mathematics

MDPI

Article

A Deep Learning Optimizer Based on Grünwald–Letnikov Fractional Order Definition

Xiaojun Zhou, Chunna Zhao * and Yaqun Huang

School of Information Science and Engineering, Yunnan University, Kunming 650500, China
* Correspondence: zhaochunna@ynu.edu.cn

Abstract: In this paper, a deep learning optimization algorithm is proposed, which is based on the Grünwald–Letnikov (G-L) fractional order definition. An optimizer fractional calculus gradient descent based on the G-L fractional order definition (FCGD_G-L) is designed. Using the short-memory effect of the G-L fractional order definition, the derivation only needs 10 time steps. At the same time, via the transforming formula of the G-L fractional order definition, the Gamma function is eliminated. Thereby, it can achieve the unification of the fractional order and integer order in FCGD_G-L. To prevent the parameters falling into local optimum, a small disturbance is added in the unfolding process. According to the stochastic gradient descent (SGD) and Adam, two optimizers' fractional calculus stochastic gradient descent based on the G-L definition (FCSGD_G-L), and the fractional calculus Adam based on the G-L definition (FCAdam_G-L), are obtained. These optimizers are validated on two time series prediction tasks. With the analysis of train loss, related experiments show that FCGD_G-L has the faster convergence speed and better convergence accuracy than the conventional integer order optimizer. Because of the fractional order property, the optimizer exhibits stronger robustness and generalization ability. Through the test sets, using the saved optimal model to evaluate, FCGD_G-L also shows a better evaluation effect than the conventional integer order optimizer.

Keywords: deep learning optimizer; stochastic gradient descent; fractional order; Adam; time series prediction

MSC: 34A08

check for updates

Citation: Zhou, X.; Zhao, C.; Huang, Y. A Deep Learning Optimizer Based on Grünwald–Letnikov Fractional Order Definition. *Mathematics* **2023**, *11*, 316. https://doi.org/10.3390/math11020316

Academic Editors: Pengyu Chen, Haiyong Qin and Haoyu Niu

Received: 3 December 2022
Revised: 25 December 2022
Accepted: 4 January 2023
Published: 7 January 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In the field of deep learning, the research of optimization algorithms has been an important direction. Among them, the optimization algorithms based on the gradient descent method have become the mainstream. Gradient Descent (GD) is the most basic optimization algorithm. Currently, various improved optimization algorithms in deep learning are based on it. It has a faster convergence speed, its convergence rule can be illustrated simply as $\theta_{n+1} = \theta_n - \eta \nabla J(\theta_n)$, where η is the learning rate and $\nabla J(\theta_n)$ is the gradient of $J(\theta)$ at θ_n [1]. GD is hardly practical in deep learning. It evaluates the entire datasets at each iteration, and the current datasets are getting larger and larger, which can easily make video memory and memory insufficient. SGD accelerates the convergence speed and solves the problem of excessive video memory and memory occupation. However, it causes the direction of the gradient to fluctuate too much at each iteration. Due to the disadvantages of GD and SGD, MBGD is proposed, which does not cause video memory and memory overflow and overcomes the problem of gradient direction fluctuation [2]. It is a compromise between GD and SGD. It degenerates to SGD when the batch size is 1. It becomes GD when the batch size is the total sample size. In deep learning, a moderate batch size can speed up sample training [3]. Thus, the batch size is generally set. Namely, SGD is also equated to MBGD in the application scenario. The same

is true for the fractional order gradient descent optimizer in the paper (FCGD_G-L). Polyak introduced the concept of momentum [4], which was discussed in detail theoretically by Nesterov in the context of convex optimization [5]. The introduction of momentum in deep learning has long been shown to be beneficial for parameters' convergence [6]. It speeds up the convergence and prevents the parameters from falling into local optimal solutions. In addition, scholars have proposed some algorithms for adaptive learning rates. For example, AdaGrad proposed by Duchi [7], RMSProp proposed by Thieleman [8], and Adadelta proposed by Zeiler [9]; all of them use the current gradient state to change the learning rate or as a calibration for changes. Kingma proposed Adam [10] by combining the momentum method and the adaptive learning rate algorithm. Although the above algorithms and their improvements have their own characteristics, their gradients are based on the first-order derivative. Therefore, further development is limited.

As the research on fractional order gradient descent and deep learning optimization algorithms has intensified, more and more scholars have introduced fractional order calculus into deep learning optimization algorithms. Thus, it is possible for deep learning optimization algorithms to rely on fractional order derivation. Some scholars have achieved some good results on related research by exploiting the fractional order property. Under the convexity condition and Caputo, Li studied the convergence rate of different orders of GD, by jointly using the integer order and fractional order, parameters that finally converge to the integer order extreme value points [11]. By using Riemann–Liouville (R-L) and Caputo, Chen studied GD under convexity conditions and proposed a deformation's formula; the formula can converge quickly to integer order extreme value points [12]. By transforming R-L and looking for special initial parameters, Wang designed a deep learning optimization algorithm that can guarantee the same convergence result as the integer order under the convexity condition, and the related optimizer is validated using the MNIST dataset by experiments [13]. Yu designed a deep learning optimizer using G-L by setting the step size to two. Its current gradient is determined by the gradient in the past fixed size time window according to a specific weight [14]. Kan studied the deep learning optimization algorithm using G-L and validated the relevant optimizers using the MNIST dataset and the CIFAR-10 dataset with the inclusion of momentum, discussing the effect of different step sizes on the results [15]. Khan designed a deep learning optimizer using a power series of fractional order, which was applied to a recommender system with good results [16–18]. Due to the constraints of the fractional order power series, it is limited in the kinds of loss functions. In addition, the update of the parameters can only be kept in the positive range [19]. Constrained optimization problems have been studied by Yaghooti [20] using Caputo, and Viola [21] using R-L.

There are various fractional order definitions. The commonly used ones are the G-L fractional order definition, R-L fractional order definition, and Caputo fractional order definition [22]. SGD is rarely used directly in deep learning, but it is the basis for improved optimization algorithms. SGDM is an optimization algorithm with momentum [23]. AdaGrad, RMSProp and Adadelta are a class of optimization algorithms with an adaptive learning rate [24]. Adam is an optimization algorithm for combining momentum and adaptive learning rate property [25,26].

Based on the above discussion, FCGD_G-L is designed in this paper using the G-L fractional order definition. Its current α order gradient is obtained by summing the current first order gradient and the first order gradients of the past 10 time steps according to the fractional order property. Compared with the integer order, which can only add momentum and disturbances to the gradient descent, FCGD_G-L can add perturbations to its own derivation process to accelerate the descent and prevent falling into the local optimum solution. At the same time, the integer order needs additional momentum per iterative process, and this increases the computational workloads. Because of the fractional order property, the fractional order is equivalent to self-contained momentum. Thus, these computational workloads are eliminated in FCGD_G-L. The designed optimizer in this paper adds small perturbations to the fractional order derivation process; this

maximizes the ability of finding the global optimal solution while ensuring the fractional order properties. The major contributions of this paper are as follows:

1. In this paper, a novel deep learning optimizer is designed, written according to Pytorch documentation specification, with the same invocation methods as the existing optimizers of Pytorch, enriching the variety of optimizers.
2. Compared to other fractional order deep learning optimizers, the use of the G-L fractional order definition reduces the work of adding momentum and perturbation to the gradient at each iteration; thus, reducing the computational workload and improving the efficiency of the fractional order deep learning optimizer.
3. The new G-L fractional order definition uses improved Grünwald coefficients, avoiding the use of the Gamma function. In addition, it solves the problem that the previous fractional order optimizer is not perfectly compatible with the integer order.
4. In this paper, obtaining the global optimal solution is the best result. However, it is easy to fall into local optimal solutions during training. Thus, a constant factor c_j is added before each term of the G-L fractional order definition, and a small internal perturbation is added to the current time step by fine-tuning c_j, which can prevent the parameters from falling into the local optimal solutions well.
5. The deep learning algorithm in this paper provides a new way of thinking; by introducing fractional order, the optimizer adds a hyperparameter α. By adjusting α, the optimizer can be adapted to different application scenarios well, and a faster convergence speed and higher convergence accuracy can be obtained than the integer order.

The remainder of this paper is organized as follows. Section 2 introduces the G-L fractional order definition and SGD and its related improvement algorithm. Section 3 introduces the definition of fractional order gradient descent in this paper and gives the corresponding fractional order optimizer algorithms for SGD and Adam. Section 4 validates the optimizers of this paper on deep neural network models using two time series datasets, compares the corresponding integer order optimizers, analyzes the train loss of each optimizer's loss function, and evaluates the effectiveness of the resulting models on test sets. Section 5 summarizes FCGD_G-L, pointing out its shortcomings and future improvement directions.

2. Derivation of Fractional Calculus Gradient Descent Based on G-L Fractional Order

To better illustrate the optimizer in this paper, this section focuses on some of the basic concepts mentioned above.

2.1. The Definition of G-L Fractional Order

Definition 1. *The G-L fractional order definition is defined as follows* [22,27]:

$$ {}_{t_0}^{G}D_t^{\alpha} f(t) = \lim_{h \to 0} h^{-\alpha} \sum_{j=0}^{[(t-t_0)/h]} (-1)^j \frac{\Gamma(\alpha+1)}{\Gamma(j+1)\Gamma(\alpha-j+1)} f(t-jh) \tag{1} $$

where h is step size, and $[t_0, t]$ *is the upper and lower bound on the number of steps. The* α *is the order of the G-L fractional order definition.*

$$ w_j = (-1)^j \frac{\Gamma(\alpha+1)}{\Gamma(j+1)\Gamma(\alpha-j+1)} \tag{2} $$

Theorem 1. *The limit finding operation in Equation (1) can be neglected if the chosen computational step is small enough. The G-L fractional order definition can be written as follows* [22]:

$$ {}_{t_0}^{G}D_t^{\alpha} f(t) \approx h^{-\alpha} \sum_{j=1}^{[(t-t_0)/h]} (1 - \frac{\alpha+1}{j}) w_{j-1} f(t-jh) \tag{3} $$

Proof. We begin by proving Theorem 1:

With Equation (2), obviously, $w_0 = 1$. Thus, Equation (4) is given in the paper.

$$\begin{aligned} \frac{w_j}{w_{j-1}} &= \frac{-\Gamma(j)\Gamma(\alpha-j+2)}{\Gamma(j+1)\Gamma(\alpha-j+1)} \\ &= -\frac{\alpha+1-j}{j} \\ &= 1-\frac{\alpha+1}{j} \end{aligned} \tag{4}$$

Namely, Theorem 1 is proved.

Through Equation (4), the calculation of Equation (2) by the Gamma function can be avoided. Since Equation (4) does not need to compute double float [20], the computational efficiency and robustness of the algorithm are improved. □

2.2. Preliminaries Algorithms

Let a data set sample be n, the $f_i(x)$ is the loss function of the training samples with index i, and t is a variable of time. This means that the parameter x is iterated at the t time node, $i \subset n$, $x_0 = 0$. Accordingly, the following definitions are obtained:

Definition 2. *SGD's parameter update equation is as follows* [1]*:*

$$x_t \leftarrow x_{t-1} - \eta \nabla f_i(x_{t-1}) \tag{5}$$

*where η is the learning rate (*lr*), and $\nabla f_i(x_{t-1})$ is the gradient of $f_i(x_{t-1})$ at parameter x_{t-1}.*

The stochastic gradient method with momentum (SGDM) [3] is the accumulation of past historical gradients on top of the current gradient, in order to achieve faster convergence and prevent falling into local optima. SGDM is defined as follows:

Definition 3. *SGDM's parameter update equation is as follows* [4]*:*

$$\begin{aligned} v_t &\leftarrow \beta v_{t-1} + \nabla f_i(x_{t-1}), \\ x_t &\leftarrow x_{t-1} - \eta v_t. \end{aligned} \tag{6}$$

where β is the momentum coefficient, and v_t is the momentum, this accumulates past gradients to the current gradient to improve the descent speed and reduce the fluctuation of parameter updates. The default $\beta = 0.9$, $v_0 = 0$.

Adaptive learning rate algorithms are a class of deep learning optimization algorithms created by adjusting the learning rate and gradient according to their current state [2]. In addition, AdaGrad, RMSProp and Adadelta are the three main representatives, which are defined as follows:

Definition 4. *AdaGrad's parameter update equation is as follows* [7]*:*

$$\begin{aligned} s_t &\leftarrow s_{t-1} + \nabla {f_i}^2(x_{t-1}), \\ x_t &\leftarrow x_{t-1} - \frac{\eta}{\sqrt{s_t+\varepsilon}} \nabla f_i(x_{t-1}). \end{aligned} \tag{7}$$

where s_t is used to accumulate the variance of past gradients and then construct a different learning rate for each iteration, to optimize the process of iteration. The default $s_0 = 0$. $\varepsilon = 1e-8$, and the ε is to prevent the denominator from being 0.

Definition 5. *RMSProP's parameter update equation is as follows* [8]*:*

$$\begin{aligned} s_t &\leftarrow \gamma s_{t-1} + (1-\gamma) \nabla {f_i}^2(x_{t-1}), \\ x_t &\leftarrow x_{t-1} - \frac{\eta}{\sqrt{s_t+\varepsilon}} \nabla f_i(x_{t-1}). \end{aligned} \tag{8}$$

where γ is the weight coefficient of s.

Definition 6. *Adadelta's parameter update equation is as follows* [9]*:*

$$\begin{aligned} &g_t = \nabla f_i(x_{t-1}) \\ &s_t \leftarrow \rho s_{t-1} + (1-\rho) g_t^2, \\ &g_t' \leftarrow \frac{\sqrt{u_{t-1}+\varepsilon}}{\sqrt{s_t+\varepsilon}} g_t, \\ &u_t \leftarrow \rho u_{t-1} + (1-\rho) g_t'^2, \\ &x_t \leftarrow x_{t-1} - \eta g_t'. \end{aligned} \tag{9}$$

where u_t is the leaked average of g_t' with rescaled gradients. Together with s_t, it constructs a learning rate for g_t. The default $u_0 = 0$.

Adam's algorithm combines the advantages of an adaptive learning rate algorithm and momentum to achieve fast convergence. However, its gradient is easy to oscillate and the convergence accuracy is slightly poor. It is defined as follows.

Definition 7. *Adam's parameter update equation is as follows* [10]*:*

$$\begin{aligned} &g_t = \nabla f_i(x_{t-1}), \\ &v_t \leftarrow \beta_1 v_{t-1} + (1-\beta_1) g_t, \\ &s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2) g_t^2, \\ &\widehat{v}_t \leftarrow \frac{v_t}{1-\beta_1^t}, \\ &\widehat{s}_t \leftarrow \frac{s_t}{1-\beta_2^t}, \\ &g_t' \leftarrow \frac{\eta \widehat{v}_t}{\sqrt{\widehat{s}_t+\varepsilon}}, \\ &x_t \leftarrow x_{t-1} - g_t'. \end{aligned} \tag{10}$$

3. G-L Fractional Order Definition

The fractional order derivative formula of this paper is first given to illustrate its rationality. Its SGD algorithm and Adam's algorithm are also given in the paper.

3.1. G-L Fractional Order Definition of the Model

From the G-L fractional order definition of Equation (1), due to the characteristics of computers, it is clear that the algorithm cannot be expanded infinitely in practical calculations; therefore, a finite expansion is required. Some scholars have shown that, in neural networks, the expansion to the 10th term already characterizes the properties of fractional order derivatives well [15,28,29]. Let $q(j) = \frac{\Gamma(\alpha+1)}{\Gamma(j+1)\Gamma(\alpha-j+1)}$ and make a graph of the variation of $q(j)$ with $j \in \mathbb{Q}^+$ for Equation (2), as shown in Figure 1.

Figure 1 shows the curves of $\alpha = [0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.3, 1.5]$. When Equation (2) is expanded to the 10th term, the effect of the coefficients on the overall fractional order derivation becomes small. Therefore, the fractional order derivation in this paper only accumulates the past 10 time steps.

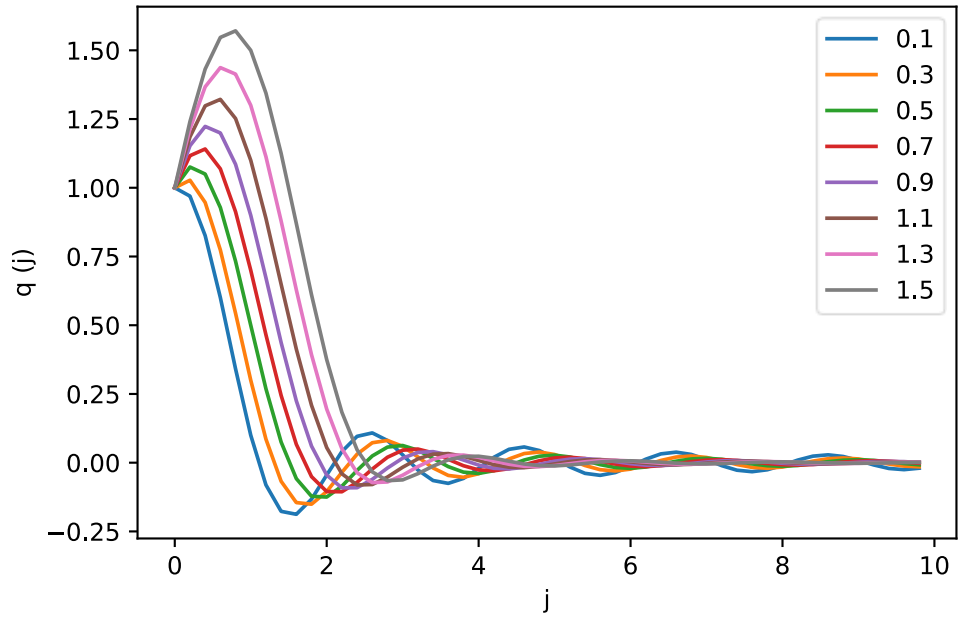

Figure 1. The variation of $q(j)$ with j.

On the other hand, the step size in Equation (1) is not a continuous value in the parameter update of the neural network [14,15], namely $h \in \mathbb{N}*$. In this paper, let the step size be the minimum value, namely $h = 1$, and according to Equations (3) and (4), a derivative equation is obtained that can be used for updating the parameters:

$$\begin{aligned} {}_a^G D_t^\alpha f(t) \approx \quad & h^{-\alpha} \sum_{j=0}^{[(t-a)/h]} w_j f(t-jh) \\ & \approx f(t) + \sum_{j=1}^{10} w_j f(t-j) \end{aligned}, \tag{11}$$

Equation (11) can eliminate the computation of the Gamma function and achieve the unification of the fractional order and integer order optimizers. In order to improve the ability of the algorithm to find the global optimal solution, a coefficient c_j is added before each accumulated term. At the same time, the probability of each coefficient having 0.9 is 1, and the probability of 0.1 is 0. To obtain Equation (12):

$${}_a^G D_t^\alpha f(t) = f(t) + \sum_{j=1}^{10} c_j w_j f(t-j) \tag{12}$$

According to Equation (12), when $\alpha = 0$, the algorithm degenerates to SGD without momentum. In order to make $\alpha = 1$ and the SGD equal, Equation (4) is improved further in the paper and obtains Equation (13):

$$w_0 = 1, w_j = (1 - \frac{\alpha + 1}{j + 1}) w_{j-1}, j = 1, 2, ...10. \tag{13}$$

By using Equation (13), when the order $\alpha = 1$, the fractional order derivative becomes the integer order derivative, and corresponds to SGD. Thus, the unification of the fractional order gradient descent and integer order gradient descent is achieved. In the original formula, $\alpha < 0$ denotes integral and $\alpha > 0$ denotes differential. Because of the transformation of Equation (4), $\alpha \leq 0$ in this paper also has good gradient descent capability, while retaining the fractional order property.

3.2. FCGD_G-L Algorithm

From Equations (12) and (13), an SGD based on FCGD_G-L and an Adam based on FCGD_G-L are proposed. In these two algorithms, the integer order derivation process becomes a fractional order derivation process. In addition, the extra momentum is no longer needed by taking advantage of the long memory property of the fractional order derivation.

The FCSGD_G-L Algorithm 1, which combines FCGD_G-L and SGD:

Algorithm 1: The SGD optimization Algorithm based on FCGD_G-L

Input: η(lr),x_0(params), $f(x)$(objective), λ(weight decay), α(order), c(disturbance coefficient), w(fractional coefficient)
Initialize: $x_0 = 0$, $g_0 = 0$,$c = [1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 0]$,$w_0 = 1$,$w_j = (1 - \frac{\alpha+1}{j+1})w_{j-1}$
$g_t = \nabla_x f_t(x_{t-1})$
for $t = 1$ to ... do
  $random.shuffle(c)$
  $${}_a^G D_{x_{t-1}}^{\alpha} f(x_{t-1}) = g_t + c[0]w_1 g_{t-1} + c[1]w_2 g_{t-2} + c[2]w_3 g_{t-3} + c[3]w_4 g_{t-4} + c[4]w_5 g_{t-5} + c[5]w_6 g_{t-6} + c[6]w_7 g_{t-7} + c[7]w_8 g_{t-8} + c[8]w_9 g_{t-9} + c[9]w_{10} g_{t-10}$$
  if $\lambda \neq 0$
    ${}_a^G D_{x_{t-1}}^{\alpha} f(x_{t-1}) \leftarrow {}_a^G D_{x_{t-1}}^{\alpha} f(x_{t-1}) + \lambda x_{t-1}$
  $x_t \leftarrow x_{t-1} - \eta {}_a^G D_{x_{t-1}}^{\alpha} f(x_{t-1})$
return x_t

In Algorithm 1, because fractional order derivatives are used, the algorithm adds a hyperparameter α that can adjust the order. In addition, the momentum and Nesterov are removed from the algorithm.

The FCAdam_G-L Algorithm 2, which combines FCGD_G-L and Adam:

Algorithm 2: The Adam optimization Algorithm based on FCGD_G-L

Input: η(lr), β_1, β_2(betas), x_0(params), $f(x)$(objective), λ(weight decay), α(order), c(disturbance coefficient), w(fractional coefficient)
Initialize: $m_0 = 0$(first moment), $v_0 = 0$(second moment), $\varepsilon = 1e-8$(eps), $x_0 = 0$, $g_0 = 0$,
$c = [1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 0]$, $w_0 = 1$, $w_j = (1 - \frac{\alpha+1}{j+1})w_{j-1}$
$g_t = \nabla_x f_t(x_{t-1})$
for $t = 1$ to ... do
  $random.shuffle(c)$
  $${}_a^G D_{x_{t-1}}^{\alpha} f(x_{t-1}) = g_t + c[0]w_1 g_{t-1} + c[1]w_2 g_{t-2} + c[2]w_3 g_{t-3} + c[3]w_4 g_{t-4} + c[4]w_5 g_{t-5} + c[5]w_6 g_{t-6} + c[6]w_7 g_{t-7} + c[7]w_8 g_{t-8} + c[8]w_9 g_{t-9} + c[9]w_{10} g_{t-10}$$
  if $\lambda \neq 0$
    ${}_a^G D_{x_{t-1}}^{\alpha} f(x_{t-1}) \leftarrow {}_a^G D_{x_{t-1}}^{\alpha} f(x_{t-1}) + \lambda x_{t-1}$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) {}_a^G D_{x_{t-1}}^{\alpha} f(x_{t-1})$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)[{}_a^G D_{x_{t-1}}^{\alpha} f(x_{t-1})]^2$
  $\widehat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
  $\widehat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$
  $x_t \leftarrow x_{t-1} - \frac{\eta \widehat{m}_t}{\sqrt{\widehat{v}_t} + \varepsilon}$
return x_t

In Algorithm 2, a hyperparameter α that can adjust the order is added.

The above two algorithms are based on FCGD_G-L and two classical gradient descent algorithms. They have the same time complexity as the original algorithm. Because of FCGD_G-L, the deep learning optimization algorithm becomes more flexible.

4. Experiment

In this section, two time series datasets are used to validate FCSGD_G-L and FCAdam_G-L. One of the datasets is the Dow Jones Industrial Average (DJIA), preprocessed with 24,298 rows of data, spanning from 3 February 1930 to 13 October 2022, with five dimensions: Open, High, Low, Volume and Close, predicting Close, and the training sets and test sets are cut according to 8:2 [30]. The other dataset is the Electricity Transformer dataset (ETTh1), with 17,420 rows of data, which has seven dimensions HUFL, HULL, MUFL, MULL, LUFL, LULL and OT, predicting OT, and the training sets and test sets are cut according to 8:2 [31].

The computer configuration for the experiment is as follows: CPU is an AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz; GPU is a RTX 3060 Laptop.

The selection criteria of the order are their fastest convergence speed and highest accuracy when training.

The neural network structure of the whole experiment consists of a three-layer LSTM [32] and two Linear, whose structure is shown in Figure 2.

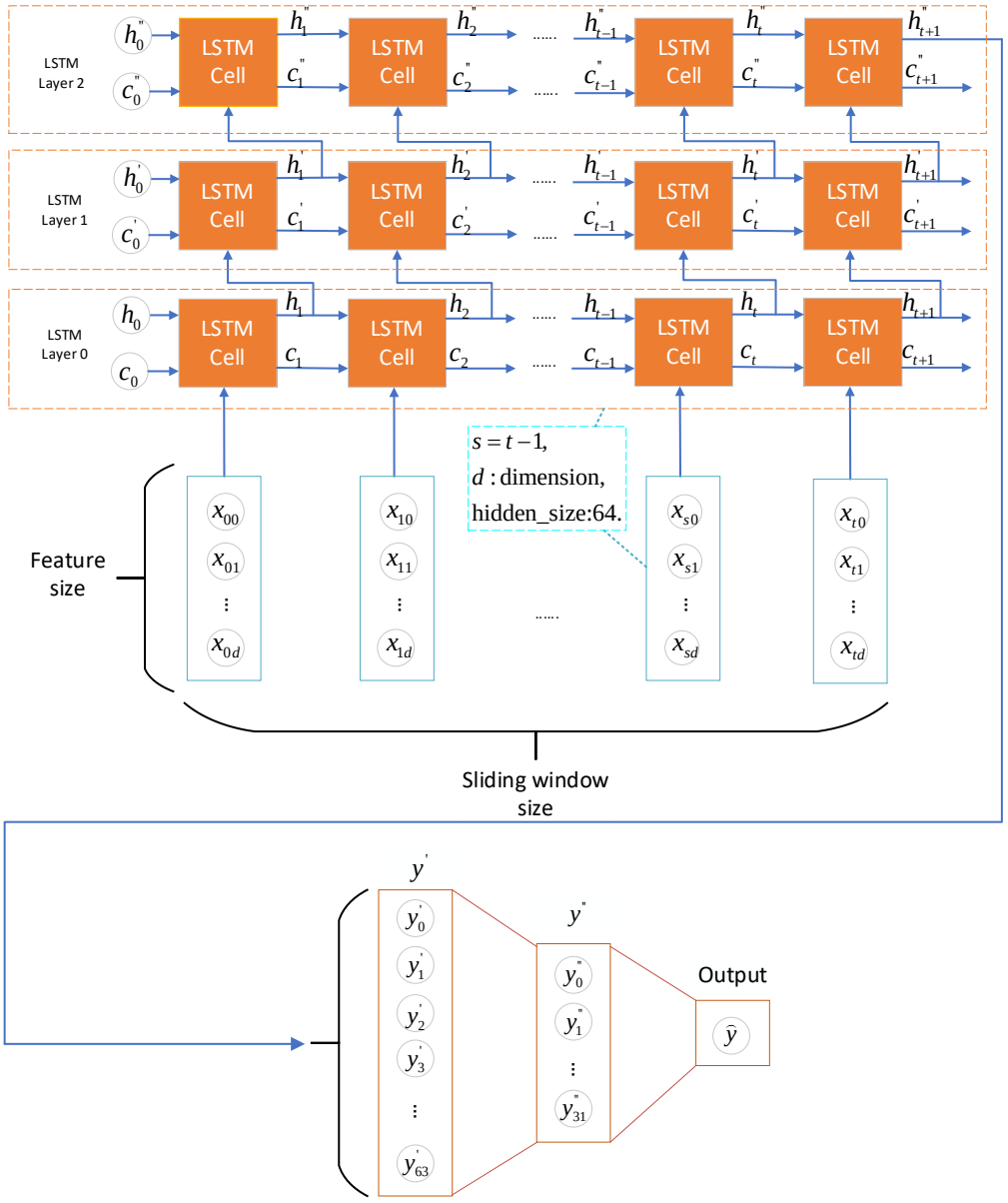

Figure 2. The neural network structure of this paper.

In Figure 2, let X be the size of each input sample; it is a matrix with rows of Feature size and columns of Sliding window size. The y' is a 64×1 vector. The y'' is a 32×1 vector. The $\widehat{y}$ is a predicted value; it is a scalar. As can be seen from Figure 2, X passes through the three-layer LSTM. After that, the h''_{t+1} is obtained on the last layer; it is equal to the y'. The y' is processed by the first Linear, and the y'' is obtained. Finally, the y'' is processed by the second Linear, and the $\widehat{y}$ is obtained.

4.1. Metrics

In this paper, the Mean Square Error (MSE), Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE) are used as evaluation metrics [32]. They are defined as follows:

Let n be the total number of samples, y_i is the true value, and $f(x_i)$ is the predicted value, and $\overline{y}$ is the sample mean. The following equation is obtained:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2 \tag{14}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2} \tag{15}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - f(x_i)| \tag{16}$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - f(x_i)}{y_i}\right| \tag{17}$$

4.2. Training for DJIA

In the training of DJIA, FCSGD_G-L and FCAdam_G-L are used as optimizers, respectively. In addition, the train loss and convergence accuracy of different optimizers are recorded. The hyperparameters are set as: $epoch = 250$; $weight\_deacy = 0$. The sliding window size is 30 and the batch size is 256. MSE is used as the loss function, and the lr of FCSGD_G-L during the training is set as in Equation (18):

$$lr = \begin{cases} 0.01, 0 \leq epoch < 50. \\ 0.005, 50 \leq epoch < 100. \\ 0.001, 100 \leq epoch < 150. \\ 0.0005, 150 \leq epoch < 200. \\ 0.0001, 200 \leq epoch < 250. \end{cases} \tag{18}$$

After 250 iterations, the train loss of different orders of FCSGD_G-L is shown in Figure 3.

Figure 3 shows the decreasing trend of train loss with epoch, where the FCSGD_G-L fractional order is -0.1, 0.0, 0.1, 0.2, 0.3, 0.4. From Figure 3, it can be seen that train loss of DJIA converges the fastest, and the convergence accuracy is the highest when the hyperparameter $\alpha = 0.1$ of FCSGD_G-L. Therefore, FCSGD_G-L with $\alpha = 0.1$ is used for comparison with SGD and SGD with $momentum = 0.9$ (SGDM); the other hyperparameters are default, as shown in Figure 4.

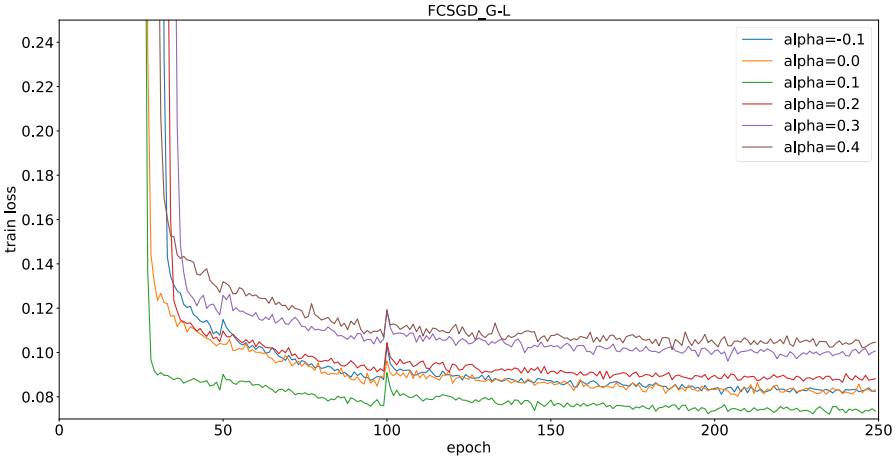

Figure 3. Train loss at DJIA by FCSGD_G-L.

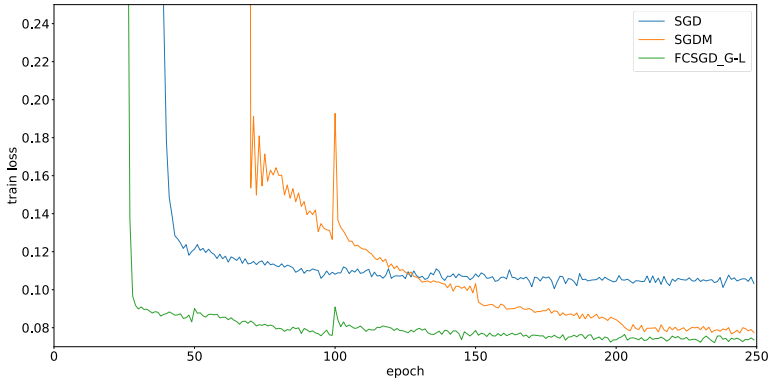

Figure 4. Train loss comparison at DJIA by SGD.

It can be seen from Figure 4, on DJIA, that the train loss convergence speed of FCSGD_G-L is faster than the SGD and the SGDM, and when $\alpha = 0.1$, FCSGD_G-L has a higher convergence accuracy.

When using FCAdam_G-L to train DJIA, due to the characteristics of Adam, the lr smaller than the SGD is selected in this paper to avoid divergence, and the other hyperparameters remain unchanged. The lr setting is as shown in Equation (19):

$$lr = \begin{cases} 0.0001, 0 \leq epoch < 50. \\ 0.00005, 50 \leq epoch < 100. \\ 0.00001, 100 \leq epoch < 150. \\ 0.000005, 150 \leq epoch < 200. \\ 0.000001, 200 \leq epoch < 250. \end{cases} \tag{19}$$

After 250 iterations, Figure 5 shows the decreasing trend of train loss with epoch, where the FCAdam_G-L fractional order is -0.1, 0.0, 0.1, 0.2, 0.3, 0.4.

As can be seen from Figure 5, when the hyperparameter $\alpha = 0.3$ of FCAdam_G-L, DJIA has the highest precision of train loss convergence, and the convergence rate of each order are roughly the same. Therefore, FCAdam_G-L with $\alpha = 0.3$ is used to compare with Adam, and the other hyperparameters are default, resulting in Figure 6.

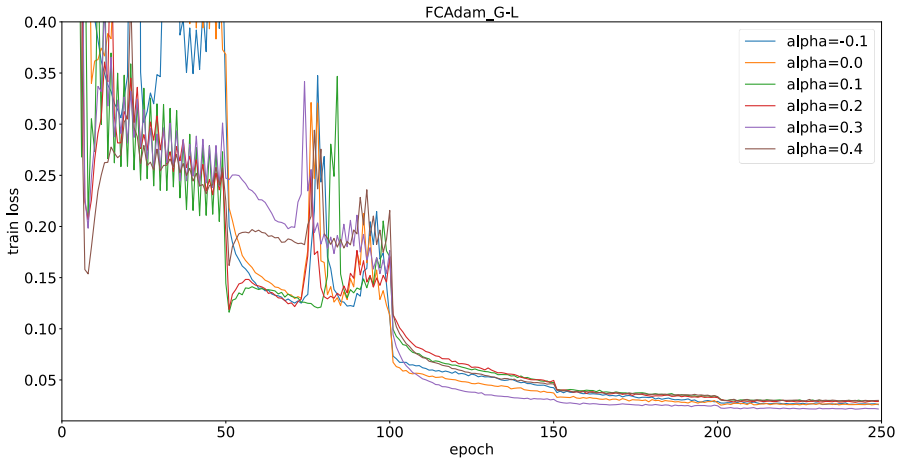

Figure 5. Train loss at DJIA by FCAdam_G-L.

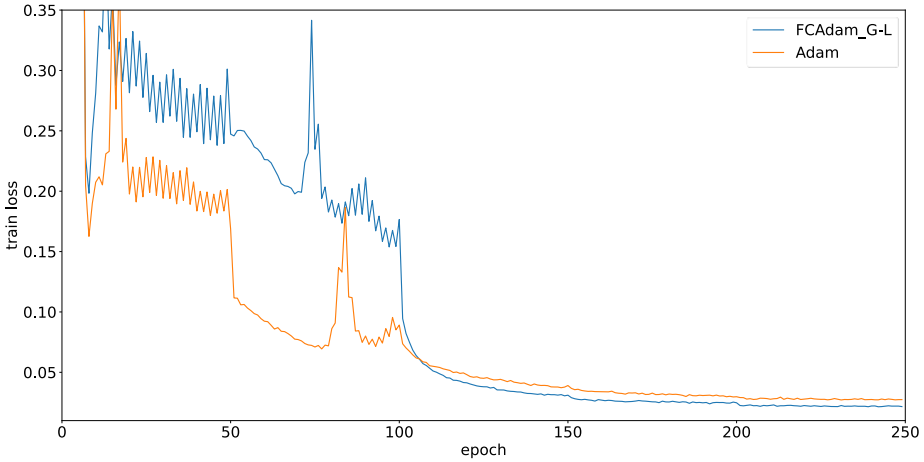

Figure 6. Train loss comparison at DJIA by Adam.

As can be seen from Figure 6, the train loss of FCAdam_G-L with $\alpha = 0.3$ demonstrates a higher convergence accuracy than Adam on DJIA. In terms of convergence speed, Adam and FCAdam_G-L with $\alpha = 0.3$ are the same; and combined with Figure 5, it can be seen that FCAdam_G-L and Adam converge at the same speed on DJIA.

4.3. Training for ETTh1

In the training of ETTh1, FCSGD_G-L and FCAdam_G-L are used as optimizers, respectively, and the train loss and convergence accuracy of different optimizers are recorded. The hyperparameters are set as: $epoch = 250$; $weight\_deacy = 0$. The sliding window size is 72 and the batch size is 256. MSE is used as the loss function. The lr of FCSGD_G-L during the training is set as in Equation (18), and the lr of FCAdam_G-L during the training is set as in Equation (19). After 250 iterations, Figure 7 shows the decreasing trend of train loss with epoch, where the FCSGD_G-L fractional order is -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0.0, 0.1, 0.2, 0.3, 0.4. Figure 8 shows the decreasing trend of train loss with epoch, where the FCAdam_G-L fractional order is -0.1, 0.0, 0.1, 0.2, 0.3, 0.4, 0.5.

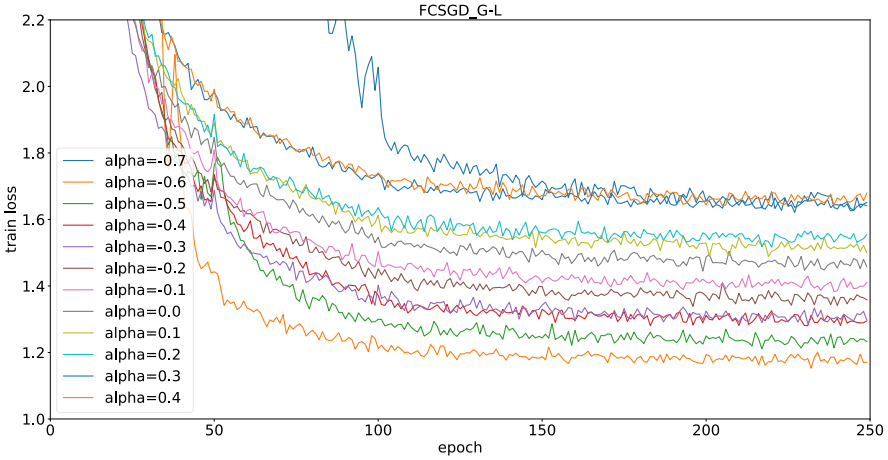

Figure 7. Train loss at ETTh1 by FCSGD_G-L.

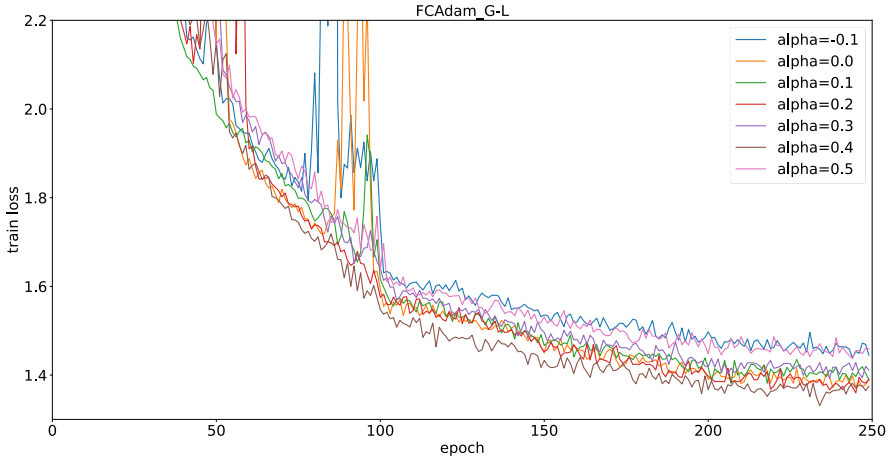

Figure 8. Train loss at ETTh1 by FCAdam_G-L.

In Figure 7, for ETTh1, FCSGD_G-L performs best when $\alpha = -0.6$, and its convergence speed and convergence accuracy reach the highest. In Figure 8, when $\alpha = 0.4$, FCSGD_G-L performs the best and its convergence accuracy reaches the highest. The two figures show roughly the same train loss descent domain. So, on ETTh1, the SGD, SGDM, Adam, FCSGD_G-L and FCAdam_G-L are compared together in this paper in Figure 9.

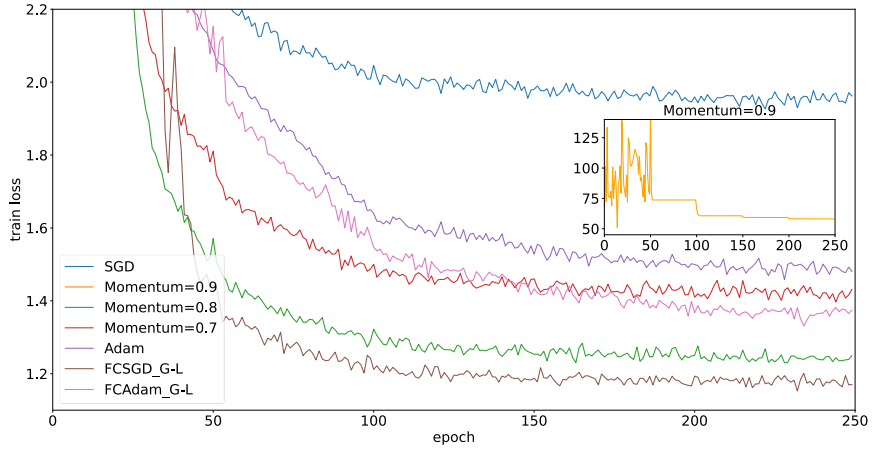

Figure 9. Train loss comparison concerning ETTh1.

In Figure 9, FCSGD_G-L with $\alpha = -0.6$ has the fastest decrease in train loss and the highest convergence accuracy. The SGDM with $momentum = 0.8$ also performs well, except that the convergence accuracy is worse than FCSGD_G-L, but at the default $momentum = 0.9$, the SGDM diverges without other hyperparameters being changed. FCGD_G-L in this paper is also essentially an algorithm with momentum, but as can be seen from Figures 7 and 8, FCGD_G-L on ETTh1 not only converges quickly and with high convergence accuracy, but is also robust and less likely to diverge. On ETTh1, both Adam and FCAdam_G-L perform poorly; however, using FCAdam_G-L is better than Adam. Among the various optimizers that achieve convergence, the SGD is the least effective, converges slowly, and has the lowest accuracy.

4.4. Evaluation of DJIA and ETTh1

Four evaluation metrics: MSE, RMSE, MAE, and MAPE were obtained by Equations (14)–(17). They are used in this paper to evaluate the effects of the two test sets. The results are recorded in Tables 1 and 2. Further, these metrics are used in order to compare the FCSGD_G-L correlation optimizer without considering the existing optimal network model. Further, the main hyperparameters are also initially set in Section 4, with the order and momentum settings based on the best results discussed above, namely, on DJIA, the lr is as in Equation (18), $momentum = 0.9$, $\alpha = 0.1$ for FCSGD_G-L and $\alpha = 0.3$ for FCAdam_G-L; on ETTh1, lr is as in Equation (19), $momentum = 0.8$, $\alpha = -0.6$ for FCSGD_G-L and $\alpha = 0.4$ for FCAdam_G-L. At the end of each epoch, the train loss is compared, the model with the smallest train loss is saved, and then the model is evaluated by using test sets.

Table 1. The best results for DJIA are bolded at MSE, RMSE, MAE, and MAPE.

Optimizer	MSE	RMSE	MAE	MAPE
SGD	0.0344	0.1855	0.1428	0.1825
SGDM	0.0313	0.1769	0.1363	0.1731
FCSGD_G-L	0.0270	0.1643	0.1268	0.1595
Adam	0.0055	0.0744	0.0564	0.0662
FCAdam_G-L	**0.0042**	0.0651	0.0502	0.0592

Table 2. The best results for ETTh1 are bolded at MSE, RMSE, MAE, and MAPE.

Optimizer	MSE	RMSE	MAE	MAPE
SGD	0.0167	0.1292	0.0984	0.7001
SGDM	0.0091	0.0953	0.0671	0.2882
FCSGD_G-L	0.0083	0.0910	0.0636	1.2089
Adam	0.0121	0.1102	0.0785	0.7112
FCAdam_G-L	0.0109	0.1044	0.0741	0.3513

In Table 1, due to the high volatility of DJIA, using the full test set is not effective and it is difficult to show the advantages and disadvantages of each optimizer. Therefore, only the first half of the test set of DJIA is used in this paper. It can be seen from Table 1, that the four metrics of FCAdam_G-L are the best among the five optimizers. For FCSGD_G-L, the results of the four metrics are all better than the SGD and the SGDM. This indicates that FCGD_G-L has obvious advantages in DJIA.

In Table 2, FCSGD_G-L's MSE, RMSE and MAE have the best results. For FCAdam_G-L, the results of the four metrics are all better than Adam. This indicates that FCGD_G-L has obvious advantages in ETTh1.

5. Conclusions

On DJIA and ETTh1, for the train loss of FCGD_G-L, its convergence speed and convergence accuracy exceed the corresponding integer order optimizer. In addition, the evaluation effect on test sets is also better than the corresponding integer order optimizer.

Taking advantage of the fractional order long memory property, FCGD_G-L does not need additional momentum, because it is equivalent to containing momentum inside. In addition, because of the properties of the G-L fractional order definition, the addition of perturbations becomes flexible in the iteration process. By using the transforming formula of the G-L fractional order definition, the Gamma function is removed in the paper. In addition, FCGD_G-L includes the integer order and the fractional order, thus achieving the unification of both.

Algorithm 1 and algorithm 2 make full use of the Autograd package of Pytorch to avoid the complicated derivation process in the complex neural network. The optimizers designed according to Algorithm 1 and algorithm 2 are very compatible with Pytorch. In Pytorch, our optimizer can be used just like any other existing optimizer. Using the order of the fractional order, we can fine-tune the results of the optimizer to obtain better convergence speed and convergence results. In Tables 1 and 2, the evaluation results on the test set also show better results than the integer order through adjusting the order.

In the foreseeable future, we will further explore the influence of the fractional calculus gradient descent on deep neural network, how to select the appropriate order quickly, and how to reduce hyperparameters. Eventually, it is also a significant research direction to make FCGD_G-L play a role in other fields.

Author Contributions: Supervision, C.Z. and Y.H.; Writing—original draft, X.Z.; Writing—review and editing, C.Z. and X.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Yunnan University Graduate Research Innovation Fund Project, grant numbers KC-22221166, and the National Natural Science Foundation of China, grant numbers 61862062 and 61104035.

Data Availability Statement: The input data files and codes, saved models, and evaluations are available at: https://github.com/zhouxiaojunlove/A-Deep-Learning-Optimizer-Based-on-Grunwald-Letnikov-Fractional-Order-Definition.git (13 November 2022).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Boyd, S.; Boyd, S.P.; Vandenberghe, L. *Convex Optimization*, 1st ed.; Cambridge University Press: New York, NY, USA, 2004.
2. Ruder, S. An overview of gradient descent optimization algorithms. *arXiv* **2017**, arXiv:1609.04747.
3. Shi, H.Z.; Yang, N.S.; Tang, H.; Yang, X. aSGD: Stochastic Gradient Descent with Adaptive Batch Size for Every Parameter. *Mathematics* **2022**, *10*, 863. [CrossRef]
4. Polyak, B.T. Some methods of speeding up the convergence of iteration methods. *Comput. Math. Math. Phys.* **1964**, *4*, 1–17. [CrossRef]
5. Nesterov, Y. *Lectures on Convex Optimization*, 2nd ed.; Springer: Cham, Switzerland, 2018; 137p.
6. Sutskever, I.; Martens, J.; Dahl, G.; Hinton, G. On the importance of initialization and momentum in deep learning. In Proceedings of the 30th International Conference on International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013.
7. Duchi, J.; Hazan, E.; Singer, Y. Adaptive Methods for Nonconvex Optimization. *J. Mach. Learn Res.* **2011**, *12*, 2121–2159.
8. Tieleman, T.; Hinton, G. Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of its Recent Magnitude. *COURSERA Neural Netw. Mach. Learn.* **2012**, *4*, 26–30.
9. Zeiler, M.D. Adadelta: An adaptive learning rate method. *arXiv* **2012**, arXiv:1212.5701.
10. Kingma, D.P.; Ba, J. Adam a method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
11. Li, J.W.; Shen, Y.J.; Yang, S.P. Variable fractional-order gradient descent method. *Shock. Vib.* **2021**, *40*, 43–47.
12. Chen, Y.Q.; Gao, Q.; Wei, Y.H.; Wang, Y. Study on fractional order gradient methods. *Appl. Math. Comput.* **2017**, *314*, 310–321. [CrossRef]
13. Wang, Y.; He, Y.L.; Zhu, Z.G. Study on fast speed fractional order gradient descent method and its application in neural networks. *Neurocomputing* **2022**, *489*, 366–376. [CrossRef]
14. Yu, Z.L.; Sun, G.H.; Lv, J.F. A fractional-order momentum optimization approach of deep neural networks. *Neural Comput. Appl.* **2022**, *34*, 7091–7111. [CrossRef]
15. Kan, T.; Gao, Z.; Yang, C.; Jian, J. Convolutional neural networks based on fractional-order momentum for parameter training. *Neurocomputing* **2021**, *449*, 85–99. [CrossRef]
16. Khan, Z.A.; Zubair, S.; Alquhayz, H.; Azeem, M.; Ditta, A. Design of Momentum Fractional Stochastic Gradient Descent for Recommender Systems. *IEEE Access* **2019**, *7*, 179575–179590. [CrossRef]

17. Khan, Z.A.; Zubair, S.; Chaudhary, N.I.; Raja, M.A.Z.; Khan, F.A.; Dedovic, N. Design of normalized fractional SGD computing paradigm for recommender systems. *Neural Comput. Appl.* **2020**, *32*, 10245–10262. [CrossRef]
18. Khan, Z.A.; Chaudhary, N.I.; Zubair, S. Fractional stochastic gradient descent for recommender systems. *Electron. Mark.* **2018**, *29*, 275–285. [CrossRef]
19. Lotfi, E.M.; Zine, H.; Torres, D.F.M.; Yousfi, N. The Power Fractional Calculus: First Definitions and Properties with Applications to Power Fractional Differential Equations. *Mathematics* **2022**, *10*, 3594. [CrossRef]
20. Yaghooti, B.; Hosseinzadeh, M. Constrained Control of Semilinear Fractional-Order Systems: Application in Drug Delivery Systems. In Proceedings of the 2020 IEEE Conference on Control Technology and Applications, Montréal, ON, Canada, 24–26 August 2020.
21. Viola, J.; Chen, Y.Q. A Fractional-Order On-Line Self Optimizing Control Framework and a Benchmark Control System Accelerated Using Fractional-Order Stochasticity. *Fractal Fract.* **2022**, *6*, 549. [CrossRef]
22. Xu, D.Y. *Fractional Calculus and Fractional-Order Control, 1st ed*; Science Press: Beijing, China, 2018.
23. Du, J. The Frontier of SGD and Its Variants in Machine Learning. In Proceedings of the 3rd International Conference on Machine Vision and Information Technology, Guangzhou, China, 22–24 February 2019.
24. Yuan, Y.; Hu, F.; Lu, L.F. A new non-adaptive optimization method: Stochastic gradient descent with momentum and difference. *Appl. Intell.* **2022**, *52*, 3939–3953. [CrossRef]
25. Zeng, K.; Liu, J.L.; Jiang, Z.X.; Xu, D. A Decreasing Scaling Transition Scheme from Adam to SGD. *Adv. Theory Simul.* **2022**, *5*, 2100599. [CrossRef]
26. Zhou, Y.F.; Zhang, M.C.; Zhu, J.L.; Zheng, R.; Wu, Q. A Randomized Block-Coordinate Adam online learning optimization algorithm. *Neural Comput. Appl.* **2020**, *32*, 12671–12684. [CrossRef]
27. Zhao, C.N.; Li, Y.S.; Lu, T. *Analysis and Control of Fractional Order Systems*, 1st ed.; National Defense Industry Press: Beijing, China, 2011.
28. Peng, Z.X.; Pu, Y.F. Convolution neural network face recognition based on fractional differential. *J. Sichuan Univ. Nat. Sci. Ed.* **2022**, *59*, 35–42.
29. Zhu, Z.G.; Li, A.; Wang, Y. Study on two-stage fractional order gradient descend method. In Proceedings of the 40th Chinese Control Conference, Shanghai, China, 26–28 July 2021.
30. Ratchagit, M.; Xu, H.L. A Two-Delay Combination Model for Stock Price Prediction. *Mathematics* **2022**, *10*, 3447. [CrossRef]
31. Zhou, H.Y.; Zhang, S.H.; Peng, J.Q.; Zhang, S.; Li, J.; Xiong, H.; Zhang, W. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. *arXiv* **2021**, arXiv:2012.07436. [CrossRef]
32. Jin, Y.C.; Wang, R.F.; Zhuang, X.D.; Wang, K.; Wang, H.; Wang, C.; Wang, X. Prediction of COVID-19 Data Using an ARIMA-LSTM Hybrid Forecast Model. *Mathematics* **2022**, *10*, 4001. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.