



An Adaptive Learning Rate Deep Learning Optimizer Using Long and Short-Term Gradients Based on G–L Fractional-Order Derivative

Shuang Chen¹ · Changlun Zhang¹ · Haibing Mu²

Accepted: 11 February 2024 / Published online: 15 March 2024
© The Author(s) 2024

Abstract

Deep learning model is a multi-layered network structure, and the network parameters that evaluate the final performance of the model must be trained by a deep learning optimizer. In comparison to the mainstream optimizers that utilize integer-order derivatives reflecting only local information, fractional-order derivatives optimizers, which can capture global information, are gradually gaining attention. However, relying solely on the long-term estimated gradients computed from fractional-order derivatives while disregarding the influence of recent gradients on the optimization process can sometimes lead to issues such as local optima and slower optimization speeds. In this paper, we design an adaptive learning rate optimizer called AdaGL based on the Grünwald–Letnikov (G–L) fractional-order derivative. It changes the direction and step size of parameter updating dynamically according to the long-term and short-term gradients information, addressing the problem of falling into local minima or saddle points. To be specific, by utilizing the global memory of fractional-order calculus, we replace the gradient of parameter update with G–L fractional-order approximated gradient, making better use of the long-term curvature information in the past. Furthermore, considering that the recent gradient information often impacts the optimization phase significantly, we propose a step size control coefficient to adjust the learning rate in real-time. To compare the performance of the proposed AdaGL with the current advanced optimizers, we conduct several different deep learning tasks, including image classification on CNNs, node classification and graph classification on GNNs, image generation on GANs, and language

S. Chen, C. Zhang and H. Mu have contributed equally to this work.

✉ Changlun Zhang
zclun@bucea.edu.cn

Shuang Chen
2102520021007@stu.bucea.edu.cn

Haibing Mu
hbm@bjtu.edu.cn

¹ Science School, Beijing University of Civil Engineering and Architecture, Beijing 100044, China

² Key Laboratory of Communication and Information System, Beijing Jiaotong University, Beijing 100044, China

modeling on LSTM. Extensive experimental results demonstrate that AdaGL achieves stable and fast convergence, excellent accuracy, and good generalization performance.

Keywords Deep learning · Optimization algorithm · Adaptive learning rate · Fractional-order derivative

1 Introduction

As a hot research direction for processing neural networks, Deep Learning [1] has received considerable attention in recent years. A deep learning model is a multi-layered neural network structure, which consists of an input layer, multiple hidden layers, and an output layer to simulate the multi-level learning process of the human brain. It has demonstrated significant advantages in computer vision, natural language processing, and other fields, and is widely applied in video stabilization, object recognition, image processing, and other areas [2–4]. Each layer of a deep neural network contains a large number of parameters, which determine the accuracy of the network's output and reflect the effectiveness of the model. In order to obtain excellent deep learning models, it is necessary to use deep learning optimizers to optimize and update the parameters during model training.

Because the deep neural networks have the characteristics of non-convexity, nonlinearity, deep hierarchical hidden structures, and a large number of parameters, the optimization problem in deep learning is quite complicated. An excellent optimizer can make the parameters converge to the target point with a low loss value and improve the speed and accuracy of the model to complete the task. At present, optimizers in deep learning can be mainly categorized into first-order methods and second-order methods [5, 6].

First-order optimization algorithms are based on Gradient Descent (GD), which was originally a fundamental method in optimization theory and later extended to the field of deep learning. Stochastic Gradient Descent (SGD) [7] is the most basic method in practical deep learning tasks. In each iteration, one or more samples (less than the total number of samples) are selected randomly, and the gradient of model parameters is calculated to update the parameters, aiming to minimize the loss function value of the model. As shown in Fig. 1, the basic SGD method has some main drawbacks [8]: (1) The gradients may be highly sensitive to certain directions in the parameter space; (2) It can get stuck in local minima or saddle points where the gradient is zero; (3) It applies the same update step size for each parameter without considering the gradient variation information, resulting in poor optimization.

Given several shortcomings of SGD, many researchers have put forward some improved optimization algorithms. Stochastic Gradient Descent with Momentum (SGDM) [9] is a

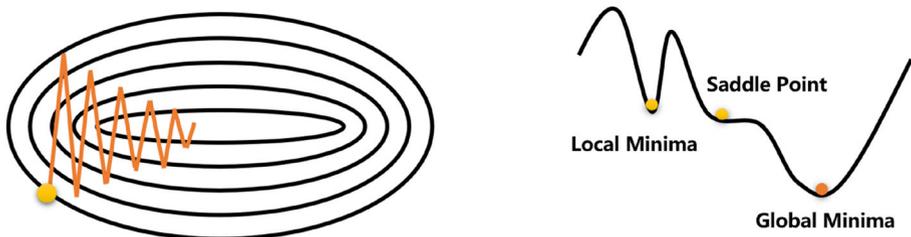


Fig. 1 SGD is sensitive to some directions (left) and can fall into local minima or saddle points (right)

popular variant that analogizes the concept of momentum in physics, considering the influence of historical gradient directions. If the gradient direction at the current moment is similar to that at the historical moment, SGDM accelerates the learning speed of parameters. It solves some problems in SGD [10]. However, SGDM, like SGD, still uses a fixed constant as the learning rate, so that all parameters change in the same step size, regardless of the gradient behavior. In recent years, adaptive learning rate optimizers have gained popularity in deep neural networks optimization, which can adjust the update step size dynamically according to the requirements of parameter update during the whole iteration process. Adam [11] is one of the most widely used adaptive learning rate optimizers, which can increase or decrease the update step size utilizing the gradient information. If the loss surface is regarded as a rugged hillside, Adam behaves like a frictional heavy ball and can find the target value faster. However, Adam tends to converge to the sharp local minima with poor generalization performance, rather than the expected flat minima, resulting in inferior performance compared to SGD-like optimizers in some tasks [12, 13].

Second-order optimization algorithms, such as Newton's method and the Conjugate Gradient method [14], require second-order derivatives to find the optimal parameter values. They utilize both gradient information and the trend of gradient changes, to some extent, to avoid local optima problems. However, the complexity of deep neural networks limits the practical application of second-order optimizers due to the high computational cost and memory requirements.

The mainstream optimizers mentioned above are all based on integer-order derivatives to obtain gradient information, which have limitations. Specifically, during the training process, they only use the gradient information of the current point to guide the optimization direction, which is calculated by the integer-order derivatives directly and lacks the global state and contextual information. This often leads to local optima issues.

Fractional-order derivative is a natural extension of integer-order derivative operation, where the order of derivative calculation is extended from integers to fractions. Fractional-order calculus has achieved excellent performance in various fields, such as data preprocessing, control systems, and time series prediction [15–17]. Recently, there has been a trend of applying fractional-order calculus methods to neural network optimization. Compared with the traditional integer-order derivative, the fractional-order derivative possesses the advantage of global memory characteristics in both time and space domains, and expands the search area around the target point [18]. So that additional information can be obtained to avoid the local optima problems caused by integer-order derivatives. Herrera-Alcántara [19] presented optimizers that introduce a fractional derivative gradient of the objective function in the parameter update rule, as well as an implementation for the Tensorflow backend. Yu et al. [20] designed a FracM optimizer based on fractional-order calculus and SGDM algorithm, using the fractional-order difference of momentum and gradient to adjust the optimization direction. Zhou et al. [21] proposed FCGD_G-L algorithm, which uses the Grünwald–Letnikov fractional-order derivative [22] to replace the first-order derivative in SGD and Adam, and adds a disturbance factor to improve the robustness of the algorithm. These are all successful algorithms using fractional-order calculus for deep learning optimization. However, these algorithms only use the historical long-term estimated gradients based on fractional-order calculations to adjust the parameter update step size, ignoring the great influence of the recent real gradients, which sometimes leads to poor optimization speed and accuracy. This is the focus of our research.

In this paper, we propose a novel adaptive learning rate optimizer called AdaGL, based on the classical Grünwald–Letnikov (G–L) fractional-order derivative. AdaGL can guide the parameter update direction and step size on the basis of both long-term and short-term

gradients information, enabling it to escape local minima and saddle points and converge to the flat target value quickly. Specifically, we start from the classical definition of the G–L fractional-order derivative in mathematics and replace the parameter update gradient with the fractional-order approximated gradient to incorporate the long memory and global correlation characteristics. Moreover, a step size control coefficient is designed to increase or decrease the update step size adaptively according to the real-time change of the short-term gradients. Theoretical analysis demonstrates the effectiveness of the proposed algorithm in addressing the problem of getting trapped in local minima and saddle points. We have also carried out extensive experiments on various deep learning tasks, including image classification, node classification, graph classification, image generation, and language modeling. The experimental results show that the proposed AdaGL optimizer can converge quickly, with excellent accuracy and good generalization performance.

In summary, the main contributions of our work are as follows:

- Based on the ability of G–L fractional derivatives to capture the historical global memory characteristics of the objective function, we derive the G–L fractional-order approximated gradient theoretically to replace the gradient of parameter updating in the neural networks, so that the curvature information of the objective function can be fully utilized.
- Considering the significant influence of recent gradient information during the optimization stage, we introduce a step size control coefficient, which can feedback and adjust the parameter update step size using the short-term gradients change. This allows us to jump out of the unexpected sharp minima and saddle points and accelerate the learning process.
- Combining G–L fractional-order approximated gradient and step size control coefficient, we propose an adaptive learning rate optimizer named AdaGL. It comprehensively utilizes both past long-term and current short-term gradients information to regulate the adaptive learning rate, preventing the optimizer from getting trapped in local minima and saddle points and ensuring rapid and stable convergence to flat optimal points.
- To evaluate the performance of the proposed AdaGL optimizer, we conduct experiments on a variety of deep learning classic architectures and datasets, comparing it with other popular optimizers. Our experiments include image classification with CNNs architectures (ResNet [23] and DenseNet [24]), node classification and graph classification with Graph Convolutional Networks (GCN) [25], image generation with Wasserstein Generative Adversarial Networks (WGAN) [26], and language modeling with Long Short-Term Memory (LSTM) [27]. AdaGL achieves advanced performance in these tasks, improving the convergence speed and accuracy of the network models.

The rest of this paper is organized as follows: Sect. 2 introduces the related work and preliminary mathematical preparation. Section 3 provides a detailed description of the proposed optimizer. Section 4 presents experimental validations of the proposed algorithm. Section 5 concludes the paper and discusses future work.

2 Preliminaries

In order to find the optimal parameters, the most fundamental method adopted in most deep learning neural networks is SGD. From all the n samples in the training set, select m ($m \leq n$) independent and identically distributed small batch samples $\{x^{(1)}, \dots, x^{(m)}\}$ randomly, where $x^{(i)}$ corresponds to the target $y^{(i)}$. In t th iteration, the defined objective optimization function L is applied to calculate the gradients of each parameter θ in the

network on the mini-batch samples, and then average their gradients to obtain the estimated gradient g_t :

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

Then, in the negative gradient direction, the parameter values are updated by using the learning rate hyperparameter and the gradient:

$$\theta_t \leftarrow \theta_{t-1} - \eta g_t \tag{2}$$

where θ_{t-1} and θ_t are the previous and updated parameter values respectively, and η is a fixed learning rate hyperparameter.

SGDM is a widely used extension of SGD that incorporates past gradient information in each dimension to maintain a momentum m , which is defined as the Exponential Moving Average (EMA) of the gradients. The parameters are updated as:

$$\begin{aligned} m_t &\leftarrow \beta m_{t-1} + g_t \\ \theta_t &\leftarrow \theta_{t-1} - \eta m_t \end{aligned} \tag{3}$$

where m_t is the momentum obtained in the t th iteration ($m_0 = 0$), and β is a hyperparameter for controlling the decay rate of the momentum.

Inspired by fractional calculus, FracM was put forward, which calculates the momentum and gradient in SGDM in fractional-order derivative instead of the traditional first-order derivative. The parameter update is defined as follows:

$$\begin{aligned} {}_{GL}D_t^{\alpha} m_t &= \beta {}_{GL}D_t^{\alpha} m_{t-1} + {}_{GL}D_t^{\alpha} g_t \\ &= \beta (m_{t-1} + \alpha_1 m_{t-3} + \alpha_2 m_{t-5} + \alpha_3 m_{t-7}) \\ &\quad + (g_t + \alpha_1 g_{t-2} + \alpha_2 g_{t-4} + \alpha_3 g_{t-6}) \\ m_{t-1} &\leftarrow {}_{GL}D_t^{\alpha} m_t \\ \theta_t &\leftarrow \theta_{t-1} - \eta m_{t-1} \end{aligned} \tag{4}$$

where ${}_{GL}D_t^{\alpha}$ represents performing a G–L fractional-order operation with a fractional order α in the t th iteration, and $\alpha_k (k = 1, 2, 3)$ is the default coefficient of the G–L fractional order.

The optimization algorithms described above use a fixed constant as the learning rate, causing all parameters to be updated with the same step size. In recent years, the adaptive learning rate algorithms have been widely used in deep learning tasks, which assign an adaptive step size to each parameter based on the current state.

Duchi et al. [28] proposed the first popular adaptive learning rate optimizer called AdaGrad. AdaGrad divides the learning rate by the square root of the accumulated sum of squared gradients for each parameter, enabling dynamic adjustment of the learning rate. The parameter update rule is as follows:

$$\begin{aligned} G_t &\leftarrow \beta G_{t-1} + g_t^2 \\ \theta_t &\leftarrow \theta_{t-1} - \frac{\eta g_t}{\sqrt{G_t} + \delta} \end{aligned} \tag{5}$$

where δ is a numerical stability constant added in the denominator to prevent division by zero.

However, AdaGrad accumulates the squared gradients continuously, leading to a sharp decline in the adaptive learning rate and hindering the learning process. To address this

drawback, RMSProp [29] was proposed as an improvement to AdaGrad. RMSProp uses the EMA to calculate the cumulative squared gradients, focusing only on the gradient information within a recent time window. The decay rate hyperparameter β is used to control the length of the time window. The parameter update rule is defined as:

$$\begin{aligned} G_t &\leftarrow \beta G_{t-1} + (1 - \beta) g_t^2 \\ \theta_t &\leftarrow \theta_{t-1} - \frac{\eta g_t}{\sqrt{G_t} + \delta} \end{aligned} \tag{6}$$

Adam can be regarded as the combination of SGDM and RMSprop, which adaptively adjusts the learning rate based on two vectors known as the first-order moment and the second-order moment. The first and second moments are defined by the EMA of the gradient and the squared gradient, respectively. However, these moments can be biased towards zero, especially during the initial iterations. To address this, a bias correction is applied. For the t th iteration, the parameter update in the Adam is defined as follows:

$$\begin{aligned} m_t &\leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &\leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\ \hat{m}_t &\leftarrow \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &\leftarrow \frac{v_t}{1 - \beta_2^t} \\ \theta_t &\leftarrow \theta_{t-1} - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \delta} \end{aligned} \tag{7}$$

where β_1 and β_2 are the decay rate hyperparameters for the first-order moment m_t and the second-order moment v_t respectively, and $m_0 = 0, v_0 = 0$.

During the later stages of training, when the gradients decrease significantly, Reddi et al. [30] observed that the adaptive learning rate of Adam increases, leading to potential divergence in parameter update. To address this issue, they proposed the AMSGrad. AMSGrad modifies the parameter update by using the maximum value of the past second-order moments, applying more friction to the optimization process to prevent overshooting the target value. The parameter update rule is defined as:

$$\begin{aligned} \tilde{v}_t^{max} &\leftarrow \max(\tilde{v}_{t-1}^{max}, v_t) \\ \theta_t &\leftarrow \theta_{t-1} - \frac{\eta \hat{m}_t}{\sqrt{\tilde{v}_t^{max}} + \delta} \end{aligned} \tag{8}$$

DiffGrad [31] is another improvement upon Adam. Unlike AMSGrad, which relies on long-term gradients information, diffGrad focuses on short-term gradients variations. It introduces a diffGrad Friction Coefficient (DFC) to control the adaptive learning rate. The DFC at the t th iteration, denoted as ξ_t , and the parameter update rule is defined as follows:

$$\begin{aligned} \xi_t &= AbsSig(\Delta g_t) = AbsSig(g_{t-1} - g_t) = \frac{1}{1 + e^{-|g_{t-1} - g_t|}} \\ \theta_t &\leftarrow \theta_{t-1} - \frac{\eta \xi_t \hat{m}_t}{\sqrt{\hat{v}_t} + \delta} \end{aligned} \tag{9}$$

Nevertheless, DFC compresses the step size of parameter update to 0.5–1 times, which further reduces the momentum and leads to slow convergence speed.

Zhou et al. [21] based on the definition of G–L fractional-order calculus, designed two optimizers called FCSGD_G–L and FCAdam_G–L by combining SGD and Adam, respectively. The gradients for both optimizers with a fractional order α are defined as:

$${}^G D_{\theta_{t-1}}^\alpha L(\theta_{t-1}) = g_t + \sum_{i=1}^{10} c[i-1] w_i g_{t-i} \tag{10}$$

where $c[\cdot]$ is the perturbation coefficient of 0 or 1, $w_i = \left(1 - \frac{\alpha+1}{i+1}\right) w_{i-1}$ and $w_0 = 0$.

Parameter update of FCSGD_G–L is defined as:

$$\theta_t \leftarrow \theta_{t-1} - \eta_\alpha^G D_{\theta_{t-1}}^\alpha L(\theta_{t-1}) \tag{11}$$

Parameter update of FCAdam_G–L is defined as:

$$\begin{aligned} m_t &\leftarrow \beta_1 m_{t-1} + (1 - \beta_1) {}^G D_{\theta_{t-1}}^\alpha L(\theta_{t-1}) \\ v_t &\leftarrow \beta_2 v_{t-1} + (1 - \beta_2) \left[{}^G D_{\theta_{t-1}}^\alpha L(\theta_{t-1}) \right]^2 \\ \hat{m}_t &\leftarrow \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &\leftarrow \frac{v_t}{1 - \beta_2^t} \\ \theta_t &\leftarrow \theta_{t-1} - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \delta} \end{aligned} \tag{12}$$

3 Algorithm

The first-order optimization algorithms only utilize the local neighborhood information of the parameters to be updated, which leads to a risk of getting trapped in local minima. The second-order optimization algorithms make better use of the curvature information of the function but are constrained by computational complexity, which limits their widespread adoption. In this section, we introduce the G–L fractional-order derivative method and use it to approximate the gradient, thereby supplementing the long-term state information lacking in the first-order optimizers. We then introduce a step size control coefficient that reflects short-term gradients changes, allowing the parameter update step size to be flexibly adjusted in real-time according to the current short-term state information. Finally, we combine these techniques to create our new adaptive learning rate optimizer AdaGL, which can accelerate convergence and prevent falling into the local optima.

3.1 G–L Fractional-Order Derivative

For a continuous function $f(x)$, the definition of the integer-order derivative with order n is as follows:

$$f^{(n)}(x) = \frac{d^n f(x)}{dx^n} = \lim_{h \rightarrow 0} \frac{1}{h^n} \sum_{r=0}^n (-1)^r \frac{n(n-1)\dots(n-r+1)}{r!} f(x-rh) \tag{13}$$

Fractional-order derivative is a classical concept in mathematics, which can be regarded as a generalized form of integer-order derivative. By extending the derivative order from integer

to arbitrary rational number, the definition of fractional-order derivative can be obtained. The Grünwald–Letnikov (G–L) fractional-order derivative of a function $f(x)$ with order α is defined as:

$${}_{G-L}D_t^\alpha f(x) = \lim_{h \rightarrow 0} \frac{1}{h^\alpha} \sum_{j=0}^{\frac{t-t_0}{h}} (-1)^j \frac{\Gamma(\alpha + 1)}{\Gamma(j + 1) \Gamma(\alpha - j + 1)} f(x - jh) \tag{14}$$

where h is the step size, and t, t_0 represents the upper and lower bounds of the steps respectively, and $\Gamma(\cdot)$ denotes the Gamma function.

When the step size h is small enough, the limit operation in Eq. 14 can be neglected. In the optimization process of neural networks, the step size h for parameter update is not a continuous value. We set h to its minimum value, that is $h = 1$. Therefore, in this case, the definition of the G–L fractional-order derivative can be approximated as follows:

$$\begin{aligned} {}_{G-L}D_t^\alpha f(x) &\approx \frac{1}{h^\alpha} \sum_{j=0}^{\frac{t-t_0}{h}} (-1)^j \frac{\Gamma(\alpha + 1)}{\Gamma(j + 1) \Gamma(\alpha - j + 1)} f(x - jh) \\ &= \sum_{j=0}^{t-t_0} (-1)^j \frac{\Gamma(\alpha + 1)}{\Gamma(j + 1) \Gamma(\alpha - j + 1)} f(x - j) \end{aligned} \tag{15}$$

Equation 15 is an infinite expansion formula. Nevertheless, when performing computations on a computer, it is necessary to convert Eq. 15 into a finite series expansion. Some studies have shown that when the fractional-order derivative formula is expanded to 10 terms, the effect of the fractional order α on the fractional-order derivative is minimal [32]. At this point, the properties of the fractional-order derivative can be well expressed in neural networks. Therefore, we set the number of expansion terms in the G–L fractional-order derivative formula to 10, resulting in:

$${}_{G-L}D_t^\alpha f(x) = \sum_{j=0}^{10} (-1)^j \frac{\Gamma(\alpha + 1)}{\Gamma(j + 1) \Gamma(\alpha - j + 1)} f(x - j) \tag{16}$$

3.2 Step Size Control Coefficient

In the adaptive learning rate optimization algorithms, the most critical thing is the control mode of learning rate. The goal of deep neural networks optimization is to find a flat minimum with low loss. An adaptive learning rate optimizer should have the ability to escape local minima and saddle points and stay away from sharp minima. To adjust the adaptive learning rate appropriately and ensure that the parameters are iteratively updated in the desired direction, we introduce a step size control coefficient.

Inspired by some common and popular activation functions in deep learning, we compare and select the softsign activation function $y = x/(1 + |x|)$ [33] experimentally. We scale and shift it to obtain the step size control coefficient C_t in the t th iteration:

$$C_t = \frac{0.5 |\nabla g_t|}{1 + |\nabla g_t|} + 0.6 = 1.1 - \frac{1}{2(1 + |g_{t-1} - g_t|)} \tag{17}$$

The step size control coefficient utilizes the short-term gradients behavior to control the learning rate. Its function image is shown in Fig. 2. By observing the formula and image

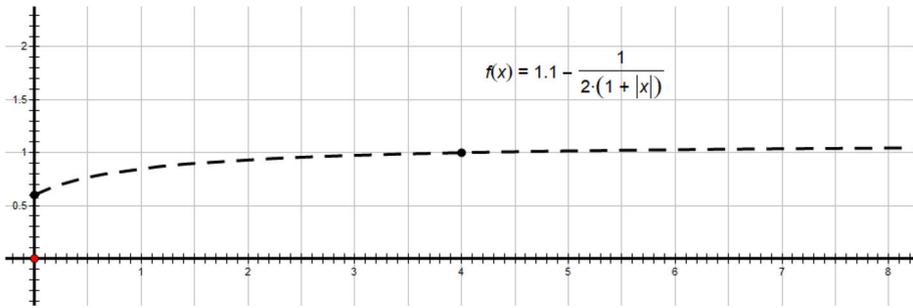
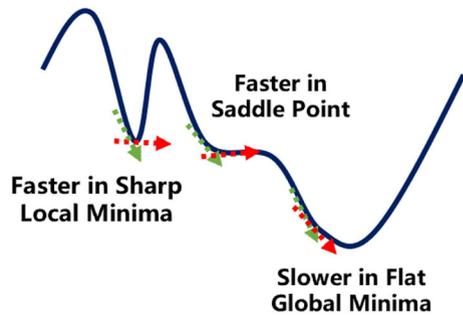


Fig. 2 Function image of the step size control coefficient

Fig. 3 Step size control coefficient dynamically adjusts the parameter update speed by short-term gradients change



of the step size control coefficient, we can find that it is a function with a value range of $[0.6, 1.1)$. This ensures that the adaptive learning rate does not decay too much, avoiding a serious slowdown of the learning process. Meanwhile, the step size control coefficient has the potential to increase the rate of moving away from unexpected regions. To illustrate this more visually, as shown in Fig. 3, when the instantaneous change of the gradient is small, i.e., $|g_{t-1} - g_t|$ is small, it indicates that the algorithm may be close to a flat minimum of the objective. In this case, the parameter update step size adaptively decreases, increasing the possibility of further exploration. On the other hand, when the instantaneous change of gradient is large, i.e., $|g_{t-1} - g_t|$ is large, it suggests that the algorithm may have reached a local (or sharp) minimum or a saddle point. In this case, the parameter update step size does not decrease or may increase slightly, providing the momentum to escape from that region and continue searching for better optima.

3.3 AdaGL Optimizer

When Eq. 16 is applied to the training process of deep neural networks, for the objective optimization function L , the G–L fractional-order approximated gradient of the parameter θ with order α can be written as:

$${}_{G-L}D_t^\alpha L(\theta_{t-1}) = \sum_{j=0}^{10} (-1)^j \frac{\Gamma(\alpha + 1)}{\Gamma(j + 1)\Gamma(\alpha - j + 1)} g_{t-j} \tag{18}$$

We adopt a similar approach to the Adam algorithm, replacing the gradient with the biased first order moment m_t and second order moment v_t calculated using the G–L fractional-order

Algorithm 1 AdaGL Optimizer

Require: objective L , learning rate η , fractional order α , initial parameter θ_0 , decay rate hyperparameters $\{\beta_1, \beta_2\}$, numerical stability constant δ

- 1: **Initialize:** first moment $m_0 = 0$, second moment $v_0 = 0$
- 2: **for** $t = 1$ to \dots **do**
- 3: $g_t \leftarrow \frac{1}{m} \nabla_{\theta} \sum L(\theta_{t-1})$
- 4: $G_{-L} D_t^{\alpha} L(\theta_{t-1}) = \sum_{j=0}^{10} (-1)^j \frac{\Gamma(\alpha+1)}{\Gamma(j+1)\Gamma(\alpha-j+1)} g_{t-j}$
- 5: $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)_{G-L} D_t^{\alpha} L(\theta_{t-1})$
- 6: $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) [G_{-L} D_t^{\alpha} L(\theta_{t-1})]^2$
- 7: $\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}, \hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2^t}$
- 8: $C_t = 1.1 - [2(1 + |g_{t-1} - g_t|)]^{-1}$
- 9: $\theta_t \leftarrow \theta_{t-1} - \frac{\eta C_t \hat{m}_t}{\sqrt{\hat{v}_t + \delta}}$
- 10: **end for**
- 11: **Return** θ_t

approximated gradient in Eq. 18. To address the issue of initial biases towards zero during the early iterations, bias corrections are separately applied, resulting in \hat{m}_t and \hat{v}_t :

$$\begin{aligned}
 m_t &\leftarrow \beta_1 m_{t-1} + (1 - \beta_1)_{G-L} D_t^{\alpha} L(\theta_{t-1}) \\
 v_t &\leftarrow \beta_2 v_{t-1} + (1 - \beta_2) [G_{-L} D_t^{\alpha} L(\theta_{t-1})]^2 \\
 \hat{m}_t &\leftarrow \frac{m_t}{1 - \beta_1^t} \\
 \hat{v}_t &\leftarrow \frac{v_t}{1 - \beta_2^t}
 \end{aligned} \tag{19}$$

Finally, by applying the step size control coefficient calculated in Eq. 17, the parameter update formula of our proposed AdaGL optimizer is obtained:

$$\theta_t \leftarrow \theta_{t-1} - \frac{\eta C_t \hat{m}_t}{\sqrt{\hat{v}_t + \delta}} \tag{20}$$

In Eq. 20, the modified first and second order moments contain the long-term memory characteristics of fractional-order gradients, while the step size control coefficient reflects the short-term variations of the real gradients. The combination of these two components enables the AdaGL optimizer to have both global and local perspectives. By utilizing the long-term gradients to maintain the overall direction and the short-term gradients to make fine adjustments in the details, the parameters are able to find the correct iterative direction more effectively and descend rapidly.

The pseudocode of the proposed AdaGL is provided in Algorithm 1.

4 Experiments

In this section, we perform various deep learning tasks to evaluate and compare the performance of the proposed AdaGL optimizer with other popular optimizers. The experiments include:

- (1) Image classification (CIFAR10 dataset [34]) with CNNs frameworks of ResNet and DenseNet;

Table 1 Network architecture and dataset settings in experimental tasks

Task	Network architecture	Dataset
Image classification	ResNet & DenseNet	CIFAR10
Node classification	GCN	Cora, Citeseer, Pubmed
Graph classification	GCN	MUTAG, PTC-MR, BZR, COX2, PROTEINS, NCI1
image production	WGAN	CIFAR10
Language modeling	LSTM	Penn TreeBank

- (2) Node classification (Cora, Citeseer and Pubmed datasets [35]) and graph classification (MUTAG [36], PTC-MR [37], BZR [38], COX2 [38], PROTEINS [39] and NCI1 [40] datasets) with GCN;
- (3) Image generation (CIFAR10 dataset) with WGAN;
- (4) Language modeling (Penn TreeBank dataset [41]) with LSTM.

The settings for each task are described in Table 1.

The compared optimizers include:

- (1) SGD-like optimizers: SGD, MSVAG [42], FracM and FCSGD_G-L;
- (2) Adaptive learning rate optimizers: Adam, Yogi [43], AdaBound [44], AdaMod [45], RAdam [46], diffGrad, AdaBelief [47], AdaDerivative [48] and FCAdam_G-L.

4.1 Image Classification with CNNs

First, using the CIFAR10 dataset, we conduct image classification experiments on the popular Convolutional Neural Networks frameworks, ResNet34 and DenseNet121.

The CIFAR10 dataset consists of 60,000 images, including 50,000 images for training and 10,000 images for validation. The size of all images is $32 \times 32 \times 3$. The ResNet architecture introduces skip connections between layers, allowing the output of one previous layer to be directly connected to the input of a later layer. This residual connection in ResNet enables the learning target to be the residual between the output and input. This design facilitates the training of deeper CNNs and achieves higher accuracy. The DenseNet architecture follows a similar idea as ResNet but establishes dense connections between all preceding layers and subsequent layers, leading to improved performance compared to ResNet.

The hyperparameters for the experiments are set as follows: epoch is 200, the batch size is 128, the initial learning rates are set to 0.1 for SGD, FracM and FCSGD_G-L, and 0.001 for other optimizers. The learning rates for all optimizers are reduced by a factor of 10 in the 150th epoch. For all optimizers in the experiment, use the default optimal parameter settings in the source code. The decay rate hyperparameter β for momentum-based optimizers is 0.9, and the moment decay rate hyperparameters β_1 and β_2 for adaptive learning rate optimizers are 0.9 and 0.999 respectively. The fractional order α is 1.5. The numerical stability constant δ is $1e-8$ and weight decay is $5e-4$.

Figure 4 shows the train and test accuracy curves on ResNet34 and DenseNet121. Table 2 presents the mean and standard deviation (in %) of the test accuracy for each optimization algorithm. The experiments verify the fast and stable convergence performance of our proposed optimizer. In comparison to SGD-like optimizers, our method ultimately achieves superior performance than traditional SGD, especially on DenseNet121. However, it does not surpass the classification accuracy achieved by two SGD improvements, FracM and

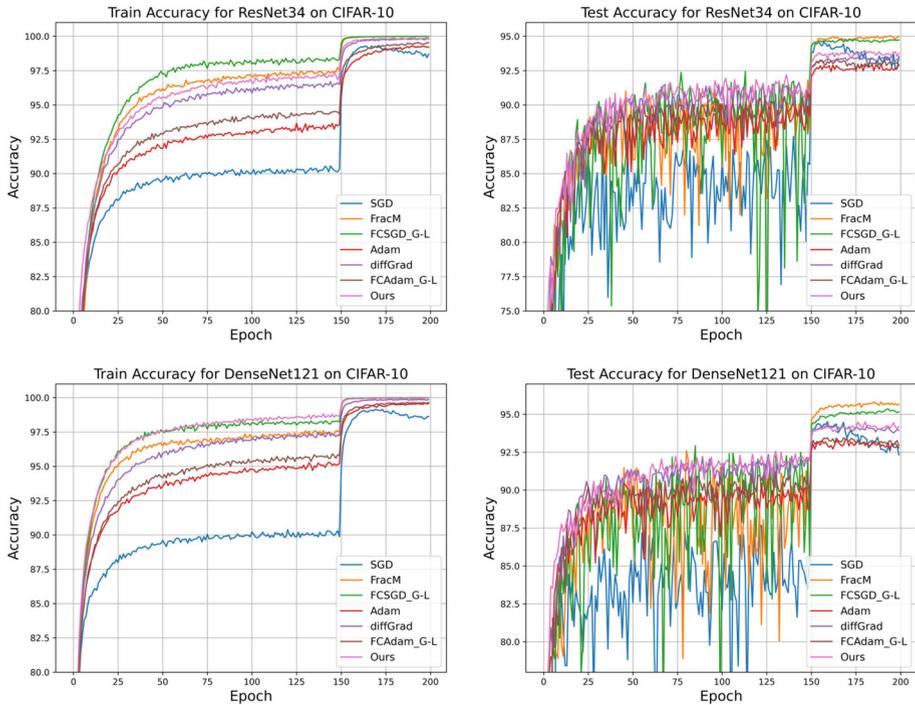


Fig. 4 Train and test accuracy curves of optimizers for ResNet34 & DenseNet121 on CIFAR10

Table 2 Test accuracy values of optimizers for ResNet34 & DenseNet121 on CIFAR10 (in %)

Optimizer	ResNet34	DenseNet121
SGD	93.62 ± 0.49	93.39 ± 0.57
FracM	94.88 ± 0.08	95.63 ± 0.09
FCSGD_G-L	94.68 ± 0.06	95.10 ± 0.11
Adam	92.69 ± 0.15	93.02 ± 0.14
diffGrad	93.35 ± 0.09	94.05 ± 0.11
FCAdam_G-L	93.35 ± 0.09	93.20 ± 0.12
AdaGL (Ours)	93.73 ± 0.11	94.15 ± 0.14

The best results among different optimizers are highlighted in bold

FCSGD_G-L. In addition, compared with other adaptive learning rate optimizers, our optimizer performs the best on the test set, achieving approximately 1.04% and 1.13% accuracy improvements over Adam on ResNet34 and DenseNet121, respectively.

Previous studies have indicated that, in image classification tasks on the CIFAR10 dataset, although adaptive learning rate algorithms converge faster than SGD-like algorithms, SGD-like algorithms typically yield better final accuracy results [49, 50]. For classification tasks in Computer Vision, adaptive learning rate optimizers tend to find sharp minima rather than flat minima, leading to poorer generalization performance compared to SGD-like optimizers. Our proposed AdaGL outperforms SGD on the test set (other adaptive learning rate optimizers do not surpass SGD), validating its ability to control step size to some extent for finding

Table 3 Statistical information of the graph benchmark datasets

Task	Dataset	Graphs	Avg. nodes	Avg. edges	Classes
Node classification	Cora	1	2708	5429	7
	Citeseer	1	3327	4732	6
	Pubmed	1	19,717	44,338	3
Graph classification	MUTAG	188	17.93	19.79	2
	PTC-MR	344	14.29	14.69	2
	BZR	405	35.75	38.36	2
	COX2	467	41.22	43.45	2
	PROTEINS	1113	39.06	72.82	2
	NCI1	4110	29.87	32.30	2

flat minima. However, its search capability is still slightly inferior to the latest non-adaptive learning rate optimizers.

4.2 Node Classification and Graph Classification with GCN

Research on the performance of different optimizers on Graph Neural Networks (GNNs) is still limited. In this section, we conduct node classification and graph classification experiments on multiple graph benchmark datasets to evaluate the performance of the current popular optimizers and our proposed optimizer.

Table 3 provides the statistical information of the graph benchmark datasets used in the experiments. For both experiments, we employ a three-layer standard GCN as the model. GCN is one of the most classic architectures in GNNs. It utilizes a first-order Chebyshev polynomial approximation and defines graph convolutional operations by mapping graph signals to the spectral domain. This allows GCN to handle non-Euclidean spatial data that traditional CNNs struggle with.

First, we consider the node classification task on the Cora, Citeseer and Pubmed datasets. All of them are citation network datasets used for semi-supervised document classification. They are undirected graphs where nodes represent papers and edges represent citation relationships. In the experiments, we use standard splits for 10 runs of experimental evaluation, that is, in each class 20 nodes are used for training, 500 nodes are used for validation, and 1000 nodes are used for testing. We train 200 epochs in each run. The initial learning rates are set to 0.07 for Cora, 0.1 for Citeseer, and 0.12 for Pubmed. The optimal accuracy of each run is recorded separately, and the mean and the standard deviation (in %) are calculated. Table 4 provides a comparison of the performance between our optimizer and existing optimizers.

By observing the experimental results in Table 4, we can see that our proposed optimizer achieves the best accuracy on both Cora and Pubmed datasets, although it does not achieve the best performance on the Citeseer dataset, it still performs well compared to other adaptive learning rate optimization algorithms.

Next, we conduct graph classification experiments to further evaluate the performance of the proposed optimization algorithm. The goal of graph classification tasks is to learn the mapping function between graphs and corresponding category labels to correctly predict the class of unlabeled graphs. We choose six benchmark datasets from bioinformatics for graph classification: MUTAG, PTC-MR, BZR, COX2, PROTEINS and NCI1. They are all datasets

Table 4 Node classification on 3-layer GCN: mean \pm standard deviation of accuracy over 10 runs (in %)

Optimizer	Cora	Citeseer	Pubmed
SGD	81.73 \pm 0.45	71.97 \pm 0.57	77.40 \pm 1.49
FracM	81.08 \pm 0.66	71.75 \pm 0.84	73.56 \pm 2.56
FCSGD_G-L	78.57 \pm 1.57	71.55 \pm 0.95	57.57 \pm 4.24
Adam	81.32 \pm 0.36	68.73 \pm 0.71	79.33 \pm 0.32
Adamod	74.47 \pm 1.19	56.01 \pm 1.03	75.66 \pm 0.64
RAdam	79.78 \pm 0.72	68.06 \pm 1.25	77.22 \pm 0.33
diffGrad	81.65 \pm 0.49	69.60 \pm 0.46	79.34 \pm 0.33
AdaBelief	81.91 \pm 0.60	70.39 \pm 0.67	78.88 \pm 0.52
AdaDerivative	81.77 \pm 0.68	70.51 \pm 0.69	78.87 \pm 0.41
FCAdam_G-L	81.73 \pm 0.27	69.59 \pm 0.62	79.47 \pm 0.36
AdaGL (ours)	81.91 \pm 0.35	69.74 \pm 0.72	79.53 \pm 0.43

The best results among different optimizers are highlighted in bold

Table 5 Graph classification on 3-layer GCN: mean \pm standard deviation of accuracy over 10 runs (in %)

Optimizer	MUTAG	PTC-MR	BZR	COX2	PROTEINS	NCI1
SGD	88.24 \pm 5.34	64.96 \pm 8.17	85.27 \pm 5.46	81.63 \pm 4.47	59.59 \pm 2.89	67.10 \pm 1.74
FracM	84.37 \pm 8.44	66.96 \pm 7.09	84.54 \pm 5.56	81.84 \pm 3.82	59.59 \pm 2.89	66.52 \pm 1.72
FCSGD_G-L	86.19 \pm 8.18	65.36 \pm 9.92	83.04 \pm 7.01	81.84 \pm 4.57	59.59 \pm 2.89	65.94 \pm 2.72
Adam	93.03 \pm 4.24	65.75 \pm 6.15	86.73 \pm 5.17	84.64 \pm 3.43	76.82 \pm 1.90	72.51 \pm 4.05
Adamod	88.30 \pm 8.74	67.44 \pm 8.35	88.37 \pm 3.36	84.20 \pm 2.90	76.64 \pm 2.78	74.22 \pm 2.66
RAdam	90.40 \pm 4.63	65.36 \pm 8.88	86.07 \pm 4.93	84.64 \pm 4.14	74.49 \pm 4.20	71.68 \pm 5.01
diffGrad	92.57 \pm 4.18	65.30 \pm 8.25	88.06 \pm 4.13	84.82 \pm 3.13	76.63 \pm 2.50	73.94 \pm 2.80
AdaBelief	90.40 \pm 3.99	65.98 \pm 7.72	85.51 \pm 5.82	84.23 \pm 4.99	76.81 \pm 2.48	73.80 \pm 2.36
AdaDerivative	92.57 \pm 4.18	65.47 \pm 7.84	87.60 \pm 3.80	83.97 \pm 4.99	77.08 \pm 2.51	73.72 \pm 3.10
FCAdam_G-L	92.51 \pm 3.62	67.59 \pm 6.60	86.31 \pm 4.02	85.05 \pm 3.83	77.07 \pm 3.03	73.55 \pm 2.48
AdaGL (Ours)	92.63 \pm 5.37	68.27 \pm 6.94	88.79 \pm 5.01	85.50 \pm 4.96	77.24 \pm 3.21	74.04 \pm 2.60

The best results among different optimizers are highlighted in bold

about chemical molecules or compounds, where nodes represent atoms and edges represent chemical bonds. The task is to judge the types or properties of compounds. In the experiments, we split the datasets into 90% training set and 10% test set randomly, and conduct 10-fold cross validation. For each fold, we train the model for 300 epochs with an initial learning rate chosen from the optimal values searched in the range of {0.007, 0.01, 0.015, 0.022}. The batch size is set to 64. We record the best accuracy achieved in each fold and calculate the mean and standard deviation over the 10-fold cross validation (in %). Table 5 provides a comparison of the performance between our optimizer and existing optimizers.

As can be seen from Table 5, in the graph classification experiments on the bioinformatics datasets, our method achieves the second-best classification accuracy on the MUTAG and NCI1 datasets, while it achieves the best performance on the other four datasets.

4.3 Image Generation with WGAN

We perform image generation tasks on the CIFAR10 dataset using a WGAN model with the original CNN generator. WGAN is a classic generative model that improves upon many issues

Table 6 Mean \pm standard deviation of FID (Lower is better) and IS (Higher is better) over 5 runs with WGAN on CIFAR10

Optimizer	FID (lower is better)	IS (higher is better)
SGD	453.54 \pm 0.61	1.04 \pm 0.01
MSVAG	449.73 \pm 0.55	1.03 \pm 0.01
FracM	363.92 \pm 0.12	1.11 \pm 0.01
FCSGD_G-L	368.84 \pm 0.08	1.04 \pm 0.02
Adam	98.02 \pm 0.09	3.20 \pm 0.02
RAdam	80.50 \pm 0.08	3.61 \pm 0.04
diffGrad	74.32 \pm 0.13	3.73 \pm 0.05
AdaBelief	82.05 \pm 0.21	2.31 \pm 0.01
FCAAdam_G-L	89.91 \pm 0.08	3.88 \pm 0.03
AdaGL (ours)	70.99 \pm 0.06	3.96 \pm 0.05

The best results among different optimizers are highlighted in bold

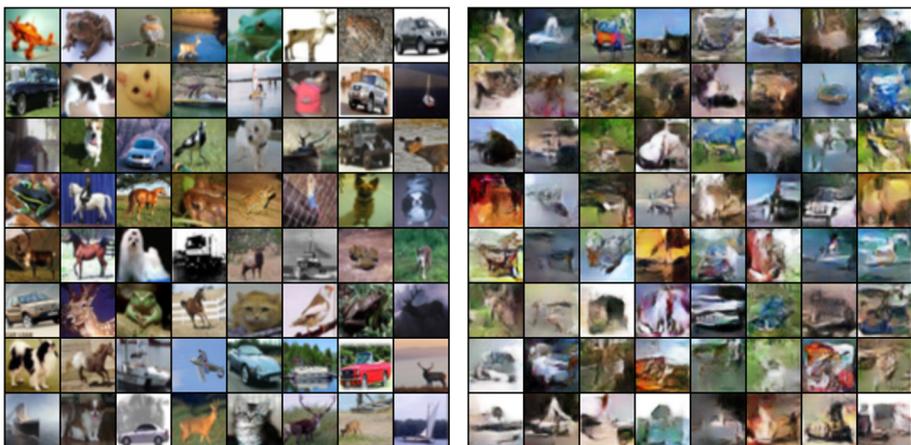


Fig. 5 Real images (left) and fake samples generated on WGAN trained by AdaGL optimizer (right)

in the original GAN. The original GAN suffers from training difficulties, heavy reliance on the design of the generator and discriminator, and lack of sample diversity, among other problems. WGAN successfully alleviates these limitations.

In the experiment, we train the model for 100 epochs and generate 64,000 fake images from noise. We calculate the Inception Score (IS) [51] for fake images, as well as the Fréchet Inception Distance (FID) [52] between the fake images and the real images. Both IS and FID are widely used metrics for evaluating the performance of generative models. IS (higher is better) measures the diversity and realism of the generated images, while FID (lower is better) reflects both the quality and diversity of the generated images.

For each optimizer, we use the optimal hyperparameter settings from the source code. We conduct 5 runs experiments, and the results are shown in Table 6. Compared to other optimizers, our proposed AdaGL optimizer significantly outperforms them on the WGAN, achieving the lowest FID and the highest IS. Figure 5 shows real images and fake samples generated by the WGAN trained with our proposed AdaGL optimizer.

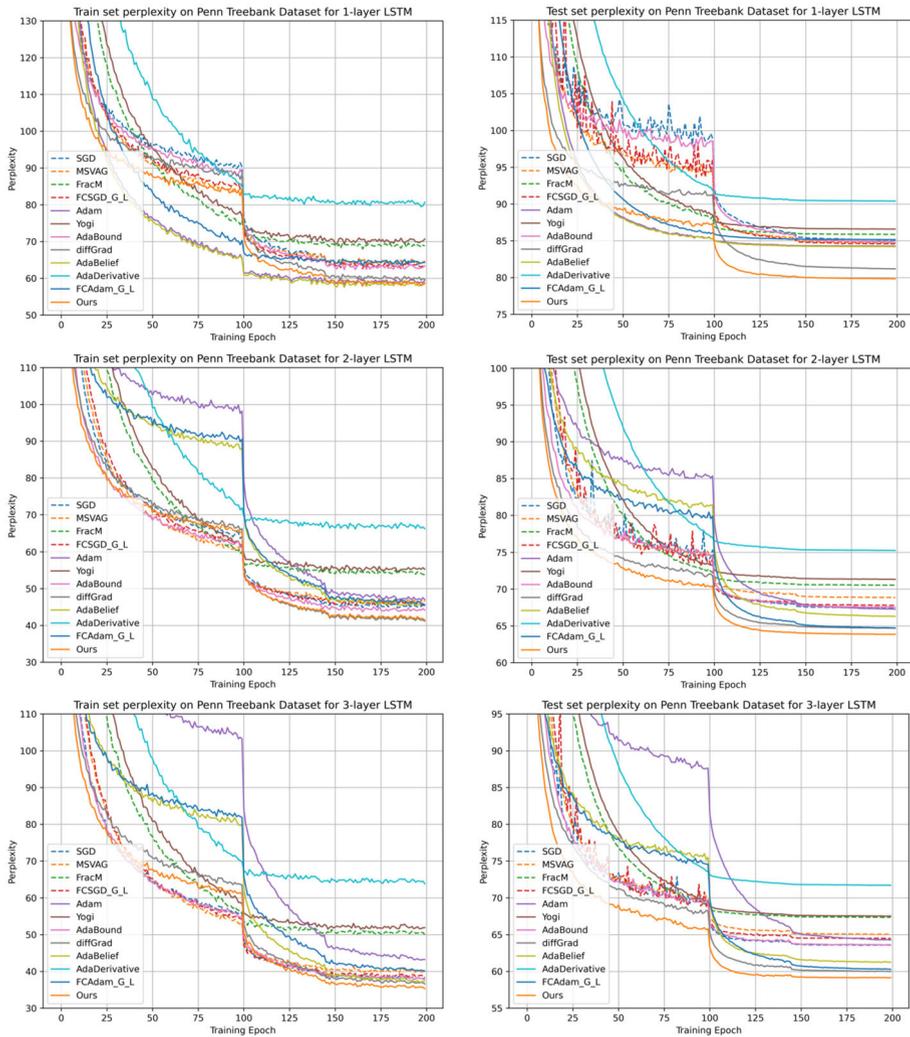


Fig. 6 PPL (lower is better) curves with different layers LSTM on Penn TreeBank

4.4 Language Modeling with LSTM

In the language modeling experiment, we use 1-layer, 2-layer and 3-layer LSTM on the Penn TreeBank dataset to verify the performance of the proposed optimizer. The Penn TreeBank dataset is a classic English language model dataset consisting of 900,000 words of English text, commonly used for Natural Language Processing tasks. LSTM is a variant of the Recurrent Neural Networks (RNNs) model that can effectively handle time series data, particularly in long sequences where it outperforms classical RNNs.

Our experiment runs 200 epochs with a batch size of 20. The initial learning rates are set to 30 for SGD, MSVAG and FCSGD_G-L, 1 for FracM, and the optimal values from

Table 7 Test PPL (lower is better) with different layers LSTM on Penn TreeBank

Optimizer	1-Layer LSTM	2-Layer LSTM	3-Layer LSTM
SGD	85.07	67.42	63.55
MSVAG	84.68	68.83	65.03
FracM	85.87	70.53	67.35
FCSGD_G–L	84.58	67.77	64.46
Adam	84.28	67.27	64.28
Yogi	86.59	71.33	67.51
AdaBound	84.78	67.53	63.58
diffGrad	81.18	64.68	59.98
AdaBelief	84.21	66.29	61.23
AdaDerivative	90.40	75.25	71.70
FCAadam_G–L	85.16	64.71	60.26
AdaGL (ours)	79.83	63.85	59.13

The best results among different optimizers are highlighted in bold

0.001 and 0.01 for other optimizers. We use perplexity (PPL) as the evaluation metric for performance. The experimental results are shown in Fig. 6 and Table 7.

Observing the experimental results, the following conclusions can be drawn: Regardless of whether it is a 1-layer, 2-layer, or 3-layer LSTM model, the proposed optimizer demonstrates fast convergence speed and achieves the lowest PPL during both training and testing stages, outperforming other optimizers. Furthermore, when comparing different numbers of layers, differences in the extent of improvement can be observed. Compared to the Adam optimizer, our optimizer reduces perplexity by 4.45% in the 1-layer LSTM and by 5.15% in the 3-layer LSTM, which indicates that our optimizer achieves better performance improvements when handling more complex models and larger-scale language modeling tasks.

5 Conclusion

This article proposes a new adaptive learning rate optimization algorithm called AdaGL. AdaGL is based on the G–L fractional-order derivative method, which approximates the gradient during parameter update and fully utilizes the long-term curvature information of the loss function. At the same time, we introduce a step size control coefficient that controls the parameter update direction and step size based on recent gradients changes, enabling acceleration to escape when encountering unexpected local (or sharp) minima or saddle points, and deceleration to explore when encountering flat target points.

Extensive experimental results on image classification, node classification, graph classification, image generation, and language modeling demonstrate that AdaGL achieves excellent performance compared to traditional optimizers, and can converge quickly and stably with high accuracy across various deep learning tasks.

In the future, we will continue to explore the integration of fractional-order calculus methods and deep learning optimizers, as well as the properties of loss surfaces that affect the optimization algorithms' ability to find target points. We plan to extend the different definitions of fractional-order calculus in mathematics to deep learning optimization methods and further investigate hyperparameters such as fractional order. By comparing and analyzing

performances, we can select the most suitable optimization method when facing different neural network architectures.

Acknowledgements This research was funded by the National Natural Science Foundation of China (Nos. 62072024 and 41971396), the Fundamental Research Funds for Municipal Universities of Beijing University of Civil Engineering and Architecture (Nos. X20084 and ZF17061), and the BUCEA Post Graduate Innovation Project (PG2023143).

Author Contributions SC contributed to the conception of the study, performed the experiments, and wrote the manuscript. CZ initiated the idea of the research, followed up its development, and supervised and organized the course of the article. HM helped perform the analysis with constructive discussions and contributed to manuscript preparation.

Availability of Data and Materials The data is based on open-source repository.

Code Availability Code could be made available at a later stage.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethics Approval Not applicable.

Consent to Participate Not applicable.

Consent for Publication Not applicable.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Hinton GE, Osindero S, Teh Y-W (2006) A fast learning algorithm for deep belief nets. *Neural Comput* 18(7):1527–1554
2. Sargül M (2023) A survey on digital video stabilization. *Multimed Tools Appl* 86:1–27
3. Zou Z, Chen K, Shi Z, Guo Y, Ye J (2023) Object detection in 20 years: a survey. In: *Proceedings of the IEEE*
4. Su J, Xu B, Yin H (2022) A survey of deep learning approaches to image restoration. *Neurocomputing* 487:46–65
5. Ruder S (2016) An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*
6. Dubey SR, Singh SK, Chaudhuri BB (2023) Adanorm: adaptive gradient norm correction based optimizer for cnns. In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pp 5284–5293
7. Bottou L (1998) Online learning and stochastic approximations. *Online Learn Neural Netw* 17(9):142
8. Dauphin YN, Pascanu R, Gulcehre C, Cho K, Ganguli S, Bengio Y (2014) Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In: *Advances in neural information processing systems*, vol 27
9. Sutskever I, Martens J, Dahl G, Hinton G (2013) On the importance of initialization and momentum in deep learning. In: *International conference on machine learning*. PMLR, pp 1139–1147

10. LeCun Y, Bottou L, Orr GB, Müller K-R (2002) Efficient backprop. In: Neural networks: tricks of the trade, pp 9–50
11. Kingma DP, Ba, J (2014) Adam: a method for stochastic optimization. arXiv preprint [arXiv:1412.6980](https://arxiv.org/abs/1412.6980)
12. Keskar NS, Mudigere D, Nocedal J, Smelyanskiy M, Tang PTP (2016) On large-batch training for deep learning: generalization gap and sharp minima. arXiv preprint [arXiv:1609.04836](https://arxiv.org/abs/1609.04836)
13. Xing C, Arpit D, Tsirigotis C, Bengio Y (2018) A walk with sgd. arXiv preprint [arXiv:1802.08770](https://arxiv.org/abs/1802.08770)
14. Le Roux N, Bengio Y, Fitzgibbon A (2011) Improving first and second-order methods by modeling uncertainty. *Optim Mach Learn* 15:403
15. Raubitsek S, Mallinger K, Neubauer T (2022) Combining fractional derivatives and machine learning: a review. *Entropy* 25(1):35
16. Ahmad J, Khan S, Usman M, Naseem I, Moinuddin M, Syed HJ (2017) Fclms: fractional complex lms algorithm for complex system identification. In: 2017 IEEE 13th International colloquium on signal processing & its applications (CSPA). IEEE, pp 39–43
17. Khan S, Ahmad J, Naseem I, Moinuddin M (2018) A novel fractional gradient-based learning algorithm for recurrent neural networks. *Circuits Syst Signal Process* 37:593–612
18. Kan T, Gao Z, Yang C, Jian J (2021) Convolutional neural networks based on fractional-order momentum for parameter training. *Neurocomputing* 449:85–99
19. Herrera-Alcántara O (2022) Fractional derivative gradient-based optimizers for neural networks and human activity recognition. *Appl Sci* 12(18):9264
20. Yu Z, Sun G, Lv J (2022) A fractional-order momentum optimization approach of deep neural networks. *Neural Comput Appl* 34(9):7091–7111
21. Zhou X, Zhao C, Huang Y (2023) A deep learning optimizer based on Grünwald–Letnikov fractional order definition. *Mathematics* 11(2):316
22. Xue D (2018) Fractional calculus and fractional-order control. Science Press, Beijing
23. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
24. Huang G, Liu Z, Van Der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4700–4708
25. Kipf TN, Welling M (2016) Semi-supervised classification with graph convolutional networks. arXiv preprint [arXiv:1609.02907](https://arxiv.org/abs/1609.02907)
26. Arjovsky M, Chintala S, Bottou L (2017) Wasserstein generative adversarial networks. In: International conference on machine learning. PMLR, pp 214–223
27. Ma X, Tao Z, Wang Y, Yu H, Wang Y (2015) Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transp Res Part C Emerg Technol* 54:187–197
28. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. *J Mach Learn Res* 12(7):2121–2159
29. Tieleman T, Hinton G (2012) Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. *COURSERA Neural Netw Mach Learn* 4(2):26–31
30. Reddi SJ, Kale S, Kumar S (2019) On the convergence of adam and beyond. arXiv preprint [arXiv:1904.09237](https://arxiv.org/abs/1904.09237)
31. Dubey SR, Chakraborty S, Roy SK, Mukherjee S, Singh SK, Chaudhuri BB (2019) diffgrad: an optimization method for convolutional neural networks. *IEEE Trans Neural Netw Learn Syst* 31(11):4500–4511
32. Peng Z, Pu Y (2022) Convolution neural network face recognition based on fractional differential. *J Sichuan Univ (Nat Sci Ed)* 59(1):35–42
33. Bergstra J, Desjardins G, Lamblin P, Bengio Y (2009) Quadratic polynomials learn better image features. Technical report, 1337
34. Krizhevsky A, Hinton G et al (2009) Learning multiple layers of features from tiny images
35. Sen P, Namata G, Bilgic M, Getoor L, Galligher B, Eliassi-Rad T (2008) Collective classification in network data. *AI Mag* 29(3):93–93
36. Debnath AK, Compadre RL, Debnath G, Shusterman AJ, Hansch C (1991) Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *J Med Chem* 34(2):786–797
37. Helma C, King RD, Kramer S, Srinivasan A (2001) The predictive toxicology challenge 2000–2001. *Bioinformatics* 17(1):107–108
38. Sutherland JJ, O'Brien LA, Weaver DF (2003) Spline-fitting with a genetic algorithm: a method for developing classification structure- activity relationships. *J Chem Inf Comput Sci* 43(6):1906–1915
39. Borgwardt KM, Ong CS, Schönauer S, Vishwanathan S, Smola AJ, Kriegel H-P (2005) Protein function prediction via graph kernels. *Bioinformatics* 21(suppl-1):47–56
40. Wale N, Watson IA, Karypis G (2008) Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl Inf Syst* 14:347–375

41. Marcus M, Santorini B, Marcinkiewicz MA (1993) Building a large annotated corpus of English: the Penn treebank
42. Balles L, Hennig P (2018) Dissecting adam: the sign, magnitude and variance of stochastic gradients. In: International conference on machine learning. PMLR, pp 404–413
43. Zaheer M, Reddi S, Sachan D, Kale S, Kumar S (2018) Adaptive methods for nonconvex optimization. In: Advances in neural information processing systems, vol 31
44. Luo L, Xiong Y, Liu Y, Sun X (2019) Adaptive gradient methods with dynamic bound of learning rate. arXiv preprint [arXiv:1902.09843](https://arxiv.org/abs/1902.09843)
45. Ding J, Ren X, Luo R, Sun X (2019) An adaptive and momental bound method for stochastic learning. arXiv preprint [arXiv:1910.12249](https://arxiv.org/abs/1910.12249)
46. Liu L, Jiang H, He P, Chen W, Liu X, Gao J, Han J (2019) On the variance of the adaptive learning rate and beyond. arXiv preprint [arXiv:1908.03265](https://arxiv.org/abs/1908.03265)
47. Zhuang J, Tang T, Ding Y, Tatikonda SC, Dvornek N, Papademetris X, Duncan J (2020) Adabelief optimizer: adapting stepsizes by the belief in observed gradients. *Adv Neural Inf Process Syst* 33:18795–18806
48. Zou W, Xia Y, Cao W (2023) Adaderivative optimizer: adapting step-sizes by the derivative term in past gradient information. *Eng Appl Artif Intell* 119:105755
49. Wilson AC, Roelofs R, Stern M, Srebro N, Recht B (2017) The marginal value of adaptive gradient methods in machine learning. *Adv Neural Inf Process Syst* 30:4148–4158
50. Keskar NS, Socher R (2017) Improving generalization performance by switching from adam to sgd. arXiv preprint [arXiv:1712.07628](https://arxiv.org/abs/1712.07628)
51. Salimans T, Goodfellow I, Zaremba W, Cheung V, Radford A, Chen X (2016) Improved techniques for training gans. *Adv Neural Inf Process Syst* 29:2234–2242
52. Heusel M, Ramsauer H, Unterthiner T, Nessler B, Hochreiter S (2017) Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Adv Neural Inf Process Syst* 30:6626–6637

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.