAR-10例子中，我们总共有30730个参数，因此我们单次迭代总共就需要计算30731次损失函数。这个问题在之后会提到的神经网络中更为严重，很可能两层神经元之间就有百万级别的参数权重，所以，计算机算起来都很耗时…人也要等结果等到哭瞎…

### 4.2 解析法计算梯度

数值梯度发非常容易实现，但是从公式里面我们就看得出来，梯度实际上是一个近似(毕竟你没办法把h取到非常小)，同时这也是一个计算非常耗时的算法。第二种计算梯度的方法是解析法，它可以让我们直接得到梯度的一个公式(代入就可以计算，非常快)，但是呢，不像数值梯度法，这种方法更容易出现错误。so，聪明的同学们，就想了一个办法，我们可以先计算解析梯度和数值梯度，然后比对结果和校正，在确定我们解析梯度实现正确之后，我们就可以大胆地进行解析法计算了(这个过程叫做梯度检查/检测)

我们拿一个样本点的SVM损失函数举例：

*Li*=∑*j*≠*yi*[max(0,*wTjxi*−*wTyixi*+Δ)]

我们可以求它对每个权重的偏导数，比如说，我们求它对*wyi*

的偏导，我们得到：

∇*wyiLi*=−(∑*j*≠*yi*1(*wTjxi*−*wTyixi*+Δ>0))*xi*

其中1

是一个bool函数，在括号内的条件为真的时候取值为1，否则为0。看起来似乎很吓人，但实际上要写代码完成的话，你只需要计算不满足指定SVM最小距离的类(对损失函数有贡献的类)的个数，然后用这个值会对数据向量*xi*做缩放即可得到梯度。但是要注意只是*W*中对应正确的类别的列的梯度。对于其他的*j*≠*yi*

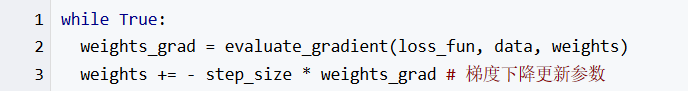
的情况，梯度为：

∇*wjLi*=1(*wTjxi*−*wTyixi*+Δ>0)*xi*

一旦得到梯度的表达式，那计算梯度和调整权重就变得非常直接和简单。熟练掌握如何在loss expression下计算梯度是非常重要的一个技巧，贯穿整个神经网络的训练实现过程，关于这个内容，下次会详细讲到。

## 5. 梯度下降

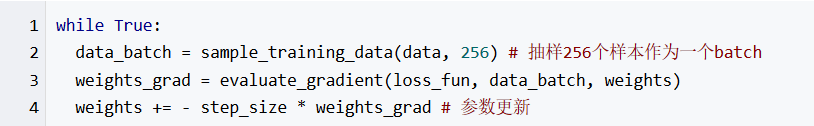
在我们有办法计算得到梯度之后，使用梯度去更新已有权重参数的过程叫做『梯度下降』，伪代码其实就是如下的样子：



这个简单的循环实质上就是很多神经网络库的核心。当然，我们也有其他的方式去实现最优化(比如说L-BFGS)，但是梯度下降确实是当前使用最广泛，也相对最稳定的神经网络损失函数最优化方法。

### 5.1 Mini-batch gradient descent

在大型的应用当中(比如ILSVRC)，训练数据可能是百万千万级别的。因此，对整个训练数据集的样本都算一遍损失函数，以完成参数迭代是一件非常耗时的事情，一个我们通常会用到的替代方法是，采样出一个子集在其上计算梯度。现在比较前沿的神经网络结构基本都是这么做的，例如ConvNets是每256张作为一个batch去完成参数的更新。参数更新的代码如下：



之所以可以这么做，是因为训练数据之间其实是关联的。我们简化一下这个问题，你想想，如果ILSVRC中的120w图片，如果只是1000张不同的图片，一直复制1200次得到的。那么其实我们在这1000张图片上算得的损失函数和120w的平均其实是一致的。当然，当然，在实际场景中，我们肯定很少遇到这种多次重复的情况，但是原数据的一个子集(mini-batch)上的梯度，其实也是对整体数据上梯度的一个很好的近似。因此，只在mini-batch上计算和更新参数，会有快得多的收敛速度。

上述算法的一个极端的情况是，如果我们的一个mini-batch里面只有一张图片。那这个过程就变成**『随机梯度下降/Stochastic Gradient Descent (SGD)』**，说起来，这个其实在实际应用中倒也没那么常见，原因是向量化之后，一次计算100张图片，其实比计算一张图片100次，要快得多。所以即使从定义上来说，SGD表示我们用一张图片上的梯度近似全局梯度，**但是很多时候人们提到SGD的时候，其实他们指的是mini-batch梯度下降**，也就是说，我们把一个batch当做1份了。额，还要稍微提一句的是，有些同学可能会问，这个batch size本身不是一个需要实验的参数吗，取多大的batch size好啊？但实际应用中，我们倒很少会用cross-validation去选择这个参数。这么说吧，我们一般是基于我们内存限制去取这个值的，比如设成100左右。

## 6. 总结

* 把损失函数在各参数上的取值，想象成我们所在山峰的高度。那么我们要**最小化损失函数**，实际上就是『要想办法下山』。
* 我们采取的下山策略是，**一次迈一小步，只要每次都往下走了，那么最后就会到山底**。
* 梯度对应函数变化最快的方向，负梯度的方向就是我们下山，环顾四周之后，发现最陡的下山路方向。
* 我们的**步长(也叫学习率)，会影响我们的收敛速度(下山速度)**，如果步伐特别特别大，甚至可能跃过最低点，跑到另外一个高值位置了。
* 我们用**mini-batch的方式，用一小部分的样本子集，计算和更新参数**，减少计算量，加快收敛速度。

## 参考资料与原文

[cs231n 最优化与随机梯度下降](http://cs231n.github.io/optimization-1/)