Artificial Intelligence and Robotics Research 人工智能与机器人研究, 2022, 11(4), 448-462
Published Online November 2022 in Hans. http://www.hanspub.org/journal/airr
https://doi.org/10.12677/airr.2022.114046

Hans汉斯

深度学习中的优化算法研究

陈　爽，张长伦，黎铭亮

北京建筑大学理学院，北京

收稿日期：2022年10月26日；录用日期：2022年11月14日；发布日期：2022年11月22日

摘　要

深度学习作为处理神经网络的一个热门研究方向，在近些年来备受关注。深度学习模型是一个多层次的网络结构，评价模型最终效果优劣的网络参数必须通过深度学习优化器进行训练，因此深度学习中的优化算法成为了国内外的研究热点。本文对深度学习中的一阶优化算法进行综述，首先介绍了经典的随机梯度下降及其动量变体优化算法，然后介绍了近年更流行的自适应学习率优化算法，最后对未来深度学习中优化算法的发展进行了总结与展望。

关键词

深度学习，优化算法

Research on Optimization Algorithms in Deep Learning

Shuang Chen, Changlun Zhang, Mingliang Li

School of Science, Beijing University of Civil Engineering and Architecture, Beijing

Received: Oct. 26th, 2022; accepted: Nov. 14th, 2022; published: Nov. 22nd, 2022

Abstract

As a popular research direction for processing neural networks, deep learning has attracted much attention in recent years. The deep learning model is a multi-layered network structure, and the network parameters that evaluate the final effect of the model must be trained by the deep learning optimizers, so the optimization algorithms in deep learning have become a research hotspot at home and abroad. In this paper, the first-order optimization algorithms in deep learning are reviewed. Firstly, the classical stochastic gradient descent and its momentum variant optimization algorithms are introduced. Then, the more popular adaptive learning rate optimization algorithms

文章引用: 陈爽, 张长伦, 黎铭亮. 深度学习中的优化算法研究[J]. 人工智能与机器人研究, 2022, 11(4): 448-462.
DOI: 10.12677/airr.2022.114046

in recent years are introduced. Finally, the development of optimization algorithms in deep learning in the future is summarized and prospected.

Keywords

Deep Learning, Optimization Algorithms

Copyright © 2022 by author(s) and Hans Publishers Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

1. 引言

2006 年，Hinton 等人提出了一种新的机器学习研究领域——深度学习[1]，通过建立多层的神经网络结构来模拟人脑的多层次学习过程，在计算机视觉、自然语言处理等方向发挥着显著的优势，广泛应用于自动驾驶、目标识别、图像处理等领域[2] [3] [4] [5]。

深度学习模型是一个多层次的网络结构，由输入层、多个隐藏层和输出层构成，网络的每一层中都含有大量的参数，主要包括权重矩阵 w 和偏差向量 b，这些参数的取值直接决定了网络模型的优劣。为了获得优秀的深度学习模型，需要在网络模型训练时使用深度学习优化算法对模型中的参数进行优化更新。

深度学习中的优化算法[6] [7] [8]主要分为一阶算法和二阶算法：一阶算法基于梯度下降法[9]，是目前深度学习理论和实际应用中的主导优化方法；二阶算法有牛顿法[10] [11]和共轭梯度法[12] [13]等，由于二阶优化算法需要进行二阶求导，受限于计算量与内存成本，很少应用在深度学习的各项任务中。本文着眼于对当前深度学习模型中广泛流行的一阶优化算法进行介绍。

2. 梯度下降法

梯度下降法(Gradient Descent)最初是最优化理论中的基础方法，常常用于求解无约束最优化问题，广泛应用于目标追踪、工程设计和产品优化等方面[14] [15] [16] [17]，后来拓展到机器学习和深度学习领域中。

深度学习网络模型训练的目标是最小化损失函数，即尽可能降低目标真实值和模型预测值之间的差距，如果将损失函数看作目标函数，那么任务目标等价于求解一个最优化问题。梯度下降法正是一个可以找到最小损失函数值的优化算法，它首先利用所定义的损失函数来计算当前参数值的损失，然后计算网络中每个参数的梯度，并且在梯度的相反方向上通过与梯度成比例的因子更新参数值，重复上述步骤直到符合需要。

网络训练时，根据每次更新使用的样本数据量大小，可以将梯度下降法分为三个种类：批量梯度下降(Batch Gradient Descent)、随机梯度下降(Stochastic Gradient Descent)以及小批量梯度下降(Mini-Batch Gradient Descent)。

批量梯度下降法是最原始的形式，指的是在每一次迭代时使用全部样本来进行模型参数的更新。它的优点在于使用全部数据可以很准确地朝目标值所在的方向梯度下降，并且实现了并行运算；缺点在于当数据量过大时，一次迭代的时间成本较大，训练过程较慢。

随机梯度下降法不同于批量梯度下降法选择全部样本进行参数更新，而是在每一次迭代时在所有样本数据中随机选择一个来进行模型参数的更新。这样的好处是可以大大加快训练过程，实现了在线更新；

但是弊端是由于每次只随机选择一个样本进行更新，会有随机的噪声波动，使得每次梯度下降的方向不准确，很可能无法收敛到全局最优值。

小批量梯度下降法，是介于批量梯度下降法和随机梯度下降法之间的一个折中方法，每一次迭代时随机使用固定数量的样本数据进行参数更新，这个固定数量被叫做批大小(Batch size)，是小于样本数据总数的一个值，实践中往往选取 16、32 和 64 等值。小批量梯度下降法兼顾了训练时间和更新速度，是深度学习中最常使用的方法。

在深度学习中，常常把随机梯度下降法和小批量梯度下降法都称为随机梯度下降法，下文中我们提到的随机梯度下降法所指代的是它们两者。

3. 历史发展与现状

为了寻找最优参数，大部分深度学习神经网络中采取的最基本方法是随机梯度下降法(SGD) [18] [19]。根据参数更新时实际学习率是否固定为常数，可以将 SGD 算法大体分为两类：第一类是随机梯度下降及其动量变体算法，第二类是自适应学习率算法。

3.1. 随机梯度下降及其动量变体算法

随机梯度下降法在训练集所有的 n 个样本中，随机选取 $m\left(m \le n\right)$ 个独立同分布的小批量样本 $\left\{x^{(1)},\cdots,x^{(m)}\right\}$，其中 $x^{(i)}$ 对应目标为 $y^{(i)}$，然后利用所定义的目标优化函数 L 在小批量样本上计算网络中每个参数的梯度，之后计算它们的梯度均值获得梯度的估计，最后在负梯度方向上通过学习率超参数和梯度均值来更新参数值，重复上述步骤直到达到停止条件。

SGD 中，参数 θ 在第 t 次迭代时更新为：$\theta_t = \theta_{t-1} - \alpha \times g_t$，其中，$\theta_{t-1}$ 和 θ_t 分别是参数先前值和更新值，α 为固定的学习率超参数，g_t 是第 t 次迭代时的参数梯度。算法 1 中展示了随机梯度下降法的伪代码流程。

算法 1：随机梯度下降算法(SGD)
输入：学习率 α，初始参数 θ_0
初始化：时间步 $t=0$
当没有达到停止准则时，执行
1) 时间步更新：$t \leftarrow t+1$
2) 计算梯度：$g_t \leftarrow \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)};\theta_{t-1}\right),y^{(i)}\right)$
3) 参数更新：$\theta_t \leftarrow \theta_{t-1} - \alpha \times g_t$

基本的随机梯度下降法有以下四个主要缺点：1) 梯度高度敏感于参数空间的某些方向[20]；2) 如果目标损失函数有局部最小值或鞍点，由于该区域中的零梯度，参数更新会卡住[21]；3) 不是在全批量而是在小批量上计算梯度，会有随机的噪声扰动；4) 每个参数采取相同的学习率超参数进行更新，需要小心设置学习率的值。

为改进随机梯度下降法的几个缺陷，设计更适合神经网络的优化方法，研究人员陆续提出了几种 SGD 变体。具有动量的随机梯度下降法(SGD with Momentum, SGDM) [22] [23]是 SGD 的一个广泛使用的拓展。鉴于神经网络中的优化目标函数并不总是严格的凸函数，所以使用经典的 SGD 时参数很可能会由于局部最小值或鞍点处梯度近似为零，困在此处无法更新。SGDM 考虑这个问题，通过结合过去的梯度信息来保持

一个“动量”，使得参数可以跳出局部最小值和鞍点。物理学中，动量表示物体的质量和速度的乘积，指的是运动物体在它运动方向上保持运动的趋势，SGDM 中把参数看作单位质点，梯度看作力，当力(梯度)改变时参数会随之加速或减速。SGDM 将动量定义为梯度的指数移动平均值(EMA)，同时引入动量超参数 β 用于调节先前梯度的衰减效果，相对于学习率超参数，动量超参数越大，历史梯度对当前方向的影响也越大。

在 SGDM 中，过去的梯度被合并以获得具有一致梯度的参数的动量，目标是在任何具有一致梯度的维度上发展高“速度”。直观的理解为：要是当前时刻的梯度与历史时刻梯度方向相似，这种趋势在当前时刻则会加强；要是不同，则当前时刻的梯度方向减弱。前者能够加速收敛，后者能够减小摆动。如图 1 所示，动量方法解决了 SGD 中梯度敏感于参数空间某些方向和易陷入局部最小值或鞍点的问题，并且加快了参数的学习和收敛[24]。SGDM 的伪代码如算法 2 所示。

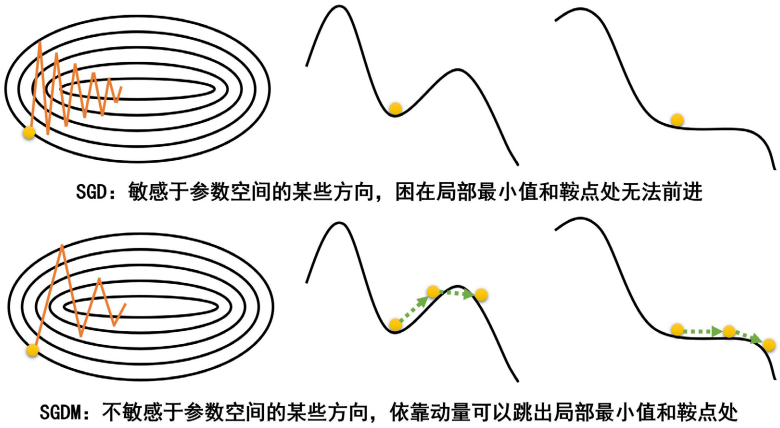

Figure 1. Comparison between SGD and SGDM
图 1. SGD 与 SGDM 的对比

算法 2：SGDM 优化器

输入：学习率 α，初始参数 θ_0，动量超参数 β

初始化：时间步 $t=0$，动量 $m_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t \leftarrow t+1$

2) 计算梯度：$g_t \leftarrow \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)};\theta_{t-1}\right),y^{(i)}\right)$

3) 计算动量：$m_t \leftarrow \beta m_{t-1}+g_t$

4) 参数更新：$\theta_t \leftarrow \theta_{t-1}-\alpha \times m_t$

2013 年，Sutskever 等[25]受到 Nesterov 动量的启发[26] [27]，提出了具有动量的随机梯度下降法的另一个变体——NAG 优化器(Nesterov’s Accelerated Gradient)。NAG 和 SGDM 的差别在于：NAG 先跟随历史累计梯度信息更新一步到临时参数点，然后再计算临时参数点处的梯度，将该梯度加入动量后更新到实际参数点；SGDM 则不“先走一步”进行临时更新，而是直接计算梯度和动量来更新参数。NAG 比 SGDM 多应用了一次梯度信息，对参数的当前梯度方向进行了修正，得到了更为精确的更新梯度，进一步地提升了 SGDM 的算法稳定性，并且加快了收敛速度。NAG 算法的伪代码如算法 3 所示。

算法 3：NAG 优化器

输入：学习率 α，初始参数 θ_0，动量超参数 β

初始化：时间步 $t=0$，动量 $m_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t \leftarrow t+1$

2) 临时参数更新：$\tilde{\theta}_{t-1} \leftarrow \theta_{t-1} - \beta m_{t-1}$

3) 计算梯度：$g_t \leftarrow \frac{1}{m}\nabla_{\tilde{\theta}} \sum_i L\left(f\left(x^{(i)};\tilde{\theta}_{t-1}\right), y^{(i)}\right)$

4) 计算动量：$m_t \leftarrow \beta m_{t-1} + g_t$

5) 参数更新：$\theta_t \leftarrow \theta_{t-1} - \alpha \times m_t$

3.2. 自适应学习率算法

上述介绍的随机梯度下降及其动量变体算法都使用固定的常数作为学习率，参数以相同的步长更新，不考虑梯度行为，因此如何设定学习率的大小在实际应用中十分重要。然而，学习率是一个难以设置的超参数，它对模型性能有显著的影响。近年来，深度网络优化的有效方法是为每个参数制定自适应步长，即在整个迭代过程中自动地调整学习率，该类算法被称为自适应学习率优化算法。

3.2.1. 经典的自适应学习率算法

2011 年，Duchi 等人[28]受到 Delta-bar-delta 算法训练早期时适应参数的学习率[29]的启发，提出了第一个经典自适应学习率优化算法——AdaGrad。AdaGrad 算法采用参数的过去累计梯度平方和的平方根来划分学习率，可以很好地处理稀疏数据。它对不频繁的参数执行较大的更新，使得步长较大，多学习一些知识；而对频繁的参数执行较小的更新，使得步长较小，学习过程更稳定，不至于被单个样本影响太多。AdaGrad 减少了传统 SGD 及其动量变体算法中需要手动调节学习率的需要，实现了学习率的动态调节，提高了超参数的鲁棒性。AdaGrad 算法的伪代码如算法 4 所示。

算法 4：AdaGrad 优化器

输入：学习率 α，初始参数 θ_0，衰减率超参数 β，数值稳定常数 δ

初始化：时间步 $t=0$，累计梯度平方 $G_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t \leftarrow t+1$

2) 计算梯度：$g_t \leftarrow \frac{1}{m}\nabla_{\theta} \sum_i L\left(f\left(x^{(i)};\theta_{t-1}\right), y^{(i)}\right)$

3) 计算累计梯度平方：$G_t \leftarrow G_{t-1} + g_t^2$

4) 参数更新：$\theta_t \leftarrow \theta_{t-1} - \frac{\alpha \times g_t}{\sqrt{G_t} + \delta}$

但是，AdaGrad 的缺点是会不断累积梯度的平方，在一段时间后分母过大，会大幅降低学习率甚至使其逐渐递减至零，反而扼杀了学习过程，导致训练提前结束。为了改进 AdaGrad 算法的上述缺点，2012

年 Adadelta 算法[30]和 RMSProp 算法[31]被提出。RMSProp 使用指数移动平均法计算累计梯度平方，替代了 AdaGrad 中直接计算梯度平方和的方法。利用指数移动平均法的好处在于，计算二阶动量时不累积全部历史梯度，而只关注最近某一段时间窗内的梯度信息，衰减率超参数 β 用来控制时间窗长短。RMSProp 算法的伪代码如算法 5 所示。

算法 5：RMSProp 优化器

输入：学习率 α，初始参数 θ_0，数值稳定常数 δ

初始化：时间步 $t=0$，累计梯度平方 $G_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t \leftarrow t+1$

2) 计算梯度：$g_t \leftarrow \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)};\theta_{t-1}\right),y^{(i)}\right)$

3) 计算累计梯度平方：$G_t \leftarrow \beta G_{t-1}+\left(1-\beta\right)g_t^2$

4) 参数更新：$\theta_t \leftarrow \theta_{t-1}-\frac{\alpha \times g_t}{\sqrt{G_t}+\delta}$

2014 年，Adam 算法[32]被提出，它是现今最为广泛使用的一种自适应学习率优化算法，在深度学习相关的各种任务实践上都很受欢迎。Adam 算法可以看作是 SGDM 算法和 RMSprop 算法的结合，基于被称为一阶矩和二阶矩的两个向量来自适应地调节学习率。一阶矩 m 和二阶矩 v 分别使用梯度和梯度的平方的指数移动平均值来定义，在迭代的初始阶段，观察到一阶矩 m 和二阶矩 v 有一个向初值的偏移(过多的偏向了零)。为解决，Adam 校正了一阶矩和二阶矩的偏差，然后通过校正的两者计算每个参数的自适应学习率。如果我们把矩想象成是一个球滚下斜坡，Adam 表现得像一个有摩擦的重球，因此可以更快速更准确地朝着目标曲面上的平坦极小值前进。算法 6 展示了 Adam 算法的伪代码流程。

算法 6：Adam 优化器

输入：学习率 α，初始参数 θ_0，矩衰减率超参数 $\{\beta_1,\beta_2\}$，数值稳定常数 δ

初始化：时间步 $t=0$，一阶矩变量 $m_0=0$，二阶矩变量 $v_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t \leftarrow t+1$

2) 计算梯度：$g_t \leftarrow \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)};\theta_{t-1}\right),y^{(i)}\right)$

3) 计算有偏一阶矩估计：$m_t \leftarrow \beta_1 m_{t-1}+\left(1-\beta_1\right)g_t$

4) 计算有偏二阶矩估计：$v_t \leftarrow \beta_2 v_{t-1}+\left(1-\beta_2\right)g_t^2$

5) 修正一阶矩和二阶矩的偏差：$\hat{m}_t \leftarrow m_t/\left(1-\beta_1^t\right),\ \hat{v}_t \leftarrow v_t/\left(1-\beta_2^t\right)$

6) 参数更新：$\theta_t \leftarrow \theta_{t-1}-\frac{\alpha \times \hat{m}_t}{\sqrt{\hat{v}_t}+\delta}$

对于标准的随机梯度下降法，L2 正则化和权重衰减正则化是等价的，但是对于如 Adam 的自适应学习率优化算法，Loshchilov 等[33]在 2017 年证明，L2 正则化和权重衰减并不相同，这也是 Adam 的泛化性能较差的一个主要原因。由于 L2 正则化对 Adam 的效果远不如对 SGD 有效，而权重衰减正则化对 Adam 和 SGD 同样有效，因此作者提出了 AdamW，将权重衰减从损失函数的优化步骤中解耦出来，恢复权重衰减正则化的原始公式。AdamW 算法使得正则化权重衰减率与学习率两个超参数的影响解耦，提升了 Adam 算法的泛化性能。

3.2.2. 对梯度信息的利用改进算法

2018 年后，研究人员陆续基于 Adam 算法进行了改进，他们考虑已有的长期或短期样本梯度信息对学习过程的调节作用，对自适应学习率进行更改，来动态加大或减小更新步长。

2018 年，Zaheer 等[34]考虑到 Adam 使用了本质上是“乘性”的指数移动平均法，导致过去的梯度以相当快的方式被“遗忘”，在稀疏情况下即梯度大多为零时会出现问题，提出了一种“加性”改进算法 Yogi。Yogi 将二阶矩 v 的计算从 Adam 中的指数移动平均乘性算法变为加性算法，两者相同的一个重要特性是 v_t 和 v_{t-1} 的差异仅取决于 v_{t-1} 和 g_t^2。然而，这种差异的大小在 Yogi 中仅取决于 g_t^2，而 Adam 中则取决于 v_{t-1} 和 g_t^2。相比于 Adam，Yogi 减轻了指数移动平均法对过去梯度信息的比重过小，同时还可以控制实际的自适应学习率的增长情况，来获得更好的性能。Yogi 的伪代码如算法 7 所示。

算法 7：Yogi 优化器

输入：学习率 α，初始参数 θ_0，矩衰减率超参数 $\{\beta_1,\beta_2\}$，数值稳定常数 δ

初始化：时间步 $t=0$，一阶矩变量 $m_0=0$，二阶矩变量 $v_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t\leftarrow t+1$

2) 计算梯度：$g_t\leftarrow\frac{1}{m}\nabla_\theta\sum_i L\left(f\left(x^{(i)};\theta_{t-1}\right),y^{(i)}\right)$

3) 计算有偏一阶矩估计：$m_t\leftarrow\beta_1 m_{t-1}+\left(1-\beta_1\right)g_t$

4) 计算有偏二阶矩估计：$v_t\leftarrow v_{t-1}+\left(1-\beta_2\right)sign\left(v_{t-1}-g_t^2\right)g_t^2$

5) 修正一阶矩和二阶矩的偏差：$\hat{m}_t\leftarrow m_t/\left(1-\beta_1^t\right)$, $\hat{v}_t\leftarrow v_t/\left(1-\beta_2^t\right)$

6) 参数更新：$\theta_t\leftarrow\theta_{t-1}-\frac{\alpha\times\hat{m}_t}{\sqrt{\hat{v}_t}+\delta}$

2018 年，Reddi 等[35]发现当 Adam 中梯度的平方大幅下降时，二阶矩变得较低，自适应学习率会变高，可能会引起参数更新超过最优解并发散的问题，于是提出了 AMSGrad 算法作为对 Adam 的改进。AMSGrad 考虑用过去二阶矩的最大值，即过去梯度的长期记忆值，来代替 Adam 算法中的二阶矩，目的是给学习过程施加更多的摩擦，来避免超过目标最小值。AMSGrad 的伪代码如算法 8 所示。

2020 年，耶鲁大学团队提出了 AdaBelief 算法[36]，同样针对 Adam 算法的二阶矩进行了更改。AdaBelief 将一阶矩看作当前梯度的预测值，根据梯度方向上的“信念”(Belief)来自适应地调节实际的学习率进行参数更新。当一阶矩和当前梯度的差值较小时，“相信”当前梯度是正确的，参数更新步长大；当一阶矩和当前梯度的差值较大时，“不相信”当前梯度，参数更新步长小。团队在论文中表示，该优

化器兼具 Adam 的快速收敛特性和 SGD 的良好泛化性。AdaBelief 算法的伪代码如算法 9 所示。

算法 8：AMSGrad 优化器

输入：学习率 α，初始参数 θ_0，矩衰减率超参数 $\{\beta_1,\beta_2\}$，数值稳定常数 δ

初始化：时间步 $t=0$，一阶矩变量 $m_0=0$，二阶矩变量 $v_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t\leftarrow t+1$

2) 计算梯度：$g_t\leftarrow\frac{1}{m}\nabla_\theta\sum_i L\left(f\left(x^{(i)};\theta_{t-1}\right),y^{(i)}\right)$

3) 计算有偏一阶矩估计：$m_t\leftarrow\beta_1 m_{t-1}+\left(1-\beta_1\right)g_t$

4) 计算有偏二阶矩估计：$v_t\leftarrow\beta_2 v_{t-1}+\left(1-\beta_2\right)g_t^2$

5) 修正一阶矩和二阶矩的偏差：$\hat{m}_t\leftarrow m_t/\left(1-\beta_1^t\right),\hat{v}_t\leftarrow v_t/\left(1-\beta_2^t\right)$

6) 计算修正二阶矩的最大值：$\hat{v}_t^{\max}\leftarrow\max\left(\hat{v}_{t-1}^{\max},\hat{v}_t\right)$

7) 参数更新：$\theta_t\leftarrow\theta_{t-1}-\frac{\alpha\times\hat{m}_t}{\sqrt{\hat{v}_t^{\max}}+\delta}$

算法 9：AdaBelief 优化器

输入：学习率 α，初始参数 θ_0，矩衰减率超参数 $\{\beta_1,\beta_2\}$，数值稳定常数 δ

初始化：时间步 $t=0$，一阶矩变量 $m_0=0$，二阶矩变量 $v_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t\leftarrow t+1$

2) 计算梯度：$g_t\leftarrow\frac{1}{m}\nabla_\theta\sum_i L\left(f\left(x^{(i)};\theta_{t-1}\right),y^{(i)}\right)$

3) 计算有偏一阶矩估计：$m_t\leftarrow\beta_1 m_{t-1}+\left(1-\beta_1\right)g_t$

4) 计算有偏二阶矩估计：$v_t\leftarrow\beta_2 v_{t-1}+\left(1-\beta_2\right)\left(g_t-m_t\right)^2+\delta$

5) 修正一阶矩和二阶矩的偏差：$\hat{m}_t\leftarrow m_t/\left(1-\beta_1^t\right),\hat{v}_t\leftarrow v_t/\left(1-\beta_2^t\right)$

6) 参数更新：$\theta_t\leftarrow\theta_{t-1}-\frac{\alpha\times\hat{m}_t}{\sqrt{\hat{v}_t}+\delta}$

Dubey 等在 2020 年提出了 Adam 算法的又一改进——diffGrad 算法[37]，不同于 AMSGrad 算法中考虑“长期梯度行为”，而是展现了如何利用“短期梯度行为”来控制优化环境中的自适应学习率，使参数更新接近或远离最优解。diffGrad 算法基于近期梯度的变化信息，引入了差分梯度摩擦系数(diffGrad Friction Coefficient, DFC)来动态调整自适应学习率，使参数按照需要的方向来更新。第 t 次迭代时的差分梯度摩擦系数 DFC 定义为：$\xi_t=AbsSig\left(\Delta g_t\right)=AbsSig\left(g_{t-1}-g_t\right)$。其中，$\Delta g_t$ 是上一次迭代和当前迭代之间的梯度变化；$AbsSig\left(x\right)=1/\left(1+\mathrm{e}^{-|x|}\right)$ 是一个非线性 Sigmoid 函数，它把每个值压缩在 0.5 到 1 之间。

如果梯度变化很大时，DFC 趋近于 1，摩擦力几乎没有，此时 diffGrad 与 Adam 相同；如果梯度变化很小时，DFC 趋近于 0.5，会产生摩擦力。这意味着，DFC 不仅允许高梯度变化表面的高学习率，而且降低了低梯度变化表面的学习率，并防止超过可能的目标值。算法 10 展示了 diffGrad 算法的伪代码。

2021 年，Khan 等提出的 AdaDB 算法[38]是针对 diffGrad 算法的改进，两者的差别在于计算差分梯度摩擦系数 DFC 的方法。diffGrad 基于上一期梯度和当前梯度的变化来计算更新梯度差，而 AdaDB 则是考虑过去三期梯度与当前梯度来计算更新梯度差。为了能进一步地加快学习过程，AdaDB 根据连续两次迭代的梯度异号或是同号，分别定义了 DFC，使其可以动态增大或减小自适应学习率，而不像 diffGrad 中只能减小而不能增大自适应学习率。算法 11 展示了 AdaDB 算法的伪代码。

算法 10：diffGrad 优化器

输入：学习率 α，初始参数 θ_0，矩衰减率超参数 $\{\beta_1,\beta_2\}$，数值稳定常数 δ

初始化：时间步 $t=0$，一阶矩变量 $m_0=0$，二阶矩变量 $v_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t\leftarrow t+1$

2) 计算梯度：$g_t\leftarrow\frac{1}{m}\nabla_\theta\sum_i L\left(f\left(x^{(i)};\theta_{t-1}\right),y^{(i)}\right)$

3) 计算差分摩擦系数：$\xi_t\leftarrow AbsSig\left(g_{t-1}-g_t\right)$

4) 计算有偏一阶矩估计：$m_t\leftarrow\beta_1 m_{t-1}+\left(1-\beta_1\right)g_t$

5) 计算有偏二阶矩估计：$v_t\leftarrow\beta_2 v_{t-1}+\left(1-\beta_2\right)g_t^2$

6) 修正一阶矩和二阶矩的偏差：$\hat{m}_t\leftarrow m_t/\left(1-\beta_1^t\right),\hat{v}_t\leftarrow v_t/\left(1-\beta_2^t\right)$

7) 参数更新：$\theta_t\leftarrow\theta_{t-1}-\frac{\alpha\times\xi_t\times\hat{m}_t}{\sqrt{\hat{v}_t}+\delta}$

算法 11：AdaDB 优化器

输入：学习率 α，初始参数 θ_0，矩衰减率超参数 $\{\beta_1,\beta_2\}$，数值稳定常数 δ，梯度差衰减率超参数 β_3，差分摩擦系数超参数 λ

初始化：时间步 $t=0$，一阶矩变量 $m_0=0$，二阶矩变量 $v_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t\leftarrow t+1$

2) 计算梯度：$g_t\leftarrow\frac{1}{m}\nabla_\theta\sum_i L\left(f\left(x^{(i)};\theta_{t-1}\right),y^{(i)}\right)$

3) 计算更新梯度差：$\Delta g_t\leftarrow\beta_3\left(g_{t-1}-g_t\right)+\beta_3^2\left(g_{t-2}-g_{t-1}\right)+\beta_3^3\left(g_{t-3}-g_{t-2}\right)$

4) 计算差分摩擦系数：

a) 连续两次迭代的梯度异号时：$\xi_t\leftarrow AbsSig\left(\Delta g_t\right)$

b) 连续两次迭代的梯度同号时：$\xi_t\leftarrow 1+\lambda\times AbsSig\left(\Delta g_t\right)$

Continued

5) 计算有偏一阶矩估计：$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$

6) 计算有偏二阶矩估计：$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$

7) 修正一阶矩和二阶矩的偏差：$\hat{m}_t \leftarrow m_t / (1-\beta_1^t), \hat{v}_t \leftarrow v_t / (1-\beta_2^t)$

8) 参数更新：$\theta_t \leftarrow \theta_{t-1} - \dfrac{\alpha \times \xi_t \times \hat{m}_t}{\sqrt{\hat{v}_t} + \delta}$

3.2.3. 对极端学习率的探索改进算法

与传统的随机梯度下降算法相比，Adam 这类自适应学习率优化算法存在泛化能力较差、甚至可能由于不稳定和极端的自适应学习率而无法收敛的问题[39] [40] [41] [42]。针对这些问题，国内外的研究学者们展开了探索，尝试设计一种兼具好的泛化性能与快的收敛速度的优化算法。

2019 年，AdaBound 算法被 Luo 等人[43]提出，旨在结合 Adam 算法和经典 SGD 算法的优势，即分别是训练早期快速的收敛速度和训练末期更好的最终性能。AdaBound 的创新在于设计了一个动态学习率裁剪函数：$Clip\left(\alpha/\sqrt{v_t}, \eta_l(t), \eta_u(t)\right) = \max\left(\min\left(\alpha/\sqrt{v_t}, \eta_u(t)\right), \eta_l(t)\right)$，其中 $\eta_l(t)$ 和 $\eta_u(t)$ 分别是非减的下界函数和非增的上界函数。在这一动态的学习率边界操作下，自适应学习率被限制在一定范围之间。训练早期时，上界和下界对学习率的影响很小，算法更加接近于 Adam；而随着训练过程的推进，裁剪区间越来越收紧，学习率逐渐趋于稳定，算法更加接近于 SGD。于是，AdaBound 算法就实现了从 Adam 到 SGD 的渐进动态平稳过渡。

在 AdaBound 算法基础上，Ding 等[44]又进一步提出了 AdaMod 算法。为了避免训练过程中可能出现的极端自适应学习率值，AdaMod 通过计算自适应学习率的指数移动平均值，来动态修剪过高的学习率，使训练过程变得更加平稳。AdaBound 算法和 AdaMod 算法的伪代码如算法 12 和算法 13 所示。

算法 12：AdaBound 优化器

输入：学习率 α，初始参数 θ_0，矩衰减率超参数 $\{\beta_1, \beta_2\}$，数值稳定常数 δ，下界函数 η_l，上界函数 η_u

初始化：时间步 $t=0$，一阶矩变量 $m_0=0$，二阶矩变量 $v_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t \leftarrow t+1$

2) 计算梯度：$g_t \leftarrow \dfrac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)}; \theta_{t-1}\right), y^{(i)}\right)$

3) 计算有偏一阶矩估计：$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$

4) 计算有偏二阶矩估计：$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$

5) 裁剪自适应学习率：$\hat{\eta}_t \leftarrow Clip\left(\alpha/\sqrt{v_t}, \eta_l(t), \eta_u(t)\right)$

6) 计算自适应学习率：$\eta_t \leftarrow \hat{\eta}_t / \sqrt{t}$

7) 参数更新：$\theta_t \leftarrow \theta_{t-1} - \eta_t \times m_t$

算法 13：AdaMod 优化器

输入：学习率 α，初始参数 θ_0，矩衰减率超参数 $\{\beta_1,\beta_2,\beta_3\}$，数值稳定常数 δ，下界函数 η_l，上界函数 η_u

初始化：时间步 $t=0$，一阶矩变量 $m_0=0$，二阶矩变量 $v_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t \leftarrow t+1$

2) 计算梯度：$g_t \leftarrow \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)};\theta_{t-1}\right),y^{(i)}\right)$

3) 计算有偏一阶矩估计：$m_t \leftarrow \beta_1 m_{t-1}+\left(1-\beta_1\right)g_t$

4) 计算有偏二阶矩估计：$v_t \leftarrow \beta_2 v_{t-1}+\left(1-\beta_2\right)g_t^2$

5) 修正一阶矩和二阶矩的偏差：$\hat{m}_t \leftarrow m_t/\left(1-\beta_1^t\right), \hat{v}_t \leftarrow v_t/\left(1-\beta_2^t\right)$

6) 计算自适应学习率：$\eta_t \leftarrow \alpha/\left(\sqrt{\hat{v}_t}+\delta\right)$

7) 计算自适应学习率的指数移动平均值：$s_t \leftarrow \beta_3 s_{t-1}+\left(1-\beta_3\right)\eta_t$

8) 修剪自适应学习率：$\hat{\eta}_t \leftarrow \min\left(\eta_t,s_t\right)$

9) 参数更新：$\theta_t \leftarrow \theta_{t-1}-\hat{\eta}_t \times \hat{m}_t$

2020 年，来自伊利诺伊州大学的团队[45]更深入地研究了 Adam 优化算法在训练早期会由于较大的学习率引起模型振荡问题的原因与解决方案。在此之前，基于工程直觉，实践中往往采用学习率预热方法(Warmup) [46]解决 Adam 的振荡问题——在训练最初几个时期使用较小的学习率，等模型相对稳定后再选择预先设置的学习率进行训练。预热一般可以使模型收敛更快、效果更佳，防止早期的梯度分布被扭曲。但是，由于缺乏理论基础，不能保证预热会为各种机器学习的任务带来一致的改进，也不能指导面对不同数据集中应该如何进行预热。伊利诺伊州大学团队在论文中给出了预热的理论依据，表明 Adam 等的收敛问题是由于使用的训练样本数量有限，在模型训练的早期阶段自适应学习率有不期望的大方差。团队将预热看作是一种方差减少技术，从统计学的角度对学习率方差进行了研究，并提出了一种新的 Adam 变体——RAdam，意为“整流版的 Adam”(Rectified Adam)。RAdam 使用简单移动平均法近似 Adam 二阶矩中的指数移动平均法，将自适应学习率的平方看作服从缩放逆卡方分布，从而估算出自适应学习率的方差，最后提出修正因子 r_t 来校正各时刻的方差使其最小。总而言之，RAdam 避免了传统需要手动选择数量超参数的预热方法，采用一个“整流器”函数，根据实际遇到的方差散度，动态地打开、关闭或者修正自适应学习率，起到如预热一样防止早期梯度分布扭曲的作用，当变差稳定下来之后，它在剩下的训练过程中基本就等效于 Adam 或 SGDM。RAdam 算法如下算法 14 所示。

多伦多大学的研究团队对于自适应学习率的较大方差问题，提出了另一种有效的优化算法 Lookahead [47]，该算法采用了一种全新的设计，与之前已有的方法都不相同。非常需要注意的是，Lookahead 严格来说不算一种优化器，更像是一种“包装”，需要搭配其他任意一种优化器(如 SGD、Adam)一起工作，从而加强已有优化算法的性能，并且有效地降低自适应学习率的方差。具体来说，如图 2 所示，Lookahead

维持快速、慢速两组权重，首先使用其内循环中的任何标准优化器更新“快速权重”k 次，然后沿最终快速权重的方向更新一次“慢速权重”，重复进行上述步骤直至完成。直观地说，该算法允许更快的那一组权重“向前看”或“探索”来选择搜索方向，同时更慢的那一组权重可以留在后面拽回，以带来更好的长期稳定性。Lookahead 实践中常常结合 SGDM、Adam 和 RAdam 等使用，通常可以提高内部优化器的性能，稳定深度神经网络的训练过程。算法 15 展示了 Lookahead 算法的伪代码。

算法 14：RAdam 优化器

输入：学习率 α，初始参数 θ_0，矩衰减率超参数 $\{\beta_1,\beta_2\}$，数值稳定常数 δ

初始化：时间步 $t=0$，一阶矩变量 $m_0=0$，二阶矩变量 $v_0=0$

当没有达到停止准则时，执行

1) 时间步更新：$t\leftarrow t+1$

2) 计算梯度：$g_t\leftarrow\frac{1}{m}\nabla_\theta\sum_i L\left(f\left(x^{(i)};\theta_{t-1}\right),y^{(i)}\right)$

3) 计算有偏一阶矩估计：$m_t\leftarrow\beta_1 m_{t-1}+\left(1-\beta_1\right)g_t$

4) 计算有偏二阶矩估计：$v_t\leftarrow\beta_2 v_{t-1}+\left(1-\beta_2\right)g_t^2$

5) 修正一阶矩和二阶矩的偏差：$\hat{m}_t\leftarrow m_t/\left(1-\beta_1^t\right),\hat{v}_t\leftarrow v_t/\left(1-\beta_2^t\right)$

6) 计算近似简单移动平均的长度：$\rho_t\leftarrow\rho_\infty-2t\beta_2^t/\left(1-\beta_2^t\right)$

7) a) 当 $\rho_t>4$ 时：计算方差修正项：$r_t\leftarrow\sqrt{\frac{(\rho_t-4)(\rho_t-2)\rho_\infty}{(\rho_\infty-4)(\rho_\infty-2)\rho_t}}$

参数更新：$\theta_t\leftarrow\theta_{t-1}-\frac{\alpha\times r_t\times\hat{m}_t}{\sqrt{\hat{v}_t}+\delta}$

b) 当 $\rho_t\le 4$ 时：参数更新：$\theta_t\leftarrow\theta_{t-1}-\alpha\times\hat{m}_t$

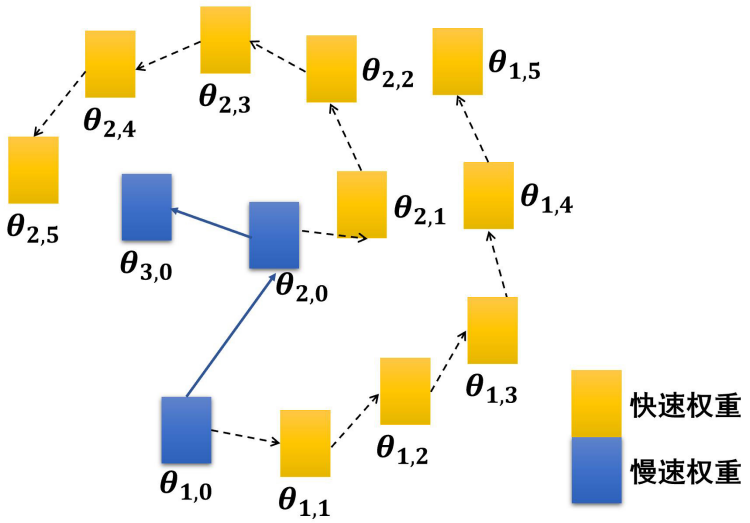

Figure 2. The parameter update process of Lookahead when synchronization period k = 5
图 2. 同步周期 k = 5 时，Lookahead 的参数更新流程

算法 15：Lookahead 优化算法

输入：学习率 α，初始参数 ϕ_0，同步周期 k，内部标准优化器 A

当 $t=1,2,\cdots$ 时，执行

1) 同步参数：$\theta_{t,0} \leftarrow \phi_{t-1}$

当 $i=1,2,\cdots,k$ 时，执行

2) 内部应用参数更新：$\theta_{t,i} \leftarrow \theta_{t,i-1} + A\left(L,\theta_{t,i-1}\right)$

3) 外部应用参数更新：$\phi_t \leftarrow \phi_{t-1} + \alpha \times \left(\theta_{t,k} - \phi_{t-1}\right)$

4. 研究展望

近年来，深度学习中的优化算法层出不穷，但是在深度学习的实际任务中，选择和设计合适的优化器仍面临着许多困难。探索影响优化算法寻找目标点的损失曲面的性质，提升优化算法的收敛速度与泛化性能是具有挑战和价值的研究方向。

1) 损失曲面的性质

深度学习神经网络的损失函数对优化算法的参数更新方向和步长有着关键影响，损失平面的几何特征与性质指导着如何设计优化算法。对于深度学习任务中非凸的目标函数，以及小批量会引起的随机噪声扰动，优化算法如何体现深度神经网络参数的分布信息、正确找到目标点是值得深入探索的问题。

2) 收敛速度与泛化性能

尽管诸如 Adam 等自适应学习率优化算法在大多深度学习实践任务中表现出优秀的性能，但 SGD 还是经常在任务训练后期时被使用。自适应学习率优化算法虽然在训练早期有着快速的收敛速度，但在测试时的泛化能力往往不如 SGD 优秀。目前，实验证实自适应学习率优化器更倾向于收敛到泛化性能效果差的尖锐极小值，而不是和 SGD 一样，收敛到所期望的平坦极小值。因此，如何提高自适应学习率优化算法的泛化效果、同时保证快速收敛是一个具有价值的研究方向。

基金项目

国家自然科学基金(No. 62072024)；北京建筑大学北京未来城市设计高精尖创新中心资助项目(UDC2017033322, UDC2019033324)；北京建筑大学市属高校基本科研业务费专项资金资助(No. X20084, ZF17061)；北京建筑大学研究生创新项目(PG2022144)；北京高等教育本科教学改革创新项目(201910016004)。

参考文献

[1] Hinton, G.E., Osindero, S. and Teh, Y.W. (2006) A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, **18**, 1527-1554. https://doi.org/10.1162/neco.2006.18.7.1527

[2] Kiran, B.R., Sobh, I., Talpaert, V., *et al*. (2021) Deep Reinforcement Learning for Autonomous Driving: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, **6**, 4909-4926. https://doi.org/10.1109/TITS.2021.3054625

[3] Zou, Z., Shi, Z., Guo, Y., *et al*. (2019) Object Detection in 20 Years: A Survey. ArXiv: 1905.05055.

[4] Su, J., Xu, B. and Yin, H. (2022) A Survey of Deep Learning Approaches to Image Restoration. *Neurocomputing*, **487**, 46-65. https://doi.org/10.1016/j.neucom.2022.02.046

[5] 殷琪林, 王金伟. 深度学习在图像处理领域中的应用综述[J]. 高教学刊, 2018(9): 72-74.

[6] Ruder, S. (2016) An Overview of Gradient Descent Optimization Algorithms. ArXiv: 1609.04747.

[7] 袁群勇. 深度神经网络的训练优化方法研究[D]: [博士学位论文]. 广州: 华南理工大学, 2020.

[8] 张慧. 深度学习中优化算法的研究与改进[D]: [硕士学位论文]. 北京: 北京邮电大学, 2018.

[9] Cauchy, M.A. (1847) Méthodegénérale pour la résolution des systems d'équations simultanées. *Comptes Rendus de l'Académie des Sciences*, **25**, 536-538.

[10] Levenberg, K. (1944) A Method for the Solution of Certain Non-Linear Problems in Least Squares. *Quarterly of Applied Mathematics*, **2**, 164-168. https://doi.org/10.1090/qam/10666

[11] Marquardt, D.W. (1963) An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics*, **11**, 431-441. https://doi.org/10.1137/0111030

[12] Møller, M.F. (1993) Efficient Training of Feed-Forward Neural Networks. Aarhus University, Aarhus. https://doi.org/10.7146/dpb.v22i464.6937

[13] Le Roux, N., Bengio, Y. and Fitzgibbon, A. (2011) Improving First and Second-Order Methods by Modeling Uncertainty. In: Sra, S., Nowozin, S. and Wright, S.J., Eds., *Optimization for Machine Learning*, The MIT Press, Cambridge, 403-429.

[14] 王帅, 向建军, 彭芳, 唐书娟. 基于新最速下降法的目标跟踪算法[J]. 系统工程与电子技术, 2022, 44(5): 1512-1519.

[15] 刘晓, 吴明儿, 张华振. 基于最速下降法的可展开索网天线型面调整方法[J]. 中国空间科学技术, 2018, 38(3): 1-7. https://doi.org/10.16708/j.cnki.1000-758X.2018.0022

[16] 于忠霞. 基于最速下降法的常州物流业优化问题研究[J]. 常州工学院学报, 2015, 28(3): 45-48.

[17] 米阳, 彭建伟, 陈博洋, 王晓敏, 刘子旭, 王育飞. 基于一致性原理和梯度下降法的微电网完全分布式优化调度[J]. 电力系统保护与控制, 2022, 50(15): 1-10. https://doi.org/10.19783/j.cnki.pspc.211371

[18] Robbins, H. and Monro, S. (1985) A Stochastic Approximation Method. Springer, New York, 1985. https://doi.org/10.1007/978-1-4612-5110-1_9

[19] Bottou, L. (1998) Online Learning and Stochastic Approximations. In: Saad, D., Ed., *On-Line Learning in Neural Networks*, Cambridge University Press, Cambridge, 9-42. https://doi.org/10.1017/CBO9780511569920.003

[20] Sutton, R. (1986) Two Problems with Back Propagation and Other Steepest Descent Learning Procedures for Networks. In: *Proceedings of the* 8*th Annual Conference of the Cognitive Science Society*, Erlbaum, Hillsdale, 823-832.

[21] Dauphin, Y., Pascanu, R., Gulcehre, C., *et al*. (2014) Identifying and Attacking the Saddle Point Problem in High-Dimensional Non-Convex Optimization. ArXiv: 1406.2572.

[22] LeCun, Y., Boser, B., Denker, J.S., *et al*. (1989) Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, **1**, 541-551. https://doi.org/10.1162/neco.1989.1.4.541

[23] Ning, Q. and Qian, N. (1999) On the Momentum Term in Gradient Descent Learning Algorithms. *Neural Networks*, **12**, 145-151. https://doi.org/10.1016/S0893-6080(98)00116-6

[24] LeCun, Y.A., Bottou, L., Orr, G.B. and Müller, K.R. (2012) Efficient BackProp. In: Montavon, G., Orr, G.B. and Müller, KR., Eds., *Neural Networks*: *Tricks of the Trade*, Springer, Berlin, 9-48. https://doi.org/10.1007/978-3-642-35289-8_3

[25] Sutskever, I., Martens, J., Dahl, G. and Hinton, G. (2013) On the Importance of Initialization and Momentum in Deep Learning. *Proceedings of the International Conference on Machine Learning Research*, **28**, 1139-1147.

[26] Nesterov, Y. (1983) A Method of Solving a Convex Programming Problem with Convergence Rate Mathcal O(1/k^2). *Proceedings of the USSR Academy of Sciences*, **269**, 543-547.

[27] Nesterov, Y. (2003) Introductory Lectures on Convex Optimization: A Basic Course. Springer Science & Business Media, Berlin. https://doi.org/10.1007/978-1-4419-8853-9

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

[29] Jacobs, R.A. (1988) Increased Rates of Convergence through Learning Rate Adaptation. *Neural Networks*, **1**, 295-307. https://doi.org/10.1016/0893-6080(88)90003-2

[30] Zeiler, M.D. (2012) Adadelta: An Adaptive Learning Rate Method. ArXiv: 1212.5701.

[31] Tieleman, T. and Hinton, G. (2012) RmsProp: Divide the Gradient by a Running Average of Its Recent Magnitude. *COURSERA*: *Neural Networks for Machine Learning*, **4**, 26-31.

[32] Kingma, D.P. and Ba, J. (2014) Adam: A Method for Stochastic Optimization. ArXiv: 1412.6980.

[33] Loshchilov, I. and Hutter, F. (2017) Decoupled Weight Decay Regularization. ArXiv: 1711.05101.

[34] Zaheer, M., Reddi, S., Sachan, D., Kale, S. and Kumar, S. (2018) Adaptive Methods for Nonconvex Optimization.

Advances in Neural Information Processing Systems, **31**, 9793-9803.

[35] Reddi, S.J., Kale, S. and Kumar, S. (2019) On the Convergence of Adam and Beyond. ArXiv: 1904.09237.

[36] Zhuang, J., Tang, T., Ding, Y., *et al*. (2020) AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients. *Advances in Neural Information Processing Systems*, **33**, 18795-18806.

[37] Dubey, S.R., Chakraborty, S., Roy, S.K., *et al*. (2019) DiffGrad: An Optimization Method for Convolutional Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, **31**, 4500-4511. https://doi.org/10.1109/TNNLS.2019.2955777

[38] Khan, M.U.S., Jawad, M. and Khan, S.U. (2021) Adadb: Adaptive Diff-Batch Optimization Technique for Gradient Descent. *IEEE Access*, **9**, 99581-99588. https://doi.org/10.1109/ACCESS.2021.3096976

[39] Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M. and Tang, P.T.P. (2016) On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. ArXiv: 1609.04836.

[40] Keskar, N.S. and Socher, R. (2017) Improving Generalization Performance by Switching from Adam to SGD. ArXiv: 1712.07628.

[41] Xing, C., Arpit, D., Tsirigotis, C. and Bengio, Y. (2018) A Walk with SGD. ArXiv: 1802.08770.

[42] Tong, Q., Liang, G. and Bi, J. (2022) Calibrating the Adaptive Learning Rate to Improve Convergence of ADAM. *Neurocomputing*, **481**, 333-356. https://doi.org/10.1016/j.neucom.2022.01.014

[43] Luo, L., Xiong, Y., Liu, Y. and Sun, X. (2019) Adaptive Gradient Methods with Dynamic Bound of Learning Rate. ArXiv: 1902.09843.

[44] Ding, J., Ren, X., Luo, R. and Sun, X. (2019) An Adaptive and Momental Bound Method for Stochastic Learning. ArXiv: 1910.12249.

[45] Liu, L., Jiang, H., He, P., *et al*. (2019) On the Variance of the Adaptive Learning Rate and Beyond. ArXiv: 1908.03265.

[46] Gotmare, A., Keskar, N.S., Xiong, C. and Socher, R. (2018) A Closer Look at Deep Learning Heuristics: Learning Rate restarts, Warmup and Distillation. ArXiv: 1810.13243.

[47] Zhang, M., Lucas, J., Ba, J. and Hinton, G.E. (2019) Lookahead Optimizer: K Steps Forward, 1 Step Back. *Advances in Neural Information Processing Systems*, **32**, 9597-9608.