Evolutionary Gradient Descent for Non-convex Optimization

Ke Xue[1], **Chao Qian**[1*], **Ling Xu**[2] and **Xudong Fei**[2]
[1]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
[2]2012 Lab, Huawei Technologies, Shenzhen 518000, China
{xuek, qianc}@lamda.nju.edu.cn, {xuling7, feixudong}@huawei.com

Abstract

Non-convex optimization is often involved in artificial intelligence tasks, which may have many saddle points, and is NP-hard to solve. Evolutionary algorithms (EAs) are general-purpose derivative-free optimization algorithms with a good ability to find the global optimum, which can be naturally applied to non-convex optimization. Their performance is, however, limited due to low efficiency. Gradient descent (GD) runs efficiently, but only converges to a first-order stationary point, which may be a saddle point and thus arbitrarily bad. Some recent efforts have been put into combining EAs and GD. However, previous works either utilized only a specific component of EAs, or just combined them heuristically without theoretical guarantee. In this paper, we propose an evolutionary GD (EGD) algorithm by combining typical components, i.e., population and mutation, of EAs with GD. We prove that EGD can converge to a second-order stationary point by escaping the saddle points, and is more efficient than previous algorithms. Empirical results on non-convex synthetic functions as well as reinforcement learning (RL) tasks also show its superiority.

1 Introduction

In many real-world tasks such as tensor decomposition [Ge *et al.*, 2015] and training deep neural networks [Dauphin *et al.*, 2014], we often need to solve non-convex optimization problems, which are much more complex than convex problems. In convex optimization, it is sufficient to find a first-order stationary point, which must be global optimal. However, a first-order stationary point can be a saddle point and arbitrarily bad when the problem is non-convex. Thus, for non-convex optimization, it often requires to escape the saddle points and find a second-order stationary point, which can approximate the global optimum well in many cases, e.g., matrix factorization [Li *et al.*, 2016], matrix completion [Ge *et al.*, 2016] and dictionary learning [Sun *et al.*, 2016].

EAs [Bäck, 1996], inspired from natural evolution, are regarded as potential tools for non-convex optimization. They maintain a set of solutions (called a population), and iteratively try to improve the quality of population by using global search operators (e.g., mutation operators) and superior selection strategies. EAs can be applied without the gradient information. In fact, they require little properties on the problem, but have a good ability to find the global optimum due to the characteristic of employing population and global search operators. Thus, EAs can be naturally applied to solve non-convex problems, e.g., subset selection [Qian *et al.*, 2015; Qian *et al.*, 2020], neural architecture search [Elsken *et al.*, 2019; Stanley *et al.*, 2019] and policy optimization in RL [Salimans *et al.*, 2017; Liu *et al.*, 2020]. However, their efficiency may often be limited, particularly when the problem dimensionality becomes high [Hansen *et al.*, 2015].

On the contrary, the first-order algorithm GD, which iteratively updates the solution using gradients, is efficient, but only guaranteed to converge to a first-order stationary point [Nesterov, 1998], which can be arbitrarily bad in non-convex optimization. Though using the second-order information (e.g., the Hessian matrix) can help find second-order stationary points [Nesterov and Polyak, 2006; Curtis *et al.*, 2017], those methods are often computationally expensive.

A natural question is then whether combining the merits of EAs and GD can lead to better algorithms for non-convex optimization. Jin *et al.* [2017; 2019] first introduced the mutation operator of EAs into GD in order to escape saddle points efficiently. They proved that the resulting algorithm called perturbed GD (PGD) can find a second-order stationary point with some probability by using almost the same amount of time that GD takes to find a first-order stationary point. Later, more components (e.g., population) of EAs are combined with GD. The main idea of these methods, e.g., ESGD [Cui *et al.*, 2018] and GADAM [Zhang and Gouza, 2018], is to alternate between the GD step and evolution step. That is, in each iteration, the solutions in the population are updated using gradients and evolutionary operators, respectively, for a predefined number of steps. Though they have shown promising empirical performance, there is no theoretical guarantee on the convergence rate.

*This work was supported by the National Key Research and Development Program of China (2018AAA0101100), the NSFC (62022039) and the Jiangsu NSF (BK20201247). Chao Qian is the corresponding author.

In this paper, we propose a new algorithm EGD by combining the components, i.e., population and mutation, of EAs with GD. EGD uses gradients to update the solutions in the population normally, but applies mutation when the solutions are close to saddle points. That is, EGD applies mutation adaptively. After mutation, EGD will replace worse solutions with the best one in the population for acceleration. We prove that EGD can converge to a second-order stationary point, and is more efficient than PGD. Experimental results on synthetic functions as well as a set of popular RL tasks show the superior performance of EGD.

2 Background

We first give some notations that will be used in the paper. Let $\|\cdot\|$ denote the ℓ_2-norm of vectors as well as the spectral norm of matrices. We use $\lambda_{\min}(\cdot)$ to denote the smallest eigenvalue of a matrix. For a function $f:\mathbb{R}^d\to\mathbb{R}$, let ∇f and $\nabla^2 f$ denote the gradient and Hessian, respectively, and let f^* denote the global minimum of f. We use $B(\mathbf{0},r)$ to denote the ball centered at $\mathbf{0}$ with radius r. The notation $\tilde{O}(\cdot)$ is used to hide constants and factors that are only polylogarithmically dependent on the problem parameters. Let $[N]$ denote the set $\{1,2,\ldots,N\}$.

We consider the general optimization problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}),$$

where $f:\mathbb{R}^d\to\mathbb{R}$ can be non-convex. As in previous studies [Nesterov and Polyak, 2006; Ge *et al.*, 2015; Jin *et al.*, 2017], we make the smooth assumption (i.e., Assumption 1) that f is twice-differentiable, and its gradient and Hessian cannot change too rapidly.

Assumption 1. *The function f is ℓ-gradient Lipschitz and ρ-Hessian Lipschitz.*

Definition 1. *A differentiable function f is ℓ-gradient Lipschitz if*

$$\forall \boldsymbol{x},\boldsymbol{y}:\|\nabla f(\boldsymbol{x})-\nabla f(\boldsymbol{y})\|\le \ell\cdot\|\boldsymbol{x}-\boldsymbol{y}\|.$$

A twice-differentiable function f is ρ-Hessian Lipschitz if

$$\forall \boldsymbol{x},\boldsymbol{y}:\|\nabla^2 f(\boldsymbol{x})-\nabla^2 f(\boldsymbol{y})\|\le \rho\cdot\|\boldsymbol{x}-\boldsymbol{y}\|.$$

When f is convex, it is sufficient to find a first-order stationary point in Definition 2, because it must be global optimal. However, when f is non-convex, a first-order stationary point can be a saddle point (e.g., satisfy $\lambda_{\min}(\nabla^2 f(\boldsymbol{x}))<0$), and thus may be arbitrarily bad. Furthermore, saddle points are ubiquitous in high-dimensional non-convex problems [Dauphin *et al.*, 2014]. Thus, the goal of non-convex optimization is often to escape saddle points and find a second-order stationary point or its ϵ-approximation in Definition 3. Note that a second-order stationary point must be local optimal, and can often approximate the global optimum well [Li *et al.*, 2016; Ge *et al.*, 2016; Sun *et al.*, 2016].

Definition 2. *For a differentiable function f, $\boldsymbol{x}$ is a first-order stationary point if $\|\nabla f(\boldsymbol{x})\|=0$, and is an ϵ-first-order stationary point if $\|\nabla f(\boldsymbol{x})\|\le\epsilon$, where $\epsilon\ge 0$.*

Definition 3. *For a twice-differentiable function f, $\boldsymbol{x}$ is a second-order stationary point if*

$$\|\nabla f(\boldsymbol{x})\|=0, \text{and } \lambda_{\min}(\nabla^2 f(\boldsymbol{x}))\ge 0.$$

For a ρ-Hessian Lipschitz function f, $\boldsymbol{x}$ is an ϵ-second-order stationary point if

$$\|\nabla f(\boldsymbol{x})\|\le\epsilon, \text{and } \lambda_{\min}(\nabla^2 f(\boldsymbol{x}))\ge -\sqrt{\rho\epsilon},$$

where $\epsilon\ge 0$.

Next, we will introduce EAs, GD, and PGD, respectively.

2.1 Evolutionary Algorithms

EAs [Bäck, 1996] are a type of general-purpose heuristic optimization algorithms, which simulate the natural evolution process with variational reproduction (e.g., mutation and crossover) and superior selection. They only require the solutions to be evaluated in order to perform the search, while the problem structure information, e.g., gradient information, can be unavailable. As the mutation operators can often generate any solution in the search space, i.e., they are global search operators, EAs can converge to the global optimum [Rudolph, 1998]. Due to the derivative-free property and the good global search ability, EAs have been applied to solve various non-convex optimization problems, e.g., neural network optimization [Stanley *et al.*, 2019] and policy optimization in RL [Salimans *et al.*, 2017]. Though achieving promising optimization performance, their convergence rate is unsatisfactory, which is usually of $O(1/d)$ [Hansen *et al.*, 2015], thus limiting their application in high-dimensional cases.

2.2 Gradient Descent

Unlike derivative-free algorithms, the first-order algorithm GD uses gradients to update the solution iteratively. It is known that GD can find an ϵ-first-order stationary point in $O(1/\epsilon^2)$ iterations [Nesterov, 1998], which is independent of the problem dimension d. As introduced before, a first-order stationary point, however, can be a saddle point and thus arbitrarily bad in non-convex optimization. In fact, Du *et al.* [2017] have proved that GD requires exponential time to escape saddle points in some situations. Though using the random initialization technique [Lee *et al.*, 2016; Lee *et al.*, 2019] or the second-order information [Nesterov and Polyak, 2006; Curtis *et al.*, 2017] can help avoid saddle points and find second-order stationary points, the resulting methods are often computationally expensive.

2.3 Perturbed Gradient Descent

The PGD algorithm (i.e., Algorithm 1) [Jin *et al.*, 2017; Jin *et al.*, 2019] utilizes the mutation operator of EAs to help GD escape saddle points. That is, when the solution is close to a saddle point, PGD updates the solution using mutation instead of gradients. As shown in line 4 of Algorithm 1, the mutation operator changes the solution by adding a perturbation vector, which is uniformly randomly sampled from the ball $B(\mathbf{0},r)$. In line 3, when $\|\nabla f(\boldsymbol{x}_i)\|\le\epsilon$, i.e., the current gradient is small enough, the solution is regarded as close to a saddle point. Furthermore, to ensure the dynamics similar to GD, the condition of applying mutation in line 3 also requires

Algorithm 1 PGD algorithm

Parameter: learning rate η, mutation strength r, time interval L for mutation, tolerance ϵ, number T of iterations
Process:
1: Initialize the solution $\boldsymbol{x}_0$, set $i = i_{\text{mutate}} = 0$;
2: **while** $i \leq T$ **do**
3: **if** $\|\nabla f(\boldsymbol{x}_i)\| \leq \epsilon$ and $i - i_{\text{mutate}} > L$ **then**
4: $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{\xi}_i$, $\boldsymbol{\xi}_i \sim \text{Uniform}(B(\mathbf{0}, r))$;
5: $i_{\text{mutate}} \leftarrow i$
6: **else**
7: $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i - \eta \nabla f(\boldsymbol{x}_i)$
8: **end if**
9: $i \leftarrow i + 1$
10: **end while**

that the interval between the current iteration and that of last mutation is larger than a predefined value L.

Theorem 1 shows that PGD has some probability to find an ϵ-second-order stationary point by using almost the same time that GD takes to find an ϵ-first-order stationary point.

Theorem 1. *[Jin* et al.*, 2019] Let f satisfy Assumption 1. Let the parameters of PGD satisfy that $\eta = \frac{1}{\ell}$, $r = \frac{\epsilon}{400\iota^3}$ and $L = \frac{\ell}{\sqrt{\rho\epsilon}} \cdot \iota$, where $\iota = c\log(\frac{d\ell(f(\boldsymbol{x}_0)-f^*)}{\rho\epsilon\delta})$, and c is an absolute constant. Then for any $\epsilon, \delta > 0$, after running*

$$\tilde{O}(\ell(f(\boldsymbol{x}_0) - f^*)/\epsilon^2)$$

iterations, PGD will find an ϵ-second-order stationary point with probability at least $1 - \delta_{pgd}$, where

$$\delta_{pgd} = \frac{T^*}{4L} \cdot \frac{\ell\sqrt{d\epsilon}}{r\sqrt{\rho}} \frac{1}{\sqrt{\pi}2^{\iota}\iota} \leq \delta,$$

and $T^* = 8\max\{50\ell(f(\boldsymbol{x}_0) - f^*) \cdot \iota^4, \ell(f(\boldsymbol{x}_0) - f^*)\}/\epsilon^2$.

3 The EGD Algorithm

In this section, we introduce the proposed algorithm EGD, which combines the components, i.e., population and mutation, of EAs with GD. As presented in Algorithm 2, EGD initializes a population of N individual solutions, i.e., $\{\boldsymbol{x}_0^{(p)}\}_{p=1}^N$, and updates them using gradients in parallel (i.e., line 8) until all the individuals are close to saddle points (i.e., line 12). Note that we use $update^{(p)} = 0$ to denote that the p-th individual in the population is close to a saddle point, whose condition in line 5 is the same as that in line 3 of PGD.

Next, EGD will enter the procedure of Algorithm 3 to perform mutation and selection, with the goal of escaping saddle points. Each individual $\boldsymbol{x}_i^{(p)}$ will be mutated by adding a random perturbation vector sampled from $B(\mathbf{0}, r^{(p)})$ in line 2 of Algorithm 3. To judge whether the individual has escaped a saddle point, it is further updated using gradients for L iterations in lines 4–6, and the resulting solution $\boldsymbol{x}_{i+1+L}^{(p)}$ is compared with $\boldsymbol{x}_i^{(p)}$. If $f(\boldsymbol{x}_{i+1+L}^{(p)})$ is smaller than $f(\boldsymbol{x}_i^{(p)})$ by at least ϵ', EGD regards that the individual has escaped successfully, denoted by $escape^{(p)} = 1$. If an individual fails to escape a saddle point, i.e., $escape^{(p)} = 0$, it will return to the state before mutation (i.e., line 10). Furthermore, if such an individual is still bad compared with other individuals in the population, i.e., its f value is higher than the average f value of population as shown in line 16, it will be replaced by the best individual (denoted by $\boldsymbol{x}_{\text{best}}$) in the population. This selection process of EGD may accelerate the optimization.

EGD will repeat the above processes until the number of iterations reaches the budget T. Note that it is often difficult

Algorithm 2 EGD algorithm

Parameter: learning rate η, population size N, mutation strength $\{r^{(p)}\}_{p=1}^N$, time interval L for mutation, tolerance ϵ, ϵ', number T of iterations
Process:
1: Initialize the population $\{\boldsymbol{x}_0^{(p)}\}_{p=1}^N$, set $i = i_{\text{mutate}} = 0$, $update^{(p)} = 1$ for $p \in [N]$;
2: **while** $i \leq T$ **do**
3: **for** $p = 1 : N$ **do**
4: **if** $update^{(p)} = 1$ **then**
5: **if** $\|\nabla f(\boldsymbol{x}_i^{(p)})\| \leq \epsilon$ and $i - i_{\text{mutate}} > L$ **then**
6: $update^{(p)} = 0$
7: **else**
8: $\boldsymbol{x}_{i+1}^{(p)} \leftarrow \boldsymbol{x}_i^{(p)} - \eta \nabla f(\boldsymbol{x}_i^{(p)})$
9: **end if**
10: **end if**
11: **end for**
12: **if** $\forall p \in [N] : update^{(p)} = 0$ **then**
13: Apply Algorithm 3 for mutation and selection;
14: Set $update^{(p)} = 1$ for $p \in [N]$;
15: $i \leftarrow i + L$
16: **end if**
17: $i \leftarrow i + 1$
18: **end while**

Algorithm 3 Mutation and Selection

1: **for** $p = 1 : N$ **do**
2: $\boldsymbol{x}_{i+1}^{(p)} \leftarrow \boldsymbol{x}_i^{(p)} + \boldsymbol{\xi}_i^{(p)}$, $\boldsymbol{\xi}_i^{(p)} \sim \text{Uniform}(B(\mathbf{0}, r^{(p)}))$;
3: $i_{\text{mutate}} \leftarrow i$;
4: **for** $j = (i+1) : (i+L)$ **do**
5: $\boldsymbol{x}_{j+1}^{(p)} \leftarrow \boldsymbol{x}_j^{(p)} - \eta \nabla f(\boldsymbol{x}_j^{(p)})$
6: **end for**
7: **if** $f(\boldsymbol{x}_{i+1+L}^{(p)}) + \epsilon' < f(\boldsymbol{x}_i^{(p)})$ **then**
8: $escape^{(p)} = 1$
9: **else**
10: $escape^{(p)} = 0$, $\boldsymbol{x}_{i+1+L}^{(p)} \leftarrow \boldsymbol{x}_i^{(p)}$
11: **end if**
12: **end for**
13: $f_{\text{mean}} = \frac{1}{N}\sum_{p=1}^N f(\boldsymbol{x}_{i+1+L}^{(p)})$;
14: $\boldsymbol{x}_{\text{best}} = \arg\min_{\boldsymbol{x} \in \{\boldsymbol{x}_{i+1+L}^{(p)}\}_{p=1}^N} f(\boldsymbol{x})$;
15: **for** $p = 1 : N$ **do**
16: **if** $escape^{(p)} = 0$ and $f(\boldsymbol{x}_{i+1+L}^{(p)}) \geq f_{\text{mean}}$ **then**
17: $\boldsymbol{x}_{i+1+L}^{(p)} \leftarrow \boldsymbol{x}_{\text{best}}$
18: **end if**
19: **end for**

to select a proper mutation strength r, i.e., the radius of the ball where the perturbation vector is sampled. Thus, by using different $r^{(p)}$ for different individuals, we can reduce the risk of setting a bad mutation strength, and increase the population diversity, which is crucial for the practical performance of EAs [Li *et al.*, 2010; Tang *et al.*, 2016].

4 Theoretical Analysis

In this section, we analyze the performance of EGD theoretically. Theorem 2 gives a lower bound on the probability of finding an ϵ-second-order stationary point after EGD runs a specific number of iterations.

Theorem 2. *Let f satisfies Assumption 1. Let the parameters of EGD satisfy that $\eta = \frac{1}{\ell}$, $L = \frac{\ell}{\sqrt{\rho\epsilon}} \cdot \iota$ and $\epsilon' = \frac{1}{100\iota^3}\sqrt{\frac{\epsilon^3}{\rho}}$, where $\iota = c\log(\frac{d\ell(f(\boldsymbol{x}_0)-f^*)}{\rho\epsilon\delta})$, and c is an absolute constant. Then for any $\epsilon, \delta > 0$, after running*

$$\tilde{O}(\ell(f(\boldsymbol{x}_0) - f^*)/\epsilon^2)$$

iterations, EGD will find an ϵ-second-order stationary point with probability at least $1 - \delta_{egd}$, where

$$\delta_{egd} = \frac{T^*}{4L} \cdot \prod_{p=1}^{N} \frac{\ell\sqrt{d\epsilon}}{r^{(p)}\sqrt{\rho}} \frac{1}{\sqrt{\pi}2^{\iota}\iota},$$

and $T^* = 8\max\{50\ell(f(\boldsymbol{x}_0) - f^*) \cdot \iota^4, \ell(f(\boldsymbol{x}_0) - f^*)\}/\epsilon^2$.

Proof. The proof is inspired by that of Theorem 1 [Jin *et al.*, 2019]. When an individual $\boldsymbol{x}^{(p)}$ is not an ϵ-second-order stationary point, it must hold that $\|\nabla f(\boldsymbol{x}^{(p)})\| > \epsilon$ or $\lambda_{\min}(\nabla^2 f(\boldsymbol{x}^{(p)})) < -\sqrt{\rho\epsilon}$ according to Definition 3. For the case where $\boldsymbol{x}^{(p)}$ has a large gradient, EGD can reach around a stationary point, i.e., $\|\nabla f(\boldsymbol{x}^{(p)})\| \leq \epsilon$, quickly by GD updating in line 8 of Algorithm 2 and line 5 of Algorithm 3. From the proof in [Jin *et al.*, 2019], we can derive the following two points. For the case where $\|\nabla f(\boldsymbol{x}^{(p)})\| \leq \epsilon$ but $\lambda_{\min}(\nabla^2 f(\boldsymbol{x}^{(p)})) < -\sqrt{\rho\epsilon}$, i.e., $\boldsymbol{x}^{(p)}$ is around a saddle point, the probability of failing to escape the saddle point (i.e., $escape^{(p)} = 0$) after performing lines 2–11 of Algorithm 3 (i.e., one mutation step and $L = \frac{\ell}{\sqrt{\rho\epsilon}} \cdot \iota$ steps of GD updating) is at most

$$\frac{\ell\sqrt{d\epsilon}}{r^{(p)}\sqrt{\rho}} \frac{1}{\sqrt{\pi}2^{\iota}\iota}. \tag{1}$$

Furthermore, it is sufficient to find an ϵ-second-order stationary point after escaping saddle points by at most

$$T^*/(4L) \tag{2}$$

times, as the value of f can decrease by at most $f(\boldsymbol{x}_0) - f^*$.

Next, we order all the saddle points by their function values in descending order. Let $\boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \ldots, \boldsymbol{x}_m^*$ denote the resulting sequence of saddle points, where $\boldsymbol{x}_1^*$ is the worst saddle point and $\boldsymbol{x}_m^*$ is the best. Based on the above analysis, we have $m \leq T^*/(4L)$. We pessimistically assume that when EGD performs Algorithm 3 for the first time, all the individuals $\{\boldsymbol{x}^{(p)}\}_{p=1}^N$ are stuck around the worst saddle point $\boldsymbol{x}_1^*$; after each execution of Algorithm 3, if an individual escapes a saddle point (denoted by $\boldsymbol{x}_i^*$) successfully, it will only arrive around the adjacent better saddle point $\boldsymbol{x}_{i+1}^*$ in the sequence. According to the selection procedure of EGD, i.e., lines 7–11 and lines 13–19 of Algorithm 3, it is not hard to verify that when all the individuals are stuck around the saddle point $\boldsymbol{x}_i^*$ before performing Algorithm 3, if at least one individual escapes successfully, the failed individuals will be replaced by the successful one and arrive around $\boldsymbol{x}_{i+1}^*$; otherwise, all the individuals will keep around $\boldsymbol{x}_i^*$. Thus, before EGD performs Algorithm 3, all individuals in the population will always be around the same $\boldsymbol{x}_i^*$ for some i.

Under the above pessimistic assumption, EGD needs to escape all the saddle points $\boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \ldots, \boldsymbol{x}_m^*$ sequentially for finding an ϵ-second-order stationary point. We consider the event that when performing Algorithm 3, EGD always escapes the current saddle point successfully. Due to Eq. (1) and the fact that EGD fails only if all the individuals fail, the failure probability of one execution of Algorithm 3 is at most

$$\prod_{p=1}^{N} \frac{\ell\sqrt{d\epsilon}}{r^{(p)}\sqrt{\rho}} \frac{1}{\sqrt{\pi}2^{\iota}\iota}.$$

As the number m of saddle points is upper bounded by $T^*/(4L)$ in Eq. (2), by the union bound, the probability of such an event, implying that an ϵ-second-order stationary point has been found, is at least

$$1 - \frac{T^*}{4L} \cdot \prod_{p=1}^{N} \frac{\ell\sqrt{d\epsilon}}{r^{(p)}\sqrt{\rho}} \frac{1}{\sqrt{\pi}2^{\iota}\iota}.$$

Under the above event, Algorithm 3 will be performed by at most $T^*/(4L)$ times. As each execution of Algorithm 3 will cost $(L+1)$ iterations, the total number of iterations is at most

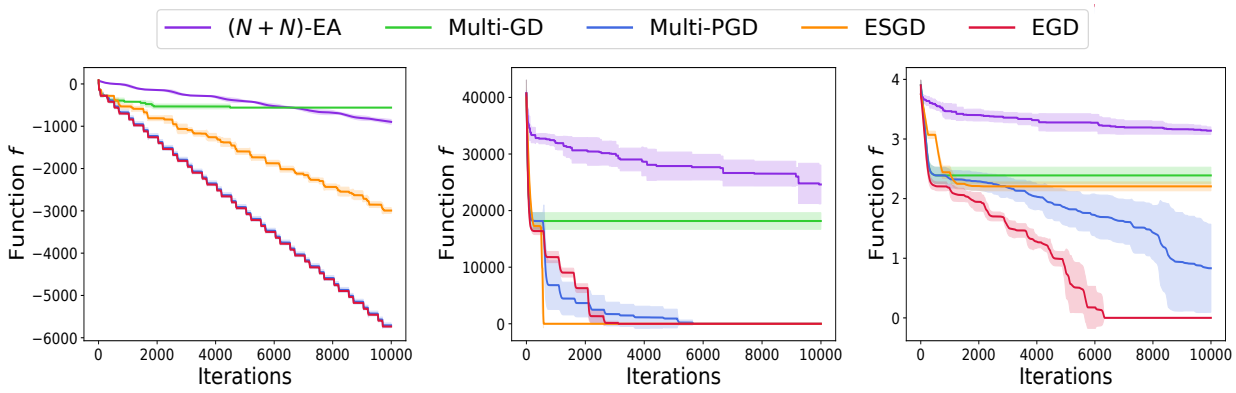
$$(L+1)T^*/(4L) = \tilde{O}(\ell(f(\boldsymbol{x}_0) - f^*)/\epsilon^2),$$

where the equality holds by $T^* = O(\ell(f(\boldsymbol{x}_0) - f^*) \cdot \iota^4/\epsilon^2)$. Note that the step of GD updating in line 8 of Algorithm 2 also costs some iterations. But it has been shown [Jin *et al.*, 2019] that the number of steps of GD updating cannot be larger than

$$T^*/4 = \tilde{O}(\ell(f(\boldsymbol{x}_0) - f^*)/\epsilon^2)$$

by Assumption 1, because otherwise the f value of the found individual will be smaller than the global optimum f^*. Thus, the theorem holds. □

By comparing Theorems 1 and 2, we find that using the same number of iterations, PGD and EGD have different probabilities of finding an ϵ-second-order stationary point, which are $1 - \delta_{pgd}$ and $1 - \delta_{egd}$, respectively. Let $q = \frac{\ell\sqrt{d\epsilon}}{r\sqrt{\rho}} \frac{1}{\sqrt{\pi}2^{\iota}\iota}$, where $r = \frac{\epsilon}{400\iota^3}$, and let $s = T^*/(4L)$. Thus, $\delta_{pgd} = sq$. For EGD, we consider a special case, where $\forall p \in [N] : r^{(p)} = r = \frac{\epsilon}{400\iota^3}$. Thus, the failure probability δ_{egd} of EGD is just $s \cdot q^N$, which is obviously smaller than δ_{pgd} as $q \leq 1$. This implies that EGD is better than PGD.



(a) *Octopus* (b) *Schwefel* (c) *Ackley*

Figure 1: Comparison of the curves averaged over five random seeds for different algorithms on three synthetic functions.

Though PGD and EGD use the same number of iterations in Theorems 1 and 2, EGD employs a population of size N, i.e., updates N solutions in parallel in each iteration, making the comparison unfair. Thus, we consider Multi-PGD, which runs PGD N times independently, and returns the best found solution. Its failure probability, denoted by δ_{mpgd}, is $\delta_{pgd}^N = s^N q^N$, because it requires that all the N independent runs of PGD fail. As $s \geq 1$, $\delta_{egd} = sq^N \leq s^N q^N = \delta_{mpgd}$, implying that EGD is still better than Multi-PGD. According to $\delta_{egd}/\delta_{mpgd} = 1/s^{N-1}$, $s = T^*/(4L) = O((f(\boldsymbol{x}_0) - f^*)\iota^3\sqrt{\rho}/\epsilon^{1.5})$ and $\iota = c\log(\frac{d\ell(f(\boldsymbol{x}_0)-f^*)}{\rho\epsilon\delta})$, we have:

Remark 1. *EGD will have more advantage over Multi-PGD,*

(1) when the problem dimension d, the Lipschitz parameters ℓ and ρ are larger, implying that the problem is more challenging;

(2) when the population size N is larger.

5 Experiments

To examine the performance of EGD, we conduct experiments on different non-convex optimization tasks, including three synthetic functions and four popular RL tasks. We compare EGD with the following algorithms:

- **$(N + N)$-EA** [Hansen *et al.*, 2015] maintains a population of N solutions. In each iteration, it generates N new solutions by mutation, and selects the best N solutions from these new solutions and the current population to form the next population.
- **Multi-GD** runs GD multiple times independently, and returns the best found solution.
- **Multi-PGD** runs PGD multiple times independently, and returns the best found solution. PGD in Algorithm 1 combines the mutation operator of EAs with GD.
- **ESGD** [Cui *et al.*, 2018] combines EAs with GD entirely, by alternating between the GD and EA step.

Note that we use Multi-GD and Multi-PGD, instead of GD and PGD, for fair comparisons, because $(N + N)$-EA, ESGD and EGD are population-based algorithms. By running GD (or PGD) for N times independently and returning the best found solution, where N is the population size, the same computational cost is used.

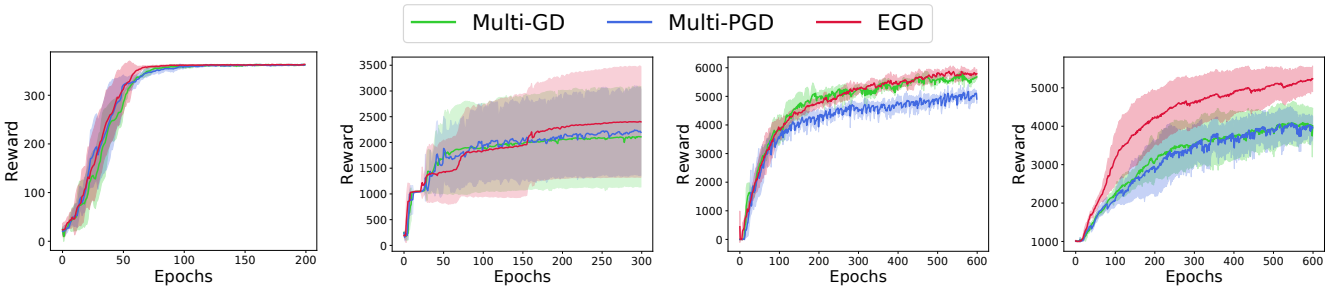
We use identical random seeds (2017, 2018, 2019, 2020, 2021) for all tasks and algorithms. The population size N is always set to 5. To make fair comparisons on each task, Multi-GD, Multi-PGD, ESGD and EGD employ the same learning rate η; $(N + N)$-EA, Multi-PGD and ESGD employ the same mutation strength r, while the N mutation strengths $\{r^{(p)}\}_{p=1}^N$ of EGD are set to uniformly discretized values between r and $1.2r$ (or $1.5r$); the parameters L and ϵ of Multi-PGD and EGD are set to the same.

5.1 Synthetic Functions

First, we compare these algorithms on three synthetic functions: *Octopus*, *Ackley* and *Schwefel*. The *Octopus* function [Du *et al.*, 2017] is designed to have a sequence of saddle points, which must be escaped sequentially for finding the global optimum. *Ackley* and *Schwefel* are two popular non-convex functions with many bad local minima. The dimension of each function is set to 100.

Figure 1 shows the curve (i.e., the change of f value of the best found solution over the number of iterations) of each algorithm on each function. Note that each curve is averaged over five random seeds. We can observe that $(N + N)$-EA and Multi-GD are always the worst, disclosing the benefit of combining EAs with GD. Compared with Multi-PGD, EGD performs better on *Schwefel* and *Ackley*, while performs similarly on the relatively easy function *Octopus*. In fact, the similar performance on *Octopus* is expected, because *Octopus* was designed so that the probability of failing to escape saddle points by mutation is small, and thus Multi-PGD utilizing mutation only has already performed well.

Compared with ESGD, EGD is significantly better on *Octopus* and *Ackley*, while is a little worse on *Schwefel*. During the running of EGD, once one mutation step is performed (i.e., line 2 of Algorithm 3), the steps of gradient updating (i.e., lines 4–6 of Algorithm 3) will be followed. Different from EGD, ESGD can perform several mutation steps continuously. Because the local minima of *Schwefel* are flat, one mutation step may be not sufficient for escaping, leading to



(a) *Swimmer-v2* (b) *Hopper-v2* (c) *HalfCheetah-v2* (d) *Ant-v2*

Figure 2: Comparison of the curves averaged over five random seeds for different algorithms on four MuJoCo locomotion tasks.

Dimension d	200	400	600	800	1000
$f(\boldsymbol{x}) < 2$	1.79	2.05	2.71	2.78	2.88
$f(\boldsymbol{x}) < 1$	1.70	2.04	2.24	2.30	2.34
$f(\boldsymbol{x}) < 0.1$	1.51	1.88	2.10	2.20	2.24

Table 1: On the *Ackley* function with different dimensions, the ratio of the number of iterations of Multi-PGD and EGD until finding a solution with the f value smaller than a threshold in $\{2, 1, 0.1\}$.

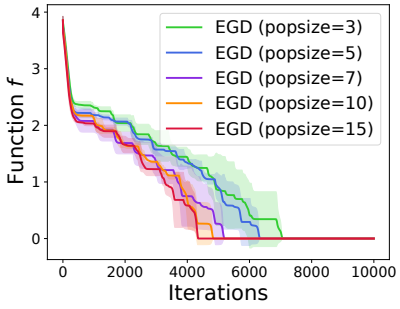


Figure 3: The curves of EGD with different population size N on the *Ackley* function.

the worse performance of EGD. This observation also gives some guide to improve EGD.

Influence of Dimension d. In the above experiments, the dimension d of each function is set to 100. To examine the influence of d, we compare EGD with Multi-PGD on the *Ackley* function with $d \in \{200, 400, 600, 800, 1000\}$. We record the number of iterations of Multi-PGD and EGD (denoted by T_{mpgd} and T_{egd}, respectively) until finding a solution $\boldsymbol{x}$ with $f(\boldsymbol{x})$ smaller than a threshold. Table 1 shows the ratio of T_{mpgd} and T_{egd}. We can observe that under different thresholds, the ratio always increases with the dimension d, implying that the advantage of EGD over Multi-PGD increases with d, consistent with the theoretical result (1) in Remark 1.

Influence of Population Size N. In all experiments, the population size N is 5. To examine the influence of N on the performance of EGD, we run EGD with different values of N on the *Ackley* function. The results in Figure 3 show that when the population size N increases, EGD will converge to the global optimum faster, consistent with Theorem 2 that the failure probability δ_{egd} of EGD decreases with N.

5.2 Reinforcement Learning Tasks

Next, we examine the performance of EGD on four MuJoCo locomotion tasks: *Swimmer-v2*, *Hopper-v2*, *HalfCheetah-v2* and *Ant-v2* [Todorov *et al.*, 2012]. The primary goal in RL is to find an optimal policy that can maximize the expected total reward. The linear policy structure is employed here, since it is sufficient to capture diverse behaviors on the MuJoCo tasks [Mania *et al.*, 2018]. For these tasks, the gradient cannot be obtained exactly, and we use the vanilla ES gradient estimator, where the smoothing parameter and the sample size are set as recommended by Mania *et al.* [2018]. Note that $(N+N)$-EA and ESGD require evaluating the objective value (i.e., the expected total reward) frequently, which is very time-consuming in RL. Thus, we only compare Multi-GD, Multi-PGD and EGD due to limited computing resources.

The results are shown in Figure 2. It can be observed that EGD always performs the best, while Multi-PGD even does not show the advantage over Multi-GD. In RL tasks, the estimated gradient is not exact, making that the condition, i.e., $\|\nabla f(\boldsymbol{x}_i)\| \leq \epsilon$, of using mutation cannot be judged correctly. Thus, the misuse of mutation leads to the bad performance of Multi-PGD. Though EGD may also misuse mutation, the employed population mechanism increases the robustness and keeps the best performance of EGD.

6 Conclusion

In this paper, we propose a new algorithm EGD for non-convex optimization. The idea of EGD is to combine the derivative-free EAs with the first-order algorithm GD. EGD maintains a population of solutions, and updates the solutions using gradients normally but using mutation when the solutions are close to saddle points. In theory, EGD can converge to a second-order stationary point more efficiently than previous algorithms. In experiments, EGD shows the superior performance on non-convex optimization tasks, including synthetic benchmark functions and RL tasks.

This work is a preliminary attempt to combine EAs with gradient-based algorithms. EGD only uses the two components, i.e., population and mutation, of EAs. It would be interesting to incorporate crossover operators, a characterizing feature of EAs [Qian *et al.*, 2013; Huang *et al.*, 2019], into EGD, which may further improve the performance. EGD combines EAs with the basic GD algorithm. It would also be interesting to combine EAs with advanced variants of GD.

References

[Bäck, 1996] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[Cui *et al.*, 2018] X. Cui, W. Zhang, Z. Tüske, and M. Picheny. Evolutionary stochastic gradient descent for optimization of deep neural networks. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, pages 6048–6058, Montréal, Canada, 2018.

[Curtis *et al.*, 2017] F. E. Curtis, D. P. Robinson, and M. Samadi. A trust region algorithm with a worst-case iteration complexity of $\mathcal{O}(\epsilon^{-3/2})$ for nonconvex optimization. *Mathematical Programming*, 162(1-2):1–32, 2017.

[Dauphin *et al.*, 2014] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems 27 (NIPS)*, pages 2933–2941, Montréal, Canada, 2014.

[Du *et al.*, 2017] S. S. Du, C. Jin, J. D. Lee, M. I. Jordan, A. Singh, and B. Poczos. Gradient descent can take exponential time to escape saddle points. In *Advances in Neural Information Processing Systems 30 (NIPS)*, pages 1067–1077, Long Beach, CA, 2017.

[Elsken *et al.*, 2019] T. Elsken, J. H. Metzenn, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[Ge *et al.*, 2015] R. Ge, F. Huang, C. Jin, and Y. Yuan. Escaping from saddle points - online stochastic gradient for tensor decomposition. In *Proceedings of the 28th Conference on Learning Theory (COLT)*, pages 797–842, Paris, France, 2015.

[Ge *et al.*, 2016] R. Ge, J. D. Lee, and T. Ma. Matrix completion has no spurious local minimum. In *Advances in Neural Information Processing Systems 29 (NIPS)*, pages 2973–2981, Barcelona, Spain, 2016.

[Hansen *et al.*, 2015] N. Hansen, D. Arnold, and A. Auger. Evolution strategies. In *Springer Handbook of Computational Intelligence*, pages 871–898. Springer, 2015.

[Huang *et al.*, 2019] Z. Huang, Y. Zhou, Z. Chen, and X. He. Running time analysis of MOEA/D with crossover on discrete optimization problem. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 2296–2303, Honolulu, HI, 2019.

[Jin *et al.*, 2017] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan. How to escape saddle points efficiently. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1724–1732, Sydney, Australia, 2017.

[Jin *et al.*, 2019] C. Jin, P. Netrapalli, R. Ge, S. M. Kakade, and M. I. Jordan. On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points. *CoRR abs/1902.04811*, 2019.

[Lee *et al.*, 2016] J. D. Lee, M. Simchowitz, M. I. Jordan, and B. Recht. Gradient descent only converges to minimizers. In *Proceedings of the 29th Conference on Learning Theory (COLT)*, pages 1246–1257, New York, NY, 2016.

[Lee *et al.*, 2019] J. D. Lee, I. Panageas, G. Piliouras, M. Simchowitz, M. I. Jordan, and B. Recht. First-order methods almost always avoid strict saddle points. *Mathematical Programming*, 176(1-2):311–337, 2019.

[Li *et al.*, 2010] M. Li, J. Zheng, K. Li, Q. Yuan, and R. Shen. Enhancing diversity for average ranking method in evolutionary many-objective optimization. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 647–656, Krakow, Poland, 2010.

[Li *et al.*, 2016] X. Li, Z. Wang, J. Lu, R. Arora, J. Haupt, H. Liu, and T. Zhao. Symmetry, saddle points, and global geometry of nonconvex matrix factorization. *CoRR abs/1612.09296*, 2016.

[Liu *et al.*, 2020] F.-Y. Liu, Z.-N. Li, and C. Qian. Self-guided evolution strategies with historical estimated gradients. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1474–1480, Yokohama, Japan, 2020.

[Mania *et al.*, 2018] H. Mania, A. Guy, and B. Recht. Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, pages 1805–1814, Montréal, Canada, 2018.

[Nesterov and Polyak, 2006] Y. Nesterov and B. T. Polyak. Cubic regularization of Newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.

[Nesterov, 1998] Y. Nesterov. *Introductory Lectures on Convex Programming Volume I: Basic Course*. Springer, 1998.

[Qian *et al.*, 2013] C. Qian, Y. Yu, and Z.-H. Zhou. An analysis on recombination in multi-objective evolutionary optimization. *Artificial Intelligence*, 204:99–119, 2013.

[Qian *et al.*, 2015] C. Qian, Y. Yu, and Z.-H. Zhou. Subset selection by Pareto optimization. In *Advances in Neural Information Processing Systems 28 (NIPS)*, pages 1765–1773, Montréal, Canada, 2015.

[Qian *et al.*, 2020] C. Qian, C. Bian, and C. Feng. Subset selection by Pareto optimization with recombination. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, pages 2408–2415, New York, NY, 2020.

[Rudolph, 1998] G. Rudolph. Finite Markov chain results in evolutionary computation: A tour d'horizon. *Fundamenta Informaticae*, 35(1-4):67–89, 1998.

[Salimans *et al.*, 2017] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *CoRR abs/1703.03864*, 2017.

[Stanley *et al.*, 2019] K. O Stanley, J. Clune, J. Lehman, and R. Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[Sun *et al.*, 2016] J. Sun, Q. Qu, and J. Wright. Complete dictionary recovery over the sphere I: Overview and the geometric picture. *IEEE Transactions on Information Theory*, 63(2):853–884, 2016.

[Tang *et al.*, 2016] K. Tang, P. Yang, and X. Yao. Negatively correlated search. *IEEE Journal on Selected Areas in Communications*, 34(3):542–550, 2016.

[Todorov *et al.*, 2012] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Proceedings of the 25th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, Vilamoura, Portugal, 2012.

[Zhang and Gouza, 2018] J. Zhang and F. B. Gouza. GADAM: Genetic-evolutionary ADAM for deep neural network optimization. *CoRR abs/1805.07500*, 2018.