余昌黔|书山有路

个人资料

ycszen

关注 发私信

访问： 59471次
积分： 624
等级： BLOG 3
排名： 千里之外

原创： 10篇 转载： 0篇
译文： 1篇 评论： 49条

文章搜索

阅读排行

TensorFlow高效读取数据的方... (16344)
图像语义分割之FCN和CRF (14200)
MXNet安装教程 (7174)
深度学习最全优化方法总结比... (5049)
MXNet数据加载 (4877)
【解决】Ubuntu安装NVIDIA... (4343)
深度学习框架Torch7解析-- Te.. (4140)
图像语义分割之特征整合和结... (977)
Scrapy 安装问题集锦 (726)
PyTorch预训练 (548)

文章分类

mxnet (2)
深度学习 (7)
scrapy (1)
Torch7解析 (1)
tensorflow (1)
深度学习理论 (3)
环境配置 (3)
PyTorch (2)

文章存档

2017年03月 (3)
2016年11月 (1)

深度学习最全优化方法总结比较（SGD，Adagrad，Adadelta，Adam，Adamax，Nadam）

标签： 深度学习理论 优化方法

2016-08-24 18:20 5057人阅读 评论(1) 收藏 举报

分类：
深度学习（6） 深度学习理论（2）

前言

（标题不能再中二了）本文仅对一些常见的优化方法进行直观介绍和简单的比较，各种优化方法的详细内容及公式只好去认真啃论文了，在此我就不赘述了。

SGD

此处的SGD指mini-batch gradient descent，关于batch gradient descent, stochastic gradient descent, 以及 mini-batch gradient descent的具体区别就不细说了。现在的SGD一般都指mini-batch gradient descent。

SGD就是每一次迭代计算mini-batch的梯度，然后对参数进行更新，是最常见的优化方法了。即：

$$g_t = \nabla_{\theta_{t-1}} f(\theta_{t-1})$$
$$\Delta\theta_t = -\eta * g_t$$

其中，η是学习率，g_t是梯度

SGD完全依赖于当前batch的梯度，所以η可理解为允许当前batch的梯度多大程度影响参数更新

缺点：（正因为有这些缺点才让这么多大神发展出了后续的各种**算法**）

- 选择合适的learning rate比较困难
- 对所有的参数更新使用同样的learning rate。对于稀疏数据或者特征，有时我们可能想更新快一些对于不经常出现的特征，对于常出现的特征更新慢一些，这时候SGD就不太能满足要求了
- SGD容易收敛到局部最优，在某些情况下可能被困在鞍点【但是在合适的初始化和学习率设置下，鞍点的影响其实没这么大】

Momentum

momentum是模拟物理里动量的概念，积累之前的动量来替代真正的梯度。公式如下：

$$m_t = \mu * m_{t-1} + g_t$$

$$\Delta\theta_t = -\eta * m_t$$

其中，μ是动量因子

特点：

- 下降初期时，使用上一次参数更新，下降方向一致，乘上较大的μ能够进行很好的加速
- 下降中后期时，在局部最小值来回震荡的时候，$gradient \to 0$，μ使得更新幅度增大，跳出陷阱
- 在梯度改变方向的时候，μ能够减少更新

总而言之，momentum项能够在相关方向加速SGD，抑制振荡，从而加快收敛

Nesterov

nesterov项在梯度更新时做一个校正，避免前进太快，同时提高灵敏度。

将上一节中的公式展开可得：

$$\Delta\theta_t = -\eta * \mu * m_{t-1} - \eta * g_t$$

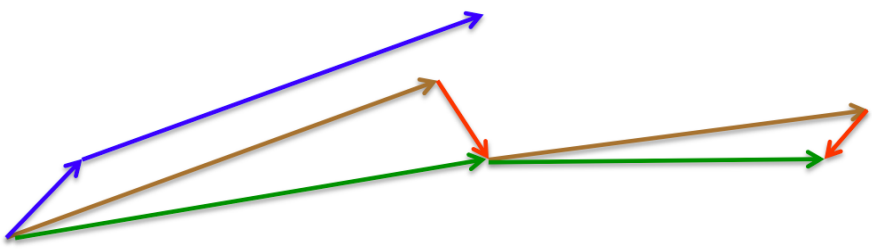
可以看出，m_{t-1}并没有直接改变当前梯度g_t，所以Nesterov的改进就是让之前的动量直接影响当前的动量。即：

$$g_t = \nabla_{\theta_{t-1}} f(\theta_{t-1} - \eta * \mu * m_{t-1})$$

$$m_t = \mu * m_{t-1} + g_t$$

$$\Delta\theta_t = -\eta * m_t$$

所以，加上nesterov项后，梯度在大的跳跃后，进行计算对当前梯度进行校正。如下图：



momentum首先计算一个梯度(短的蓝色向量)，然后在加速更新梯度的方向进行一个大的跳跃(长的蓝色向量)，nesterov项首先在之前加速的梯度方向进行一个大的跳跃(棕色向量)，计算梯度然后进行校正(绿色梯向量)

其实，momentum项和nesterov项都是为了使梯度更新更加灵活，对不同情况有针对性。但是，人工设置一些学习率总还是有些生硬，接下来介绍几种自适应学习率的方法

Adagrad

Adagrad其实是对学习率进行了一个约束。即：

$$n_t = n_{t-1} + g_t^2$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{n_t + \epsilon}} * g_t$$

此处，对g_t从1到t进行一个递推形成一个约束项regularizer，$-\frac{1}{\sqrt{\sum_{r=1}^{t}(g_r)^2+\epsilon}}$，$\epsilon$用来保证分母非0

特点：

- 前期g_t较小的时候， regularizer较大，能够放大梯度
- 后期g_t较大的时候，regularizer较小，能够约束梯度
- 适合处理稀疏梯度

缺点：

- 由公式可以看出，仍依赖于人工设置一个全局学习率
- η设置过大的话，会使regularizer过于敏感，对梯度的调节太大
- 中后期，分母上梯度平方的累加将会越来越大，使$gradient \to 0$，使得训练提前结束

Adadelta

Adadelta是对Adagrad的扩展，最初方案依然是对学习率进行自适应约束，但是进行了计算上的简化。

Adagrad会累加之前所有的梯度平方，而Adadelta只累加固定大小的项，并且也不直接存储这些项，仅仅是近似计算对应的平均值。即：

$$n_t = \nu * n_{t-1} + (1-\nu) * g_t^2$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{n_t + \epsilon}} * g_t$$

在此处Adadelta其实还是依赖于全局学习率的，但是作者做了一定处理，经过近似牛顿迭代法之后：

$$E|g^2|_t = \rho * E|g^2|_{t-1} + (1-\rho) * g_t^2$$

$$\Delta x_t = -\frac{\sqrt{\sum_{r=1}^{t-1} \Delta x_r}}{\sqrt{E|g^2|_t + \epsilon}}$$

其中，E代表求期望。

此时，可以看出Adadelta已经不用依赖于全局学习率了。

特点：

- 训练初中期，加速效果不错，很快
- 训练后期，反复在局部最小值附近抖动

RMSprop

RMSprop可以算作Adadelta的一个特例：

当$\rho = 0.5$时，$E|g^2|_t = \rho * E|g^2|_{t-1} + (1-\rho) * g_t^2$就变为了求梯度平方和的平均数。

如果再求根的话，就变成了RMS(均方根)：

$$RMS|g|_t = \sqrt{E|g^2|_t + \epsilon}$$

此时，这个RMS就可以作为学习率η的一个约束：

$$\Delta x_t = -\frac{\eta}{RMS|g|_t} * g_t$$

特点：

- 其实RMSprop依然依赖于全局学习率
- RMSprop算是Adagrad的一种发展，和Adadelta的变体，效果趋于二者之间
- 适合处理非平稳目标
- 对于RNN效果很好

Adam

Adam(Adaptive Moment Estimation)本质上是带有动量项的RMSprop，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。Adam的优点主要在于经过偏置校正后，每一次迭代学习率都有个确定范围，使得参数比较平稳。公式如下：

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

$$n_t = \nu * n_{t-1} + （1-\nu） * g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\mu^t}$$

$$\hat{n}_t = \frac{n_t}{1-\nu^t}$$

$$\Delta\theta_t = -\frac{\hat{m}_t}{\sqrt{\hat{n}_t}+\epsilon} * \eta$$

其中，m_t，n_t分别是对梯度的一阶矩估计和二阶矩估计，可以看作对期望$E|g_t|$，$E|g_t^2|$的估计；$\hat{m}_t$，$\hat{n}_t$是对m_t，n_t的校正，这样可以近似为对期望的无偏估计。

可以看出，直接对梯度的矩估计对内存没有额外的要求，而且可以根据梯度进行动态调整，而$-\frac{\hat{m}_t}{\sqrt{\hat{n}_t}+\epsilon}$对学习率形成一个动态约束，而且有明确的范围。

特点：

- 结合了Adagrad善于处理稀疏梯度和RMSprop善于处理非平稳目标的优点
- 对内存需求较小
- 为不同的参数计算不同的自适应学习率
- 也适用于大多非凸优化
- 适用于大数据集和高维空间

Adamax

Adamax是Adam的一种变体，此方法对学习率的上限提供了一个更简单的范围。公式上的变化如下：

$$n_t = max(\nu * n_{t-1}, |g_t|)$$

$$\Delta x = -\frac{\hat{m}_t}{n_t+\epsilon} * \eta$$

可以看出，Adamax学习率的边界范围更简单

Nadam

Nadam类似于带有Nesterov动量项的Adam。公式如下：

$$\hat{g}_t = \frac{g_t}{1-\Pi_{i=1}^t \mu_i}$$

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

$$\hat{m}_t = \frac{m_t}{1 - \Pi_{i=1}^{t+1}\mu_i}$$

$$n_t = \nu * n_{t-1} + （1 - \nu） * g_t^2$$

$$\hat{n}_t = \frac{n_t}{1 - \nu^t}$$

$$\bar{m}_t = (1 - \mu_t) * \hat{g}_t + \mu_{t+1} * \hat{m}_t$$

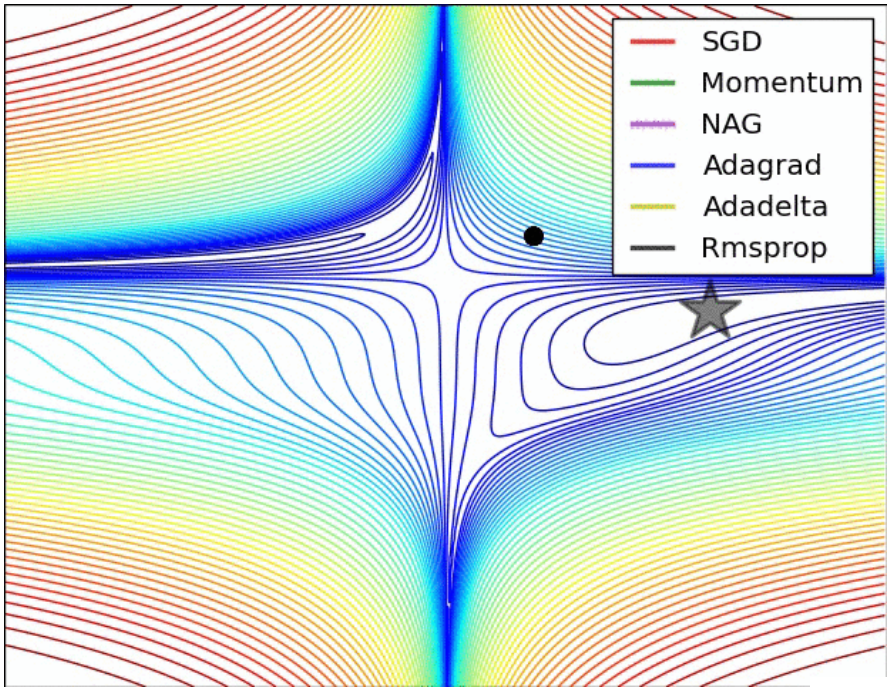
$$\Delta\theta_t = -\eta * \frac{\bar{m}_t}{\sqrt{\hat{n}_t} + \epsilon}$$

可以看出，Nadam对学习率有了更强的约束，同时对梯度的更新也有更直接的影响。一般而言，在想使用带动量的RMSprop，或者Adam的地方，大多可以使用Nadam取得更好的效果。

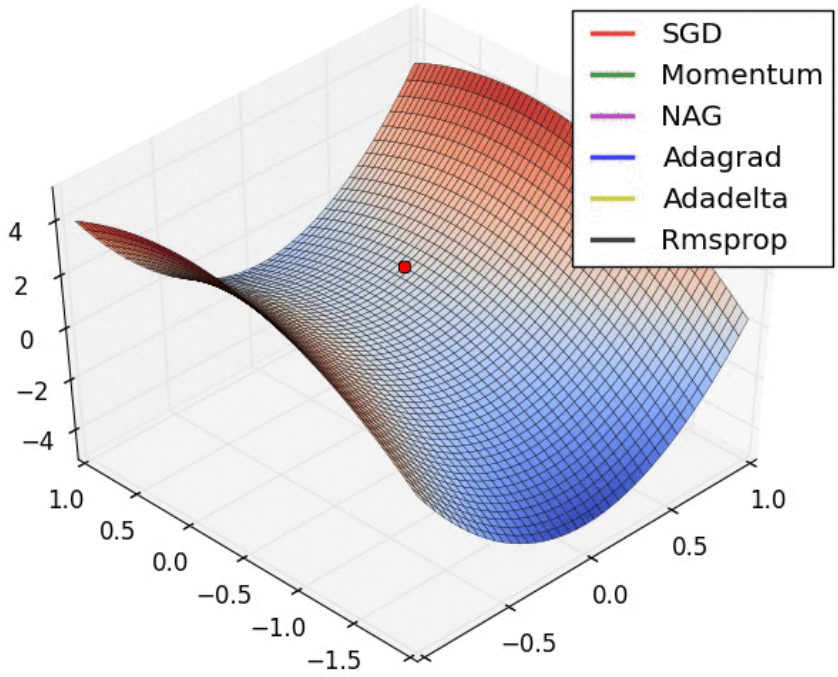
经验之谈

- 对于稀疏数据，尽量使用学习率可自适应的优化方法，不用手动调节，而且最好采用默认值
- SGD通常训练时间更长，容易陷入鞍点，但是在好的初始化和学习率调度方案的情况下，结果更可靠
- 如果在意更快的收敛，并且需要训练较深较复杂的网络时，推荐使用学习率自适应的优化方法。
- Adadelta，RMSprop，Adam是比较相近的算法，在相似的情况下表现差不多。
- 在想使用带动量的RMSprop，或者Adam的地方，大多可以使用Nadam取得更好的效果

最后展示两张可厉害的图，一切尽在图中啊，上面的都没啥用了... ...



损失平面等高线



在鞍点处的比较

引用

[1]Adagrad

[2]RMSprop[Lecture 6e]

[3]Adadelta

[4]Adam

[5]Nadam

[6]On the importance of initialization and momentum in deep learning

[7]Keras 中文文档

[8]Alec Radford(图)

[9]An overview of gradient descent optimization algorithms

[10]Gradient Descent Only Converges to Minimizers

[11]Deep Learning:Nature

- 上一篇 TensorFlow高效读取数据的方法
- 下一篇 图像语义分割之FCN和CRF

相关文章推荐

- 深度学习最全优化方法总结比较（SGD，Adagrad...
- 深度学习与计算机视觉系列(3)_线性SVM与SoftMa...
- 深度学习与计算机视觉系列(5)_反向传播与它的直...
- 深度学习情感分析
- 深度学习中的数学与技巧(0)：优化方法总结比较（...
- 深度学习与计算机视觉系列(4)_最优化与随机梯度
- 转：深度学习笔记：优化方法总结(BGD,SGD,Mom...
- 深度学习与计算机视觉系列(2)_图像
- 深度学习框架Keras使用心得
- java 动态代理深度学习[转]

查看评论

xiaohe_nh　　1楼 2016-10-09 17:20发表

关于Nadam，μi的值是如何得到的？