深度学习中的优化算法

概述

基于随机梯度下降的优化算法是机器学习和深度学习中的核心优化算法。该领域中的许多问题都可以看作是最大化和最小化一个参数化的损失函数，在大多数情况下损失函数本身是可微的，由于问题数据规模极大，所以一般情况下获得二阶信息是很难的，存储相应的二阶信息也是消耗非常大的一件事，所以梯度下降，这样一种仅仅要求一阶偏导数的优化算法在这些问题上很有效。而在深度学习领域，即使数据集和网络模型架构完全相同，使用不同的优化算法，也可能得到相差非常大的最终结果，此外，在数据收集和采样进行训练的时候可能会有比较严重的噪声影响，简单的梯度下降方法往往无法奏效，因此，为了弥补朴素梯度下降的种种缺陷，就需要一系列对梯度下降系列优化算法的的改进，也就因此诞生了SGD with momentum， Adagrad，RMSprop，Adam等一系列变种算法

算法介绍

朴素SGD及mini-batch SGD

朴素SGD即最基本的最速下降法，使用一个梯度步作为方向，步长作为一个可调控因子随时调整，那么可简单写为：

- $W_{k+1} = W_k - \alpha_k g_k$
- 其中W是参数，α 是学习率，即步长，g是梯度

在深度学习的实践当中，通常选择学习率是一个根据迭代次数逐渐衰减的量，即给定一个初始学习率然后让它随时迭代次 数逐渐衰减，或者是训练到一定迭代次数后进行衰减。朴素的SGD的问题就是其收敛速度非常慢，而且容易陷入局部最优点和鞍点，很难从其中逃出。同时，由于学习率在实践中往往是根据经验来调整，而一个合适的学习率调整策略对整个问题的优化过程是十分重要的，所以如何选择这样合适的学习率更是SGD的最大难点。

在深度学习环境下，我们有非常多的训练数据，朴素的想法是对每一个数据使用SGD更新一次，这样能获得很好的对数据的泛化能力，但是与之而相对的，如果数据的分布极不均衡，数据中有很多的噪声，那么这样的策略将会“带偏”我们的模型，因此，一个自然而然的想法就是每次使用一批数据（batch）来计算梯度，然后将梯度平均之后再使用SGD去更新参数，通过平均一批样本的梯度结果来减少震荡和其他影响，这就是所谓的mini-batch SGD的基本思想。在实际操作中，batch的大小一般根据计算资源的存储能力，训练时间开销，泛化能力要求综合来定，batch越小泛化能力越强，但时间开销就大且容易发生震荡，batch越大训练速度越快但是可能丧失一定的泛化能力，所以一般使用权衡两边的mini-batch。

SGD with momentum[1]

上文中提到朴素SGD收敛速度很慢且容易震荡，借用物理中动量的概念，我们希望结合历史的梯度信息和当前梯度信息来共同指导更新，通俗来讲，就是遇到陡坡时，希望运动得更快，遇到复杂的沟壑时，希望借助之前的惯性尽快冲出，遇到突然变化的情况，也希望由于惯性的作用方向不要变化太大。借由以上直觉和概念，就产生带有动量的梯度下降，即SGD with momentum，其表达式为：

- $m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k$
- $W_{k+1} = W_k - \alpha_k m_k$

这里β_1 是一个加权因子，一般取0.9，这个形式其实就是对历史的梯度信息做因子为β_1的指数加权平均，然后结合当前的梯度信息构成下一步更新的方向，由于这个因子大于0.5，这就意味着下降方向主要是此前累积的下降方向，然后略微偏向当前时刻的梯度方向。这样的一种改进，使得参数中那些梯度方向变化不大的维度可以加速更新，同时减少梯度方向变化较大的维度上的更新幅度，由此产生了加速收敛和减小震荡的效果。

Adagrad[2]

之前我们提到的优化算法都是对整个模型的所有参数，以相同的学习率去更新，但是深度学习模型是非常大的，往往有上亿个参数，采用统一的学习率是一种很简单粗暴的策略。而在不同的任务中，不同的网络层的参数所需要的更新频率往往有所区别。对于那些更新不频繁的参数，我们了解到的信息较少，所以希望每次更新时单次步长更大，能够多学习到一些知识，同时对于那些更新非常频繁的参数，我们已经累积了大量关于它的学习经验和知识，所以希望采用较小的步长，不要被一部分样本影响过大，能够学到更稳定的参数值。由此，我们需要一个量来衡量历史更新频率，直观的想法就是二阶动量，即某一维上，迄今为止所有梯度值的平方和，可以写作：

- $V_t = diag(\sum_{i=1}^t g_{1i}^2, \sum_{i=1}^t g_{2i}^2, \cdots, \sum_{i=1}^t g_{ni}^2)$
- 其中V_t是一个对角矩阵，每个对角线上的元素为对应维初始时刻到时刻t的梯度平方和

有了这样一个度量，我们就可以将momentum中的学习率进行替换，得到完整的表达式为：

- $\eta_k = \frac{\alpha_k}{\sqrt{V_t + \epsilon}}$
- $W_{k+1} = W_k - \eta_k g_k$
- 其中$\epsilon = 1e-8$ 为了保证数值计算的稳定性。

对于以前更新频繁的参数，其二阶动量对应的分量就比较大，所以对应的学习率就较小，反之就较大。Adagrad也可以说是进入所谓“自适应学习率”的时代。

RMSprop[3]

对于Adagrad，有一个很明显的问题，就是V_t 是一个只会累积增加的量，即是单调递增的，这在稀疏数据场景下表现非常好，但是对于稠密的数据集，这会导致学习率在训练的中后期单调递减至0，对于后续的训练样本，由于学习率已经接近于0，所以模型很难学到新的必要的知识。因此，我们需要对Adagrad做一定的修正，借鉴momentum中对历史信息的处理方式，我们对于二阶动量也采用指数加权平均的方法来处理，不累积全部的历史梯度，而只关注最近某一时间窗口内的梯度累积量，由此便得到了RMSprop：

- $V_t = \beta_2 V_{t-1} + (1-\beta_2) diag(g_t^2)$
- $\eta_k = \frac{\alpha_k}{\sqrt{V_t + \epsilon}}$
- $W_{k+1} = W_k - \eta_k g_k$

Adam[4]

在上面的方法中我们讲到了基于一阶动量和二阶动量的改进方法，那么如果将他们结合起来呢，Adam就是这样一种结合两者的改进优化算法，也是整个SGD系列的集大成者，既包含来自momentum的累积历史梯度对方向的修正，又包含来自RMSprop的对于学习率的修正，最终的完整表达式如下：

- $m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k$
- $V_t = \beta_2 V_{t-1} + (1-\beta_2) diag(g_t^2)$
- $\eta_k = \frac{\alpha_k}{\sqrt{V_t + \epsilon}}$
- $W_{k+1} = W_k - \eta_k m_k$
- 其中$\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-8$ （一般情况）

算法对比结果

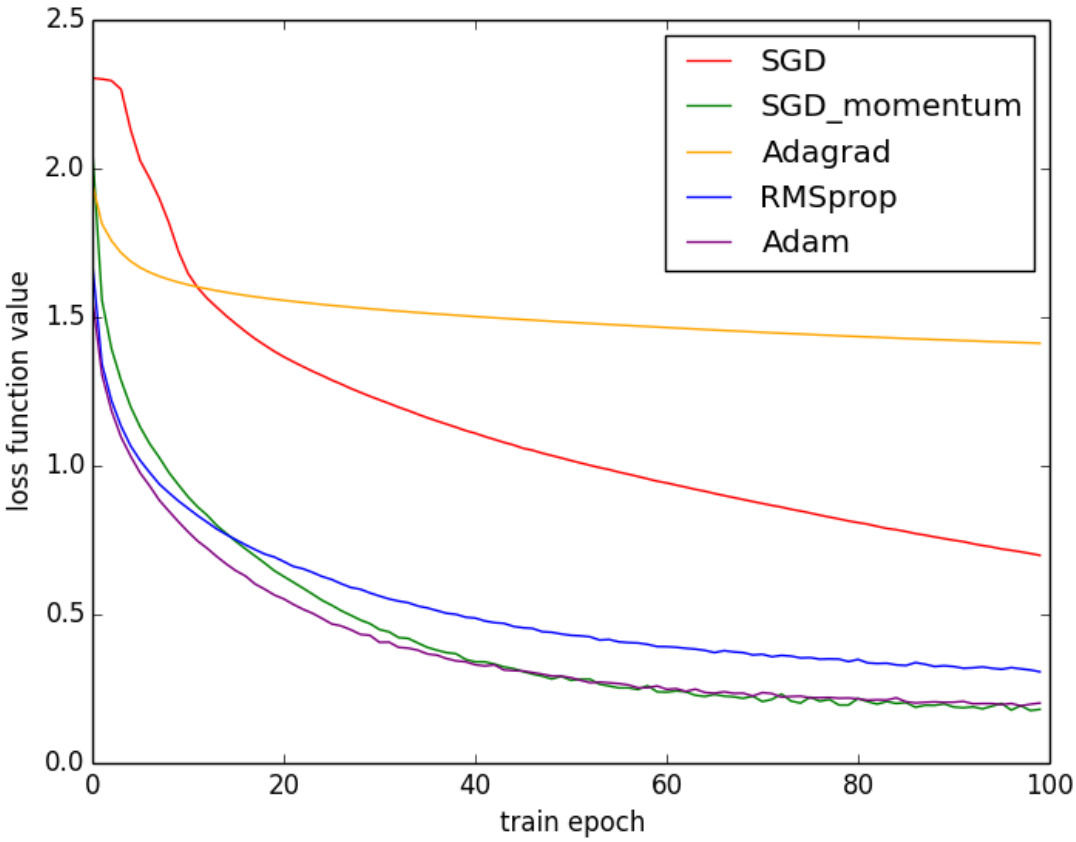
实验环境

- 对于上文中提到的五种优化算法，使用pytorch深度学习框架中的相应实现进行测试，选定实验所用数据集CIFAR-10图像分类数据集，使用相同的Lenet-5网络结构进行对比试验。
- 训练具体参数配置：网络模型权重初始化使用默认初始化，即从[-1, 1]的均匀分布中进行采样来初始化权重。初始学习率为1e-3，没有算法本身之外的学习率调控，batch大小取16，共计训练100轮（epoch），使用参数为1e-5的L2正则对损失函数加以处理，损失函数（即目标函数）选择图像分类时常用的CrossEntropy交叉熵损失函数。

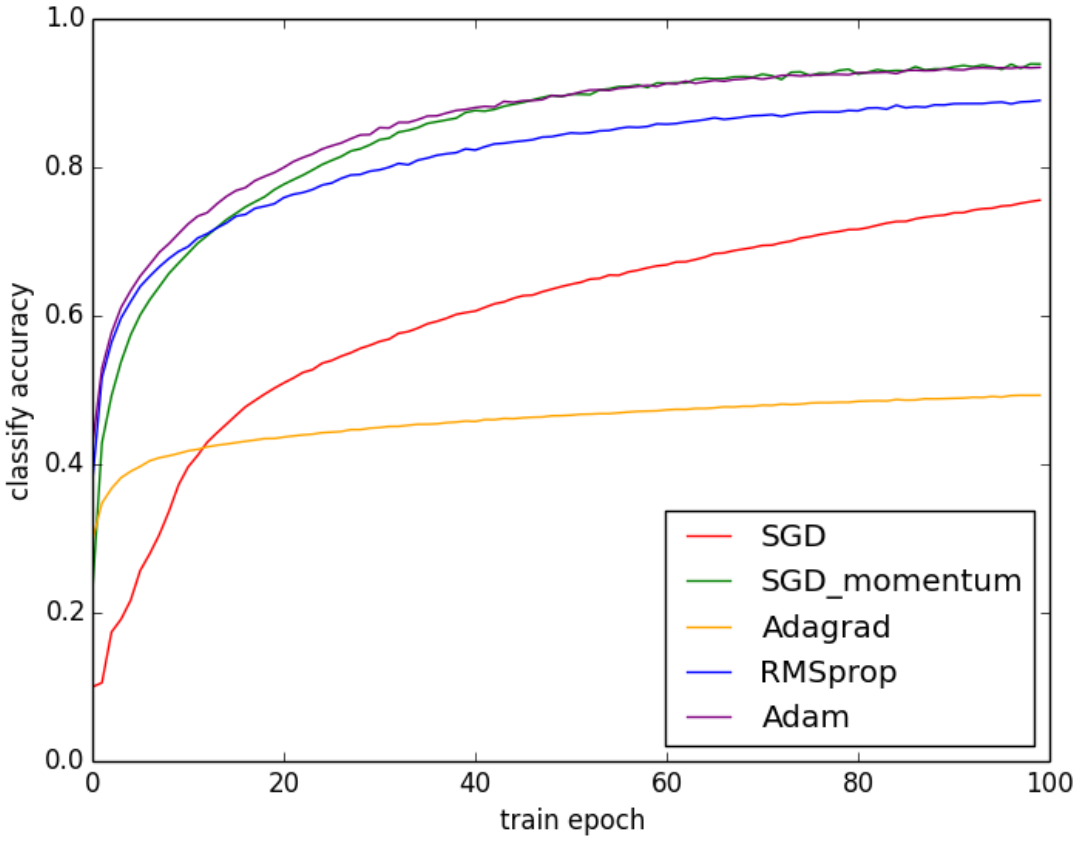
实验结果

训练100轮之后将损失函数值和图像分类的准确率结果为纵轴，训练轮次为横轴，得到下列结果图：

训练轮次-损失函数值图：


SGD
SGD_momentum
Adagrad
RMSprop
Adam
loss function value
train epoch
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100

训练轮次-分类准确率图：


SGD
SGD_momentum
Adagrad
RMSprop
Adam
classify accuracy
train epoch
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100

内容分析

- 从上述结果图可以看出，以红色的朴素SGD为基准，几种改进方法在图中初期都有更快的收敛速度。
- Adagrad在稠密数据集上的收敛速度到了中后期就会非常慢，这说明了直接暴力地累积二阶梯度确实存在问题
- 基于一阶动量的momentum和Adam有着最优秀的收敛结果，这说明加入一阶动量来修正方向确认可以帮助训练更好的跳出局部最优点和鞍点，收敛到更好的结果
- 基于二阶动量的RMSprop在初期拥有比不使用二阶动量方法更快的收敛速度，但是由于控制学习率的缘故所以在本次实验中后期收敛结果要部分低于momentum
- 综合两者的Adam算法结合了两者的优势，既有更快的收敛速度又有很好的收敛结果
- 由于本次实验的数据集和网络模型本身属于较简单的种类，因此关于SGD相比Adam更强的泛化能力，二阶动量方法对于复杂模型不同参数不同学习率的优势等并没有明显体现出来。

参考文献

[1] 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.

[2] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159.

[3] Tieleman, T. and Hinton, G. Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Technical report, 2012.

[4] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13