Generative Adversarial Optimization

Ying Tan(✉) and Bo Shi

Key Laboratory of Machine Perception (Ministry of Education), School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China
{ytan,pkushibo}@pku.edu.cn

Abstract. Inspired by the adversarial learning in generative adversarial network, a novel optimization framework named Generative Adversarial Optimization (GAO) is proposed in this paper. This GAO framework sets up generative models to generate candidate solutions via an adversarial process, in which two models are trained alternatively and simultaneously, i.e., a generative model for generating candidate solutions and a discriminative model for estimating the probability that a generated solution is better than a current solution. The training procedure of the generative model is to maximize the probability of the discriminative model. Specifically, the generative model and the discriminative model are in this paper implemented by multi-layer perceptrons that can be trained by the back-propagation approach. As of an implementation of the proposed GAO, for the purpose of increasing the diversity of generated solutions, a guiding vector ever introduced in guided fireworks algorithm (GFWA) has been employed here to help constructing generated solutions for the generative model. Experiments on CEC2013 benchmark suite show that the proposed GAO framework achieves better than the state-of-art performance on multi-modal functions.

Keywords: Generative Adversarial Optimization (GAO) · Adversarial Learning · Generative adversarial network (GAN) · Guiding vector · Multi-modal functions

1 Introduction

Continuously-valued function optimization problem [20] has long been an important problem in mathematics and computer science. With the development of deep learning in recent years, continuously-valued function optimization problem has become more and more important [23,37]. For continuously-valued function optimization problems, gradient-based methods are commonly used, such as stochastic gradient descent (SGD), Newton's method, conjugate gradient (CG), BFGS and so on [33]. However, for more complex functions and multi-modal functions, gradient-based methods can only find local optimal solutions. For these problems, the algorithm needs to be able to better deal with the balance between exploration and exploitation [4].

Y. Tan et al. (Eds.): ICSI 2019, LNCS 11655, pp. 3–17, 2019.
https://doi.org/10.1007/978-3-030-26369-0_1

In order to solve the problem, more and more meta-heuristic algorithms have been proposed. Meta-heuristic algorithms are usually inspired by biological or human behaviors. By designing a sophisticated mechanism to guide algorithms to find solutions, so as to avoid local optimal solutions and find global optimal solutions. The most critical component for meta-heuristic algorithms is generating solutions and retaining solutions. For the part of generating solutions, the algorithm should generate better solutions as many as possible, but at the same time, it is also hoped that the generated solutions have a rich diversity and will not cluster in local optimal spaces. For the part of retaining solutions, the algorithm should retain better solutions, but it is also hoped that potential solutions which are not so good currently can be retained, because solutions which is better than the current optimal solution may be found in the local searches around them later.

In the early meta-heuristic algorithms, various methods to generate solutions were proposed. Particle swarm optimization (PSO) [19] mimics migration and clustering in the foraging of birds to generate solution. The genetic algorithm (GA) [8] targets all individuals in a group and uses randomization techniques to efficiently search a coded parameter space. The fireworks algorithm (FWA) [40] employs the fitness value of each firework to dynamically calculate the explosion radius and the number of sparks in an explosion as local search. In recent years, research on methods to generate solutions tends to be more refined. Guided firework algorithm (GFWA) [26] employs the fitness value information obtained by the explosion sparks to construct the guiding vector (GV) with promising direction and adaptive length, and an elite solution called guiding spark (GS) is generated by adding the guiding vector to the corresponding firework position.

In recent years, generative adversarial network (GAN) [13] has been proposed as a new generating model, with its outstanding performance proving its powerful ability in generative tasks. Different from the previous generative model, GAN guides a generator to automatically learn how to generate by setting a loss function. In GAN, a discriminator and a generator are alternatively trained, where the discriminator is used to discriminate between the generated sample and the real sample, the generator is used to generate samples as real as possible, so as to deceive the discriminator. GAN has been widely used in the fields of image generation [10,12,32], video synthesis [41,42], text generation [3,21,44], music generation [14], semi-supervised learning [5,9], medical image [7] and information security [16,35].

Inspired by the adversarial learning in GAN, a feasible optimization framework, so-called Generative Adversarial Optimization (GAO), is proposed in this paper. The framework sets up generative models to generate candidate solutions via an adversarial process, in which two models are trained alternatively and simultaneously, i.e., a generative model $\mathcal{G}$ to generate candidate solutions, and a discriminative model $\mathcal{D}$ to estimates the probability that a generated solution is better than a current solution. The training procedure for $\mathcal{G}$ is to maximize the probability of $\mathcal{D}$. In our case, $\mathcal{G}$ and $\mathcal{D}$ are defined by multi-layer perceptrons, which can be trained with back-propagation. To improve the quality of generated

solutions, the guiding vectors introduced in GFWA are employed to help constructing generated solutions. Experiments on CEC2013 benchmark suite show that the proposed framework achieves impressive performance on multi-modal functions.

The main contributions of this paper are as follows:

1. Inspired by adversarial learning and GAN, a novel optimization framework so-called Generative Adversarial Optimization, GAO, for short, is proposed.
2. The guiding vectors introduced in GFWA [26] are employed to help constructing generated solutions, which improves the training stability and generative diversity of $\mathcal{G}$.
3. Experiments show that for GAO, multiple rounds of updates are necessary to obtain better generative capabilities.
4. Compared with current famous optimization algorithms, GAO achieves better than state-of-the-art performance on multi-modal functions.

The remainder of this paper is organized as follows. Section 2 presents related works of meta-heuristic algorithms and GAN. Section 3 describes the detail of GAO, a novel optimization framework proposed for continuously-valued function optimization. Experimental settings and results are presented and discussed in Sect. 4. Conclusions are given in Sect. 5.

2 Related Works

2.1 Meta-heuristic Algorithms

Inspired by biological and human behaviors, meta-heuristic algorithms are a kind of algorithms that can be used to better solve continuous optimization problems by simulating agents' behaviors in order to balance "exploration" and "exploitation" [4]. In recent years, researches on meta-heuristic algorithms for optimization problem have developed rapidly, more and more meta-heuristic algorithms have been proposed. According to the mechanism of the agents' behaviour, meta-heuristic algorithms can be divided into swarm intelligence algorithms [18] and evolutionary computation algorithms.

Swarm intelligence algorithms are usually inspired by the behavior of biological groups in natural world to seek the optimum in search space by employing programs to simulate the interaction among biological individuals. Swarm intelligence algorithms mainly focus on biological groups such as ant colony [11], bird flock [19], fish school [29], etc. In addition, some non-biological group systems also belong to the scope of the researches on swarm intelligence, such as multi-robot systems, fireworks [40] and other unnatural phenomena. there are many famous swarm intelligence algorithms such as particle swarm optimization (PSO) [19], ant colony optimization (ACO) [11], fireworks algorithm (FWA) [24–26,39,40,45–47], etc. Specially, GFWA [26] proposed the guiding vector to help constructing solutions for the first time.

Evolutionary computation algorithms are primarily inspired by biological evolution, which solves the global optimal solution by simulating the evolution

of organisms. Specific algorithms include genetic algorithm (GA) [8], evolution strategy (ES) [38], genetic programming, evolutionary programming, differential evolution (DE) [31], etc.

2.2 Generative Adversarial Networks

Generative adversarial network (GAN), which was first proposed by Goodfellow in 2014 [13], provides a new method of learning deep representations based on extensively unlabeled data. The basic idea of GAN is derived from the minimax two-player game in game theory, consisting of a generator $\mathcal{G}$ and a discriminator $\mathcal{D}$. GAN is trained by means of adversarial learning, with the goal of estimating the potential distribution of the data samples and generating new data samples. The discriminator $\mathcal{D}$ of the original GAN can be regarded as a function $\mathcal{D} : \mathcal{D}(x) \rightarrow (0, 1)$, which maps the sample to the discriminant probability that whether the sample is from the real data distribution or the generator distribution. The generator $\mathcal{G}$ is trained to reduce the discriminator's accuracy. If the generator distribution is sufficient to perfectly match the real data distribution, then the discriminator will be most confused and give a probability value of 0.5 to all inputs.

Since GAN was proposed, it has quickly become a hot research issue. A large number of researches based on GAN have sprung up, mainly focusing on optimizing GAN's structure [10,32,43] and loss function [1,28,30], proposing some tricks to assist the training of GAN [15,32,34], and using GAN to solve specific problems. GAN is widely used in many fields, and there are many impressive works on different tasks. For image Synthesis, there are LapGAN [10], DCGAN [32] and etc. For image-to-image translation, there are pix2pix [17], cycleGAN [48], etc. For super-resolution, there are SRGAN [22]. For text generation and NLP, there are seqGAN [44], maliGAN [3], Gumbel-softmax GAN [21], etc. For information security, there is MalGAN [16].

3 GAO: Generative Adversarial Optimization

GAO and its detailed implementation are presented in this section. First, the model architectures are described in Sect. 3.1, then the training procedure of GAO is discussed in details in Sect. 3.2.

3.1 Model Architectures

Different from the existing meta-heuristic algorithms which mainly adopt random sampling to generate elite solutions or guiding vectors [26], in GAO, we adopt a generative network $\mathcal{G}$ to generate effective guiding vectors for current solutions to move towards. Simultaneously, a discriminative network $\mathcal{D}$ is trained to evaluate whether the generated solution is better than a current one. The generative network $\mathcal{G}$ is trained by computing gradients from the feedback of $\mathcal{D}$, which means that $\mathcal{G}$ learns how to generate better guiding vectors under the guidance of $\mathcal{D}$.

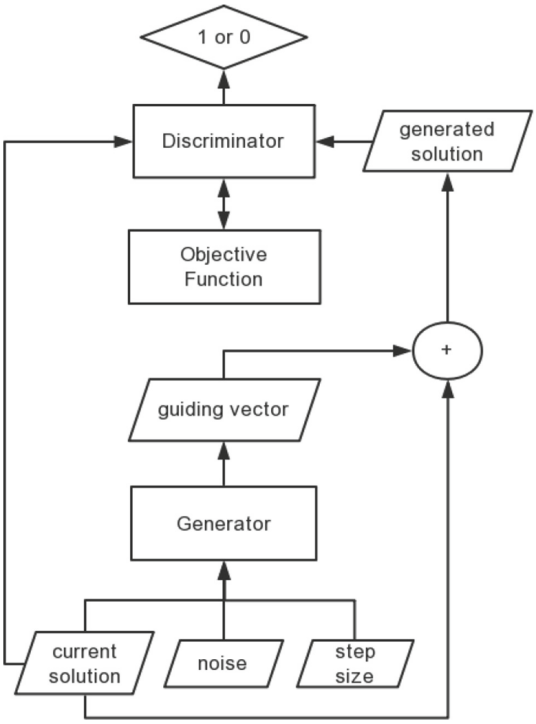

Fig. 1. Architecture of GAO

Given a objective function f, an optimization problem seeks to find the global minimum $x_* \in A$ which satisfies:

$$f(x_*) \leq f(x), \quad \forall x \in A \tag{1}$$

where A is the searching space.

As illustrated in Fig. 1, $\mathcal{G}$ gets the input, which includes a current solution x_c, a noise z and a step size l, and outputs a guiding vector g. This procedure can be expressed in Eq. 2:

$$g = \mathcal{G}(x_c, z, l) \tag{2}$$

Then the guiding vector g is added to the current solution x_c to get the generated solution x_g, as shown in Eq. 3:

$$x_g = x_c + g \tag{3}$$

$\mathcal{D}$ receives a current solution x_c and a generated solution x_g, then outputs a prediction p that whether the generated solution x_g is better than the current solution x_c as shown in Eq. 4. If the generated solution x_g is better than the current solution x_c, let $p = 1$, otherwise $p = 0$.

$$p = \mathcal{D}(x_c, x_g) = \begin{cases} 1, & x_g \ is \ better \ than \ x_c \\ 0, & else \end{cases} \tag{4}$$

In order to train $\mathcal{D}$, labels y^i for tuples of current solution and generated solution $\{x_c^i, x_g^i\}$ are required. The objective function f is employed to label the two-tuple set $\{x_c^i, x_g^i\}$ as expressed in Eq. 5. The training of $\mathcal{D}$ will be detailedly discussed in Sect. 3.2.

$$y^i = \begin{cases} 1, \ if \ f(x_g^i) < f(x_c^i) \\ 0, \ else \end{cases} \tag{5}$$

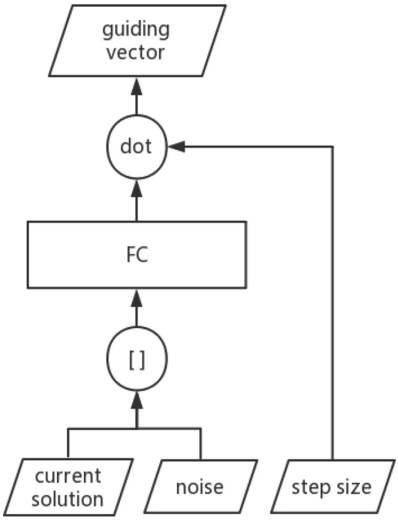

Fig. 2. Architecture of $\mathcal{G}$

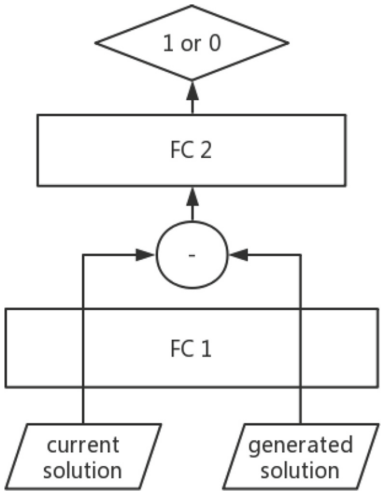

Fig. 3. Architecture of $\mathcal{D}$

The architecture of $\mathcal{G}$ is illustrated in Fig. 2. First, $\mathcal{G}$ concatenate the current solution x_c and noise z included in the input, then feed the concatenated vector to a fully-connected layer (denoted as FC). Finally $\mathcal{G}$ dot the concatenated vector with step size l and get the guiding vector g as $\mathcal{G}$'s output. This procedure can be expressed in Eq. 6.

$$g = \mathcal{G}(x_c, z, l) = FC([x_c^T, z^T]^T) \cdot l \tag{6}$$

The architecture of $\mathcal{D}$ is illustrated in Fig. 3. First, $\mathcal{D}$ feed two solutions x_c, x_g to the same fully-connected layer denoted as FC_1, then subtract the output of x_c with the output of x_g. Finally, $\mathcal{D}$ feed the subtracted vector to a fully-connected layer denoted as FC_2 and get the prediction p as $\mathcal{D}$'s output. This procedure can be expressed in Eq. 7. The activation function for the final layer of FC_2 should be sigmoid function to regularize the prediction.

$$p = \mathcal{D}(x_c, x_g) = FC_2(FC_1(x_c) - FC_1(x_g)) \tag{7}$$

3.2 Training of GAO

The complete training procedure of GAO is shown in Algorithm 1. At the beginning, μ solutions are randomly sampled in searching space to make up the solution set $C = \{x_c^i,\ i = 1, 2, ..., \mu\}$, calculate each solution's fitness value $f(x_c^i)$ and initialize the step size l. Then we repeatedly do adversarial training of $\mathcal{D}$ and $\mathcal{G}$, select solutions to be retained and reduce step size l as the iteration progresses. When the termination criterion is met, the algorithm exit the loop. Since the time allowed to evaluate the solution using fitness function is limited as $MaxFES = 10000 * D$, in which D is the evaluation dimension of fitness function [27], the termination criterion always refers to whether the limited evaluation time is used up. Details of training $\mathcal{D}$ and $\mathcal{G}$, selecting solutions and reducing step size are discussed below.

Algorithm 1. Training procedure of GAO

Require: μ: number of current solutions
Require: β: number of solutions generated at each iteration
Require: l_{init}: initial value of step size l
1: randomly sample μ solutions in searching space A as set $C = \{x_c^i\}$
2: calculate fitness value $f(x_c^i)$ for each solution x_c^i in C
3: initialize the step size $l = l_{init}$
4: **while** termination criterion is not met **do**
5: generate β solutions and train $\mathcal{D}$
6: train $\mathcal{G}$ with fitted $\mathcal{D}$
7: select μ solutions for next iteration from μ current solutions and β generated solutions
8: reduce step size l
9: **end while**

Training of $\mathcal{D}$. $\mathcal{D}$ is trained to evaluate whether the generated solution x_g will be better than the current solution x_c. Train $\mathcal{D}$ requires employing $\mathcal{G}$ to generate solutions first. In this paper, the number of solutions to be generated totally at each iteration is denoted as β. Since $\mathcal{D}$ receives two solutions as input and output a prediction, training $\mathcal{D}$ requires triplets composed of two solutions x_c^i and x_g^i and a label y^i, in which y^i can be calculated with Eq. 5. For a triplet $\{x_c^i, x_g^i, y^i\}$, the loss function of $\mathcal{D}$ can be calculated with Eq. 8:

$$\max_{\mathcal{D}}\ loss_{\mathcal{D}} = y^i \log(D(x_c^i, x_g^i)) + (1 - y^i) \log(1 - D(x_c^i, x_g^i)) \tag{8}$$

When training with batches, the loss of a batch is the average loss for each triplet in batch.

Training of $\mathcal{G}$. As mentioned above, $\mathcal{G}$ learns how to generate better guiding vectors under the guidance of $\mathcal{D}$, which means that $\mathcal{G}$ is trained by computing

gradients from the feedback of $\mathcal{D}$. $\mathcal{G}$ is trained to generate elite guiding vectors for current solutions, so it's hoped that the generated solutions perform better than current solutions. For a current solution x_c^i, the loss function of $\mathcal{G}$ can be calculated with Eq. 9:

$$\max_{\mathcal{G}} \ loss_{\mathcal{G}} = \log(D(x_c^i, x_c^i + \mathcal{G}(x_c^i, z, l))) \tag{9}$$

In which, z is a random Gaussian noise, l is the step size. When training with batches, the loss of a batch is the average loss for each triplet in the batch.

Selecting Solutions. In general, solutions with better fitness values should be retained, so we calculate the probability to be selected for each solution x^i in Eq. 10 and select solutions using the calculated probability:

$$p_r(x^i) = \frac{\gamma_{f(x^i)}^{-\alpha}}{\sum_{i=1}^{n} \gamma_{f(x^i)}^{-\alpha}} \tag{10}$$

where $\gamma_{f(x^i)}$ means the rank of fitness value for x^i among all solutions, n is the total number of candidate solutions, α is a hyper-parameter to control the shape of the distribution. The larger α is, the probability of solutions with better fitness values is larger as well.

Reducing Step Size. In GAO, the guiding vector introduced in GFWA [26] is employed to control the searching radius at each iteration. In a general searching process, searching radius should be larger at the beginning and gradually reduced to a smaller value, which coincides with keeping the balance between exploration and exploitation. At the beginning, the algorithm needs to explore the searching space to avoid missing any local optimal region where the global optimum may exist. As the algorithm goes on, it has accumulated some information about the searching space and tends to exploit more in existing local optimal regions. Thus several different schemes to adjust step size are designed, for which the basic principle is to gradually decrease the step size as the algorithm goes on. The detailed introduction and experiments will be discussed in Sect. 4.

4 Experiments

In this section, principles on how to set parameters and construct $\mathcal{D}$ and $\mathcal{G}$ are given. In more detail, we first introduce the model architecture specifically and give principles for setting parameters. Secondly, the benchmark the experiment taken on is introduced. Finally, we compare GAO with other famous optimization algorithms.

In our experiment, the architecture of $\mathcal{D}$ and $\mathcal{G}$ are mainly fully-connected layers. In this section, we denote the number of hidden layers as L, the sizes of each hidden layer as H, the sizes of output layer as O ,the activation functions of each hidden layer as AH and the activation functions of output layer as AO.

each of them is introduced respectively as follows. For FC in $\mathcal{G}$, we set $L = 1$, $H = [64]$, $O = dimension\ of\ objective\ function$, $AH = [relu]$, $AO = tanh$. For $FC1$ in $\mathcal{D}$, we set $L = 2$, $H = [64, 64]$, $O = 10$, $AH = [relu, relu]$, $AO = relu$. For $FC2$ in $\mathcal{D}$, we set $L = 1$, $H = [10]$, $O = 1$, $AH = [relu]$, $AO = sigmoid$.

The number of solutions retained at each iteration is denoted as μ, which mainly keeps the balance between "exploration" and "exploitation" [4]. Since the time allowed to evaluate is limited to $MaxFES = 10000 * D$, where D is the dimension of the objective function, smaller μ allows more solutions to be generated from one solution, which focuses on "exploitation", while larger μ allows generating solutions from more locations in search space, which focuses on "exploration". In this paper, we follow the suggestion in [40] and set $\mu = 5$.

To train $\mathcal{D}$, we need to label the tuple of $\{x_c^i, x_g^i\}$ with y^i, which requires using objective function to evaluate the fitness value of x_g^i, since fitness value of x_c^i have been calculated at the former iteration. To make $\mathcal{D}$ learn how to generate solutions better, we not only generate x_g^i from $\mathcal{G}$, but also generate x_g^i from local search and global search at each iteration. When generating solution, x_g^i calculated from Eq. 3 have to be clipped to the boundary once it exceeds the search space. In this paper, we denote the number of solutions to be generated totally at each iteration as β. On account of the limit of $MaxFES$, the iteration number $MaxIter = \frac{MaxFES}{\beta}$. In this paper, we set $\beta = 30$.

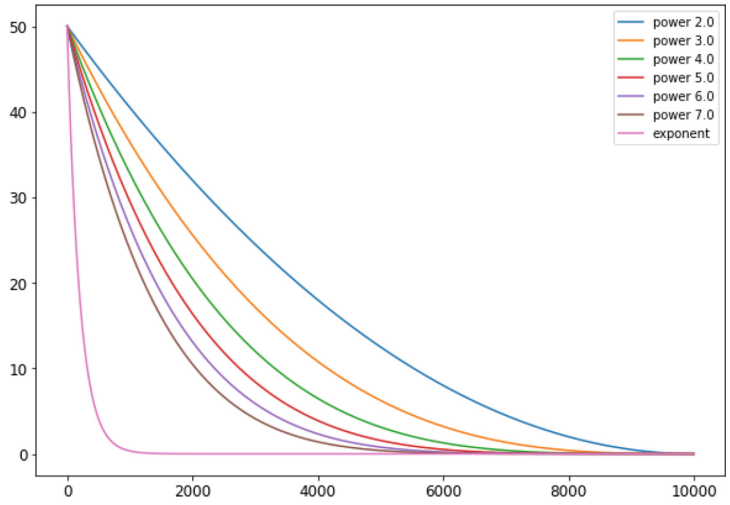

Fig. 4. How step size changes with different monotone functions

As discussed in Sect. 3.2, when selecting solutions, we calculate a probability to be selected for each solution x^i as expressed in Eq. 10 and select solutions in accordance with that probability. We denote the parameter controlling the shape of the distribution as α. The larger α is, the probability of solutions with better fitness values is larger as well. In this paper, we set $\alpha = 2$ as suggested in [24].

In our experiment, step size l have to be set as l_{init} at the beginning of the algorithm. In general, we set $l_{init} = \frac{1}{2} \cdot radius\ of\ search\ space$. Specifically for CEC2013 [27] in this paper, we set $l_{init} = 50$. To gradually reduce the step size as the algorithm goes on, we map the iteration count to $[\epsilon, l_{init}]$ with a monotone function F, here ϵ is a small positive number set to 10^{-20} in this paper. In practice, we compare exponential function and power function with different power. Figure 4 illustrates how step size changes with iteration count increases when using different functions. It shows that using exponential function make the step size drop rapidly. And when using power function, the step size drop faster as the power increases.

We compare different monotone functions on CEC2013 benchmark suite and the average ranks (ARs) are shown in Figs. 5 and 6, in which AR-uni, AR-multi and AR-all indicate average ranks for uni-modal, multi-modal and all functions, respectively. It shows that using power function performs better than exponential function and using 4.5 as power is comprehensively best. In this paper, we use power function and set power to 4.5.

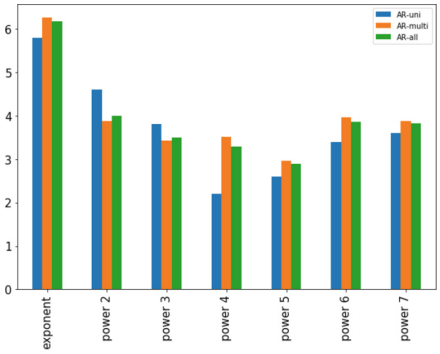

Fig. 5. Average ranks for different monotone functions

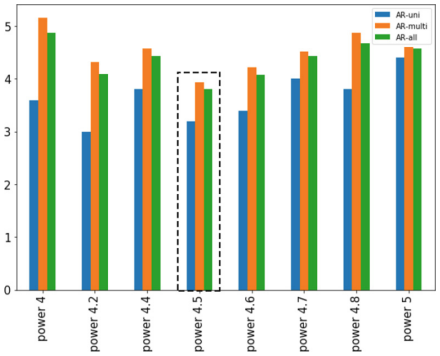

Fig. 6. Average ranks for power function with different power

We choose CEC2013 single objective optimization benchmark suite [27] as the test suite for the following experiments. CEC2013 single objective optimization benchmark suite includes 5 uni-modal functions and 23 multi-modal functions, whose optimal values range from -1400 to 1400 and searching range is $[-100,\ 100]$. According to the requirements of the benchmark suite, all the algorithms should run 51 times for each function to calculate average and variance. The maximal number of function evaluations in each run, which is denoted as $MaxFES$, is set as $10000 * D$, where D is the dimension of the objective function. The benchmark suite supports 10, 30 and 50 as the dimension of the objective function.

Table 1. Mean error, standard variance and average ranks of the chosen algorithms on CEC2013 benchmark suite

	ABC		SPSO2011		IPOP-CMAES		DE		LoT-FWA		GAO	
CEC2013	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
1	**0.00E+00**	0.00E+00	**0.00E+00**	1.88E−13	**0.00E+00**	0.00E+00	1.89E−03	4.65E−04	**0.00E+00**	0.00E+00	**0.00E+00**	0.00E+00
2	6.20E+06	1.62E+06	3.38E+05	1.67E+05	**0.00E+00**	0.00E+00	5.52E+04	2.70E+04	1.19E+06	4.27E+05	1.02E+06	6.89E+05
3	5.74E+08	3.89E+08	2.88E+08	5.24E+08	**1.73E+00**	9.30E+00	2.16E+06	5.19E+06	2.23E+07	1.91E+07	7.98E+06	1.01E+07
4	8.75E+04	1.17E+04	3.86E+04	6.70E+03	**0.00E+00**	0.00E+00	1.32E−01	1.02E−01	2.13E+03	8.11E+02	3.17E+03	1.49E+03
5	**0.00E+00**	0.00E+00	5.42E−04	4.91E−05	**0.00E+00**	0.00E+00	2.48E−03	8.16E−04	3.55E−03	5.01E−04	2.95E−03	4.70E−04
AR. uni	4		3.4		1		3.2		3.8		3.4	
6	1.46E+01	4.39E+00	3.79E+01	2.83E+01	**0.00E+00**	0.00E+00	7.82E+00	1.65E+01	1.45E+01	6.84E+00	1.73E+01	1.53E+01
7	1.25E+02	1.15E+01	8.79E+01	2.11E+01	1.68E+01	1.96E+01	4.89E+01	2.37E+01	5.05E+01	9.69E+00	**1.08E+01**	8.15E+00
8	2.09E+01	4.97E−02	2.09E+01	5.89E−02	2.09E+01	5.90E−02	2.09E+01	5.65E−02	2.09E+01	6.14E−02	**2.09E+01**	6.96E−02
9	3.01E+01	2.02E+00	2.88E+01	4.43E+00	2.45E+01	1.61E+01	1.59E+01	2.69E+00	1.45E+01	2.07E+00	**1.09E+01**	2.10E+00
10	2.27E−01	6.75E−02	3.40E−01	1.48E−01	**0.00E+00**	0.00E+00	3.24E−02	1.97E−02	4.52E−02	2.47E−02	1.09E−02	1.05E−02
11	**0.00E+00**	0.00E+00	1.05E+02	2.74E+01	2.29E+00	1.45E+00	7.88E+01	2.51E+01	6.39E+01	1.04E+01	5.84E+01	1.84E+01
12	3.19E+02	5.23E+01	1.04E+02	3.54E+01	**1.85E+00**	1.16E+00	8.14E+01	3.00E+01	6.82E+01	1.45E+01	5.60E+01	1.26E+01
13	3.29E+02	3.91E+01	1.94E+02	3.86E+01	**2.41E+00**	2.27E+00	1.61E+02	3.50E+01	1.36E+02	2.30E+01	1.09E+02	2.61E+01
14	**3.58E−01**	3.91E−01	3.99E+03	6.19E+02	2.87E+02	2.72E+02	2.38E+03	1.42E+03	2.38E+03	3.13E+02	2.38E+03	4.39E+02
15	3.88E+03	3.41E+02	3.81E+03	6.94E+02	**3.38E+02**	2.42E+02	5.19E+03	5.16E+02	2.58E+03	3.83E+02	2.43E+03	4.78E+02
16	1.07E+00	1.96E−01	1.31E+00	3.59E−01	2.53E+00	2.73E−01	1.97E+00	2.59E−01	**5.74E−02**	2.13E−02	7.72E−02	4.24E−02
17	**3.04E+01**	5.15E−03	1.16E+02	2.02E+01	3.41E+01	1.36E+00	9.29E+01	1.57E+01	6.20E+01	9.45E+00	9.40E+01	1.80E+01
18	3.04E+02	3.52E+01	1.21E+02	2.46E+01	8.17E+01	6.13E+01	2.34E+02	2.56E+01	**6.12E+01**	9.56E+00	8.92E+01	2.64E+01
19	**2.62E−01**	5.99E−02	9.51E+00	4.42E+00	2.48E+00	4.02E−01	4.51E+00	1.30E+00	3.05E+00	6.43E−01	3.68E+00	8.05E−01
20	1.44E+01	4.60E−01	1.35E+01	1.11E+00	1.46E+01	3.49E−01	1.43E+01	1.19E+00	1.33E+01	1.02E+00	**1.10E+01**	6.91E−01
21	**1.65E+02**	3.97E+01	3.09E+02	6.80E+01	2.55E+02	5.03E+01	3.20E+02	8.55E+01	2.00E+02	2.80E−03	2.94E+02	6.29E+01
22	**2.41E+01**	2.81E+01	4.30E+03	7.67E+02	5.02E+02	3.09E+02	1.72E+03	7.06E+02	3.12E+03	3.79E+02	2.99E+03	5.34E+02
23	4.95E+03	5.13E+02	4.83E+03	8.23E+02	**5.76E+02**	3.50E+02	5.28E+03	6.14E+02	3.11E+03	5.16E+02	2.67E+03	6.25E+02
24	2.90E+02	4.42E+00	2.67E+02	1.25E+01	2.86E+02	3.02E+01	2.47E+02	1.54E+01	**2.37E+02**	1.20E+01	2.39E+02	6.32E+00
25	3.06E+02	6.49E+00	2.99E+02	1.05E+01	2.87E+02	2.85E+01	2.80E+02	1.57E+01	2.71E+02	1.97E+01	**2.60E+02**	1.74E+01
26	2.01E+02	1.93E−01	2.86E+02	8.24E+01	3.15E+02	8.14E+01	2.52E+02	6.83E+01	**2.00E+02**	1.76E−02	2.00E+02	4.32E−02
27	**4.16E+02**	1.07E+02	1.00E+03	1.12E+02	1.14E+03	2.90E+02	7.64E+02	1.00E+02	6.84E+02	9.77E+01	6.45E+02	5.58E+01
28	**2.58E+02**	7.78E+01	4.01E+02	4.76E+02	3.00E+02	0.00E+00	4.02E+02	3.90E+02	2.65E+02	7.58E+01	2.96E+02	2.77E+01
AR. multi	3.57		4.87		2.87		4.04		2.61		2.48	
AR. all	3.64		4.61		2.54		3.89		2.82		2.64	

We compared GAO with the famous optimization algorithms including the artificial bee colony algorithm (ABC), the standard particle swarm optimization 2011 (SPSO2011) [6], the restart CMA-ES with increasing population size (IPOP-CMA-ES) [2], the differential evolution algorithm (DE) [36] and the loser-out tournament based FWA (LoT-FWA) [24]. The parameters of these algorithms are set as suggested in [2,6,24,36]. All these algorithms are tested under same conditions with GAO. The mean errors, standard deviations and average ranks are shown in Table 1. Average ranks are calculated separately for uni-modal functions and multi-modal functions, denoted as AR.uni and AR.multi, respectively. Average rank for all functions are denoted as AR.all. The minimal mean errors for each function are shown in **bold**.

As illustrated in Table 1, on all functions, IPOP-CMA-ES performs best, followed by GAO and LoT-FWA, while SPSO2011 is the worst one. IPOP-CMA-ES, ABC, GAO and LoT-FWA achieve 11, 10, 6 and 5 of 28 minimal mean errors on all functions, respectively. Specifically on uni-modal functions, IPOP-CMA-ES performs best as well, followed by DE, SPSO2011 and GAO performing comparable, while ABC is the worst one. IPOP-CMA-ES achieves all minimal errors on uni-modal functions, while ABC, SPSO2011, LoT-FWA and GAO achieve 1 of 5 the minimal mean errors.

On multi-modal functions, GAO performs best, followed by LoT-FWA and IPOP-CMA-ES, while SPSO2011 is the worst one. ABC achieves 8 of 23 minimal mean errors on multi-modal functions, followed by IPOP-CMA-ES, GAO and LoT-FWA, achieving 6, 5, 4 of 23 minimal mean errors, respectively. SPSO2011 and DE performs worst, achieving none minimal mean errors on multi-modal functions. Although ABC achieves 8 minimal mean errors on multi-modal functions, it also achieves 10 maximal mean error, which shows that ABC is not stable enough. At the same time, GAO achieves none maximal mean errors on all functions, which shows that GAO is quite stable and can be adapted to various problems.

It turns out from the experimental results that the proposed GAO framework performs quite very well on multi-modal functions. This is mainly due to the adversarial learning procedure, which enables $\mathcal{G}$ to learn how to generate elite and diverse solutions under the supervision of $\mathcal{D}$, rather than to follow an artificially-designed meta-heuristic rule directly. In our implementation, the guiding vector introduced in GFWA [26] has been employed to improve the quality and diversity of generated solutions, which can also certainly be replaced by other feasible methods. In this paper, the exploration on hyper-parameters is greatly simplified, with a main focus on presenting the proposed optimization framework.

5 Conclusion

Inspired by the adversarial learning in generative adversarial network, this paper proposed a novel optimization framework, so-called GAO, for short, which is the first attempt to employ adversarial learning for continuously-valued function optimization. In order to improve the quality of generated solutions, a guiding vector appeared in GFWA is employed in this paper to help constructing

generated solutions. Experiments on CEC2013 benchmark suite shew that the proposed GAO algorithm performs quite well, especially on multi-modal functions, it gave the best performance over some famous optimization approaches. Meanwhile, the performance of the GAO framework on uni-modal functions indicates that there is still room for improvement. It is worth noting that the proposed GAO framework should be further studied since it can be easily embedded into any iterative algorithms as an operator to generate solutions. We hope this paper can be regarded as a start point to attract more research on solving various optimization problems using adversarial learning strategy.

Acknowledgement. This work was supported by the Natural Science Foundation of China (NSFC) under grant no. 61673025 and 61375119 and also Supported by Beijing Natural Science Foundation (4162029), and partially supported by National Key Basic Research Development Plan (973 Plan) Project of China under grant no. 2015CB352302.

References

1. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein GAN. arXiv preprint arXiv:1701.07875 (2017)
2. Auger, A., Hansen, N.: A restart cma evolution strategy with increasing population size. In: 2005 IEEE Congress on Evolutionary Computation, vol. 2, pp. 1769–1776. IEEE (2005)
3. Che, T., et al.: Maximum-likelihood augmented discrete generative adversarial networks. arXiv preprint arXiv:1702.07983 (2017)
4. Chen, J., Xin, B., Peng, Z., Dou, L., Zhang, J.: Optimal contraction theorem for exploration-exploitation tradeoff in search and optimization. IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. **39**(3), 680–691 (2009)
5. Chongxuan, L., Xu, T., Zhu, J., Zhang, B.: Triple generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 4088–4098 (2017)
6. Clerc, M.: Standard particle swarm optimisation from 2006 to 2011. Part. Swarm Cent. **253** (2011)
7. Dai, W., et al.: Scan: structure correcting adversarial network for chest X-rays organ segmentation. arXiv preprint arXiv:1703.08770 (2017)
8. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York (1991)
9. Denton, E., Gross, S., Fergus, R.: Semi-supervised learning with context-conditional generative adversarial networks. arXiv preprint arXiv:1611.06430 (2016)
10. Denton, E.L., Chintala, S., Fergus, R., et al.: Deep generative image models using a Laplacian pyramid of adversarial networks. In: Advances in Neural Information Processing Systems, pp. 1486–1494 (2015)
11. Dorigo, M., Di Caro, G.: Ant colony optimization: a new meta-heuristic. In: Proceedings of the 1999 Congress on Evolutionary Computation-CEC 1999 (Cat. No. 99TH8406), vol. 2, pp. 1470–1477. IEEE (1999)
12. Ganin, Y., et al.: Domain-adversarial training of neural networks. J. Mach. Learn. Res. **17**(1), 2096–3030 (2016)
13. Goodfellow, I., et al.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)

14. Guimaraes, G.L., Sanchez-Lengeling, B., Outeiral, C., Farias, P.L.C., Aspuru-Guzik, A.: Objective-reinforced generative adversarial networks (organ) for sequence generation models. arXiv preprint arXiv:1705.10843 (2017)
15. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of Wasserstein GANs. In: Advances in Neural Information Processing Systems, pp. 5767–5777 (2017)
16. Hu, W.W., Tan, Y.: Generating adversarial malware examples for black-box attacks based on GAN (2017)
17. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1125–1134 (2017)
18. Kennedy, J.: Swarm intelligence. In: Zomaya, A.Y. (ed.) Handbook of Nature-Inspired and Innovative Computing, pp. 187–219. Springer, Boston (2006). https://doi.org/10.1007/0-387-27705-6_6
19. Kennedy, J.: Particle swarm optimization. In: Sammut, C., Webb, G.I. (eds.) Encyclopedia of Machine Learning, pp. 760–766. Springer, Boston (2010). https://doi.org/10.1007/978-0-387-30164-8
20. Koziel, S., Yang, X.S.: Computational Optimization, Methods and Algorithms, vol. 356. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20859-1
21. Kusner, M.J., Hernández-Lobato, J.M.: GANs for sequences of discrete elements with the Gumbel-softmax distribution. arXiv preprint arXiv:1611.04051 (2016)
22. Ledig, C., et al.: Photo-realistic single image super-resolution using a generative adversarial network. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp. 4681–4690 (2017)
23. Lehman, J., Chen, J., Clune, J., Stanley, K.O.: Safe mutations for deep and recurrent neural networks through output gradients. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 117–124. ACM (2018)
24. Li, J., Tan, Y.: Loser-out tournament-based fireworks algorithm for multimodal function optimization. IEEE Trans. Evol. Comput. **22**(5), 679–691 (2018)
25. Li, J., Zheng, S., Tan, Y.: Adaptive fireworks algorithm. In: 2014 IEEE Congress on Evolutionary Computation (CEC), pp. 3214–3221. IEEE (2014)
26. Li, J., Zheng, S., Tan, Y.: The effect of information utilization: Introducing a novel guiding spark in the fireworks algorithm. IEEE Trans. Evol. Comput. **21**(1), 153–166 (2017)
27. Liang, J., Qu, B., Suganthan, P., Hernández-Díaz, A.G.: Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. Technical report 201212(34), Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, pp. 281–295 (2013)
28. Mao, X., Li, Q., Xie, H., Lau, R.Y., Wang, Z., Paul Smolley, S.: Least squares generative adversarial networks. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2794–2802 (2017)
29. Neshat, M., Sepidnam, G., Sargolzaei, M., Toosi, A.N.: Artificial fish swarm algorithm: a survey of the state-of-the-art, hybridization, combinatorial and indicative applications. Artif. Intell. Rev. **42**(4), 965–997 (2014)
30. Nowozin, S., Cseke, B., Tomioka, R.: f-GAN: training generative neural samplers using variational divergence minimization. In: Advances in Neural Information Processing Systems, pp. 271–279 (2016)
31. Qin, A.K., Huang, V.L., Suganthan, P.N.: Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans. Evol. Comput. **13**(2), 398–417 (2009)

32. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
33. Ruder, S.: An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747 (2016)
34. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training GANs. In: Advances in Neural Information Processing Systems, pp. 2234–2242 (2016)
35. Shi, H., Dong, J., Wang, W., Qian, Y., Zhang, X.: SSGAN: secure steganography based on generative adversarial networks. In: Zeng, B., Huang, Q., El Saddik, A., Li, H., Jiang, S., Fan, X. (eds.) PCM 2017. LNCS, vol. 10735, pp. 534–544. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77380-3_51
36. Storn, R., Price, K.: Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**(4), 341–359 (1997)
37. Such, F.P., Madhavan, V., Conti, E., Lehman, J., Stanley, K.O., Clune, J.: Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv preprint arXiv:1712.06567 (2017)
38. Tan, K.C., Chiam, S.C., Mamun, A., Goh, C.K.: Balancing exploration and exploitation with adaptive variation for evolutionary multi-objective optimization. Eur. J. Oper. Res. **197**(2), 701–713 (2009)
39. Tan, Y.: Fireworks Algorithm. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46353-6
40. Tan, Y., Zhu, Y.: Fireworks algorithm for optimization. In: Tan, Y., Shi, Y., Tan, K.C. (eds.) ICSI 2010. LNCS, vol. 6145, pp. 355–364. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13495-1_44
41. Tulyakov, S., Liu, M.Y., Yang, X., Kautz, J.: MoCoGAN: decomposing motion and content for video generation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1526–1535 (2018)
42. Vondrick, C., Pirsiavash, H., Torralba, A.: Generating videos with scene dynamics. In: Advances In Neural Information Processing Systems, pp. 613–621 (2016)
43. Wu, J., Zhang, C., Xue, T., Freeman, B., Tenenbaum, J.: Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In: Advances in Neural Information Processing Systems, pp. 82–90 (2016)
44. Yu, L., Zhang, W., Wang, J., Yu, Y.: SeqGAN: sequence generative adversarial nets with policy gradient. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)
45. Zheng, S., Janecek, A., Li, J., Tan, Y.: Dynamic search in fireworks algorithm. In: 2014 IEEE Congress on Evolutionary Computation (CEC), pp. 3222–3229. IEEE (2014)
46. Zheng, S., Janecek, A., Tan, Y.: Enhanced fireworks algorithm. In: 2013 IEEE Congress on Evolutionary Computation, pp. 2069–2077. IEEE (2013)
47. Zheng, S., Li, J., Janecek, A., Tan, Y.: A cooperative framework for fireworks algorithm. IEEE/ACM Trans. Comput. Biol. Bioinform. (TCBB) **14**(1), 27–41 (2017)
48. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: 2017 IEEE International Conference on Computer Vision (ICCV) (2017)