The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Noisy Derivative-Free Optimization with Value Suppression*

Hong Wang, Hong Qian, Yang Yu
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210023, China
waghon@outlook.com, {qianh,yuy}@lamda.nju.edu.cn

Abstract

Derivative-free optimization has shown advantage in solving sophisticated problems such as policy search, when the environment is noise-free. Many real-world environments are noisy, where solution evaluations are inaccurate due to the noise. Noisy evaluation can badly injure derivative-free optimization, as it may make a worse solution looks better. Sampling is a straightforward way to reduce noise, while previous studies have shown that delay the noise handling to the comparison time point (i.e., threshold selection) can be helpful for derivative-free optimization. This work further delays the noise handling, and proposes a simple noise handling mechanism, i.e., *value suppression*. By value suppression, we do nothing about noise until the best-so-far solution has not been improved for a period, and then suppress the value of the best-so-far solution and continue the optimization. On synthetic problems as well as reinforcement learning tasks, experiments verify that value suppression can be significantly more effective than the previous methods.

Introduction

Solving sophisticated and complex optimization tasks plays a crucial role in artificial intelligence. Let $f\colon \mathcal{X} \subseteq \mathbb{R}^n \to \mathbb{R}$ be an objective function and we assume that a global minimum value x^* always exists. An optimization task can be formulized as $x^* = \operatorname{argmin}_{x\in\mathcal{X}} f(x)$. This paper regards f as a black-box function and focuses on derivative-free optimization methods. Namely, f can be evaluated point-wisely, and optimization methods are performed only on the basis of the function values $f(x)$ and the sampled solutions x. Because derivative-free methods do not rely on derivatives, they are especially suitable for sophisticated optimization problems which have many local optima and are non-differentiable. Successful examples include hyper-parameter tuning in machine learning, direct policy search in reinforcement learning, and scientific experiments (Snoek, Larochelle, and Adams 2012; Hu, Qian, and Yu 2017; Parkinson, Mukherjee, and Liddle 2006). Under noise-free environments (i.e., solution evaluations are accurate), derivative-free optimization methods have achieved desirable and impressive performance. However, many real-world environments are *noisy* and we can only access noisy objective function $f^N(x)$ instead of true objective function $f(x)$. For example, every prototype is evaluated by simulations in industrial design such as VLSI design (Guo et al. 2014). The result is inaccurate due to the simulation error. And the evaluation of a prediction model in machine learning can be inaccurate because of the limited amount of data it uses (Qian, Yu, and Zhou 2015b). Inaccurate solution evaluations resulting from noise can badly injure derivative-free optimization methods, since it may make a worse solution looks better and mislead the search process.

There are two popular mechanisms to handle noise in derivative-free optimization: sampling and threshold selection equipped with re-evaluation. Sampling is a straightforward way to reduce noise (Arnold and Beyer 2006). It independently evaluates a given solution multiple times, and then uses the average of noisy function values to approximate the true function value. In order to achieve relatively accurate estimate of true function value, more sample size (i.e., times of function evaluations) is needed. Since in derivative-free optimization function evaluation often requires expensive computational and time cost (He and Yao 2001; Yu and Zhou 2008), thus, the time and computational cost of sampling is high. Another popular mechanism to handle noise is threshold selection equipped with re-evaluation (Markon et al. 2001; Beielstein and Markon 2002; Jin and Branke 2005; Goh and Tan 2007; Gießen and Kötzing 2016). It independently reevaluates solutions whenever the comparison among solutions occurs, and replaces an old solution only when the value of a new solution is smaller than that of the old one by at least a fixed or dynamic threshold value. This mechanism delays the noise handling to the comparison time, and has been shown to be helpful for derivative-free optimization (Qian, Yu, and Zhou 2015a). Compared with sampling, due to the delay of noise handling, threshold selection equipped with re-evaluation requires less computational and time cost and seems to be more efficient.

In this paper, we further delay the noise handling in derivative-free optimization process, and propose a generic, simple yet efficient noise handling mechanism called *value suppression*. This mechanism can cooperate with most

*This research was supported by National Key Research and Development Program (2017YFB1001903), NSFC (61422304), Jiangsu SF (BK20170013), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

derivative-free optimization methods. By value suppression, we do nothing about noise until the best-so-far solution has not been improved for a period, and then suppress the value of the best-so-far solution and continue the optimization. The value suppression mechanism can not only help derivative-free optimization keep updating their best-so-far solutions, but also record some suppressed solutions, from which we can pick up the best solution whose value is more reliable. Compared with sampling and threshold selection equipped with re-evaluation, value suppression may require least computational and time cost and still be effective. Furthermore, we inject this mechanism into one state-of-the-art derivative-free algorithm SRACOS (Hu, Qian, and Yu 2017), and result in the suppressed SRACOS (SSRACOS) for optimization under noise. To compare value suppression with the other mechanisms on SRACOS, we conduct experiments on two synthetic functions and controlling tasks of reinforcement learning in OpenAI Gym. Experimental results indicate that, for noisy derivative-free optimization, the proposed value suppression mechanism can improve the performance of SRACOS more significantly than the others.

The consequent sections respectively introduce the related work, describe the proposed value suppression mechanism for noisy optimization, present the empirical results, and finally conclude the paper.

Related Work

Handling noise for derivative-free optimization, there are two main mechanisms: sampling and threshold selection equipped with re-evaluation.

Sampling. One popular mechanism to reduce noise is sampling (Arnold and Beyer 2006). Given a solution $x \in \mathcal{X}$, it independently evaluates this solution n times, and then uses the average of noisy function values to approximate the true function value (i.e., $f(x) \approx \frac{1}{n}\sum_{i=1}^{n} f^N(x)$). Sampling can reduce the standard deviation of noise and make the value estimation closer to true value. In order to achieve relatively accurate estimate of true function value, more sample size (i.e., times of function evaluations) is needed. Since in derivative-free optimization function evaluation often requires expensive computational and time cost (He and Yao 2001; Yu and Zhou 2008), thus, the time and computational cost of sampling is high. To reduce the cost of sampling, many smart sampling strategies have been proposed like adaptive and sequential ways, which decide n in each iteration dynamically (Aizawa and Wah 1994; Stagge 1998; Branke and Schmidt 2004). However, these smart sampling strategies handle noise at once without delay, and thus their cost is still relative high.

Threshold Selection Equipped with Re-evaluation. Another popular mechanism to handle noise is threshold selection equipped with re-evaluation (Markon et al. 2001; Beielstein and Markon 2002; Bartz-Beielstein 2005; Jin and Branke 2005; Goh and Tan 2007; Doerr, Hota, and Kötzing 2012). It independently reevaluates solutions whenever the comparison among solutions occurs, and replaces an old solution only when the value of a new solution is smaller than that of the old one by at least a threshold value τ. re-evaluation can prevent the fake better solutions misleading the searching, as they are reevaluated and the fake advantage resulting from noise will be reduced with high probability. Threshold selection can reduce the risk of accepting a bad solution resulting from noise, and increase the chance of keeping good solutions. Choosing a proper threshold value is crucial. It usually requires domain knowledge (Qian, Yu, and Zhou 2015a). Recently, parent populations technique, which can be regarded as adjusting threshold dynamically, is proposed (Gießen and Kötzing 2016). It maintains a set of best-so-far solutions, so as long as a solution is better than the worst one in the set, it will be kept even though it is worse than the best solution. Utilizing threshold selection alone has the negative effect that if some fake better solution is accepted, it will be hard to rule it out (Qian, Yu, and Zhou 2015a). Hence, threshold selection equipped with re-evaluation is more desirable. This mechanism delays the noise handling to the comparison time, and has been shown to be helpful for derivative-free optimization (Qian, Yu, and Zhou 2015a). Compared with sampling, due to the delay of noise handling, threshold selection equipped with re-evaluation requires less computational and time cost and seems to be more efficient.

In a nutshell, sampling mechanism provides a way to evaluate solutions more accurately. But it takes expensive computational and time cost, and may lose its power and be inefficient when the solution evaluations budget is limited. Threshold selection equipped with re-evaluation shares the merit of threshold selection helping reduce the risk of good solution being replaced by fake better solutions, and re-evaluation helping rule out some fake better solution. This mechanism delays the noise handling to the comparison time point, and thus requires less computational and time cost and seems to be more efficient. On the other hand, since re-evaluation is computationally expensive (a solution may be evaluated multiple times), there still exists room to further improve the efficiency.

Value Suppression for Noisy Optimization

In this section, we first introduce the proposed generic value suppression mechanism, and show how it can be applied to derivative-free algorithms. Compared with other two popular noise handling mechanisms, it needs less computational cost (i.e., less function evaluations). Then, we embed the value suppression in one state-of-the-art derivative-free optimization algorithm SRACOS (Hu, Qian, and Yu 2017), and result in the suppressed SRACOS (SSRACOS).

Value Suppression

This paper considers minimization problems in noisy environments. The idea of value suppression arises from the phenomenon that if an algorithm has not updated its best-so-far solution for a period, then the observed value of the best-so-far solution is likely to be much smaller than its true value. Thus we suppress the value of the best-so-far solution when it stays the best-so-far for a long period, then the algorithm could be resumed to work and find a better solution.

The principle of value suppression is to suppresses the overestimated value towards the true value. One way to im-

plement this is to re-sample the value of the solution sufficient times, in the situation of unbiased random noise. In other situations, the suppression can be implemented by, e.g., multiplying a discount factor.

This simple mechanism can help the algorithm keep generating new solutions with better observed values. However, when the optimization finishes, the algorithm needs to choose the best solution among those it has generated. Since most of the solutions have only a noisy observed value, we thus only choose the best solution from the suppressed solutions whose values are more reliable.

Algorithm 1 Value Suppression Framework for Derivative-Free Optimization

Input:
f^N: Noisy objective function.
Procedure:
1: $S =$ generate a set of solutions and evaluate them
2: $S^+ =$ best k solutions in S
3: **while** termination condition is not met **do**
4: $x =$ generate a new solution based on S^+
5: evaluate x and use it to update S^+
6: **if** S^+ does not update for a period **then**
7: suppress the function values of solutions in S^+
8: **end if**
9: **end while**
10: **return** the best among all the suppressed solutions

The value suppression mechanism can be easily applied to derivative-free optimization algorithms which keep tracking the best-so-far solution. The framework of derivative-free optimization with the value suppression is shown in Algorithm 1. Firstly, the algorithm samples a set of solutions S and evaluates them (line 1). Let S^+ denote the best k solutions in S (line 2). In the following loop, the algorithm generates a new solution based on S^+, evaluates it and uses it to update S^+ (line 4 to 5). If S^+ does not update for a period, it suppresses the values of these samples in S^+ by re-sampling, and saves these suppressed solutions (line 6 to 8). At last, the algorithm returns the best among all the suppressed solutions (line 10).

Under mild conditions, we can prove that Algorithm 1 is convergent. This indicates that the value suppression does not hurt optimization and is effective.

Theorem 1 (Convergence)**.** *For a derivative-free optimization algorithm $\mathcal{A}$ that generates any solution $x \in \mathcal{X}$ with non-zero probability, assume that the noise follows the same i.i.d. and unbiased distribution for all solutions, and the value suppression assigns the true value to the solution, then the algorithm $\mathcal{A}$ with value suppression is convergent under noise, i.e., with probability 1 it will eventually output the optimal solution x^*.*

Proof. By the assumption, once the true optimal solution x^* is generated during optimization under noise, x^* could be better than the best-so-far solution with a non-zero probability. Thus, by Algorithm 1, x^* could be absorbed into S^+ with probability 1 after a sufficient number of steps. Let S' denote the set of suppressed solutions and is initialed as $S' = \emptyset$. For a fixed maximum allowed non-update iterations u, there exists a non-zero probability that S^+ will not be updated during u iterations, and thus x^* could be further absorbed into S' with probability 1 after a sufficient number of steps. By Algorithm 1, the solutions cannot be removed from S' once absorbed into. Note that the value suppression step for solutions in S' discloses the true function values as assumed. Since the algorithm will finally return the best in S', i.e., x^*, thus the value suppression is convergent. □

Algorithm 2 SRACOS

Input:
f: Objective function to be minimized;
$\mathcal{C}$: A binary classification algorithm;
λ: Balancing parameter;
$m \in \mathbb{N}^+$: Sample size in each iteration;
$k \in \mathbb{N}^+ (\leq m)$: Number of positive samples;
$r = m + k$;
$N \in \mathbb{N}^+$: Budget;
`Sampling`: Sampling sub-procedure;
`Selection`: Decide the positive/negative solutions;
`Replace`: Replacing sub-procedure.
Procedure:
1: Collect $S = \{x_1, \ldots, x_r\}$ by i.i.d. sampling from $\mathcal{U}_X$
2: $B = \{(x_1, y_1), \ldots, (x_r, y_r)\}, \forall x_i \in S : y_i = f(x_i)$
3: $(B^+, B^-) =$ `Selection`$(B; k)$
4: Let $(\tilde{x}, \tilde{y}) = \operatorname{argmin}_{(x,y)\in B^+} y$
5: **for** $t = r + 1$ to N **do**
6: $h = \mathcal{C}(B^+, B^-)$
7: $x = \begin{cases} \texttt{Sampling}(\mathcal{U}_{D_h}) & \text{w.p. } \lambda \\ \texttt{Sampling}(\mathcal{U}_X) & \text{w.p. } 1 - \lambda \end{cases}$
8: $y = f(x)$
9: $[(x', y'), B^+] =$ `Replace`$((x, y), B^+,$ 'strategy_P')
10: $[, B^-] =$ `Replace`$((x', y'), B^-,$ 'strategy_N')
11: $(\tilde{x}, \tilde{y}) = \operatorname{argmin}_{(x,y)\in B^+ \cup \{(\tilde{x},\tilde{y})\}} y$
12: **end for**
13: **return** $(\tilde{x}, \tilde{y})$

SSRACOS: SRACOS with Value Suppression

We implement the generic value suppression framework with one state-of-the-art derivative-free optimization algorithm SRACOS (Hu, Qian, and Yu 2017). The procedure of SRACOS is presented in Algorithm 2. SRACOS maintains two sets of solutions: good solutions set (positive set) and bad solutions set (negative set). A binary classifier is trained on the basis of these two solution sets to learn the potential high-quality region in the solution space. The learned region contains one selected good solution in the positive set and rules out all the bad solutions in the negative set. After that, a new solution is uniformly sampled from this learned region with high probability or uniformly sampled from the whole solution space with the remaining probability. Then, SRACOS evaluates this new sampled solution and updates both the positive and negative sets.

Algorithm 3 Suppressed SRACOS (SSRACOS)

Input: (extra input than SRACOS)
$u \in \mathbb{N}^+$: Maximum allowed non-update iterations;
$v \in \mathbb{N}^+$: Re-sample size;
α: Balancing parameter;
`Re-sample`: Re-sample sub-procedure.

Procedure:
1: initialize SRACOS
2: $B^S = \emptyset$
3: **for** $t = r + 1$ to $N - v$ **do**
4: (x, y) = generate a new solution as in SRACOS
5: use (x, y) to update $B^+, B^-, (\tilde{x}, \tilde{y})$ in SRACOS
6: **if** B^+ does not update for u iterations **then**
7: **for** (x_i, y_i) in B^+ **do**
8: $\hat{y}_i = \texttt{Re-sample}(x_i, v)$
9: $y_i = (1 - \alpha)y_i + \alpha\hat{y}_i$
10: $B^S = B^S \cup \{(x, \hat{y}_i)\}$
11: $t = t + v$
12: **end for**
13: **end if**
14: **end for**
15: re-sample $(\tilde{x}, \tilde{y})$ and put it in B^S
16: **return** $\text{argmin}_{(x,\hat{y})\in B^S} \hat{y}$

To initialize, SRACOS samples a batch of solutions and evaluates their function values to form a solution-value tuple set B (line 1 to 2). After that, `Selection` sub-procedure is used to split B into B^+ and B^- according to the function values of solutions, where the positive set B^+ is consisted of the best k solutions and the negative set B^- is consisted of the rest (line 3). The best-so-far solution-value tuple is recorded (line 4). In the following loop, SRACOS trains a binary classifier $\mathcal{C}$ to learn a region which contains a randomly selected positive solution in B^+ and rules out all the negative solutions in B^- (line 6). More details about the classifier $\mathcal{C}$ can be found in (Yu, Qian, and Hu 2016). Then, a new solution is uniformly sampled from this learned region with probability λ or uniformly sampled from the whole solution space with probability $1 - \lambda$ (line 7). The function value of this new sampled solution is evaluated (line 8), and B^+, B^-, as well as the best-so-far solution-value tuple are updated accordingly (line 9 to 11). `Replace`$(a, A,$'s') sub-procedure replaces a tuple in the set A with a according to a strategy 's'. There are three strategies which are proposed in (Hu, Qian, and Yu 2017): replacing the worst solution in A (WR-), replacing a solution in A randomly (RR-), and replacing the solution in A which has the largest margin from the best-so-far solution (LM-). Note that 'strategy_P' can only be 'WR-', and 'strategy_N' can be any one of these three strategies. At last, the best found solution-value tuple $(\tilde{x}, \tilde{y})$ is returned as output (line 13).

We now show how to inject the proposed value suppression into SRACOS, and result in the suppressed SRACOS (SSRACOS). Since we can observe the positive set B^+ in SRACOS, if B^+ does not update for a period, we suppress these solutions in B^+. In the end, the best solution among all the suppressed ones is returned as output. The procedure of SSRACOS is presented in Algorithm 3. The set B^S is used to collect the suppressed solutions and is initialized as the empty set (line 2). During the loop, after generating a new solution and updating $B^+, B^-, (\tilde{x}, \tilde{y})$, SSRACOS checks if the positive set B^+ has been updated (line 6). If it does not update for u iterations, SSRACOS re-samples the solutions in B^+, suppresses their values, and saves them with their mean values $\hat{y}$ in B^S (line 7 to 12). `Re-sample`(x, n) sub-procedure computes $f^N(x)$ for n times independently, and returns the mean value $\hat{y}$. After that, it re-samples the best-so-far solution $(\tilde{x}, \tilde{y})$ and saves it in B^S (line 15). At last, the best solution in B^S is returned (line 16).

Experiments

This section shows the effectiveness of the value suppression mechanism in reducing the negative effect of noise and saving the cost empirically. The value suppression is compared with the other noise handling mechanisms mentioned above. We inject these mechanisms into SRACOS respectively. Specifically, SSRACOS with the number of solutions in the positive set $\#B^+ > 1$ is abbreviated to VS. SRACOS with $\#B^+ > 1$ is abbreviated to MPS. SRACOS with $\#B^+ = 1$ is abbreviated to NO_MPS. SSRACOS with $\#B^+ = 1$ is abbreviateNd to VS+NO_MPS. SRACOS equipped with sampling is abbreviated to SAMPLING. SAMPLING evaluates a solution n times and uses the average value to approximate its true function value (Arnold and Beyer 2006). SRACOS equipped with re-evaluation is abbreviated to REEVAL. REEVAL makes an independent evaluation of a solution whenever the function value is required (Jin and Branke 2005; Goh and Tan 2007; Doerr, Hota, and Kötzing 2012). SRACOS equipped with threshold selection is abbreviated to TS. TS regards a solution better than another only when its function value is better than that of another by at least a threshold τ (Markon et al. 2001; Beielstein and Markon 2002; Bartz-Beielstein 2005). SRACOS equipped with the combination of re-evaluation and threshold selection is abbreviated to REEVAL+TS.

We conduct experiments on both synthetic functions and reinforcement learning controlling tasks in OpenAI Gym to investigate their ability of noise handling. Additive Gaussian noise is used to create a noisy environment for synthetic functions. OpenAI Gym environment is considered to be noisy because a policy has different total rewards under different initial states (more details can be found in the subsection of On Controlling Tasks in OpenAI Gym). In addition to the noise from the original environment, we also add extra Gaussian noise to observe their performances under different noise levels. Moreover, we also analysis the sensitivity of the hyper-parameter maximum non-update iterations u on OpenAI Gym tasks.

On Synthetic Functions

We choose Ackley function and Sphere function to investigate the ability of noise handling for each mechanism. Ackley function is defined as:

$$f(x) = -20e^{(-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-0.2)^2})} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi(x_i-0.2)} + e + 20,$$

Table 1: The function value for each noise handling mechanism. For Ackley function, we set the dimension size $n = 100$ and 1000, the standard deviation of noise $\sigma = 0.1$. For Sphere function, we set the dimension size n =100 and 1000, the standard deviation of noise $\sigma = 1$. The number of function evaluations is set to be $200,000$.

Function_DimSize_Noise	VS	VS+NO_MPS	SAMPLING	MPS	REEVAL+TS	REEVAL	TS	NO_MPS
Ackley_100_0.1	**0.93**	2.43	1.32	2.95	3.71	3.82	3.75	3.69
Ackley_1000_0.1	**3.82**	3.90	3.93	3.96	3.99	4.01	4.01	3.99
Sphere_100_1	**4.17**	7.41	6.53	8.74	20.65	24.07	24.88	15.41
Sphere_1000_1	**72.41**	104.81	81.78	97.16	196.14	246.22	294.42	172.98

and Sphere function is defined as:

$$f(x) = \sum_{i=1}^{n} (x_i - 0.2)^2.$$

We choose the dimension sizes $n = 100$ and 1000 for both functions in the experiment. To create a noisy environment, we use additive Gaussian noise, i.e., the noisy function $f^N(x) = f(x) + \mathcal{N}(0, \sigma^2)$. For Ackley function, the standard deviation $\sigma = 0.1$, and for Sphere function $\sigma = 1$.

The parameters of these noise handling mechanisms are set as follows. For threshold selection, we set the threshold value $\tau = \sigma$, because a solution that passes the threshold may be truly better with high probability. For MPS, the number of positive solutions is set to be 5, which is a trade-off between the computational cost and the chance of keeping good solutions. For sampling, the sample size is set to be 10, which is a trade-off between the accuracy of function evaluation and the computational cost. For value suppression, we set the maximum allowed non-update iterations $u = 500$, the re-sample size $v = 100$, and the balance parameter $\alpha = 0.5$. Other parameters are set as default.

For each setting with different dimension size n or standard deviation of noise σ, we run each mechanism 10 times independently to minimize the noisy function $f^N(x)$. The total number of function evaluations is set to be $200,000$. The true function value $f(x)$ of the best found solution during the process is shown in Figure 1, and the true function value of the returned best found solution at last is listed in Table 1. SRACOS with the number of solutions in the positive set $\#B^+ = 1$ (NO_MPS) is chosen as the baseline. From the results, we can find that VS achieves the best performance in all the settings. SAMPLING and MPS show a similar performance and are able to reduce the effects of noise. VS+NO_MPS performs better than MPS, and is competitive with SAMPLING. However, REEVAL, TS and REEVAL+TS are worse than the baseline or not significantly different. From Figure 1, we can observe that VS needs significantly less iterations to achieve a good performance than the other mechanisms. Specifically, on the Ackley function with dimension size 100 and noise level 0.1, we can find that VS only needs less than half function evaluations to have the function value reach below 2 compared with the second best mechanism SAMPLING. This indicates that the the proposed VS mechanism is able to significantly reduce the computational and time cost.

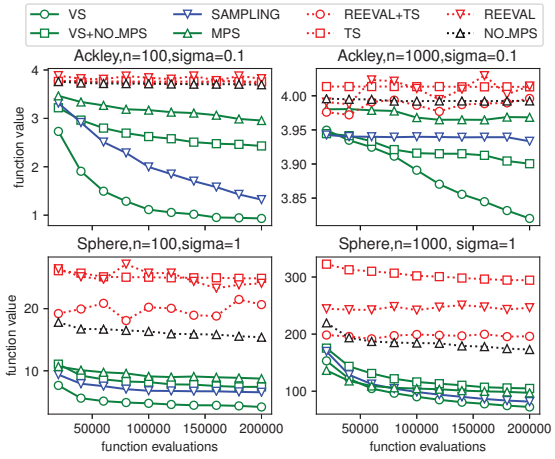


Figure 1: The function value of each noise handling mechanism during the optimization process. For Ackley function, we set the dimension size $n = 100$ and 1000, the standard deviation of noise $\sigma = 0.1$. For Sphere function, we set the dimension size $n = 100$ and 1000, the standard deviation of noise $\sigma = 1$.

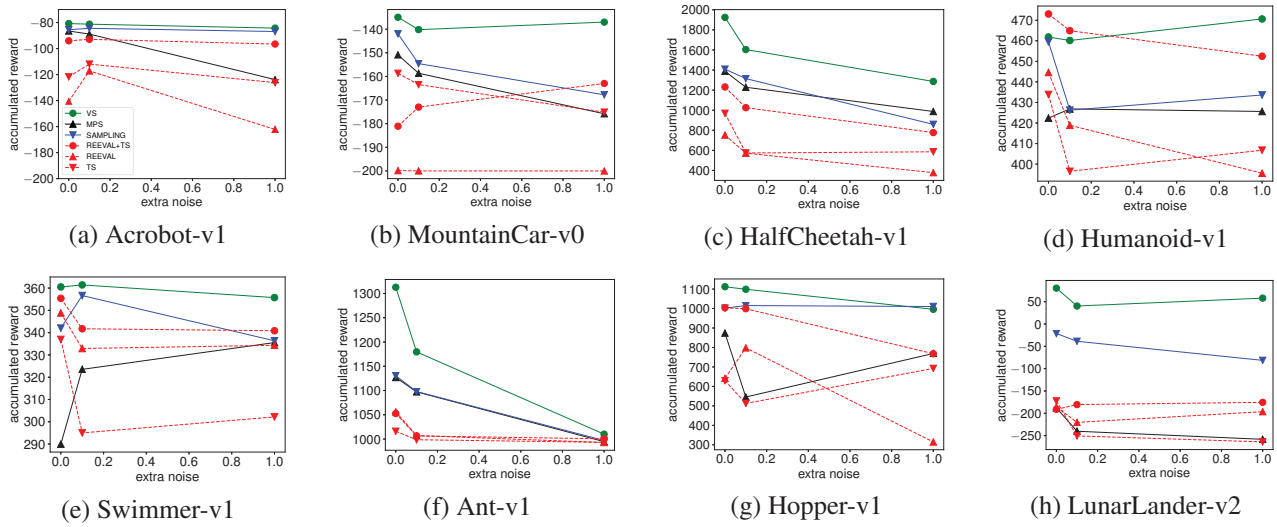
Table 2: Parameters of the Gym tasks

Task	d_{State}	#Actions	NN nodes	#Weights	Horizon
Acrobot-v1	6	1	5, 3	48	500
MountainCar-v0	2	1	5	15	200
HalfCheetah-v1	17	6	10	230	1,000
Humanoid-v1	376	17	25	9825	1,000
Swimmer-v1	8	2	5, 3	61	1,000
Ant-v1	111	8	15	1785	1,000
Hopper-v1	11	3	9, 5	159	1,000
LunarLander-v2	8	1	5, 3	58	1,000

On Controlling Tasks in OpenAI Gym

OpenAI Gym provides a toolkit for reinforcement learning research (https://gym.openai.com). There are many controlling tasks, from which we choose Acrobot, MountainCar, HalfCheetah, Humanoid, Swimmer, Ant, Hopper, and LunarLander to compare the ability of reducing the effects of noise for each noise handling mechanism.

We use the framework of direct policy search to solve these tasks. Direct policy search applies optimization algorithms to search on the parameter space of a policy which is



(a) Acrobot-v1 (b) MountainCar-v0 (c) HalfCheetah-v1 (d) Humanoid-v1

(e) Swimmer-v1 (f) Ant-v1 (g) Hopper-v1 (h) LunarLander-v2

Figure 2: Comparison of the performance under extra noises of 0, 0.1, and 1 times of the noise level respectively

Table 3: Parameters of SRACOS and noise level

Task	$\#B^-$	$\#B^+$	U-bits	Noise Level
Acrobot-v1	20	2	1	28.0
MountainCar-v0	20	2	1	10.0
HalfCheetah-v1	50	3	3	200.0
Humanoid-v1	20	2	3	56.0
Swimmer-v1	50	4	2	10.0
Ant-v1	20	2	3	46.0
Hopper-v1	50	6	4	60.0
LunarLander-v2	50	5	3	50.0

often presented by a neural network (El-Fakdi, Carreras, and Palomeras 2005). The object of the optimization is to maximize the accumulated reward of a policy. Specifically, a policy is represented by a neural network, which is constructed by an input layer of the observation of the state, an output layer of the available action and several other hidden layers. In each step, an agent will receive an observation of the state, and implement an action according to its policy. After that, it will get the reward of that action together with the observation of the next state. This interaction can be repeated until the maximum step is reached or the game is over. The accumulated reward is used to evaluate the performance of a policy. The agent would have different accumulated rewards if the initial state is reset to be different. Therefore, we regard that the environment is noisy. To summarize, our goal is to find the best parameter w for this neural network so as to achieve the best performance. But the difficulty lies in that the accumulated reward $f^N(w)$ used to evaluate its performance can be noisy during the optimization. Thus, we use the noise handling mechanisms to reduce the effect of noise in this environment and compare their performances. The settings of neural network and OpenAI Gym tasks is listed in Table 2, where $\mathrm{d}_{\mathrm{State}}$, #Actions, NN nodes, #Weights and Horizon denote the dimension size of observation, the dimension size of action, the hidden layers of the neural network, the total number of parameters in the neural network and the maximum step, respectively.

We compare these mechanisms under the same parameter setting of SRACOS, which is listed in Table 3, where $\#B^-$ and $\#B^+$ denote the size of negative set and positive set respectively, and U-bits denotes the number of bits that can be changed when generating a new solution from a positive solution. From the experimental results of synthetic functions, we note that VS achieves the best performance. Thus, we combine the other mechanisms with MPS to see if they can improve the performance of MPS better compared with the value suppression. On OpenAI Gym tasks, the total number of function evaluations is set to be $20,000$.

The parameters of these mechanisms are set as follows. For sampling, the sample size is set to be 10. For threshold selection, the noise level is estimated in order to choose a proper threshold value. To estimate the standard deviation of the noise, we draw 10 samples from the solution space, evaluate each sample 1000 times independently and compute the standard deviation. The average standard deviation of these 10 samples is used to estimate the standard deviation of the noise. The values are listed in Table 3 as noise level σ. We round these estimated values to the nearest integers, and set the threshold value $\tau = \sigma$. For value suppression, we set the maximum allowed non-update iterations $u = 500$, the resample size $v = 100$, and the balancing parameter $\alpha = 0.5$.

For each mechanism, the optimization algorithm is independently run for 10 times. And in the end of each run, the average accumulated reward of 1000 simulations is used to estimate the performance of the found policy. The mean ϵ and standard deviation of the 10 policies are reported in Table 4. The mean value of a mechanism is in bold type if not significantly worse than that of the mechanism with the maximal mean value under t-test. We firstly compute the difference of 10 pair results of two mechanisms, $\Delta_i = \epsilon_i^A - \epsilon_i^B$, then we compute the mean μ_Δ and standard deviation σ_Δ of

Table 4: The mean value and the standard deviation of reward achieved by each mechanism. Each method is repeated 10 times. The standard deviation of additional Gaussian noise is set to be 0, 0.1, and 1 times of the noise level in Table 3 respectively. The mark ↓ means in this task the smaller reward the better, and ↑ means the larger reward the better. The mean value of a mechanism is in bold type if not significantly worse than that of the mechanism with the maximal mean value under t-test, where the significance level $\gamma = 10\%$.

Additional Noise	Task	VS	SAMPLING	REEVAL+TS	REEVAL	TS	MPS
0	Acrobot-v1↓	**80.76**±1.38	85.51±3.46	94.03±13.10	140.58±120.76	121.74±40.45	86.50±4.45
	MountainCar-v0↓	**134.92**±3.87	141.94±8.29	181.07±22.61	199.86±0.40	158.67±17.10	150.85±13.33
	HalfCheetah-v1↑	**1924.60**±278.08	1408.54±383.85	1231.46±209.50	752.38±346.83	968.50±427.66	1388.27±479.94
	Humanoid-v1↑	**461.85**±23.92	**459.53**±22.81	**473.05**±34.70	444.60±39.12	433.87±32.57	422.40±41.84
	Swimmer-v1↑	**360.51**±3.45	342.02±20.25	355.40±3.38	348.84±9.70	336.94±16.33	289.97±71.70
	Ant-v1↑	**1312.85**±90.16	1130.54±55.35	1052.64±95.64	1056.09±78.96	1016.05±36.28	1126.89±123.11
	Hopper-v1↑	**1111.91**±117.69	1002.03±48.55	1003.73±12.84	641.23±406.58	630.02±242.86	873.87±186.46
	LunarLander-v2↑	**80.40**±54.51	-21.23±91.65	-191.23±45.77	-185.59±24.45	-172.36±97.17	-187.70±107.00
0.1	Acrobot-v1↓	**81.26**±1.44	**84.45**±5.48	92.91±11.18	117.18±61.64	111.94±51.53	88.87±5.12
	MountainCar-v0↓	**140.18**±9.20	**154.56**±24.52	172.97±22.88	200.00±0.04	163.45±21.21	158.65±16.69
	HalfCheetah-v1↑	**1603.95**±469.26	**1314.68**±674.56	1025.08±372.57	572.12±755.55	573.43±687.99	1228.45±579.65
	Humanoid-v1↑	**460.15**±25.12	426.24±19.89	**464.92**±28.66	418.89±45.39	396.49±39.76	426.85±21.51
	Swimmer-v1↑	**361.42**±2.38	356.64±4.22	341.74±18.11	332.89±40.37	294.99±54.03	323.52±39.40
	Ant-v1↑	**1179.63**±93.48	1097.92±78.16	1006.45±22.73	1007.23±18.04	998.59±3.10	1097.26±95.47
	Hopper-v1↑	**1098.84**±92.24	1015.18±38.21	998.98±13.98	797.96±288.05	512.60±273.42	545.55±251.03
	LunarLander-v2↑	**40.39**±65.07	-38.60±69.64	-180.50±25.73	-220.36±81.02	-250.69±107.65	-240.22±79.50
1	Acrobot-v1↓	**84.28**±2.05	86.89±2.54	96.53±12.12	162.16±119.12	126.18±52.74	123.88±77.87
	MountainCar-v0↓	**136.99**±5.02	167.77±27.89	162.97±22.92	200.02±0.34	175.12±24.49	175.80±16.00
	HalfCheetah-v1↑	**1286.29**±440.46	858.87±468.68	776.81±400.41	377.91±585.26	585.48±611.87	**987.11**±424.63
	Humanoid-v1↑	**470.63**±36.79	**433.63**±39.01	**452.51**±37.53	395.58±57.60	406.76±43.01	425.62±33.90
	Swimmer-v1↑	**355.73**±5.86	336.34±38.57	340.85±16.27	334.31±12.90	302.17±35.86	335.54±20.73
	Ant-v1↑	**1010.00**±15.27	997.07±3.57	**1000.48**±9.38	992.90±3.55	993.30±2.98	994.67±2.64
	Hopper-v1↑	**996.14**±70.48	**1010.32**±30.11	767.96±350.41	314.72±339.94	692.43±190.35	769.20±285.23
	LunarLander-v2↑	**58.02**±74.66	-81.39±73.79	-175.45±18.49	-196.36±70.33	-264.02±126.33	-258.27±91.00

Δ. If $\tau_{10} = |\sqrt{10}\mu_\Delta/\sigma_\Delta| > 1.833$, which is the critical value of the two-tailed t-test with degree of freedom $v = 9$ and significance level $\gamma = 10\%$, the two mechanisms are considered as significantly different; otherwise they are not. On the task of Acrobot-v1 and MountainCar-v0, the smaller the mean value is, the better the performance is. On the other tasks, the larger the better. SRACOS with the number of solutions in the positive set $\#B^+ > 1$ (MPS) is chosen as the baseline for comparison. We can find that VS performs best on all the tasks, while SAMPLING and REEVAL+TS get the best performance only on the Humanoid-v1 task. REEVAL, TS and REEVAL+TS perform even worse than the baseline on some tasks. Since VS achieves the best performance within a given solution evaluation budget, compared with the other mechanisms, it needs the least computational and time cost and thus is the most efficient one.

We then further add Gaussian noise on these tasks and observe their performances under additional noise. Two experiments are conducted on different levels of extra noise. For the first one, the standard deviation of additional Gaussian noise is set to be 0.1 times of the noise level in Table 3. For the second one, the standard deviation is set to be 1 times of the noise level. We keep the other parameters as same as the above experiment for OpenAI Gym. SRACOS with the number of solutions in the positive set $\#B^+ > 1$ (MPS) is chosen as the baseline for comparison. The result is listed in Table 4, and the comparison of mechanisms on different extra noise is shown in Figure 2. We can observe that VS achieves the best or the same best performance under t-test in all the tasks, although in tasks like Ant-v1 and HalfCheetah-v1, it performs worse than the environment without extra noise, it achieves almost the same or even better performance in other tasks. However, SAMPLING and REEVAL+TS perform worse as the extra noise increases in most tasks. And in tasks like HalfCheetah-v1 and Swimmer-v1, they do not perform better than the baseline which does not handle noise.

Table 5: Hyper-parameter analysis of maximum allowed non-update iterations u, where $u \in \{100, 500, 1000\}$. The mark ↓ means in this task the smaller reward the better, and ↑ means the larger reward the better. The mean value of a mechanism is in bold type if not significantly worse than that of the mechanism with the maximal mean value under t-test, where the significance level $\gamma = 10\%$.

Task	500	100	1000
Acrobot-v1↓	**80.76**±1.38	**79.52**±2.54	82.06±1.39
MountainCar-v0↓	**134.92**±3.87	**132.30**±4.28	**134.96**±4.52
HalfCheetah-v1↑	**1924.60**±278.08	1554.27±486.50	**1773.89**±548.06
Humanoid-v1↑	**461.85**±23.92	**460.39**±34.97	**455.46**±35.26
Swimmer-v1↑	**360.51**±3.45	**360.91**±2.33	**359.35**±5.09
Ant-v1↑	**1312.85**±90.16	**1239.24**±119.53	1181.04±91.45
Hopper-v1↑	**1111.91**±117.69	**1046.35**±27.49	**1058.87**±30.77
LunarLander-v2↑	**80.40**±54.51	-23.02±62.22	**21.04**±80.90

Hyper-parameter Analysis

We also investigate the sensitivity of hyper-parameter u, i.e., the maximum allowed non-update iterations, in OpenAI Gym. The experimental setting is as same as that without extra noise, and u is chosen from $\{100, 500, 1000\}$. In the

previous experiments, u is always set to be 500. The results are shown in Table 5.

Table 5 indicates that the results are not significantly different in most tasks whenever $u \in \{100, 500, 1000\}$. This implies that the hyper-parameter maximum allowed non-update iterations u is indeed not so sensitive. Moreover, if u is too small, the confidence that the solution is trapped due to the noise is low, and thus it may waste samples to accurately evaluate the solution that would be replaced soon. If u is too large, the confidence is high, but it may waste samples in waiting the confidence. Therefore, the choice of u should be balanced. According to the results of experiments, the default setting $u = 500$ should be fine in many cases.

Conclusion

In many real-world applications such as policy search in reinforcement learning, the environment is noisy and noise can badly injure the performance of derivative-free optimization methods. This paper proposes a generic, simple yet effective noise handling mechanism called value suppression. The value suppression can be embedded into most derivative-free optimization methods to handle and reduce noise. To verify the effectiveness of this mechanism, we inject it into one state-of-the-art derivative-free optimization algorithm SRACOS and result in suppressed SRACOS (SSRACOS). Experimental results in both synthetic functions and controlling tasks in OpenAI Gym indicate that value suppression could perform better than other popular noise handling mechanisms, e.g., sampling and threshold selection equipped with re-evaluation. In the future, we will further explore if value suppression can be helpful under noise-free environment. Intuitively, value suppression may help the algorithm jump out of the local optima even if there is no noise.

References

Aizawa, A. N., and Wah, B. W. 1994. Scheduling of genetic algorithms in a noisy environment. *Evolutionary Computation* 2(2):97–122.

Arnold, D. V., and Beyer, H.-G. 2006. A general noise model and its effects on evolution strategy performance. *IEEE Transactions on Evolutionary Computation* 10(4):380–391.

Bartz-Beielstein, T. 2005. Evolution strategies and threshold selection. In *International Workshop on Hybrid Metaheuristics*, 104–115.

Beielstein, T., and Markon, S. 2002. Threshold selection, hypothesis tests, and doe methods. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*, 777–782.

Branke, J., and Schmidt, C. 2004. Sequential sampling in noisy environments. In *Parallel Problem Solving from Nature VIII*, 202–211.

Doerr, B.; Hota, A.; and Kötzing, T. 2012. Ants easily solve stochastic shortest path problems. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, 17–24.

El-Fakdi, A.; Carreras, M.; and Palomeras, N. 2005. Direct policy search reinforcement learning for robot control. In *Proceedings of the 8th International Conference of the CCIA*, 9–16.

Gießen, C., and Kötzing, T. 2016. Robustness of populations in stochastic environments. *Algorithmica* 75(3):462–489.

Goh, C. K., and Tan, K. C. 2007. An investigation on noisy environments in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation* 11(3):354–381.

Guo, W.; Liu, G.; Chen, G.; and Peng, S. 2014. A hybrid multi-objective pso algorithm with local search strategy for vlsi partitioning. *Frontiers of Computer Science* 8(2):203–216.

He, J., and Yao, X. 2001. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence* 127(1):57–85.

Hu, Y.-Q.; Qian, H.; and Yu, Y. 2017. Sequential classification-based optimization for direct policy search. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2029–2035.

Jin, Y., and Branke, J. 2005. Evolutionary optimization in uncertain environments-a survey. *IEEE Transations on Evolutionary Computation* 9(3):303–317.

Markon, S.; Arnold, D. V.; Back, T.; Beielstein, T.; and Beyer, H.-G. 2001. Thresholding-a selection operator for noisy es. In *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, 465–472.

Parkinson, D.; Mukherjee, P.; and Liddle, A. R. 2006. Bayesian model selection analysis of wmap3. *Physical Review D* 73(12):123523.

Qian, C.; Yu, Y.; and Zhou, Z.-H. 2015a. Analyzing evolutionary optimization in noisy environments. *Evolutionary Computation* 1–41.

Qian, C.; Yu, Y.; and Zhou, Z.-H. 2015b. Pareto ensemble pruning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2935–2941.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2951–2959.

Stagge, P. 1998. Averaging efficiently in the presence of noise. In *Parallel Problem Solving from Nature V*, 188–197.

Yu, Y., and Zhou, Z.-H. 2008. A new approach to estimating the expected first hitting time of evolutionary algorithms. *Artificial Intelligence* 172(15):1809–1832.

Yu, Y.; Qian, H.; and Hu, Y.-Q. 2016. Derivative-free optimization via classification. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2286–2292.