

Fork me on GitHub

河沟大河沟

----- 梦想还是要有的，万一实现了呢！纸上得来终觉浅 绝知此事要躬行！

博客园 首页 新随笔 联系 管理 订阅 XML

随笔- 783 文章- 51 评论- 42

梯度下降优化算法综述

1. An overview of gradient descent optimization algorithms

<http://ruder.io/optimizing-gradient-descent/index.html><https://arxiv.org/abs/1609.04747>

2. 中文翻译《梯度下降优化算法综述》：

<http://blog.csdn.net/heyongluoyao8/article/details/52478715>

3. 深度学习笔记：优化方法总结(BGD,SGD,Momentum,AdaGrad,RMSProp,Adam)



2. 随机梯度下降加速

如果把要优化的目标函数看成山谷的话，可以把要优化的参数看成滚下山的石头，参数随机化为一个随机数可以看做在山谷的某个位置以0速度开始往下滚。目标函数的梯度可以看做给石头施加的力，由力学定律知： $F=m*a$ ，所以梯度与石头下滚的加速度成正比。因而，梯度直接影响速度，速度的累加得到石头的位置，对这个物理过程进行建模，可以得到参数更新过程为：

```
# Momentum update
v = momentum * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

代码中 v 指代速度，其计算过程中有一个超参数 $momentum$ ，称为动量（momentum）。虽然名字为动量，其物理意义更接近于摩擦，其可以降低速度值，降低了系统的动能，防止石头在山谷的最底部不能停止情况的发生。动量的取值范围通常为 $[0.5, 0.9, 0.95, 0.99]$ ，一种常见的做法是在迭代开始时将其设为0.5，在一定的迭代次数（epoch）后，将其值更新为0.99。

在实践中，一般采用SGD+momentum的配置，相比普通的SGD方法，这种配置通常能极大地加快收敛速度。

3. 学习率的更新

在算法迭代过程中逐步降低学习率（step_size）通常可以加快算法的收敛速度。常用的用来更新学习率的方法有三种：

逐步降低（Step decay），即经过一定迭代次数后将学习率乘以一个小的衰减因子。典型的做法包括经过5次迭代（epoch）后学习率乘以0.5，或者20次迭代后乘以0.1。

指数衰减（Exponential decay），其数学表达式可以表示为： $\alpha = \alpha_0 e^{-kt}$ ，其中， α_0 和 k 是需要设置的超参数， t 是迭代次数。

倒数衰减（1/t decay），其数学表达式可以表示为： $\alpha = \alpha_0 / (1+kt)$ ，其中， α_0 和 k 是需要设置的超参数， t 是迭代次数。

实践中发现逐步衰减的效果优于另外两种方法，一方面在于其需要设置的超参数数量少，另一方面其可解释性也强于另两种方法。

4. 二阶更新方法

提升梯度下降法收敛速度的方法还包括将其由一阶提升为二阶，也就是牛顿法或者拟牛顿法，比如常用的 L-BFGS。然而，牛顿法和L-BFGS不适用于解决大规模训练数据集和大规模问题。

比如，常用的深度网络包括数百万个参数，每次迭代都要计算大小为 $[1,000,000 \times 1,000,000]$ 的Hessian矩阵，需要占用3G多的内存，严重影响了计算效率。

L-BFGS法不需要计算完全的Hessian矩阵，虽然没有了内存的担忧，但这种方法通常类似于批量梯度下降法，需要在计算整个训练集（通常为几百万个样本）的梯度后才能更新一次参数，严重影响了收敛速度。因而在深度神经网络领域很少使用L-BFGS来优化目标函数。

参考的SGD+momentum的代码是Andrew Ng的深度学习教程给出的matlab代码。

关于博主：爱生活，积极学习各种技术
IT技能：能够较熟练使用C和C++编程，熟悉常见的数据结构和算法，了解Linux和Git操作

个人技能：熟悉常用的图像处理，机器学习，计算机视觉算法，了解常见超分辨率重建，分类，检测，分割网络；能运用Python, Matlab语言和OpenCV, Tensorflow, Pytorch等工具进行计算机视觉相关课题研究；

努力方向：深入学习深度学习框架，积累实际工程能力，熟练刷题...

职业目标：希望从事计算机视觉领域算法岗（image processing, machine learning, deep learning）

Github: <https://github.com/ranjiewwen/>

个人站: <https://ranjiewwen.github.io/>



昵称: ranjiewen

园龄: 2年9个月

粉丝: 113

关注: 107

+加关注

<	2018年10月						>
日	一	二	三	四	五	六	
30	1	<u>2</u>	3	4	<u>5</u>	6	
7	8	9	10	11	12	13	
14	15	16	17	18	19	20	
21	22	23	24	25	26	27	
28	29	30	31	1	2	3	
4	5	6	7	8	9	10	

```

function [opttheta] = minFuncSGD(funObj,theta,data,labels,options)
% Runs stochastic gradient descent with momentum to optimize the
% parameters for the given objective.
%
% Parameters:
% funObj      - function handle which accepts as input theta,
%               data, labels and returns cost and gradient w.r.t
%               to theta.
% theta       - unrolled parameter vector
% data        - stores data in m x n x numExamples tensor
% labels      - corresponding labels in numExamples x 1 vector
% options     - struct to store specific options for optimization
%
% Returns:
% opttheta    - optimized parameter vector
%
% Options (* required)
% epochs*     - number of epochs through data
% alpha*      - initial learning rate
% minibatch*  - size of minibatch
% momentum    - momentum constant, defaults to 0.9

%%=====
%% Setup
assert(all(isfield(options',{'epochs','alpha','minibatch'})), 'Some options not
defined');
if ~isfield(options,'momentum')
    options.momentum = 0.9;
end;
epochs = options.epochs;
alpha = options.alpha;
minibatch = options.minibatch;
m = length(labels); % training set size
% Setup for momentum
mom = 0.5;
momIncrease = 20;
velocity = zeros(size(theta));

%%=====
%% SGD loop
it = 0;
for e = 1:epochs
    % randomly permute indices of data for quick minibatch sampling
    rp = randperm(m);
    for s=1:minibatch:(m-minibatch+1)
        it = it + 1;
        % increase momentum after momIncrease iterations
        if it == momIncrease
            mom = options.momentum;
        end;
        % get next randomly selected minibatch
        mb_data = data(:, :, rp(s:s+minibatch-1));
        mb_labels = labels(rp(s:s+minibatch-1));
        % evaluate the objective function on the next minibatch
        [cost grad] = funObj(theta,mb_data,mb_labels);
        % update the current weights theta
        velocity = mom*velocity+alpha*grad;
        theta = theta-velocity;
        fprintf('Epoch %d: Cost on iteration %d is %f\n',e,it,cost);
    end;
    % anneal learning rate by factor of two after each epoch
    alpha = alpha/2.0;
end;
opttheta = theta;
end

```

搜索

我的标签

[中国大学MOOC-陈越、何钦铭-数据结构-2017春\(42\)](#)
[图像处理算法\(28\)](#)
[树\(27\)](#)
[递归+DFS+BFS+回溯\(26\)](#)
[tensorflow\(21\)](#)
[dp算法\(21\)](#)
[数据结构\(21\)](#)
[算法之美\(17\)](#)
[字符串\(16\)](#)
[操作系统\(16\)](#)
[更多](#)

随笔分类

[C/C++\(84\)](#)
[CrackingtheCodingInterview\(20\)](#)
[CV经典问题实践\(8\)](#)
[GAN\(3\)](#)
[Git\(4\)](#)
[Java\(2\)](#)
[Kaggle\(1\)](#)
[leetcode\(168\)](#)
[Linux\(16\)](#)
[Matlab\(18\)](#)
[MFC\(24\)](#)
[Oj\(11\)](#)
[OpenCV\(12\)](#)
[Python\(32\)](#)
[Qt\(5\)](#)
[RNN\(1\)](#)
[STL\(26\)](#)
[笔试，面试题\(50\)](#)
[操作系统 operating system\(14\)](#)
[常见易错点&&函数实现\(16\)](#)
[多线程与并发\(8\)](#)
[环境搭建\(28\)](#)
[机器学习 Machine learning\(50\)](#)
[计算机网络 computer networks\(16\)](#)
[剑指offer\(11\)](#)
[牛客网--课程学习\(18\)](#)
[其他技术\(6\)](#)
[设计模式\(12\)](#)
[深度学习DeepLearning\(34\)](#)
[生活和学习计划\(18\)](#)
[数据库系统 database system\(18\)](#)
[数学\(8\)](#)
[数字信号处理\(7\)](#)
[算法分析与设计\(2\)](#)
[算法与数据结构\(48\)](#)
[算法之美-22个经典问题 +45个算法\(17\)](#)
[图像处理 image processing\(37\)](#)
[总结概述\(13\)](#)

文章分类

[C/C++\(3\)](#)
[paper-分类，检测，分割，GAN...\(20\)](#)
[典型算法题目\(2\)](#)
[机器学习\(13\)](#)
[深度学习\(14\)](#)
[图像处理\(9\)](#)

机器学习达人

寒小阳

动量momentum在迭代开始时将其设为0.5，在20次迭代（epoch）后，其值更新为0.99。
学习率的更新采用逐步降低（step decay）法，即每次迭代（epoch）后将学习率乘以0.5。



梯度下降优化算法综述

该文翻译自[An overview of gradient descent optimization algorithms](#)。

众所周知，梯度下降算法是机器学习中使用非常广泛的优化算法，也是众多机器学习算法中最常用的优化方法。几乎当前每一个先进的(state-of-the-art)机器学习库或者深度学习库都会包括梯度下降算法的不同变种实现。但是，它们就像一个黑盒优化器，很难得到它们优缺点的实际解释。

这篇文章旨在提供梯度下降算法中的不同变种的介绍，帮助使用者根据具体需要进行使用。

这篇文章首先介绍梯度下降算法的三种框架，然后介绍它们所存在的问题与挑战，接着介绍一些如何进行改进来解决这些问题，随后，介绍如何在并行环境中或者分布式环境中使用梯度下降算法。最后，指出一些有利于梯度下降的策略。

梯度下降算法是通过沿着目标函数 $J(\theta)$ 参数 $\theta \in \mathbb{R}$ 的梯度（一阶导数）相反方向 $-\nabla J(\theta)$ 来不断更新模型参数来达到目标函数的极小值点（收敛），更新步长为 η 。详细的介绍参见：[梯度下降](#)。

三种梯度下降优化框架

有三种梯度下降算法框架，它们不同之处在于每次学习（更新模型参数）使用的样本个数，每次更新使用不同的样本会导致每次学习的准确性和学习时间不同。

- 全量梯度下降(Batch gradient descent)
每次使用全量的训练集样本来更新模型参数，即：
$$\theta = \theta - \eta \cdot \nabla J(\theta)$$

其代码如下：

```
for i in range(epochs):
    params_grad = evaluate_gradient(loss_function, data, params)
    params = params - learning_rate * params_grad
```

epochs 是用户输入的最大迭代次数。通过上述代码可以看出，每次使用全部训练集样本计算损失函数loss_function的梯度params_grad，然后使用学习速率learning_rate朝着梯度相反方向去更新模型的每个参数params。一般各现有的一些机器学习库都提供了梯度计算api。如果想自己亲手写代码计算，那么需要在程序调试过程中验证梯度计算是否正确，具体验证方法可以参见：[这里](#)。

全量梯度下降每次学习都使用整个训练集，因此其优点在于每次更新都会朝着正确的方向进行，最后能够保证收敛于极值点（凸函数收敛于全局极值点，非凸函数可能会收敛于局部极值点），但是其缺点在于每次学习时间过长，并且如果训练集很大以至于需要消耗大量的内存，并且全量梯度下降不能进行在线模型参数更新。

- 随机梯度下降(Stochastic gradient descent)

随机梯度下降算法每次从训练集中随机选择一个样本来进行学习，即： $\theta = \theta - \eta \cdot \nabla J(\theta; x_i; y_i)$

批量梯度下降算法每次都会使用全部训练样本，因此这些计算是冗余的，因为每次都使用完全相同的样本集。而随机梯度下降算法每次只随机选择一个样本来更新模型参数，因此每次的学习是非常快速的，并且可以进行在线更新。

其代码如下：

```
for i in range(epochs):
    np.random.shuffle(data)
    for example in data:
        params_grad = evaluate_gradient(loss_function, example, params)
        params = params - learning_rate * params_grad
```

随机梯度下降最大的缺点在于每次更新可能并不会按照正确的方向进行，因此可以带来优化波动(扰动)，如下图所示：

AI之路

DFan的NoteBook(细致paper阅读)

July's Pensieve

Poll的笔记

Rachel Zhang的专栏

Real_Myth--CV

zouxy09的专栏

白马负金羁

红色石头的专栏

拾毅者

系统学习深度学习

刷题达人

Code Ganker-leetcode

Grandyang-LeetCode

Zhu Li

我的链接

CSDN

Github

ranjiewen-知乎

个人主页

牛课网

阅读排行榜

1. 模拟退火算法(20100)
2. MultiByteToWideChar和WideCharToMultiByte用法详解(17725)
3. matlab 重命名文件和文件夹(12599)
4. Python in/not in --- if not/if + for...[if]...构建List+ python的else子句(12132)
5. Python 实现二维码生成和识别(11756)
6. 大津法---OTSU算法(10716)
7. C++用 _findfirst 和 _findnext 查找文件(10193)
8. 梯度下降优化算法综述(9178)
9. 损失函数(loss function)(8109)
10. C++实现ping功能(7612)

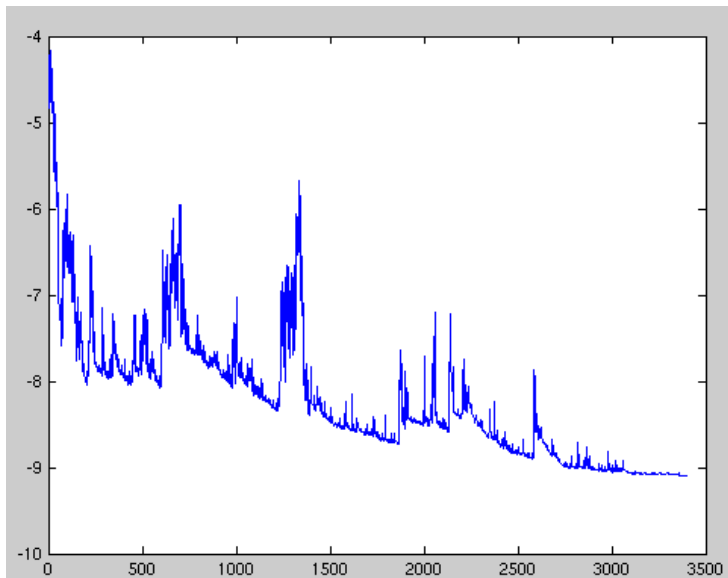


图1 SGD扰动来源

不过从另一个方面来看，随机梯度下降所带来的波动有个好处就是，对于类似盆地区域（即很多局部极小值点）那么这个波动的特点可能会使得优化的方向从当前的局部极小值点跳到另一个更好的局部极小值点，这样便可能对于非凸函数，最终收敛于一个较好的局部极值点，甚至全局极值点。

由于波动，因此会使得迭代次数（学习次数）增多，即收敛速度变慢。不过最终其会和全量梯度下降算法一样，具有相同的收敛性，即凸函数收敛于全局极值点，非凸损失函数收敛于局部极值点。

- 小批量梯度下降(Mini-batch gradient descent)

Mini-batch梯度下降综合了batch梯度下降与stochastic梯度下降，在每次更新速度与更新次数中间取得一个平衡，其每次更新从训练集中随机选择 $m, m < n$ 个样本进行学习，即：

$$\theta = \theta - \eta \nabla \theta J(\theta; x_{i:i+m}; y_{i:i+m})$$

其代码如下：

```
for i in range(epochs):
    np.random.shuffle(data)
    for batch in get_batches(data, batch_size=50):
        params_grad = evaluate_gradient(loss_function, batch, params)
        params = params - learning_rate * params_grad
```

相对于随机梯度下降，Mini-batch梯度下降降低了收敛波动性，即降低了参数更新的方差，使得更新更加稳定。相对于全量梯度下降，其提高了每次学习的速度。并且其不用担心内存瓶颈从而可以利用矩阵运算进行高效计算。一般而言每次更新随机选择[50,256]个样本进行学习，但是也要根据具体问题而选择，实践中可以进行多次试验，选择一个更新速度与更新次数都较适合的样本数。

mini-batch梯度下降虽然可以保证收敛性。mini-batch梯度下降常用于神经网络中。

问题与挑战

虽然梯度下降算法效果很好，并且广泛使用，但同时其也存在一些挑战与问题需要解决：

- 选择一个合理的学习速率很难。如果学习速率过小，则会导致收敛速度很慢。如果学习速率过大，那么其会阻碍收敛，即在极值点附近会振荡。
- 学习速率调整(又称学习速率调度，Learning rate schedules)[11]试图在每次更新过程中，改变学习速率，如退火。一般使用某种事先设定的策略或者在每次迭代中衰减一个较小的阈值。无论哪种调整方法，都需要事先进行固定设置，这边便无法自适应每次学习的数据集特点[10]。
- 模型所有的参数每次更新都是使用相同的学习速率。如果数据特征是稀疏的或者每个特征有着不同的取值统计特征与空间，那么便不能在每次更新中每个参数使用相同的学习速率，那些很少出现的特征应该使用一个相对较大的学习速率。
- 对于非凸目标函数，容易陷入那些次优的局部极值点中，如在神经网络中。那么如何避免呢。Dauphin[19]指出更严重的问题不是局部极值点，而是鞍点(These saddle points are usually surrounded by a plateau of the same error, which makes it notoriously hard for SGD to escape, as the gradient is close to zero in all dimensions.)。

梯度下降优化算法

下面将讨论一些在深度学习社区中经常使用用来解决上述问题的一些梯度优化方法，不过并不包括在高维数据中不可行的算法，如牛顿法。

Momentum

如果在峡谷地区(某些方向较另一些方向上陡峭得多，常见于局部极值点)[1]，SGD会在这些地方附近振荡，而导致收敛速度慢。这种情况下，动量(Momentum)便可以解决[2]。动量在参数更新项中加上一次更新量(即动

量项),即:

$$v_t = \gamma v_t - 1 + \eta \nabla \theta J(\theta)$$

$$\theta = \theta - v_t$$

其中动量项超参数 $\gamma < 1$ 一般是小于等于0.9。

其作用如下图所示:



图2 没有动量

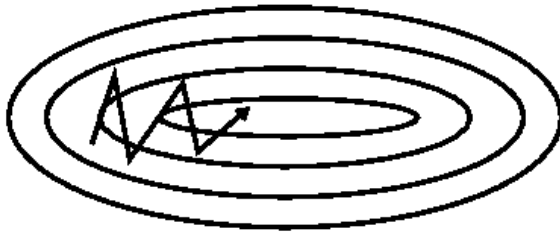


图3 加上动量

加上动量项就像从山顶滚下一个球,求往下滚的时候累积了前面的动量(动量不断增加),因此速度变得越来越快,直到到达终点。同理,在更新模型参数时,对于那些当前的梯度方向与上一次梯度方向相同的参数,那么进行加强,即这些方向上更快了;对于那些当前的梯度方向与上一次梯度方向不同的参数,那么进行削减,即这些方向上减慢了。因此可以获得更快的收敛速度与减少振荡。

NAG[7]

从山顶往下滚的球会盲目地选择斜坡。更好的方式应该是在遇到倾斜向上之前应该减慢速度。

Nesterov accelerated gradient(NAG,涅斯捷罗夫梯度加速)不仅增加了动量项,并且在计算参数的梯度时,在损失函数中减去了动量项,即计算 $\nabla \theta J(\theta - \gamma v_t - 1)$,这种方式预估了下次参数所在的位置。即:

$$v_t = \gamma v_t - 1 + \eta \nabla \theta J(\theta - \gamma v_t - 1)$$

$$\theta = \theta - v_t$$

如下图所示:

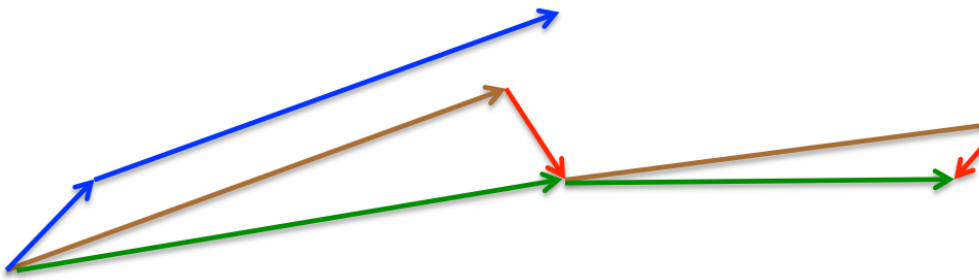


图4 NAG更新

详细介绍可以参见Ilya Sutskever的PhD论文[9]。假设动量因子参数 $\gamma = 0.9$,首先计算当前梯度项,如上图小蓝色向量,然后加上动量项,这样便得到了大的跳跃,如上图大蓝色的向量。这便是只包含动量项的更新。而NAG首先来一个大的跳跃(动量项),然后加上一个小的使用了动量计算的当前梯度(上图红色向量)进行修正得到上图绿色的向量。这样可以阻止过快更新来提高响应性,如在RNNs中[8]。

通过上面的两种方法,可以做到每次学习过程中能够根据损失函数的斜率做到自适应更新来加速SGD的收敛。下一步便需要对每个参数根据参数的重要性进行各自自适应更新。

Adagrad

Adagrad[3]也是一种基于梯度的优化算法,它能够对每个参数自适应不同的学习速率,对稀疏特征,得到大的学习更新,对非稀疏特征,得到较小的学习更新,因此该优化算法适合处理稀疏特征数据。Dean等[4]发现Adagrad能够很好的提高SGD的鲁棒性,google使用起来训练大规模神经网络(看片识猫:recognize cats in Youtube videos)。Pennington等[5]在GloVe中便使用Adagrad来训练得到词向量(Word Embeddings),频繁出现的单词赋予较小的更新,不经常出现的单词则赋予较大的更新。

在前述中,每个模型参数 θ_i 使用相同的学习速率 η ,而Adagrad在每一个更新步骤中对于每一个模型参数 θ_i 使用不同的学习速率 η_i ,设第 t 次更新步骤中,目标函数的参数 θ_i 梯度为 $g_{t,i}$,即:

$$g_{t,i} = \nabla \theta_j(\theta_i)$$

那么SGD更新方程为：

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

而Adagrad对每一个参数使用不同的学习速率，其更新方程为：

$$\theta_{t+1,i} = \theta_{t,i} - \eta G_{t,ii} + \epsilon - \sqrt{G_{t,ii}} \cdot g_{t,i}$$

其中， $G_t \in \mathbb{R}^d \times \mathbb{R}^d$ 是一个对角矩阵，其中第*i*行的对角元素 e_{ii} 为过去到当前第*i*个参数 θ_i 的梯度的平方和，epsilon是一个平滑参数，为了使得分母不为0(通常 $\epsilon = 1e-8$)，另外如果分母不开根号，算法性能会很糟糕。

进一步，将所有 $G_{t,ii}, g_{t,i}$ 的元素写成向量 G_t, g_t ，这样便可以使用向量点乘操作：

$$\theta_{t+1} = \theta_t - \eta G_t + \epsilon - \sqrt{G_t} \odot g_t$$

Adagrad主要优势在于它能够为每个参数自适应不同的学习速率，而一般的人工都是设定为0.01。同时其缺点在于需要计算参数梯度序列平方和，并且学习速率趋势是不断衰减最终达到一个非常小的值。下文中的Adadelta便是用来解决该问题的。

Adadelta Adadelta[[6]](#reference_6)是Adagrad的一种扩展，为了降低Adagrad中学习速率衰减过快问题，其改进了三处，一是使用了窗口w；二是对于参数梯度历史窗口序列(不包括当前)不再使用平方和，而是使用均值代替；三是最终的均值是历史窗口序列均值与当前梯度的时间衰减加权平均。即：

$$E[g_2]_t = \gamma E[g_2]_{t-1} + (1-\gamma)g_2$$

其中 γ 与动量项中的一样，都是

RMSprop

其实RMSprop是Adadelta的中间形式，也是为了降低Adagrad中学习速率衰减过快问题，即：

$$E[g_2]_t = \gamma E[g_2]_{t-1} + (1-\gamma)g_2$$

$$\theta_{t+1} = \theta_t - \eta E[g_2]_t + \epsilon - \sqrt{E[g_2]_t} \odot g_t$$

Hinton建议 $\gamma=0.9, \eta=0.001$

Adam

Adaptive Moment Estimation(Adam) 也是一种不同参数自适应不同学习速率方法，与Adadelta与RMSprop区别在于，它计算历史梯度衰减方式不同，不使用历史平方衰减，其衰减方式类似动量，如下：

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t$$

m_t 与 v_t 分别是梯度的带权平均和带权有偏方差，初始为0向量，Adam的作者发现他们倾向于0向量(接近于0向量)，特别是在衰减因子(衰减率) β_1, β_2 接近于1时。为了改进这个问题，对 m_t 与 v_t 进行偏差修正(bias-corrected)：

$$\hat{m}_t = m_t / (1 - \beta_1^t)$$

$$\hat{v}_t = v_t / (1 - \beta_2^t)$$

最终，Adam的更新方程为：

$$\theta_{t+1} = \theta_t - \eta \hat{v}_t + \epsilon + \hat{m}_t$$

论文中建议默认值： $\beta_1=0.9, \beta_2=0.999, \epsilon=10^{-8}$ 。论文中将Adam与其它的几个自适应学习速率进行了比较，效果均要好。

各优化方法比较

下面两幅图可视化形象地比较上述各优化方法，详细参见[这里](#)，如图：

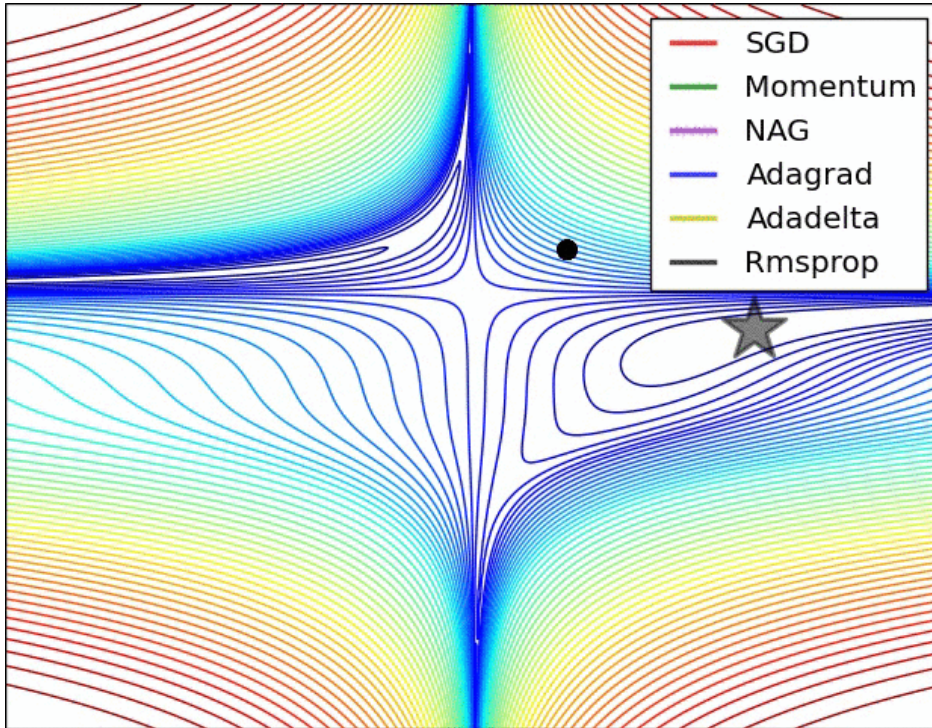


图5 SGD各优化方法在损失曲面上的表现

从上图可以看出，Adagrad、Adadelta与RMSprop在损失曲面上能够立即转移到正确的移动方向上达到快速的收敛。而Momentum与NAG会导致偏离(off-track)。同时NAG能够在偏离之后快速修正其路线，因为其根据梯度修正来提高响应性。

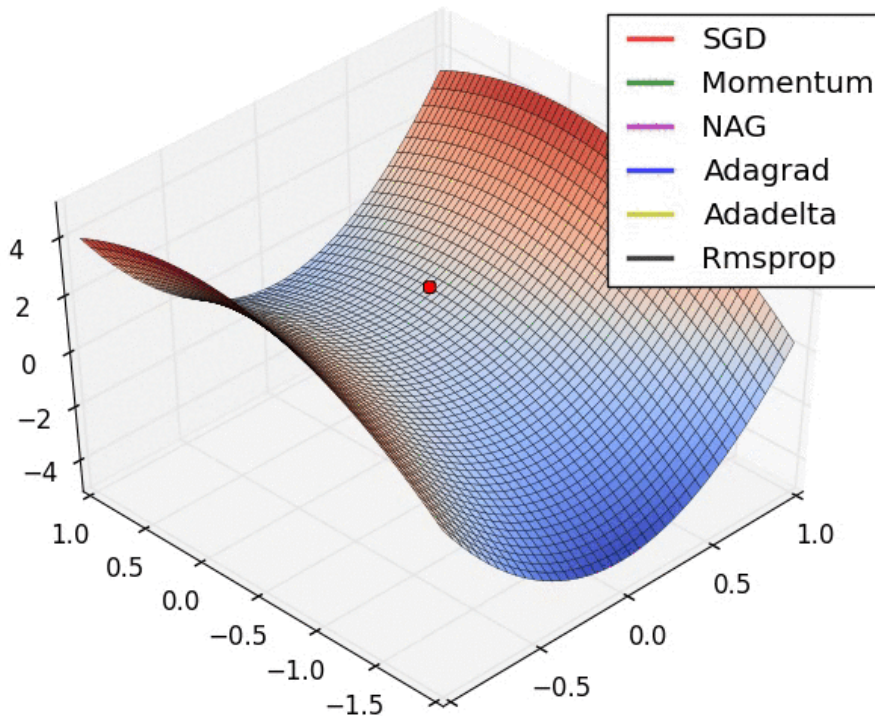


图6 SGD各优化方法在损失曲面鞍点处上的表现

从上图可以看出，在鞍点（saddle points）处（即某些维度上梯度为零，某些维度上梯度不为零），SGD、Momentum与NAG一直在鞍点梯度为零的方向上振荡，很难打破鞍点位置的对称性；Adagrad、RMSprop与Adadelta能够很快地向梯度不为零的方向上转移。

从上面两幅图可以看出，自适应学习速率方法(Adagrad、Adadelta、RMSprop与Adam)在这些场景下具有更好的收敛速度与收敛性。

如何选择SGD优化器

如果你的数据特征是稀疏的，那么你最好使用自适应学习速率SGD优化方法(Adagrad、Adadelta、RMSprop与Adam)，因为你不需要在迭代过程中对学习速率进行人工调整。

RMSprop是Adagrad的一种扩展，与Adadelta类似，但是改进版的Adadelta使用RMS去自动更新学习速率，并且不需要设置初始学习速率。而Adam是在RMSprop基础上使用动量与偏差修正。RMSprop、Adadelta与Adam在类似的情形下的表现差不多。[Kingma\[15\]](#)指出收益于偏差修正，Adam略优于RMSprop，因为其在接近收敛时梯度变得更加稀疏。因此，Adam可能是目前最好的SGD优化方法。

有趣的是，最近很多论文都是使用原始的SGD梯度下降算法，并且使用简单的学习速率退火调整（无动量

项)。现有的已经表明：SGD能够收敛于最小值点，但是相对于其他的SGD，它可能花费的时间更长，并且依赖于鲁棒的初始值以及学习速率退火调整策略，并且容易陷入局部极小值点，甚至鞍点。因此，如果你在意收敛速度或者训练一个深度或者复杂的网络，你应该选择一个自适应学习速率的SGD优化方法。

并行与分布式SGD

如果你处理的数据集非常大，并且有机器集群可以利用，那么并行或分布式SGD是一个非常好的选择，因为可以大大地提高速度。SGD算法的本质决定其是串行的(step-by-step)。因此如何进行异步处理便是一个问题。虽然串行能够保证收敛，但是如果训练集大，速度便是一个瓶颈。如果进行异步更新，那么可能会导致不收敛。下面将讨论如何进行并行或分布式SGD，并行一般是指在同一机器上进行多核并行，分布式是指集群处理。

- Hogwild

[Niu\[23\]](#)提出了被称为Hogwild的并行SGD方法。该方法在多个CPU时间进行并行。处理器通过共享内存来访问参数，并且这些参数不进行加锁。它为每一个cpu分配不重叠的一部分参数（分配互斥），每个cpu只更新其负责的参数。该方法只适合处理数据特征是稀疏的。该方法几乎可以达到一个最优的收敛速度，因为cpu之间不会进行相同信息重写。

- Downpour SGD

Downpour SGD是[Dean\[4\]](#)提出的在DistBelief(Google TensorFlow的前身)使用的SGD的一个异步变种。它在训练子集上训练同时多个模型副本。这些副本将各自的更新发送到参数服务器(PS,parameter server)，每个参数服务器只更新互斥的一部分参数，副本之间不会进行通信。因此可能会导致参数发散而不利于收敛。

- Delay-tolerant Algorithms for SGD

[McMahan与Streeter\[12\]](#)扩展AdaGrad，通过开发延迟容忍算法(delay-tolerant algorithms)，该算法不仅自适应过去梯度，并且会更新延迟。该方法已经在实践中表明是有效的。

- TensorFlow

[TensorFlow\[13\]](#)是Google开源的一个大规模机器学习库，它的前身是DistBelief。它已经在大量移动设备上或者大规模分布式集群中使用了，已经经过了实践检验。其分布式实现是基于图计算，它将图分割成多个子图，每个计算实体作为图中的一个计算节点，他们通过Rend/Receive来进行通信。具体参见[这里](#)。

- Elastic Averaging SGD

[Zhang等\[14\]](#)提出Elastic Averaging SGD(EASGD)，它通过一个elastic force(存储参数的参数服务器中心)来连接每个work来进行参数异步更新。This allows the local variables to fluctuate further from the center variable, which in theory allows for more exploration of the parameter space. They show empirically that this increased capacity for exploration leads to improved performance by finding new local optima. 这句话不太懂，需要去看论文。

更多的SGD优化策略

接下来介绍更多的SGD优化策略来进一步提高SGD的性能。另外还有众多其它的优化策略，可以参见[\[22\]](#)。

- Shuffling and Curriculum Learning

为了使得学习过程更加无偏，应该在每次迭代中随机打乱训练集中的样本。

另一方面，在很多情况下，我们是逐步解决问题的，而将训练集按照某个有意义的顺序排列会提高模型的性能和SGD的收敛性，如何将训练集建立一个有意义的排列被称为[Curriculum Learning\[16\]](#)。[Zaremba与Sutskever\[17\]](#)在使用Curriculum Learning来训练LSTMs以解决一些简单的问题中，表明一个相结合的策略或者混合策略比对训练集按照按照训练难度进行递增排序要好。（表示不懂，袁）

- Batch normalization

为了方便训练，我们通常会对参数按照0均值1方差进行初始化，随着不断训练，参数得到不同程度的更新，这样这些参数会失去0均值1方差的分布属性，这样会降低训练速度和放大参数变化随着网络结构的加深。

Batch normalization[\[18\]](#)在每次mini-batch反向传播之后重新对参数进行0均值1方差标准化。这样可以使用更大的学习速率，以及花费更少的精力在参数初始化点上。Batch normalization充当着正则化、减少甚至消除Dropout的必要性。

- Early stopping

在验证集上如果连续的多次迭代过程中损失函数不再显著地降低，那么应该提前结束训练，详细参见[NIPS 2015 Tutorial slides](#)，或者参见[防止过拟合的一些方法](#)。

- Gradient noise

Gradient noise[\[21\]](#)即在每次迭代计算梯度中加上一个高斯分布 $N(0, \sigma^2 t)$ 的随机误差，即

$$g_t = g_t + N(0, \sigma^2 t)$$

高斯误差的方差需要进行退火：

$$\sigma^2 t = \eta(1+t)^{-\gamma}$$

对梯度增加随机误差会增加模型的鲁棒性，即使初始参数值选择地不好，并适合对特别深层次的负责的网络进行训练。其原因在于增加随机噪声会有更多的可能性跳过局部极值点并去寻找一个更好的局部极值点，这种可能性在深层次的网络中更常见。

总结

在上文中，对梯度下降算法的三种框架进行了介绍，并且mini-batch梯度下降是使用最广泛的。随后，我们重点介绍了SGD的一些优化方法：Momentum、NAG、Adagrad、Adadelata、RMSprop与Adam，以及一些异

步SGD方法。最后，介绍了一些提高SGD性能的其他优化建议，如：训练集随机洗牌与课程学习(shuffling and curriculum learning)、batch normalization、early stopping与Gradient noise。

希望这篇文章能给你提供一些关于如何使用不同的梯度优化算法方面的指导。如果还有更多的优化建议或方法还希望大家提出来？或者你使用什么技巧和方法来更好地训练SGD可以一起交流？Thanks。

引用

- [1] Sutton, R. S. (1986). Two problems with backpropagation and other steepest-descent learning procedures for networks. Proc. 8th Annual Conf. Cognitive Science Society.
- [2] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. Neural Networks : The Official Journal of the International Neural Network Society, 12(1), 145–151. [http://doi.org/10.1016/S0893-6080\(98\)00116-6](http://doi.org/10.1016/S0893-6080(98)00116-6).
- [3] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159. Retrieved from <http://jmlr.org/papers/v12/duchi11a.html>.
- [4] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V, ... Ng, A. Y. (2012). Large Scale Distributed Deep Networks. NIPS 2012: Neural Information Processing Systems, 1–11. <http://doi.org/10.1109/ICDAR.2011.95>.
- [5] Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 1532–1543. <http://doi.org/10.3115/v1/D14-1162>.
- [6] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. Retrieved from <http://arxiv.org/abs/1212.5701>.
- [7] Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$. Doklady ANSSSR (translated as Soviet.Math.Docl.), vol. 269, pp. 543– 547.
- [8] Bengio, Y., Boulanger-Lewandowski, N., & Pascanu, R. (2012). Advances in Optimizing Recurrent Networks. Retrieved from <http://arxiv.org/abs/1212.0901>.
- [9] Sutskever, I. (2013). Training Recurrent neural Networks. PhD Thesis.
- [10] Darken, C., Chang, J., & Moody, J. (1992). Learning rate schedules for faster stochastic gradient search. Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop, (September), 1–11. <http://doi.org/10.1109/NNSP.1992.253713>.
- [11] H. Robbins and S. Monro, "A stochastic approximation method," Annals of Mathematical Statistics, vol. 22, pp. 400–407, 1951.
- [12] McMahan, H. B., & Streeter, M. (2014). Delay-Tolerant Algorithms for Asynchronous Distributed Online Learning. Advances in Neural Information Processing Systems (Proceedings of NIPS), 1–9. Retrieved from <http://papers.nips.cc/paper/5242-delay-tolerant-algorithms-for-asynchronous-distributed-online-learning.pdf>.
- [13] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). TensorFlow : Large-Scale Machine Learning on Heterogeneous Distributed Systems.
- [14] Zhang, S., Choromanska, A., & LeCun, Y. (2015). Deep learning with Elastic Averaging SGD. Neural Information Processing Systems Conference (NIPS 2015), 1–24. Retrieved from <http://arxiv.org/abs/1412.6651>.
- [15] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13
- [16] Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. Proceedings of the 26th Annual International Conference on Machine Learning, 41–48. <http://doi.org/10.1145/1553374.1553380>.
- [17] Zaremba, W., & Sutskever, I. (2014). Learning to Execute, 1–25. Retrieved from <http://arxiv.org/abs/1410.4615>.
- [18] Ioffe, S., & Szegedy, C. (2015). Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv Preprint arXiv:1502.03167v3.
- [19] Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., & Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. arXiv, 1–14. Retrieved from <http://arxiv.org/abs/1406.2572>.
- [20] Sutskever, I., & Martens, J. (2013). On the importance of initialization and momentum in deep learning. <http://doi.org/10.1109/ICASSP.2013.6639346>.
- [21] Neelakantan, A., Vilnis, L., Le, Q. V., Sutskever, I., Kaiser, L., Kurach, K., & Martens, J. (2015). Adding Gradient Noise Improves Learning for Very Deep Networks, 1–11. Retrieved from <http://arxiv.org/abs/1511.06807>.
- [22] LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R. (1998). Efficient BackProp. Neural Networks: Tricks of the Trade, 1524, 9–50. http://doi.org/10.1007/3-540-49430-8_2.

[23] Niu, F., Recht, B., Christopher, R., & Wright, S. J. (2011). Hogwild ! : A Lock-Free Approach to Parallelizing Stochastic Gradient Descent, 1–22.

[24] Duchi et al. [3] give this matrix as an alternative to the full matrix containing the outer products of all previous gradients, as the computation of the matrix square root is infeasible even for a moderate number of parameters d .

深入梯度下降(Gradient Descent)算法

参考：<http://blog.csdn.net/heyongluoyao8/article/details/52478715>

C/C++基本语法学习 STL C++ primer

分类: [机器学习 Machine learning](#),[总结概述](#)

好文要顶

关注我

收藏该文

[ranjiewen](#)
关注 - 107
粉丝 - 113
[+加关注](#)

3

0

« 上一篇：[机器学习中的范数规则化之 L0、L1与L2范数](#)
» 下一篇：[损失函数\(loss function\)](#)

posted @ 2016-10-08 15:36 ranjiewen 阅读(9178) 评论(2) 编辑 收藏

评论

#1楼 2017-07-26 16:56 | TensorSense

总结得非常好！

支持(0) 反对(0)

#2楼 2018-04-16 00:05 | 逐浪大白鲨

非常牛逼的样子，链接也不错，图文并茂，谢谢博主

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】超50万VC++源码: 大型组态工控、电力仿真CAD与GIS源码库！
【推荐】华为云11.11普惠季 血拼风暴 一促即发
【拼团】腾讯云服务器拼团活动又双叒叕来了！
【推荐】腾讯云新注册用户域名抢购1元起

腾讯云

腾讯云AMD云服务器

节省IT成本30%

1核1G AMD机型0.57元/天起

立即抢购

最新IT新闻:

- IBM营收规模再度下滑 股价跟着继续缩水
- 续航最长，华为手表却变得更薄了
- 法官批准马斯克与SEC和解 特斯拉市值损失120亿美元找谁哭
- 腾讯上线音乐短视频应用音兔
- 亚马逊更新Kindle Paperwhite，带来IPX8级防水和最高32GB空间

» [更多新闻...](#)



最新知识库文章:

- [为什么说 Java 程序员必须掌握 Spring Boot ?](#)
- [在学习中, 有一个比掌握知识更重要的能力](#)
- [如何招到一个靠谱的程序员](#)
- [一个故事看懂“区块链”](#)
- [被踢出去的用户](#)
- » [更多知识库文章...](#)

Copyright ©2018 ranjiewen