Journal of Computers Vol. 35 No. 3, June 2024, pp. 107-127
doi: 10.53106/199115992024063503009

Improved Gradient Descent Optimizer with Adaptive Parameter Setting and Swarm Intelligence

Xiang-Yu Deng*, You-Min Fan

College of Physics and Electronic Engineering, Northwest Normal University,
Lanzhou, 730070, China

dengxy000@126.com, fanyoumin1997@163.com

Received 4 October 2023; Revised 22 February 2024; Accepted 14 March 2024

Abstract. The gradient descent method suffers from the defects that the parameters must be set manually and easily fall into local minima in neural network training. This paper proposes an improved gradient descent method with adaptive parameter setting and global search capability. The proposed adaptive algorithm does not require any prior knowledge to obtain suitable parameters. Parameters such as learning rate are modified according to different training stages without human intervention. The structure of the gradient descent method is improved by using evolutionary computation, and the idea of lightweight populations is introduced into the method for global search and fast convergence. In addition, we introduce the meta-heuristic idea to make the swarm more intelligent and achieve better robustness. Experimental analysis shows that the proposed adaptive algorithm performs excellently with outstanding global search and generalization capabilities.

Keywords: gradient descent, self-adaptive mechanism, global search optimizer, population-based method, modified metaheuristic

1 Introduction

The past decades have witnessed the rise of artificial intelligence. Artificial neural networks [1] have been gradually applied in areas such as computer vision [2], industrial production [3], natural language processing [4], and so on. In recent years, various variants of neural networks have been proposed, such as AlexNet [5], GoogLeNet [6, 7], and ResNet [8]. They all employ a backpropagation training strategy and have achieved good results. The performance of neural networks is affected by the optimizer during the training process, whose task is to minimize the loss function by tuning the model parameters, thus facilitating model learning and optimization. Training a neural network is essentially an optimization problem to find the global optimal solution, and the optimizer gradually reduces the loss function by iteratively updating the model parameters. Choosing an appropriate optimizer significantly impacts the performance of neural networks, as different optimization algorithms have unique characteristics and are applicable to different scenarios. Therefore, researchers must explore and compare the performance attributes of various optimization algorithms in depth, considering data distribution, model complexity, and training time, to ensure that the selected optimizer can perform optimally in a given task.

As an optimizer, gradient descent is the most widely used optimization strategy in neural networks and an essential technique for convolutional neural networks [9]. The method of small batch stochastic gradient descent (MSGD) is one of the most widely used optimizers in neural networks [10]. Still, it is sensitive to parameter variations and slow to train [11]. Moreover, although the gradient descent method has good convergence properties and can converge quickly in the interval with a small resource footprint, its global search capability is relatively limited [22]. Hinton and Salakhutdinov [23] suggested that gradient descent can be used to fine-tune the weights, but it only works when the initial weights are close to a good solution. Y. Lecun et al. [24] suggested in their paper that gradient descent may fall into localized minima, thus affecting the system's quality. Therefore, improving the global optimization ability of the gradient descent method is an improvement direction of the gradient descent method.

Evolutionary computation (EC) [25], such as genetic algorithms (GA) [26], evolutionary strategies (ES) [27], and particle swarm optimization (PSO) [28], as one of the meta-heuristics [29], can achieve a more robust global search due to its use of swarm intelligence for parallel optimization [30]. With robustness, ease of implementa-

* Corresponding Author

tion, and no need for gradients, EC has been widely used in engineering, economics, manufacturing, marketing, finance, transportation, communication networks, and other fields [31-33].

Therefore, combining gradient-based algorithms with population has a broad research prospect. On the one hand, the gradient descent method has the advantage of faster convergence, but its global search ability is poor; on the other hand, the heuristic search method has a better global search ability, but its computational cost is more significant. The two search methods are complementary and have more excellent application prospects if combined and applied to the optimization problem. The contributions of this paper are as follows:

1) The proposed method has an adaptive mechanism and does not require artificial parameterization. First, our algorithm requires no a priori knowledge when initializing the parameters. Second, our method considers more factors, such as first-order derivatives, second-order derivatives, and historical information, which can produce an adaptive learning rate for use in gradient descent methods. Then, our algorithm can adjust the parameters in real time with training.

2) Combining the gradient descent method with a simplified population search strategy, an optimization strategy capable of global search is proposed. Centered on the current point, we generate sample points by mutation to search in parallel. We perform a fitness comparison of the loss values and filter the population for use in the next iteration, ultimately enabling global search. In addition, we improve the generation distribution of sample points with a modified meta-heuristic mechanism to enhance the method's robustness.

The rest of the paper is organized as follows. Chapter 2 presents several shortcomings of MSGD, describes the work of other scholars in the literature, and explains the strengths and differences of this paper compared to other work. Chapter 3 introduces the method's adaptive mechanism and analyzes each parameter's influence on the algorithm. Chapter 4 introduces the global optimization improvement of the technique and makes the process more stable by introducing the meta-heuristic idea. Chapter 5 verifies the effectiveness and robustness of the proposed method by analyzing the training in various functions. Compared with other methods, our method has the advantages of training with faster convergence, more stable training, and better global search capability. Chapter 6 summarizes the article and indicates future work.

2 Related Works

2.1 Stochastic Gradient Descent and Its Problems

Effect of Learning Rate on Training. The iterative equations of Gradient descent (GD) [40] are as (1), (2):

$$Grad_{GD}(w_n) = \frac{\partial L(w_n, \beta)}{\partial w_n} \ . \tag{1}$$

$$w_{n+1} = w_n - LR \times Grad_{GD}(w_n) \ . \tag{2}$$

ln (2), $Grad_{GD}(w_n)$ is the derivation of the objective function $L(w_n, \beta)$ to the current weight w_n. w_{n+1} is the weight after one iteration, *LR* is the learning rate, and $-LR \times Grad_{GD}(w_n)$ is the increment of one iteration. The learning rate is the only parameter to be set in this method and is very important in neural networks. Different learning rates are selected under the same function conditions, and the comparison effect is shown in Fig. 1.

In this figure, the training model is $f(x) = -Sa(x)$, the initial training point is $x = 4$, and the objective function is $f(x) = 1 - Sa(x)$. Comparing the training curve with $LR = 1$ and $LR = 2$, it can be seen that increasing the learning rate will increase the increment of each iteration, thereby speeding up the training. Comparing the training curve of $LR = 7$ and $LR = 8$, it can be seen that increasing the learning rate will make the oscillation range more significant when the function converges, thereby reducing the training accuracy. Through experiments and analyses, the following conclusions can be obtained in the gradient descent method: when the function is continuously differentiable, there is a point that satisfies the condition of Equation (3).

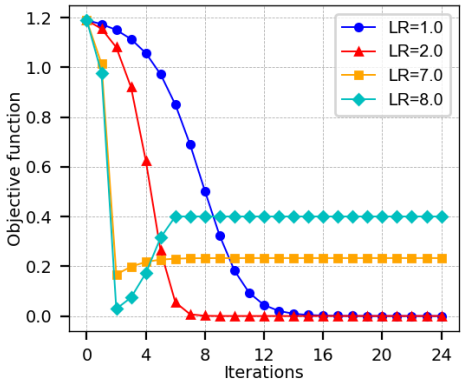

Fig. 1. Training process curves with different learning rates

$$f'(x_n) = -f'(x_n - LR \times f'(x_n)) \quad . \tag{3}$$

The meaning of (3) is that when training under a continuously derivable model, the first derivative of the training point after one iteration and the first derivative of the current training point are the inverse of each other when training converges. In (3), $x_n - LR \times f'(x_n)$ is the training point after one iteration equal to x_{n+1}. Part $LR \times f'(x_n)$ in (3) is the increment of one iteration, which is also the range of oscillation when the function converges, and it is related to the learning rate LR and the current training point x_n. However, x_n depends on the specific function conditions, therefore the choice of the learning rate affects the training accuracy of the gradient descent method. The selection of the learning rate needs to adapt to different situations, and it directly determines the speed and accuracy of the gradient descent. Although some methods reduce the learning rate according to the number of iterations to improve accuracy [41], the change is unidirectional. If the learning rate is reduced untimely, it will affect the training speed [42].

Because of the poor stability of stochastic gradient descent (SGD) [43], it is often combined with batch gradient descent (BGD) to form a highly stable and faster mini-batch stochastic gradient descent method (MSGD) [10]. However, MSGD has the following problems.

Affection of Random Range on Training. The generation range of random sample points in MSGD is manually set, and it will have different effects on the training process of MSGD. The comparison effects are shown in Fig. 2.

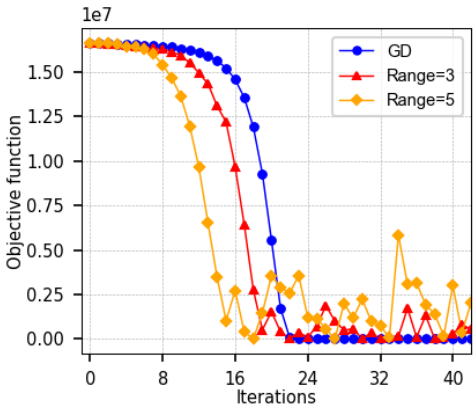

Fig. 2. The effect of different random ranges on the training process

The model for training is $f(x) = (x^3 - x - 4080)^2$, and the initial training point is $x = 2$, where the function is relatively flat, which can better reflect the effect of different random ranges. Comparing the curves in the early training phase, increasing the random range can make the objective function enter the descending phase earlier. The curve in the descending phase shows that increasing the random range does not significantly speed up the descent of the objective function. Comparing the curves after converging to the minimum, it can be found that increasing the random range will increase the fluctuation of the objective function in the late training period. Therefore, in the MSGD method, increasing the random range can make the training response faster, but a vast random range will lead to poor convergence in the later stage of function training. Therefore, in our algorithm, it is essential to make full use of randomness while ensuring stability as much as possible.

Oscillations in the Late Training Period. MSGD is faster than GD but oscillates continuously after convergence, as shown in Fig. 2. The reason is shown in Fig. 3. In this figure, the training model is $f(w_n) = -Sa(w_n)$, which has a global optimum when $w_n = 0$. The current training point is $w_n = 5.5$, which belongs to the convergence interval of the local minima. MSGD generates five random sample points w_n^i within the random range.

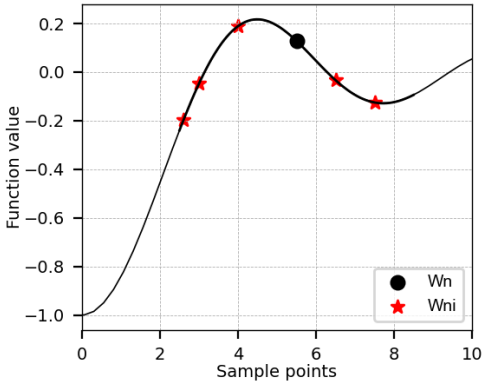

Fig. 3. Reversed iteration direction with MSGD

It can be seen that using MSGD, $Grad_{MSGD} > 0$ here by analyzing the random sample points. Consequently $-LR \times Grad_{MSGD}(w_n) < 0$, and w_n will decreases after one iteration, increasing the objective function. However, when using GD for training, the weights will increase after one iteration, reducing the objective function and converging to local minima. Therefore, when $Grad_{MSGD}$ and $Grad_{GD}$ are positive and negative, the next iteration will make the objective function not decrease but increase. The training is accelerated due to randomness, but randomness has the side effect of making the training oscillate in the later period. This oscillation will accumulate through the various layers of the network and have an impact on training. Therefore, one of our improvement goals is to avoid oscillation after convergence.

Easily Trapped in Local Minima. Although MSGD has the mechanism of randomness, it has a great possibility of falling into local minima. Fig. 3 shows a case in which MSGD jumps out of the local minima. When using GD, w_n will fall into local minima after training. However, as shown in Fig. 3, when MSGD is used for training, the increment is less than 0, and the weight will be decreased. Therefore, the weights can enter the global optima's convergence interval after training. However, this is a small probability case because the parameters need to be set properly and should be suitable w_n and w_n^i Therefore, the ability of MSGD to jump out of local minima is not ideal, and it is hard to jump out when converging to local minima with a large convergence interval, such as a saddle point.

The momentum gradient descent method (CM) [44] was proposed to solve the problem that MSGD can easily fall into local minima. CM adds a momentum term based on MSGD, and its increment is the weighted sum of the previous iteration increment and the gradient obtained by MSGD. The oscillations on the short axis are effectively reduced, and the contribution on the long axis is increased [45]. When training near the local minima, CM does not converge and oscillate like GD but continues to rise against the ramp by inertia. When the range of local minima is small, CM can jump out of local minima. However, due to the inertia of CM, the convergence speed becomes slower when trained near the minimum, and it needs repeated oscillations to converge.

The Nesterov accelerated gradient (NAG) [46] was proposed to make CM more intelligent. NAG replaces the reference point of the MSGD part in CM with an imaginary point after training with GD, making the acceleration or deceleration of NAG more predictable. Therefore, NAG can start faster at the beginning of training and converge faster when training near local minima. The improvement of NAG is a double-edged sword. On the one hand, the improvement can make the training more quickly accumulate energy and better skip the local minima. On the other hand, its convergence becomes faster, and its ability to jump out of local minima worsens. Therefore, one of the innovations of this paper is to find a good idea to make the improved optimizer have global optimization ability and to speed up the training while causing the training no longer oscillate.

2.2 Improvements in Gradient Descent by Scholars

To improve the optimization performance of the gradient descent method, scholars have introduced various adaptive mechanisms in neural network optimizers. A commonly used gradient descent algorithm that adaptively adjusts the learning rate is Adagrad, which uses a different learning rate for each component, improving generalization performance [12]. RMSProp reduces storage costs and gradually improves adaptive performance as iterations are added [13]. Adadelta [14] combines Adagrad and RMSProp. Adam uses higher-order information and combines the idea of moment estimation to achieve good adaptive performance [15]. It has recently been widely used in neural network training, but its global optimization-seeking performance needs to be improved. In recent decades, various novel improvements have been proposed with good results. Senior et al. [16] proposed a more accurate and robust AdaDec method based on the Adagrad method, which separates the long-term learning rate scheduling from the learning rate variation of each parameter and achieves higher frame accuracy. George and Powell [17] improved convergence speed by directly reducing the step size and achieved good training accuracy. Roux and Fitzgibbon [18] combine the gradient mean square error with second-order information to enable the method to achieve higher robustness. Schaul et al. [19] adaptively vary the learning rate according to the function distribution by analyzing the gradient expectation leveling method and the gradient leveling method expectation. Wu et al. [20] propose an adaptive learning rate-tuning strategy using categorical confidence and variance. Bordes et al. [21] propose an SGD-QN method that utilizes second-order information and reduces resource consumption.

2.3 Studies of Combining Gradient Descent with Evolutionary Computation

Shubham Gupta et al. [34] modified the population-based meta-heuristic algorithm Sine Cosine Algorithm (SCA) and generated populations of different polarities for global optimization. Heuristic algorithms require a large amount of computational resources due to groupthink, which reduces the training speed in practical applications. As the number of iterations increases, the number of individuals close to the optimum increases, and screening the population is also a problem that needs to be solved [35]. Polakova et al. [36] adaptively reduce or increase the population size during the search process, which improves the algorithm efficiency.

Heuristic algorithms have proved to be an effective strategy for use in optimization. In recent years, various scholars have often combined them with gradient descent, which is capable of exact optimization and has achieved better results. Seyedali et al. [37] combined PSO with the Gravity Search Algorithm (GSA), giving their method the advantages of fast convergence and avoiding local minima. Gianni et al. [38] took advantage of the GD's ability to refine the local solutions and use it as a more favorable starting point in genetic algorithms to propose a GGA algorithm capable of global search. Ahmadianfar et al. [39] utilized the idea of heuristic algorithms to explore the search space using a Local Escape Operator (LEO) in combination with gradient descent, which allows the gradient descent method to eliminate local minima.

2.4 Work on This Paper

In this paper, we discuss the shortcomings of the gradient descent method, such as the problems that the parameters need to be set artificially and it is difficult to jump out of the local minima, as well as the disadvantages of the evolutionary computation, such as the sizeable computational loss and the lack of accuracy. To overcome these problems, we propose a gradient descent method that incorporates the population idea. The method uses the population idea to optimize multiple individuals, thus overcoming the problem that a single individual can easily fall into local minima. Meanwhile, the method still adopts the gradient descent method for parameter updating, which retains the advantages of the gradient descent method. Compared with the traditional evolutionary computation method, the method has less computational loss and higher accuracy. By reasonably combining both advantages, our method shows better performance in optimization.

The proposed adaptive algorithm includes several adaptive modules. First, an adaptive range is generated based on the historical information of the function and the characteristics of the current point, and the range is used to generate random sample points for training. The adaptive range is used to create the adaptive variance to measure the function's volatility. Next, the adaptive learning rate for training is generated by combining the adaptive variance, the historical information of the function, and the first-order and second-order derivatives of the current point. We propose various adaptive algorithms to generate adaptive range, variance, and learning rate. These adaptive mechanisms can initialize the parameters without prior knowledge and automatically adjust the parameters according to different training states without human intervention. In addition, our proposed learning rate not only accelerates the convergence of early training states but also improves the accuracy of later training. At the same time, the adaptive variance reflects the fluctuations of different perspectives.

In addition, our method introduces the idea of EC to modify the structure of MSGD, which achieves better global optimization ability and late stability. During the training process, each sample point is independently trained multiple times and then sorted according to the loss function to select the best sample point for the next training iteration, thus enhancing the ability of the algorithm to jump out of local minima. In addition, the distribution of generated sample points is improved by introducing the idea of meta-heuristic algorithm, which enables the global search method to utilize the search space better and enhances the algorithm's robustness.

Different from others' methods, firstly, our method not only has an adaptive mechanism during local searching, but secondly, during global searching, our process is centered on the gradient descent method and introduces the idea of the simplified population by reconfiguring the parts of the MSGD method to mimic the operations in the genetic algorithm, and then, introduces the concept of meta-heuristic algorithms, so that the searching will be more stable.

3 Design of Self-adaptive Mechanism

3.1 Design of Adaptive Performance for Learning Rate

For convenience of presentation and expression, we illustrate for the time being under the condition of unitary functions. We want the increment to be a manageable size per iteration for training stability and adjustability. Therefore, we make improvements based on the method of determining increments, as shown in (4), (5):

$$LR = 1/\left|f'\right| . \tag{4}$$

$$Delta = LR \times f' = \pm 1 . \tag{5}$$

In Equation (4), f' is the first derivative of the unary function, and the increment $Delta$ of one iteration of the gradient descent method using this method is determined. On this basis, we hope that the learning rate will not only not decrease too fast in the early stage of training to increase the training speed but also adaptively decrease in the late stage of training to increase the training accuracy.

Some scholars used the annealing method to reduce the learning rate gradually, but this reduction needs to be accurately adjusted according to the training process. We decided to reflect the training process by the first derivative because it tends to be numerically large when away from the minimum and small when approaching

the minimum. The introduced first-order lead components' input-output characteristics must satisfy the following conditions. The input-output characteristics of Equation (4) are not ideal, and we hope that the learning rate does not tend to infinity when the first derivative is close to 0 because this is not conducive to convergence. For large values of the first derivative, we want the learning rate to increase to speed up training, but not too much, as the increment of iterations is too large. The process from decreasing to increasing the learning rate is smooth and linear. We choose the following improvements as alternatives. The comparison is shown in Fig. 4.

$$LR = \left(e^{|f'|} - 1\right) \Big/ |f'| \ . \tag{6}$$

$$LR = \arctan\left(|f'|\right) \Big/ |f'| \ . \tag{7}$$

$$LR = \log_2\left(1 + |f'|\right) \Big/ |f'| \ . \tag{8}$$

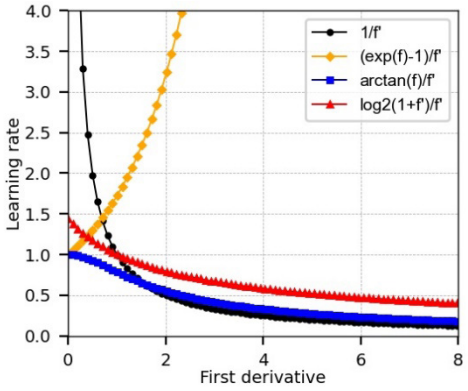

Fig. 4. Alternatives of learning rate

From Fig. 4, we can obtain the following conclusions. Observing the curve of Equation (6), it can be seen that when the first derivative is greater than 1, the learning rate is overly amplified. This is an undesirable option as our entire neural network will run into errors due to the learning rate being too high. Observing the curve of Equation (7), it can be seen that although the first derivative is small, the learning rate is reduced to increase the training accuracy. However, when the first derivative gradually increases, its acceleration effect on training is not apparent due to the limited amplitude of the function. By observing the curve of Equation (8), it can be seen that the effect of increasing accuracy is better when the first derivative is small, and the acceleration effect of training is ideal with the increase of the first derivative. In addition, when the first derivative is close to 0, the learning rate approaches the value $\log_2 10$ to prevent itself from going to infinity. This reduces the increment when it approaches the minimum and makes the function converge more accurately. Besides, compared with the trigonometric function, the logarithm has a minor time complexity of the algorithm, so with the increase of the input, the computational load will not be too large, which is convenient for further improvement.

3.2 Design of Parameters for Self-adaptive Mechanism

Parameters that Reflect Fluctuation Degree. In order to set the learning rate more accurately and reasonably, various scholars have proposed ideas for improvement. MSGD averages the gradients of the points in the batch to make the learning rate better act on the population during training. However, it tends to oscillate after convergence. The NAG method introduces the idea of momentum and inertia and performs hypothetical training during iteration, which has a certain acceleration effect on training. The Adagrad method uses different gradients for different components and increases the computational pressure of the system. Adam method introduces the idea

of momentum and second moment estimation, which converges fast and is more robust. Roux and Fitzgibbon combine gradient mean squared error and second-order information to improve robustness. Schaul et al. reduce the dependence of the learning rate on the function distribution by analyzing the squared norm of the gradient expectation and the expectation of the squared norm of the gradient. The SGD-QN method utilizes second-order information to reduce resource usage. Many of the above methods combine high-order information and try to obtain more information through the population.

As an improvement, our algorithm absorbs the advantages of the above methods and considers many elements when generating parameters. Not only the first derivatives but also historical information, neighborhood variance, and second derivatives. Based on Equation (8), our method adds a part of the fluctuation degree. As the degree of fluctuation increases, we may be at the minimum, and we need to decrease the learning rate to avoid crossing the minimum because the increment is too large. Therefore, the improvement of Equation (8) is as follows:

$$LR = \frac{\log_{(2+FlucDegree)}(1+|f'|)}{|f'|+FlucDegree} \quad . \tag{9}$$

$$Ratio = \frac{\log_{(2+FlucDegree)}(1+|f'|)/(|f'|+FlucDegree)}{\log_2(1+|f'|)/|f'|} \quad . \tag{10}$$

$Ratio$ is obtained by dividing Equation (9) by Equation (8). By fixing other parameters, we analyze the effect of $FlucDegree$ in Equation (9). The influence of the fluctuation degree is shown in Fig. 5.

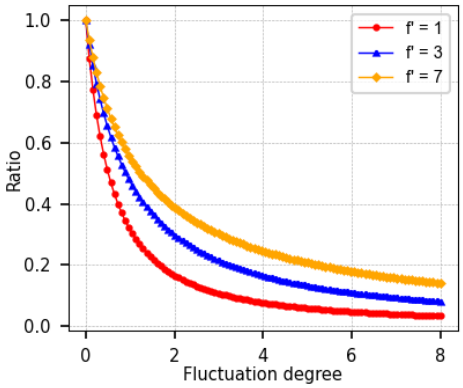

Fig. 5. Effect of fluctuation degree

The curves in Fig. 5 show that when the first derivative is fixed, the adaptive learning rate will decrease with the increase of the degree of fluctuation. With the same degree of fluctuation, the learning rate will decrease with the gradient decreases. With this improvement, the learning rate can adaptively decrease as the degree of fluctuation increases to avoid stepping over the minimum and retain the function of increasing accuracy with the training process.

We define the degree of fluctuation through the following two aspects. The first is the curvature of the current point of the function because the higher the curvature of the current point, the greater the tendency of the function's first derivative to change. Secondly, sampling is carried out in the range around the current point, the objective function value of the sampling point is recorded, and then the variance of the difference is calculated to determine whether the sampling point is distributed in the same convergence interval. When the sampling points are distributed in different convergence intervals, they tend to have large fluctuations, and their variance is also large. When the sampling points are distributed in a relatively flat interval, the fluctuation of $[D]$ is also tiny, and its variance is slight. The specific method is as follows:

$$Cur = \left|f''\right| / (1 + f'^2)^{3/2} \quad . \tag{11}$$

$$w_n^i \sim U(w_n - Range, w_n + Range) \quad . \tag{12}$$

$$[Loss]_k = [Loss(w_n^1), Loss(w_n^2), \cdots, Loss(w_n^k)] \quad . \tag{13}$$

$$[D]_{k-1} = [Loss(w_n^2) - Loss(w_n^1), Loss(w_n^3) - Loss(w_n^2), \cdots, Loss(w_n^k) - Loss(w_n^{k-1})] \quad . \tag{14}$$

$$Var = \frac{1}{N}\sum_{i=1}^{k-1}[D(w_n^i) - \frac{1}{N}\sum_{i=1}^{k-1}D(w_n^i)]^2 \quad . \tag{15}$$

$$AdaVar = \log_{(2\times Range)}(1 + Var) \quad . \tag{16}$$

$$LR = \frac{\log_{(2+Cur+Var)}(1 + \left|f'\right|)}{\left|f'\right| \times (1 + Cur + Var)} \quad . \tag{17}$$

In (16), $Range$ is the adaptive neighborhood used to generate random sample points. $Var \in [0, +\infty)$ and varies with the order of magnitude of the function. The proposed method introduces the logarithmic function and uses its characteristics to limit the output of larger values. When the function is relatively flat and Var is small, $AdaVar$ outputs Var linearly. When the function fluctuates drastically and Var is large, $AdaVar$ outputs Var by limiting.

Parameters to Generate Random Points. Finally, we have one last parameter to define: the range of samples to get. When the function is close to the minimum, $Range$ will be accompanied by a decrease in the gradient and the function value. Therefore, it is necessary to obtain information from current and past training points. Many methods with adaptive mechanisms combine historical gradient information to obtain the relative information between the current training value and the value that has appeared in the past and achieve good results.

Our method is combined with the current training point's gradient, objective function, and historical information when obtaining the adaptive neighbor range. The historical information will reflect whether the function is converged and measure the relative degree of fluctuation. The adaptive range needs to be adaptively increased when the function is close to the minimum, which is used in the subsequent improvement to enhance its ability to jump out of the local minima. The acquisition method is shown as follows:

$$R_b = 1 + \left|f'\right| / MaxGrad \quad . \tag{18}$$

$$R_r = 1 + (\left|f\right| + MaxLoss) / MaxLoss \quad . \tag{19}$$

$$Range = \log_2(1 + \log_{R_b} R_r) \quad . \tag{20}$$

In the above method of setting $Range$, f is the value of the objective function, f' is the first derivative of the function, $MaxLoss$ is the maximum value of the objective function value that has occurred, $MaxGrad$ records the maximum value of the absolute value of the first derivative that has occurred, and $Range$ is the adaptive neighbor range.

In (18), f' is the variate, and $MaxGrad$ is a constant. It is easy to get $\left|f'\right| / MaxGrad \in (0,1]$ and $R_b \in (1,2]$. $\left|f'\right| / MaxGrad$ reflects the training progress from the gradient aspect. When approaching the minimum, $\left|f'\right|$ will close to 0, $\left|f'\right| / MaxGrad$ will decrease, and R_b will close to 1. When R_r does not change, $\log_{R_b} R_r$ in-

creases, and *Range* increases accordingly. Therefore, when the function is close to the minimum, *Range* will increase to enhance the ability to jump out of local minima.

In (19), f is the input, and *MaxLoss* is a constant item, the range of $(|f|+MaxLoss)/MaxLoss$ is [1,2] and the range of R_r is [2,3]. $(|f|+MaxLoss)/MaxLoss$ reflects the training progress from the aspect of the objective function value. The farther the current training point is from the minimum, the more likely that $|f|$ is big. And as $|f|$ increases, $(|f|+MaxLoss)/MaxLoss$ approaches 2, and R_r is closer to 3. When R_b remains unchanged, $\log_{R_b} R_r$ increases, and *Range* increases accordingly. So, when the function is far from the minimum, the random range does not change significantly.

In (20), $R_b \in (1,2]$, $R_r \in [2,3]$, therefore $\log_{R_b} R_r \in [1,+\infty)$, $Range \in [1,+\infty)$. When $\log_{R_b} R_r$ is close to 1, it means that R_r is close to 2 and R_b is close to 2. It can be deduced that the first derivative of the function is larger at this time, and the function value is close to the *MaxLoss* at this time. The possibility of being close to the minimum is small, so there is no need to modify *Range*. When $\log_{R_b} R_r$ is close to +∞, it means that R_b is close to 1, and $|f'|$ is close to 0. It can be inferred that the function is close to the minimum at this time, and *Range* needs to be increased.

Considering $\log_{R_b} R_r$ as a whole input item, (6) can be simplified to $Range = \log_2(1+x)$. According to the characteristics of the logarithmic function, when the input range is (0,1), the function have good linearity. When the input range is $[1,+\infty]$, the algorithm can limit the output. This method is used for different situation functions, and the effect is shown in Fig. 6.

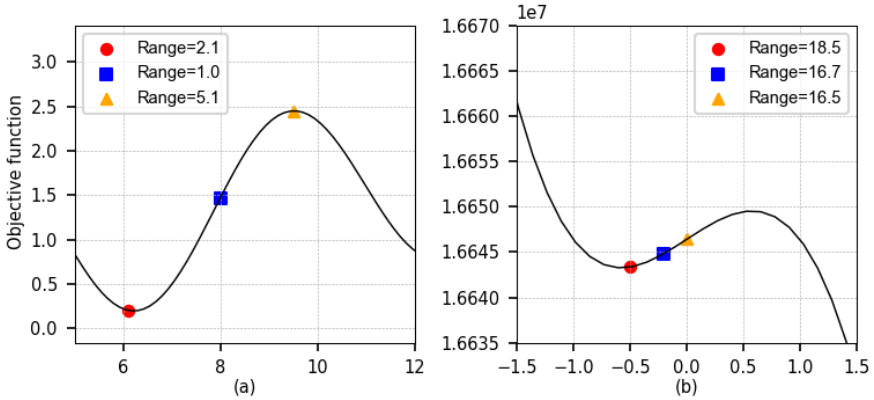

Fig. 6. Effect of adaptive range algorithm

The training model in subfigure (a) is the *Girewank* function, and the training model in subfigure (b) is $f(x) = (x^3 - x - 4080)^2$. By observing subgraph (a), it can be seen that when this method is applied to the function with a smaller order of magnitude span, *Range* increases accordingly as the function approaches the minimum. This is because the function value will not be very large, $|f|$ does not change significantly relative to *MaxLoss*, while $|f'|$ near the minimum changes significantly relative to *MaxGrad*. At this time, *Range* is more determined by the first derivatives of the function. By observing subgraph (b), it can be seen that when this method is applied to the function with larger order of magnitude span, *Range* increases accordingly as the function approaches both the local and global minimum. Because the function fluctuates drastically around the minimum, $|f'|$ does not change significantly relative to *MaxGrad*. However, $|f|$ near the minimum changes significantly relative to *MaxLoss*. At this time, *Range* is more determined by the function values.

It can be seen that when the method of generating the adaptive range proposed in this section is applied to different functions, the *Range* can be adaptively increased when close to the minimum to enhance the ability to jump out of the local minima and can cover the convergence interval of the global optima. In addition, *MaxLoss* and *MaxGrad* will be continuously updated with iterations, and the adaptive range will increase. Even if the order of magnitude of the function is large, the adaptive neighbor range will be a manageable size and will have strong usability.

4 Improved Global Search Method with Swarm Intelligence

4.1 Swarm-based Gradient Method

Due to the population's intelligence, EC and its derived algorithms can realize global search, such as Evolutionary Strategies (ES) and Particle Swarm Optimization (PSO). EC has the searching strategy of gradient-free searching, parallel search, and survival of the fittest [47]. The training process of EC includes population initialization, individual fitness, selection, and mutation. ES that simulates biological evolution and uses Gaussian variation to generate offspring is proposed by Schwefel [48]. Different from GA, the method of selection uses a definite strategy. Through selection, the superior individuals are selected from the population, and the individuals who do not meet the target conditions are eliminated. Avoiding selecting individuals that converge to the same minimum is critical, and a suitable selective strategy is the key.

Inspired by bird foraging behavior, PSO utilizes individual training and swarm cooperation to find the minimum solution [49]. PSO is unidirectional, and the distribution of all points will only become more and more concentrated with iteration. If the convergence fails to reach the global optima when the process is over, the process cannot restart again. Therefore, one-way training should also be avoided when drawing on PSO.

Gradient descent and population-based optimization methods have complementary advantages. Therefore, combining the gradient descent method in neural networks with lightweight metaheuristics has excellent research prospects. Our method absorbs the idea of swarm cooperation to improve the structure of EC and combines it with the training process of MSGD to produce a population-based gradient method. We introduced swarm intelligence to solve the problem of later-period oscillations. On the one hand, compared with the population strategy of metaheuristics, the gradient descent method can make full use of the characteristics of the current point to achieve fast convergence. On the other hand, the population idea of metaheuristics can overcome the drawback that gradient descent is difficult to overcome local minima. We introduce the concept of lightweight population into each iteration of the gradient descent method, which can solve the problem that particle swarm optimization can not restart and the calculation loss is manageable. A finite number of random individuals are generated at each iteration and trained separately, corresponding to population initialization. The value of the loss function after training random individuals corresponds to the individual fitness. Sorting the individuals according to the value of the loss function and selecting the minimum value for the next iteration is the selection strategy of the improved method.

In the first iteration, our method regards all sample points A as the center, and each center generates $N-1$ random sample points w_n^i within range $[w_n - Range_n, w_n + Range_n]$. Then, combined with the idea of population, the operation of the sample points in the batch is no longer the average of the gradients like MSGD, but all sample points are considered individually and equally trained with the modified gradient descent method with the self-adaptive mechanism. Therefore, the swarm size after the first iteration is N. Ranking the result pairs and deduplication of the swarm with similar position and training results. This is to avoid multiple training points belonging to the same convergence interval and reduce computational consumption. Then, keep the best K individuals as the swarm of nest iteration. Besides, in order to avoid random sample points with farther distances being discarded prematurely, w_n^i is trained for *i* times so that it has a greater chance to obtain better results than w_n and makes better use of randomness. By the way, *K* should be no more than *N*.

In the next iteration, take *K* w_{n+1}^i as the center, and each center generates *N* random sample points w_{n+1}^{ij} independently within range $[w_{n+1}^i - Range_{n+1}^i, w_{n+1}^i + Range_{n+1}^i]$, and use w_{n+1}^{ij} to train. Therefore, there are $K \times N$ result pairs in the first iteration, and the swarm size is $K \times N$. The best *K* groups of result pairs are saved as the

next swarm after reduplicating. Using the next swarm to train likewise. Do this cycle training.

Because some functions have drastic changes in the value of the objective function and the gradient near the global optima, therefore, the traditional method that uses the accurate function value or gradient to terminate training may only sometimes be sufficiently trained. Our method records the w_n value of the most recent T times of training and finds its variance. When the variance is smaller than a certain threshold for T times in a row, it can be considered that the function has been fully trained, and the training is terminated.

Unlike other adaptive algorithms, our method uses information in the swarm to adjust the self-adaptive mechanism. The global variables $MaxLoss$ and $MaxGrad$ are introduced to better reflect the situation that the current value is close to the minimum. These two items are updated through the population-based method and used in (4) and (5) to obtain the adaptive neighbor range and other parameters. The swarm becomes intelligent because of its indirect impact. Unlike other global search method, our method can arrange and save the data after one iteration, and can obtain sample points randomly again in the next training, restarting the process of multi-point training. Unlike MSGD, the random neighbor range is adaptively set and dynamic; the training points are not directly used to obtain the average gradient but are compared and deduplicated as a swarm after training, increasing population availability. The flowchart of the global search method is shown in Fig. 7.

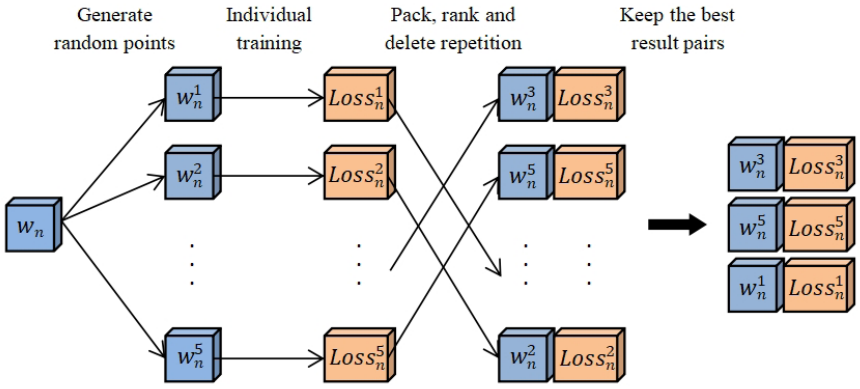

Fig. 7. Flowchart of global search method

4.2 Improvement of Swarm Randomness with Metaheuristic

Sometimes, the current training points in our method will jump to the point far from the global optima after an iteration, and it is hard to return to the interval convergence of the global optima with the training process. Fig. 8 illustrates the cause of this problem. In Fig. 8, the training result of black point w_n is not better than the results of red points w_n^i. If GD is used for training, the current training point w_n will converge to the global optima. However, due to the randomness, the results of other sample points in the swarm after training may be better than those of w_n. Because the method keeps the best K result pairs, if the training result pair of w_n is not included in the best K result pairs, sample points that converge to the global optima will be discarded. As a result, the current training point will jump to a position far from the global optima after one iteration.

In Fig. 8, the training model is $f(x) = -Sa(x)$. w_n is the current training point, which belongs to the convergence interval of the global optima. w_n^i is random sample points generated by this method, which belongs to the convergence interval of local minima and is far from the global optima. After one iteration, assume it is in the convergence interval of the local minima, which is far from the global optima. At this time, in range $[w_{n+1} - Range_{n+1}, w_{n+1} + Range_{n+1}]$, the convergence interval of the global optima is minimal compared to the

whole range or is not included in the entire range, so it isn't easy to return to the convergence interval of the global optima through randomness. So, after jumping to the convergence interval of a local minima far from the global optima, the training point may have difficulty returning to the convergence interval of the global optima.

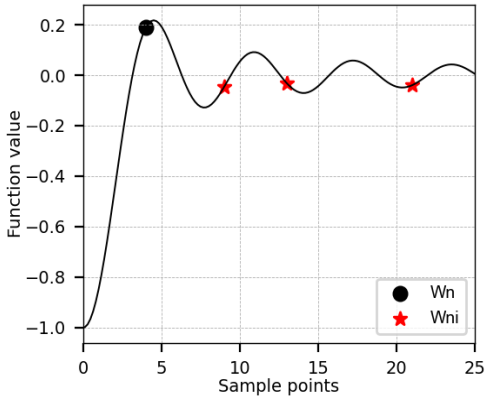

Fig. 8. Cause of unsteadiness

This kind of problem will scale up exponentially under the bivariate function model. Therefore, improving the probability distribution in the random range is necessary, making the probability near the current training point more significant and the probability far away smaller. Gaussian variation, which ES uses to generate offspring, is worth reference. Metaheuristics strives to explore the whole search space and use adequate information as much as possible. Gaussian distribution meets the improvement requirements of exploration. It is because, in Gaussian distribution, the probability within $[w_n - Range_n, w_n + Range_n]$ is 0. 682, the probability in $[w_n - 2\times Range_n, w_n - Range_n] \bigcup [w_n + Range_n, w_n + 2\times Range_n]$ is 0. 272, and it still has the probability to search the other search space. Besides, $i \times Range_n$ was used to change the variance of the Gaussian distribution so that random points may be generated both near and far. By training w_n^i for i times, the randomness is better used to meet the exploitation requirements. The population generation method is improved by using Gaussian distribution with $Range$. The specific method is shown in Equation (21). The range of i is $[0, N-1]$, and when $i = 0$, the first random point is w_n. The specific effect is shown in Fig. 9.

$$w_n^i \sim N(w_n, i \times Range_n) \ . \tag{21}$$

In Fig. 9, subgraph (a) shows the distribution of random sample points without metaheuristic. Subfigure (b) shows the distribution of random sample points with the idea of metaheuristic. It can be seen that the sample points (x_n^i, y_n^i) in the swarm randomly obtained by our method conform to the Gaussian distribution and belong to different convergence intervals. By modifying metaheuristic, when i in (14) $w_n^i \sim N(w_n, i \times Range_n)$ is small, our method prone to generate nearby points and train them with less iteration, when i in (14) $w_n^i \sim N(w_n, i \times Range_n)$ is large, our method prone to generate points far away from current (x_n, y_n) and train them with more iteration. Compare the objective function of the current training point (x_n, y_n) after training once with the objective function value of other random points after training. If the training results of other sample points are better than the current point, the current point will be updated to the result point of the corresponding random point after the iteration. Besides, by deduplicating the operation, nearby sample points after training have a higher possibility of being abandoned. Therefore, our method is not easy to fall into local minima, but it has an excellent ability to find global optima. This meets our expectations for improvement.

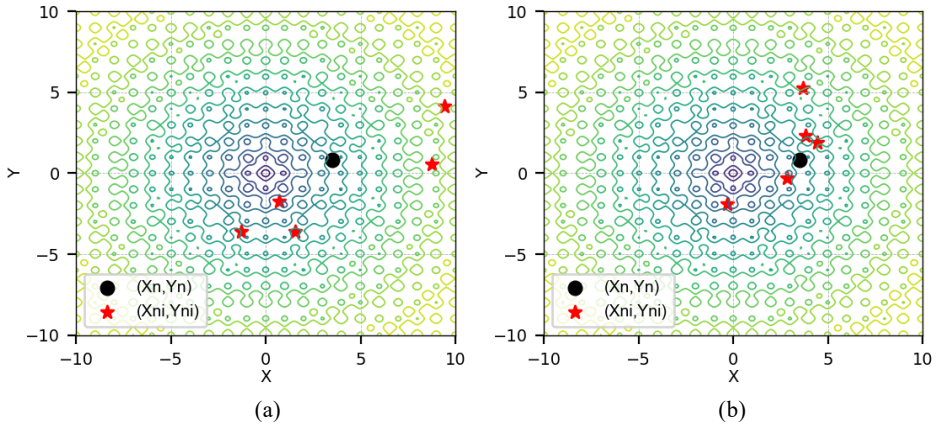

Fig. 9. Effects of metaheuristics

5 Experimental Simulation and Validation

Since the work of the article consists of two parts, the validation is also divided into two parts. The first part is the improvement of the adaptive algorithm, where the reasonableness of the learning rate generated at this point is first analyzed by observing the learning rate generated at the crucial point of the particular function about the function. The overall effectiveness of the adaptive algorithm in training is then verified through a complete training process. The second part is the improvement of the global optimization search. Firstly, experiments are carried out under the unitary and binary functions to verify the advantages of the method proposed in this paper by comparing it with other commonly used methods. Then, the generalization ability of the proposed method in this paper is illustrated through experiments under different functions and given Errorbar functions. Finally, as a complement, the method proposed in this paper is applied to neural networks, and the applicability of the proposed method is illustrated by comparing the loss functions of other functions.

5.1 Validation of Adaptive Learning Rate

Learning Rate Analysis of Function Keypoints. The adaptive performance of the algorithm proposed in this paper is first justified by analyzing the adaptive performance of the algorithm and by observing the effect of the change in the analysis function on the learning rate. The training model in subfigure (a) is $f(x) = -Sa(x) + 1$, with a minor function of the order of magnitude and a smoother convergence interval for its global minima. The training results are shown in Fig. 10, and the training model in subfigure (b) is $f(x) = (x^3 - x - 4080)^2$, with a large order of magnitude of the function and considerable fluctuations in the convergence interval of its global minima. These two functions are more typical of the one-dimensional functions used to test adaptive learning rates.

Observing Fig. 10 reveals that the adaptive algorithm has several highlights: First, the learning rate generated by the adaptive algorithm decreases adaptively as the training points gradually approach the local minima, with good followability. Second, the resulting learning rate is applicable under different order of magnitude conditions, ensuring that each training iteration increment is not too large or too small when training using gradient descent. Third, the process that generates the learning rate is one in which there is no input a priori knowledge, the algorithm derives all intermediate variables of the algorithm, and the algorithm's tuning of the parameters is real-time and independent.

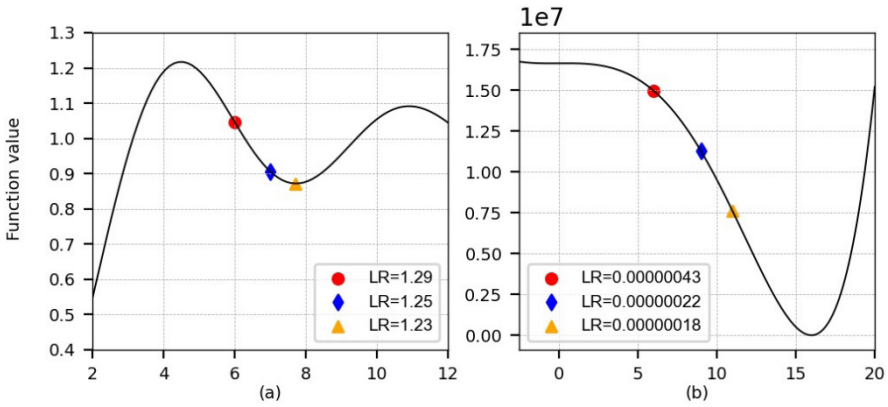

Fig. 10. Adaptive learning rate of training points

Observing the points drawn with diamonds and triangles in subfigure (a), the loss function values of these two points are relatively close to each other due to the smaller order of magnitude of the function. Still, the gap between the gradients is also amplified by the smaller order of magnitude. At this point, the component changes of the gradients in the adaptive algorithm are more pronounced, affecting the generation of the learning rate to a greater extent. Observing the points drawn with diamonds and triangles in subfigure (b), the gap between the loss functions of these two points is more pronounced due to the larger order of magnitude of the function, while the gap in the loss functions is weakened due to the larger order of magnitude of the function, at which point the relationship between the loss functions the learning rate of the generation of the impact is more significant.

After experimentation and analysis, the proposed adaptive algorithm can generate appropriate learning rates according to different functions with good generalization performance, both for functions with a small and a large span of variation. It can also adaptively reduce the learning rate as the training gradually approaches the local minima, thus increasing the training accuracy.

Analysis through Training. Because the population-based optimization method can accelerate the training by selective operation, it will cause interference when experimentally verifying the adaptive learning rate algorithm. So, we temporarily remove the global optimization mechanism in the proposed method and only validate the effectiveness of the adaptive learning rate algorithm in training. The superiority of the adaptive algorithm proposed in this paper can be highlighted by comparing it with other methods. The global optimum of the test model must have a large convergence interval to ensure the test method can converge. In addition, the chosen model needs to be more sensitive to the learning rate to facilitate screening for a more appropriate learning rate. The model used in the experiment is $f(x) = (x^3 - x - 4080)^2$, and the initial point is $x = 3$. The chosen comparison test function should be able to adjust the learning rate or increment adaptively. The NAG can tune increments and introduce an improved inertia mechanism for faster convergence. RMSProp can adaptively adjust the learning rate with several pre-set parameters. In addition, the original method GD was used as one of the comparison test functions. The experimental results are shown in Fig. 11.

The curves show that the proposed adaptive algorithm can achieve faster convergence without any prior knowledge compared to RMSProp, which requires manual tuning of the parameters for different situations. Compared with the NAG method, although it does not start as fast as NAG, the proposed adaptive mechanism can increase the learning rate at the beginning of training to speed up the training while avoiding the disadvantage of NAG, which requires oscillations to consume inertia. Compared with GD, the proposed algorithm can flexibly adjust the learning rate in real-time, which makes up for the defect that GD adjusts the learning rate many times by experience.

After experimental comparison and analysis, the adaptive algorithm proposed in this paper has the following advantages: First, the proposed adaptive algorithm can satisfy the needs of each training phase by adaptively increasing or decreasing the learning rate. The learning rate is increased at the early stage of training to speed up the training. At the later stage of training, the proposed method can narrow down the increment by decreasing the

learning rate to improve the training accuracy. Second, the whole process is free of human intervention and does not require a priori knowledge, which saves the cost of use and has good generalization performance.

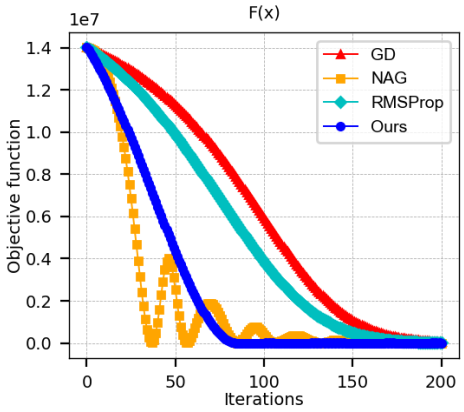

Fig. 11. Performance of adaptive algorithm

5.2 Validation of Global Search Ability

The Girewank [50] and the AckLey [51] functions are commonly used multivariate optimization test functions for performance testing of EC genetic methods [52]. They are continuous and have ample search space. The function has many local minima near the global minimum, training under these models will inevitably fall into local minima. This dramatically increases the optimization algorithm's difficulty finding the global minimum, making them ideal test functions in experiments testing global optimization capabilities. In the experiment, the Girewank function is chosen as a one-dimensional test function, and the Ackley function is selected as a binary test function to compare the performance of each optimization algorithm from different dimensions. The test functions are shown in Fig. 12.

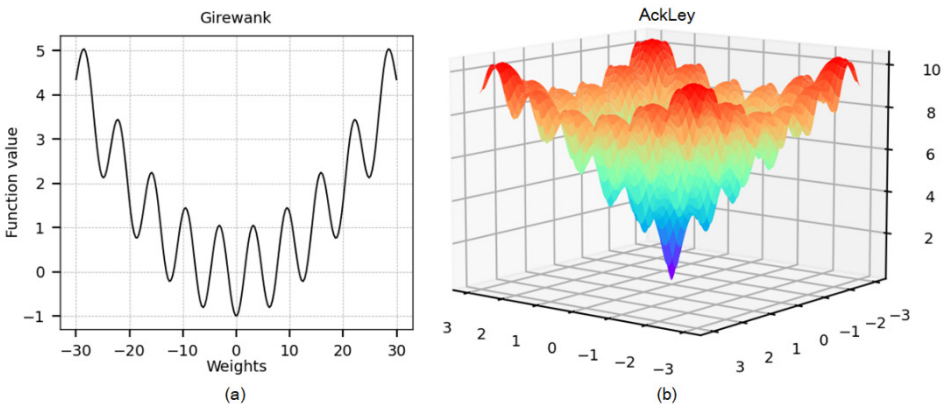

Fig. 12. Testing functions

The results of the tests compared to the MSGD, NAG, and RMSProp methods are shown in Fig. 13. Subgraph (a) starts at $w_n = 22$, and subgraph (b) starts at $(9,9)$, which belongs to the interval of convergence of local minima. From subfigure (a), it can be seen that the MSGD method is slower to train and needs to set the appropriate

batch size and learning rate, and the effect will only improve with the increase of labor cost. The NAG method can continue to train against the trend even after training into local minima by inertia and eventually crosses the local minima. However, this method still has the disadvantage of slower convergence at a later stage. Although both NAG and MSGD can jump out of local minima, they are not very effective, and their objective functions even rise in the later stages of training. Although the RMSProp method can generate a suitable learning rate and converge to the local minima relatively quickly after training, it does not have the ability to jump out of the local minima and the loss function fluctuates significantly when the training is carried out about 15 times. In comparison, the method proposed in the article has the following advantages. First, the intelligent adaptive algorithms allow more appropriate parameters to be set and modified in real-time without any a priori knowledge. Second, by introducing a population-based gradient method and taking full advantage of stochasticity, the proposed method can continuously jump out of the local minima and eventually enter the convergence interval of the global minima. Finally, the proposed method can avoid oscillations in the late stage of training by sorting and repetition. The method proposed in this paper can converge to the global optimum quickly and stably and does not oscillate in the late training period. The method proposed in this paper has more robust optimization capability and faster optimization speed than other algorithms.

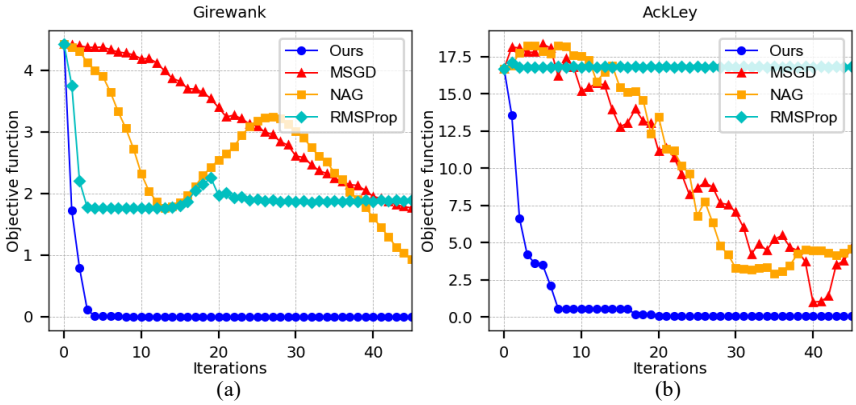

Fig. 13. Performance of global search ability

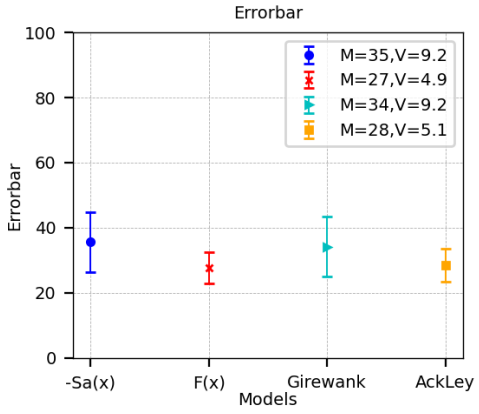

Fig. 14. Performance of generalization ability

Four functions are chosen as training models to demonstrate the generalization ability of this method through experiments. The reason for choosing the $f(x) = -Sa(x) + 1$, is that when training under this function, the train-

ing points tend to randomize to places far away from the global minima, are difficult to return to, and eventually fall into the local minima, for the reasons described in Chapter 4. The use of this function for training reflects the contribution of the work we did in Chapter 4 on the stability improvement of the global optimization algorithm, and it can also be used as a function for the global optimization ability test. $f(x) = (x^3 - x - 4080)^2$, as a representative of a larger order of magnitude unitary function, is a better representation of the effectiveness of our adaptive algorithm. The reason for using the Girewank and Ackley functions as binary functions for training is because of the complex overall shape of the Girewank and Ackley functions, which can be represented better among the standard test functions, thus allowing the observer to make a better judgment of the training. Fig. 14 shows the error bar function, where the height of the middle point of the bar is the average number of times the training converged to the global optimum point. The length of the bar is the standard deviation of the training convergence to the global optimum point, which shows the performance of the training applied to the different training models. It can be observed that when proposed algorithm is applied to different function models, the number of iterative steps required for convergence is more stable, and the training speed is faster with good generalization performance.

5.3 Validation in Neural Network

The computational burden is increased because the proposed method needs to compute and store multiple data sets for each iteration. Therefore, lightweight needs to be achieved when this approach is applied to neural networks. First, second-order derivatives are used to approximate the curvature, and the method of obtaining the second-order derivatives is improved from the more computationally intensive Hessian matrices to first-order derivative inner products to model the second-order derivatives. Second, K in Section 4 can be set to 1, which reduces the storage and computational pressure while fully preserving the algorithmic mechanism. By terminating training early, the proposed method can speed up training overall. For the validation of the neural network, a handwriting recognition neural network is used to demonstrate the effectiveness of the proposed algorithm quickly and clearly. It is specialized to recognize Arabic numerals 0 to 9. It has an input size of $20 \times 20 \times 3$, with ten classification results through two convolutional layers and three linear layers. The MSGD method and the Adam method were compared, and the experimental results are shown in Fig. 15.

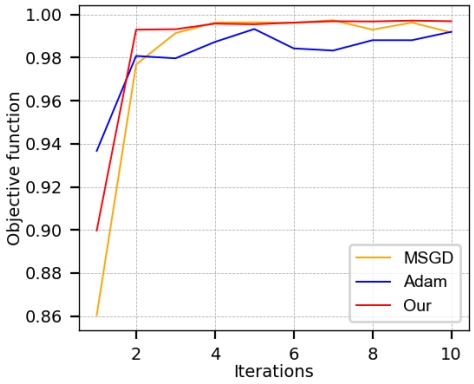

Fig. 15. Performance in neural network

From the figure, although the accuracy of the proposed method is lower than the other methods in the first iteration due to the random setting of the weights, the accuracy is better than the other methods after one training. This indicates that the proposed method starts faster and trains faster when the loss function is higher. At the late stage of training, the proposed method is more stable compared to the MSGD method and compared to the Adam method, and the proposed method is more accurate, as can be obtained from the accuracy curve.

The effectiveness of the method's adaptive learning rate, convergence speed, and ability to jump out of local minima are verified through the above three-part experimental comparison. However, the proposed method still suffers from training overfitting and thus needs further modification to avoid overfitting in neural networks. In addition, the proposed method needs to be adapted and improved in neural networks and more scenarios later to realize batch applications.

6 Conclusion

In this paper, we give an improved gradient descent optimizer with adaptive parameter setting and global search ability.

The proposed adaptive algorithm includes several adaptive modules. First, an adaptive range is generated based on the historical information of the function and the characteristics of the current point, and this range is used to generate random sample points for training. The adaptive range is used to generate the adaptive variance to measure the volatility of the function. Then, the adaptive learning rate for training is generated by combining the adaptive variance, the historical information of the function, and the first-order derivative and second-order derivative of the current point. Several adaptive algorithms are proposed to generate adaptive range, adaptive variance, and adaptive learning rate. Our adaptive mechanisms can initialize the parameters without prior knowledge and modify them to match different training states without human intervention. In addition, our proposed learning rate can both speed up the training of early training states and improve the accuracy of later training, while the adaptive variance reflects the fluctuations of different perspectives.

In addition, our method incorporates the idea of EC to modify the structure of MSGD, which achieves better global optimization ability and late stability. During training, each sample point is independently trained for different numbers of times and then ranked according to the loss function, and the best ones are retained for the next training iteration, which empowers the algorithm to jump out of the local minima. In addition, by introducing the idea of meta-heuristic algorithms, the distribution of generated sample points is improved so that the global search method can better utilize the search space while enhancing robustness.

The two are organically combined. Each sample point feeds back the acquired information, which can set the adaptive range in a more timely and reasonable way, affecting other adaptive variables more quickly. At the same time, modifying the adaptive range can also affect the generation of sample points in real-time. In this way, our method empowers the population intelligence and adjusts the adaptive algorithm independently.

The experiment is divided into two aspects: firstly, the validation of the adaptive algorithm, and secondly, the validation of the global optimization seeking ability. By analyzing the learning rate of the sample points in the function, it is obtained that our method can produce appropriate learning rates that can meet the requirements of different periods of training. By analyzing the training in the function, it can be obtained that the proposed adaptive algorithm has a good performance in different function conditions. The training in one-dimensional binary complex functions shows that the proposed method has a better global optimization ability, and compared with other algorithms, our algorithm has an obvious advantage in global optimization and generalization ability. In addition, our method can reduce the consumption of computational resources through lightweight improvement and has good applicability in neural networks.

However, this method does not solve some drawbacks of neural networks, such as overfitting. Therefore, for the further development of neural networks, we will focus on improving neural networks in the future. Improve the neural network to realize the adaptive setting of layers and simplify the network structure. Improve various functions and mechanisms in the network to minimize the loss of information caused by the transmission of various parts of the network and improve training accuracy.

7 Acknowledgement

This work is supported by the Industrial Support Plan of Education Department of Gansu Province (No. 2021CYZC-30).

References

[1] N. Hu, C. Fan, Y. Ming, F. Feng, MAENet: A novel multi-head association attention enhancement network for completing intra-modal interaction in image captioning, Neurocomputing 519(2023) 69-81.

[2] J. Chai, H. Zeng, A. Li, E.-W. Ngai, Deep learning in computer vision: A critical review of emerging techniques and application scenarios, Machine Learning with Applications 6(2021) 100134.

[3] N. Zobeiry, K.-D. Humfeld, A physics-informed machine learning approach for solving heat transfer equation in advanced manufacturing and engineering applications, Engineering Applications of Artificial Intelligence 101(2021) 104232.

[4] D.-W. Otter, J.-R. Medina, J.-K. Kalita, A survey of the usages of deep learning for natural language processing, IEEE transactions on neural networks and learning systems 32(2)(2020) 604-624.

[5] A. Krizhevsky, I. Sutskever, G.-E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proc. Advances in neural information processing systems 25, 2012.

[6] H. Yang, K.-R. Schell, GHTnet: Tri-Branch deep learning network for real-time electricity price forecasting, Energy 238(2022) 122052.

[7] M.-N. Islam, R. Shrestha, S.-R. Chowdhury, A New Hardware-Efficient VLSI-Architecture of GoogLeNet CNN-Model Based Hardware Accelerator for Edge Computing Applications, in: Proc. 2022 IEEE Computer Society Annual Symposium on VLSI, 2022.

[8] K. Lin, Y. Zhao, L. Wang, W. Shi, F. Cui, T. Zhou, MSWNet: A visual deep machine learning method adopting transfer learning based upon ResNet 50 for municipal solid waste sorting, Frontiers of Environmental Science & Engineering 17(6)(2023) 77.

[9] Y. Xue, Y. Wang, J. Liang, A self-adaptive gradient descent search algorithm for fully-connected neural networks, Neurocomputing 478(2022) 70-80.

[10] M. Li, T. Zhang, Y. Chen, A. J. Smola, Efficient mini-batch training for stochastic optimization, in: Proc. of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014.

[11] L. Yang, D. Cai, AdaDB: An adaptive gradient method with data-dependent bound, Neurocomputing 419(2021) 183-189.

[12] S. Long, W. Tao, Z.-D. Zhang, Q. Tao, Adaptive NAG Methods Based on AdaGrad and Its Optimal Individual Convergence, Journal of Software 33(4)(2021) 1231-1243.

[13] D.-S. Kwon, C. Jin, M.-H. Kim, Prediction of dynamic and structural responses of submerged floating tunnel using artificial neural network and minimum sensors, Ocean Engineering 244 (2022) 110402.

[14] Z. Qu, S. Yuan, R. Chi, L. Chang, L. Zhao, Genetic optimization method of pantograph and catenary comprehensive monitor status prediction model based on adadelta deep neural network, IEEE access 7 (2019) 23210-23221.

[15] A. Barakat, P. Bianchi, Convergence and dynamical behavior of the ADAM algorithm for nonconvex stochastic optimization, SIAM Journal on Optimization 31.1(2021) 244-274.

[16] A. Senior, G. Heigold, M.-A. Ranzato, K. Yang, An empirical study of learning rates in deep neural networks for speech recognition, in: Proc. 2013 IEEE international conference on acoustics, speech and signal processing, 2013.

[17] A.-P. George, W.-B. Powell, Adaptive stepsizes for recursive estimation with applications in approx-imate dynamic programming, Machine Learning 65(1)(2006) 167-198.

[18] N.-L. Roux, A.-W. Fitzgibbon, A fast natural Newton method, in: Proc. International Conference on Machine Learning DBLP, 2010.

[19] T. Schaul, S. Zhang, Y. LeCun, No more pesky learning rates, in: Proc. International conference on machine learning, PMLR, 2013.

[20] Y. Wu, L. Liu, J. Bae, K.-H. Chow, A. Iyengar, C. Pu, W. Wei, L. Yu, Q. Zhang, Demystifying learning rate policies for high accuracy training of deep neural networks, in: Proc. 2019 IEEE International conference on big data, 2019.

[21] A. Bordes, L. Bottou, P. Gallinari, SGD-QN: Careful Quasi-Newton Stochastic Gradient Descent, Journal of Machine Learning Research 10(3)(2009) 1737-1754.

[22] D.-H. Salunkhe, G. Michel, S. Kumar, M. Sanguineti, D. Chablat, An efficient combined local and global search strategy for optimization of parallel kinematic mechanisms with joint limits and collision constraints, Mechanism and Machine Theory 173(2022):104796.

[23] G.-E. Hinton, R.-R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, science 313(5786) (2006) 504-507.

[24] Y. Lecun, Y. Bengio, G. Hinton, Deep learning, nature 521.7553 (2015) 436-444.

[25] J.-R. Jian, Z.-G. Chen, Z.-H. Zhan, J. Zhang, Region Encoding Helps Evolutionary Computation Evolve Faster: A New Solution Encoding Scheme in Particle Swarm for Large-Scale Optimization, IEEE Transactions on Evolutionary Computation 25(4)(2021) 779-793.

[26] Y. Ji, S. Liu, M. Zhou, Z. Zhao, X. Guo, L. Qi, A machine learning and genetic algorithm-based method for predicting width deviation of hot-rolled strip in steel production systems, Information Sciences 589(2022) 360-375.

[27] K.-H. Rahi, H.-K. Singh, T. Ray, Partial Evaluation Strategies for Expensive Evolutionary Con-strained Optimization, IEEE Transactions on Evolutionary Computation 25(6)(2021) 1103-1117.

[28] W.-Y. Dong, R.-R. Zhang, Stochastic stability analysis of composite dynamic system for particle swarm optimization, Information Sciences 592(2022) 227-243.
[29] P. Wang, Y. Zhou, Q. Luo, C. Han, Y. Niu, M. Lei, Complex-valued encoding metaheuristic optimization algorithm: A comprehensive survey, Neurocomputing 407(2020) 313-342.
[30] A. Telikani, A. Tahmassebi, W. Banzhaf, A.-H. Gandomi, Evolutionary Machine Learning: A Survey, ACM Computing Surveys (CSUR) 54(8)(2021) 1-35.
[31] F. Han, J. Jiang, Q.-H. Ling, B.-Y. Su, A survey on metaheuristic optimization for random single-hidden layer feedforward neural network, Neurocomputing 335(2019) 261-273.
[32] M.-B. Bonab, G. Bok-Min, M. A. L. B. Nair, C.-K. Huat, W.-C. Chwee, A New Genetic-Based Hyper-Heuristic Algorithm for Clustering Problem, in: Proc. of the 12th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2020) 12. Springer International Publishing, 2021.
[33] H. Pourvaziri, H. Pierreval, Combinning metaheuristic search and simulation to deal with capacitated aisles in facility layout, Neurocomputing 452(2021) 443-449.
[34] S. Gupta, K. Deep, A hybrid self-adaptive sine cosine algorithm with oposition based learning, Expert Systems with Applications 119(2019) 210-230.
[35] F. Xu, H. Li, C.-M. Pun, H. Hu, Y. Li, Y. Song, H. Gao, A new global best guided artificial bee colony algorithm with application in robot path planning, Applied Soft Computing 88(2020) 106037.
[36] R. Poláková, J. Tvrdík, P. Bujok, Differential evolution with adaptive mechanism of population size according to current population diversity, Swarm and Evolutionary Computation 50(2019) 100519.
[37] S. Mirjalili, S. Z. M. Hashim, H.-M. Sardroudi, Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm, Applied Mathematics & Computation 218(22)(2012) 11125-11137.
[38] G. D'Angelo, F. Palmieri, GGA: A modified genetic algorithm with gradient-based local search for solving constrained optimization problems, Information Sciences 547(2021) 136-162.
[39] I. Ahmadianfar, O. Bozorg-Haddad, X. Chu. Gradient-based optimizer: A new metaheuristic optimization algorithm, Information Sciences 540(2020) 131-159.
[40] X. Li, Y. Liu, Z. Liu, Physics-informed neural network based on a new adaptive gradient descent algorithm for solving partial differential equations of flow problems, Physics of Fluids 35(6)(2023).
[41] W. Lin, T. Chen, Analysis of two restart algorithms, Neurocomputing 69(16-18)(2006) 2301-2308.
[42] L.-N. Smith, Cyclical learning rates for training neural networks, in: Proc. 2017 IEEE winter conference on applications of computer vision, 2017.
[43] K. Ahn, J. Zhang, S. Sra, Understanding the unstable convergence of gradient descent, in: Proc. 2022 International Conference on Machine Learning, 2022.
[44] W. Liu, L. Chen, Y. Chen, W. Zhang, Accelerating Federated Learning via Momentum Gradient Descent, IEEE Transactions on Parallel and Distributed Systems 31(8)(2020) 1754-1766.
[45] A. Botev, G. Lever, D. Barber, Nesterov's Accelerated Gradient and Momentum as Approximations to Regularised Update Descent, in: Proc. 2017 International Joint Conference on Neural Networks, 2017.
[46] H. Li, H. Cheng, Z. Wang, G.-C. Wu, Distributed Nesterov Gradient and Heavy-Ball Double Accelerated Asynchronous Optimization, IEEE Transactions on Neural Networks and Learning Systems 32(12)(2020) 5723-5737.
[47] F. Dignum, Review of Heuristics: The Foundations of Adaptive Behavior, Journal of Artificial Societies and Social Simulation 15(1)(2012) https://www.jasss.org/15/1/reviews/3.html.
[48] H.-G. Beyer, H.-P. Schwefel, Evolution strategies – A comprehensive introduction, Natural Computing 1(2002) 3-52.
[49] R. Poli, J. Kennedy, T. Blackwell, Particle swarm optimization: An overview, Swarm intelligence 1(2007) 33-57.
[50] J. Duchi, E. Hazan, Y. Singer, Adaptive Subgradient Methods Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, Journal of Machine Learning Research 12(2011) 2121-2159.
[51] W. Cai, L. Yang, Y. Yu, Solution of ackley function based on particle swarm optimization algorithm, in: Proc. 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications, 2020.
[52] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Transactions on Evolutionary computation 3(2) (1999) 82-102.