LEARNING SAMPLING POLICY TO ACHIEVE FEWER QUERIES FOR ZEROTH-ORDER OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Zeroth-order (ZO) methods, which use the finite difference of two function evaluations (also called ZO gradient) to approximate first-order gradient, have attracted much attention recently in machine learning because of its broad applications. The accurateness of ZO gradient highly depends on how many finite differences are averaged, which are intrinsically determined by the number of perturbations randomly drawn from a distribution. Existing ZO methods try to learn a data-driven distribution for sampling the perturbations to improve the efficiency of ZO optimization (ZOO) algorithms. In this paper, we explore a new and parallel direction, *i.e.*, learn an optimal sampling policy instead of using totally random strategy to generate perturbations based on the techniques of reinforcement learning (RL), which makes it possible to approximate the gradient with only two function evaluations. Specifically, we first formulate the problem of learning a sampling policy as a Markov decision process. Then, we propose our ZO-RL algorithm, *i.e.*, using deep deterministic policy gradient, an actor-critic RL algorithm to learn a sampling policy which can guide the generation of perturbed vectors in getting ZO gradients as accurate as possible. Importantly, the existing ZOO algorithms of learning a distribution can be plugged in to improve the exploration of ZO-RL. Experimental results with different ZO estimators show that our ZO-RL algorithm can effectively reduce the query complexity of ZOO algorithms and converge faster than existing ZOO algorithms especially in the later stage of the optimization process.

1 INTRODUCTION

Gradient based optimization is dominant in machine learning. However, in many fields of science and engineering, explicit gradients $\nabla f(x)$ are difficult or even infeasible to obtain. Zeroth-order (ZO, also known as derivative-free) optimization, where the optimizer is provided with only function values $f(x)$ and use the finite difference Berahas et al. (2021) (also called ZO gradient) to approximate the explicit (first-order) gradients $\nabla f(x)$. ZO optimization (ZOO) has a lot applications including hyperparameter optimization Gu et al. (2021); Koch et al. (2018) and bandit problems (Bubeck & Cesa-Bianchi, 2012; Lattimore & Gyorgy, 2021). One of the famous applications is to generate prediction-evasive adversarial examples in the black-box setting Liu et al. (2018a); Papernot et al. (2017), *e.g.*, crafted images with imperceptible perturbations to deceive a well-trained image classifier into misclassification. Thus, ZOO has attracted a lot of attention in recent years due to a growing requirement in real-world applications.

Specifically, the *ZO gradient* is usually with the following formula.

$$\hat{\nabla} f(x) = \frac{1}{\mu q} \sum_{i=1}^{q} [f(x + \mu u_i) - f(x)] u_i \tag{1}$$

where $\mu > 0$ is the smoothing parameter, $\{u_i\}_{i=1}^q$ are the random perturbed vectors drawn from a distribution $p(u)$ which could be a Gaussian distribution or a uniform distribution on unit sphere Nesterov & Spokoiny (2017); Duchi et al. (2012). Essentially, if the smoothing parameter μ approaches infinitesimal, $\lim_{\mu \to 0} f(x + \mu u) - f(x)$ is exactly calculating the directional derivative $\nabla_u f(x)$ along a direction u, which means we use q directional derivatives to approximate the ground truth gradient $\nabla f(x)$ due to $\nabla f(x) = \int_u (\nabla_u f(x) u) p(u) \mathrm{d}u$ Nesterov & Spokoiny (2017). From

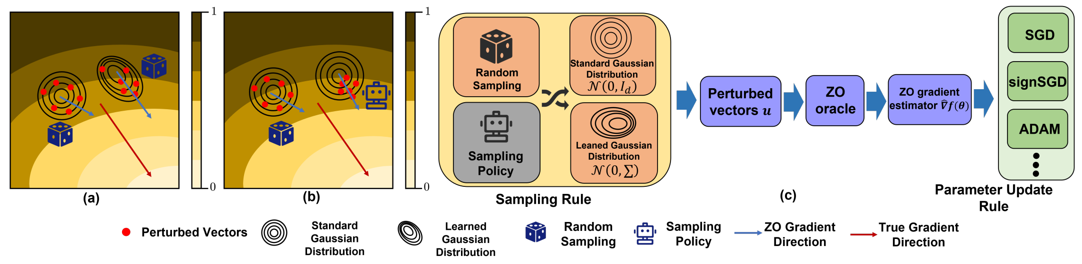


Figure 1: (a) Comparison of the ZO gradient directions obtained by sampling perturbed vectors from the standard Gaussian distribution and a learned Gaussian distribution. (b) Comparison of the ZO gradient directions obtained by a random sampling policy and a learned sampling policy. (c) The architecture of ZO optimizer.

this perspective, the accurateness of ZO gradient highly depends on the number of perturbed vectors randomly sampled from the distribution. For a practical ZOO algorithm, *e.g.*, we should balance the number q of perturbed vectors and the accurateness of ZO gradient to make the overall query complexity[1] of ZOO algorithm as less as possible.

To improve the efficiency of ZOO algorithms, there have been much effort made recently. The main idea of these efforts Maheswaranathan et al. (2019); Ruan et al. (2019) is trying to learn a non-isotropic Gaussian distribution $\mathcal{N}(0, \Sigma)$ (the co-variance matrix Σ may not be a scale of the identity matrix as shown in Fig. 1.a.) to generate perturbed vectors, instead of using the standard Gaussian distribution $\mathcal{N}(0, I)$. By using the learned Gaussian distribution, one more accurate ZO gradient can be obtained for a fixed query budget, which could improve the convergence of ZOO algorithms. Specifically, a lot of evolutionary strategies (ES) Maheswaranathan et al. (2019) such as Natural ES Wierstra et al. (2008), CMA-ES Hansen (2006), and Guided ES Maheswaranathan et al. (2019) were proposed to let co-variance matrix Σ track a low-dimensional subspace and further use this to guide the sampled perturbed vectors. Ruan et al. (2019) utilized RNN to learn an adaptive co-variance matrix Σ and dynamically guide the sampled perturbed vectors.

As discussed above, learning a good sampling distribution helps to calculate a more accurate ZO gradient. Parallel to learn a sampling distribution, learning a sampling policy to generate perturbed vectors instead of using multiple randomly sampled perturbed vectors (please refer to Fig 1.b.) would also be a promising direction for accelerate ZO algorithms. Theoretically, if the perturbation vector u' is with the same direction of $\nabla f(x)$, we can have that the directional derivative $\nabla_{u'} f(x)$ is equal to the true gradient $\nabla f(x)$, i.e., $\nabla_{u'} f(x) u' = \nabla f(x)$. Thus, $\nabla f(x)$ can be approximated by the ZO gradient with only two queries as follows.

$$\hat{\nabla} f(x) = \frac{1}{\mu}[f(x + \mu u') - f(x)]u' \tag{2}$$

The key is to design some mechanism to find such perturbed vector u' which is an unstudied problem in the community as far as we know.

In this paper, we explore Reinforcement Learning (RL) to learn a good sampling policy to solve the above issue. We propose a ZOO algorithm based on RL (ZO-RL) to learn a sampling policy using the policy gradient algorithm. Specifically, we first formulate the problem of learning a sampling policy as a Markov decision process (MDP). Then, we use an actor-critic algorithm called deep deterministic policy gradient (DDPG) Lillicrap et al. (2015), with two neural network function approximators to learn a sampling policy which can guide the generation of perturbed vectors in getting ZO gradients as accurate as possible. Importantly, the existing ZOO algorithms of learning a distribution can be plugged in to improve the exploration of ZO-RL. Experimental results on three ZOO applications show that our ZO-RL algorithm can effectively reduce the query complexity of ZOO algorithms and converge faster than existing ZOO algorithms especially in the later stage of the optimization process.

[1]The overall number of queries of function evaluations is called query complexity.

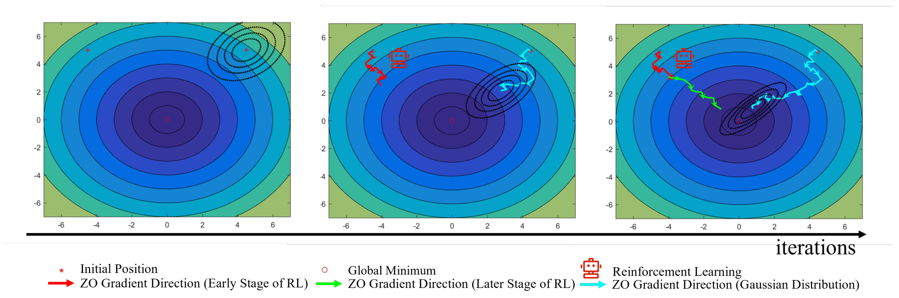


Figure 2: Illustration of the change of ZO gradient direction.

2 Learning Sampling Policy for ZOO

2.1 Principle of Using RL to Accelerate ZOO

As it regains its popularity recently because of star projects like AlphaGo Wang et al. (2016) and AlphaStar Arulkumaran et al. (2019), RL is a dominant framework which can continuously optimize a policy by maximizing the expected cumulative reward while interacting with the environment. If we take the mechanism of generating perturbed vectors as the RL agent whose goal is to make the direction of perturbed vector close to the true gradient. The target function f and the updating rules of ZOO algorithm are all parts of the environment. At each step, the agent picks a perturbation vector and pass it over to the environment for execution. The environment takes its step to calculate ZO gradient and update x_k and output x_{k+1} as the new observation for the RL agent. The RL agent then receives a reward of $f(x_{k+1}) - f(x_k)$. Please refer to Fig. 3 for the above procedure. Basically, this is a typical RL problem, which can be solved by model-free RL methods. The work we present here can also be regarded as the work of model-free learning to optimize (L2O) Chen et al. (2021); Li & Malik (2016), which aims to learn a better optimization algorithm by observing its execution. The RL is an effective way to learn policies in a data-driven manner based on the execution of algorithms.

Intuitive reason of the advantage of ZO-RL. Even RL can work on ZOO as mentioned above, why ZO-RL can beat the traditional ZOO algorithms in terms of query complexity? We try to answer this question intuitively in the below. In practice, our extensive experimental results (some of them are provided in appendix) support this advantage.

Essentially, ZO-RL uses one perturbed vector generated by the learned policy function and two function queries to approximate the true gradient as shown in (2). Ideally, if the agent becomes smart in selecting perturbed vectors, which could produce a competitive approximation of the truth gradient as accurate as the traditional ZO gradient (1) with $q + 1$ function queries. Thus, an effective prediction of gradient oriented perturbed vector by RL can reduce the query complexity and speed up the convergence of the ZOO algorithms. As show in Fig 2, in the later stage of RL training, the RL agent could learn to reduce the random behavior (i.e., be smart) which makes ZO-RL possible to save query complexity compared to the traditional ZOO algorithms.

2.2 Formulation with MDP

We consider the problem of finding a sampling policy that encourages the sampled perturbed vectors to be more efficient in calculating the ZO gradients. As discussed in previous subsection, RL provides a framework in which the agent can learn the best action to take by subsequently receiving rewards from the environment with which it interacts. Specifically, at each query of ZO-RL algorithm, the agent outputs the perturbed vectors u' according to the current state x_k. Then, the agent receives rewards and next state x_{k+1} by interacting with the environment, and learns the sampling policy to maximize rewards. We will use MDP to formalize this procedure with the tuple $(\mathcal{S}, \mathcal{A}, \mathbb{P}_{sa}, R)$ as follows.

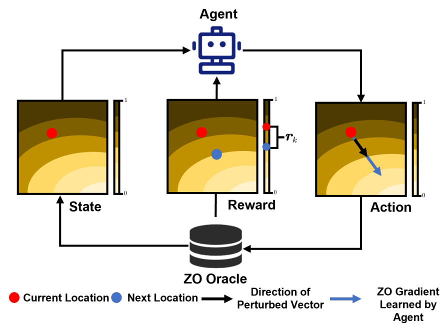


Figure 3: Illustration of learning sampling policy for ZOO base on RL.

1. State space $\mathcal{S} \subset \mathbb{R}^d$: We choose $s_k = x_k \in \mathcal{S}$ to describe the current state of ZO algorithm, where x_k is the point location for the k-th iteration [2]. The state space $\mathcal{S}$ could be $\mathbb{R}^d$ or some constraint set (e.g. $||x||_2 \leq 1$) which depends on the target problem.
2. Action space $\mathcal{A} \subset \mathbb{R}^d$: We choose $a_k = u' \in \mathcal{A}$ as the action, where u' is a perturbed vector. The action space $\mathcal{A}$ could be $\mathbb{R}^d$ regarding Gaussian distribution or the surface of unit sphere regarding uniform distribution on unit sphere. Note that, no matter which kind of action space considered, all of them are continuous.
3. Transition probability $\mathbb{P}_{sa} = \mathbb{P}(\cdot|s, a)$: We do not know the transition probability. We will use model-free RL methods to learn the policy.
4. Reward function $R(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: We consider the reward function $R(s_k, a_k) = r_k = f(x_{k+1}) - f(x_k)$ as the difference between the function values at the current point location x_k and the immediately preceding point location x_{k+1} after the action a_k is performed, which encourages the learned policy to reach the minimum of the function value as quickly as possible.

We also provide a visualization of MDP in Figure 3. Through the framework of MDP, we will propose a RL based ZOO algorithm to learn an action policy to maximize the expected cumulative reward, which will be discussed in detail in the following. The optimal policy is to sample perturbed vectors along the ground-truth gradient direction of the unknown function.

2.3 ZO-RL Algorithm

Actor-Critic Method. As mentioned above, the action space is continuous in our ZOO problem. We adopt the deterministic sampling policy π, because it has the advantages of requiring less data to be sampled compared with the stochastic policy gradient algorithm. Thus, we can achieve higher efficiency for the algorithm and have stable performance in a series of tasks with continuous action space.

To find the optimal policy to approach the true gradient direction, we use deep deterministic policy gradient (DDPG) to learn sampling policy π. DDPG is an actor-critic and model-free algorithm Konda & Tsitsiklis (2000); Lillicrap et al. (2015) for RL over continuous action spaces and output deterministic actions in a stochastic environment to maximize cumulative rewards.

The DDPG has two neural network function approximators. One is called the actor network which learns a deterministic sampling policy $\pi(x|\theta^\pi)$ with neural network weights θ^π. The other is called the critic network, which outputs a state-action value function $Q(s, a|\theta^Q)$ with neural network weights θ^Q to evaluate the value of the action performed.

In addition, DDPG creates a copy of the actor and critic networks, $Q'(s, a|\theta'^Q)$ with neural network weights θ'^π and $\pi'(x|\theta'^\pi)$ with neural network weights θ'^Q respectively, which are called target networks and used for calculating the target values. The weights of these target networks are updated by making them slowly track the learned networks: $\theta' \to \tau\theta' + (1 - \tau)\theta'$ with $\tau \ll 1$. This means that the target values are constrained to change slowly, greatly improving the stability of learning. This simple change moves the relatively unstable problem of learning the action-value function closer to the case of supervised learning.

At each iteration, we minimize a squared-error loss L to update the critic network parameter:

$$\min_{\theta^Q} L = \frac{1}{N} \sum_{k=1}^{N} (y_k - Q(s, a|\theta^Q))^2 \tag{3}$$

where y_k represents the TD target denoted as

$$y_k = r_k + \gamma Q'\Big(s_{k+1}, \pi'(x_{k+1}|\theta'^\pi)|\theta'^Q\Big) \tag{4}$$

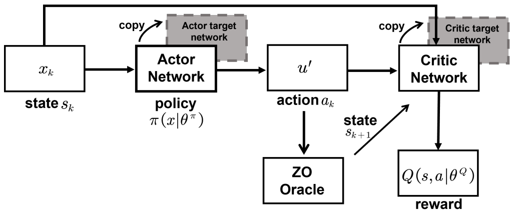


Figure 4: Illustration of of ZO-RL algorithm.

We maximize the cumulative reward using a sampled policy gradient to the actor network parameter:

$$\nabla_{\theta^\pi} J \approx \frac{1}{N} \sum_{k=1}^{N} \nabla_{\{u_i\}} Q(s, a|\theta^Q)|_{s=x_i, a=\pi(x_i)} \cdot \nabla_{\theta^\pi} \pi(x|\theta^\pi)|_{x_i}$$

[2]In the appendix, we provide the result of using the function values from the previous m iterations as states, since this state can reflect the effectiveness of the previous estimated ZO gradient.

Algorithm 1 Zeroth-Order Optimization with RL (ZO-RL)

```
Input: Smoothing parameter $\mu$, the number of sampled perturbed vectors $q$ and learning rate $\eta$,
    mini-batch size $N$ and threshold $\epsilon$, $\alpha \in [0, 1]$.
Output: RL based Sampling policy $\pi(x|\theta^\pi)$.
 1: Initialize $\Sigma = I_d$, $\rho = 0$ and sampling policy $\pi(x|\theta^\pi)$.
 2: for $k = 1$ to $K$ do
 3:     Get a perturbed vectors $u' = (1 - \alpha)\pi(x_k|\theta^\pi) + \alpha \cdot \bar{u}$, where $\bar{u} \sim \mathcal{N}(0, \Sigma_k)$.
 4:     Calculate the ZO gradient estimator $\tilde{g}_k$ according to (2) with the perturbed vectors $u'$.
 5:     if $\rho < \epsilon$ then
 6:         Randomly sample $q$ perturbed vectors $\{u_i\}_{i=1}^q$ from the Gaussian distribution $\mathcal{N}(0, \Sigma_k)$.
 7:         Calculate the ZO gradient estimator $\hat{g}_k$ according to (1) with the $q$ perturbed vectors.
 8:         Update $\rho$ based on $\hat{g}_k$ and $\tilde{g}_k$ according to (5).
 9:         Update $\tilde{g}_k = \hat{g}_k$ (use $\hat{g}_k$ to substitute $\tilde{g}_k$).
10:     end if
11:     Update $x_k$ with $\tilde{g}_k$, e.g. $x_{k+1} = x_k - \eta \cdot \tilde{g}_k$.
12:     Store transition $\{\{u_i\}_i^q/u', x_k, x_{k+1}, r_k\}$ in a replay memory buffer.
13:     Update $\Sigma$ (e.g. according to (6)).
14:     Observe $N$ transitions from replay memory buffer to update the actor network and critic
        network.
15: end for
```

where $J = \mathbb{E}_{r,s\sim\mathcal{S},a\sim\pi}[R(s, a)]$ represents the expected cumulative reward.

Initialization with Also-Ran. At each iteration of ZO-RL, we use actor network to output action u' according to current state x_k. Then, we transfer the action u' to ZO Oracle, calculate the ZO gradient $\tilde{g}$ and output the next state x_{k+1}, and use the critic network to output the reward of this action. We call $\{u', x_k, x_{k+1}, r_k\}$ a transition. Good transitions can accelerate the learning speed of the agent. The random sampling can effectively estimate the accurate gradient direction by querying multiple perturbed vectors in each iteration. Thus, in order to better initialize the actor network, we can use the random sampling to sampling perturbed vectors and interact with the current environment to obtain accurate ZO gradient estimator $\hat{g}$. We store these transitions generated by random sampling into replay memory buffer, and use them to initialize the actor network. We use a cosine similarity ρ to calculate the similarity of gradient directions obtained by plain random sampling and our sampling policy $\pi(x|\theta^\pi)$:

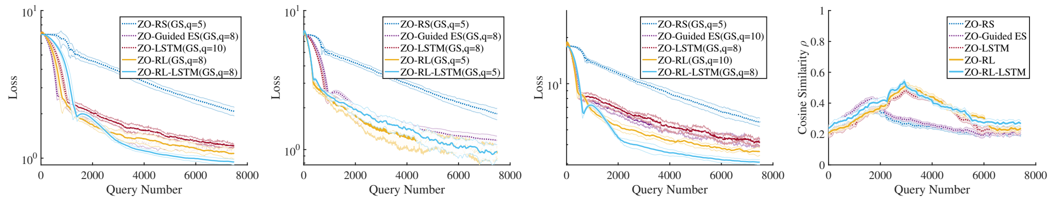
$$\rho = \frac{\hat{g} \cdot \tilde{g}}{||\hat{g}|| \cdot ||\tilde{g}||} \tag{5}$$

where $\hat{g}$ is the ZO gradient estimator defined in (1) with the perturbed vectors generated by random sampling, and $\tilde{g}$ is the ZO gradient estimator defined in (2) with the perturbed vector generated by our RL based policy. Obviously, $\rho \in [-1, 1]$. If ρ closes to 1, that means the directions between $\hat{g}$ and $\tilde{g}$ are close each other. Thus, we set a threshold ϵ to control when to switch to pure RL method. Correspondingly, if $\rho < \epsilon$, we use the random sampling policy to generate perturbed vectors and initialize the actor network. Note that once $\rho \geq \epsilon$, the codes in lines 6-9 will not be execute any more which also supports that the codes in lines 6-9 are just for initialization.

ZO-RL. We summarize our ZO-RL algorithm in Algorithm 1. Note that lines 5-10 of Algorithm 1 corresponds to the also-ran which is used for initialization. Once $\rho \geq \epsilon$, the lines 5-10 will not be executed any more, which means that we will use the perturbed vectors produced by RL based sampling policy to calculate the ZO gradients and update the solutions entirely.

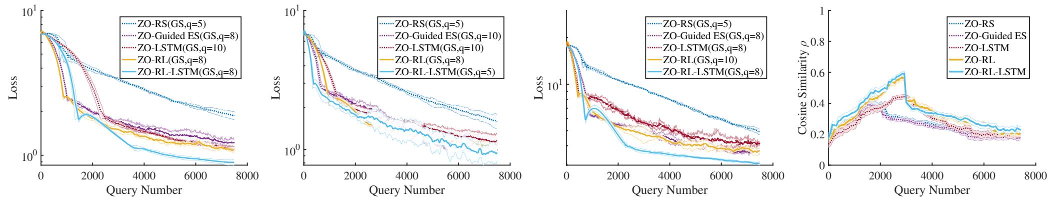
2.4 Improve Exploration

The RL usually can only evaluate the performance of an action, but does not know whether the action is the best. For this reason, the essence of RL determines that it needs exploration very much. Thus, we add a noise from an appropriate Gaussian distribution to the action performed by the agent in RL to increase the exploration, which is formalized by $u' = (1 - \alpha)\pi(x_k|\theta^\pi) + \alpha \cdot \bar{u}$, where $\bar{u} \sim \mathcal{N}(0, \Sigma_k)$. $\alpha \in [0, 1]$ is the trade-off parameter to balance the learned policy and an exploration distribution. Adding noise to the action can help RL to explore the environment more effectively and to think about all possible actions. Here, we utilize the learned sampling distribution from existing ZOO algorithms as the noise distribution to improve the quality of exploration. Specifically,



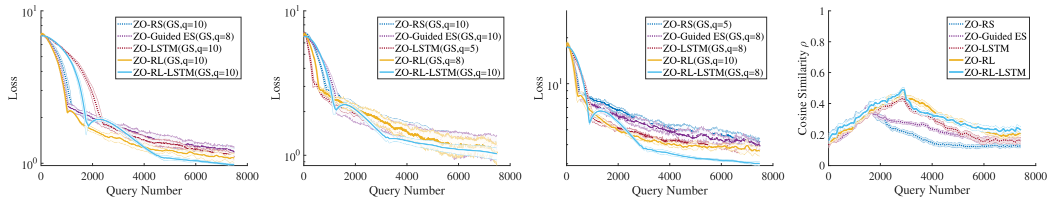
(a) MNIST TEST 1 (b) MNIST TEST 2 (c) MNIST TEST 3 (d) MNIST TEST 1

Figure 5: Adversarial attack to black-box models in SGD setting.



(a) MNIST TEST 1 (b) MNIST TEST 2 (c) MNIST TEST 3 (d) MNIST TEST 1

Figure 6: Adversarial attack to black-box models in signSGD setting.



(a) MNIST TEST 1 (b) MNIST TEST 2 (c) MNIST TEST 3 (d) MNIST TEST 1

Figure 7: Adversarial attack to black-box models in ADAM setting.

1. Ruan et al. (2019) proposed a ZOO algorithm called ZO-LSTM, which use a Long Short-Term Memory (LSTM) network called QueryRNN to learn the sampling distributions. They dynamically update the covariance matrix Σ_k as follows.

$$\Sigma_k = \text{QueryRNN}([\hat{\nabla} f(x_k), x_{k-1}]) \tag{6}$$

   QueryRNN can increase the sampling probability in the direction of the bias of the estimated gradient or the parameter update of the previous iteration.
2. Maheswaranathan et al. (2019) proposed a Guided Evolutionary Strategy for ZO-Guided ES, which incorporates surrogate gradient information (which is correlated with true gradient) into random search. It keeps track of a low dimensional guided subspace defined by k surrogate gradients, which is combined with the full space for query direction sampling. Thus, they dynamically update the covariance matrix Σ_k as follows.

$$\Sigma_k = \beta I_d + (1-\beta)UU^T \tag{7}$$

   where $U \in \mathbb{R}^{d\times k}$ is the orthonormal basis of the guided subspace (i.e., $UU^T = I_k$), β trades off between the full space and the guided space.

At each iteration of our ZO-RL algorithm, we can update Σ_k (i.e. the line 13 of Algorithm 1) to learn the corresponding Gaussian distribution.

3 Experimental Setup

3.1 Design of experiments

We empirically demonstrate the superiority of our proposed ZO-RL on three practical application problems (black-box adversarial attack, non-convex binary classification and hyperparameter optimization problem). Specifically, to show the advantages of the learned sampling policy, we compare the performance of our ZO-RL with existing ZO gradient estimators on three ZOO algorithms.

The comparative ZO gradient estimators are summarized as follows:

1. ZO-RS Wang et al. (2019): ZO gradients are calculated by (1) with randomly sampled perturbed vectors from standard Gaussian distribution.
2. ZO-LSTM Ruan et al. (2019): They learned the Gaussian sampling rule and dynamically predicted the covariance matrix Σ for query directions with recurrent neural networks, and generated ZO gradient by UpdateRNN.
3. ZO-Guided ES Maheswaranathan et al. (2019): ZO gradients are calculated by (1) with randomly sampled perturbed vectors from a learned Gaussian distribution. They let the covariance matrix Σ in Gaussian distribution be related with the recent history of surrogate gradients during optimization.
4. ZO-RL: ZO gradients are calculated by (2) with the perturbed vector generated by our ZO-RL.
5. ZO-RL-LSTM: ZO gradients are calculated by (2) with the perturbed vector generated by our ZO-RL where ZO-LSTM is used to improve the exploration.

The three ZO algorithms are summarized as follows:

1. SGD Ghadimi & Lan (2013): A stochastic gradient descent algorithm based on the ZO gradient estimators.
2. signSGD Liu et al. (2018b): A stochastic gradient descent algorithm based on the sign of the ZO gradient estimators.
3. ADAM Chen et al. (2017): A stochastic gradient descent algorithm based on adaptive estimates of lower-order moments.

To show the sensitivity of our proposed algorithm, we compare the performance of ZO-RL with different values of hyperparameters. In addition, we use ablation experiments to test the impact of each component of ZO-RL on the performance of our ZO-RL algorithm. The uniform distribution on unit sphere is also considered for generating perturbed vectors. These results are included in appendix.

3.2 Implementation

The choice of the network structure of the critic and actor for our ZO-RL is important because they are used not only to evaluate sampling policies, but also to learn sampling policies. We choose the convolutional neural network (CNN) Sezer & Ozbayoglu (2018) both for the critic network and the actor network when image data are faced. Specifically, for MNIST dataset, we use a five-layer CNN with two $(5 \times 5) \times 6$ convolutional layers with a step size of 1, each convolutional layer is subsampled using a 2×2 pooling layer with a step size of 2, and finally a 300-unit fully-connected layer.

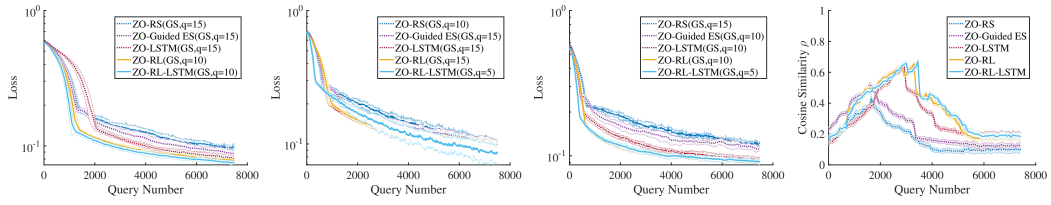
The parameters of the optimizer have different descent rates in different dimensions and the range may be different in different environments. This may make it difficult for the network to learn efficiently and find hyper-parameters that generalize the scale of state values in different environments. To overcome this problem, we adapt the batch normalization Santurkar et al. (2018) to manually scale features so that the parameters of the ZO optimizer are in a similar range across environments and units.

For each task, we tune the hyper-parameters of baseline algorithms to report the best performance. For ADAM, we tune β_1 values over $\{0.9, 0.99\}$ and β_2 values over $\{0.99, 0.996, 0.999\}$. We coarsely tune the constant δ on a logarithmic range $\{0.01; 0.1; 1; 10; 100; 1000\}$ and set the learning rate of baseline algorithms to $\eta = \delta/d$, where d is the dimension of dataset. We set the smoothing parameter $\mu = 0.0001$ in all experiments. The number q of sampled perturbed vectors is selected over $\{1, 3, 5, 8, 10, 15, 20\}$ for each algorithm. When the overall number of queries of function evaluations is fixed, the optimal number q of sampled perturbed vectors of the comparison algorithm is different. For fairness, we compare the results of each algorithm under their best number of sampled perturbed vectors. For our ZO-RL algorithm, the values for hyperparameters l_1, l_2, γ, N τ and ϵ and α are selected over $\{0.001, 0.005, 0.01, 0.05\}$, $\{0.001, 0.005, 0.01, 0.05\}$, $\{0.99, 0.95, 0.9\}$, $\{16, 32, 64\}$, $\{0.001, 0.005, 0.01, 0.05\}$, $\{0.3, 0.4, 0.5, 0.6\}$ and $\{0.33, 0.5, 0.66\}$ respectively.

4 Experimental Results and Discussion

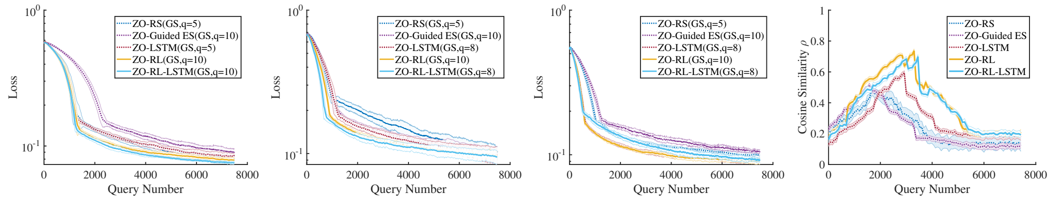
4.1 Adversarial Attack to Black-box Models

We consider generating adversarial examples to attack black-box DNN image classifier and formulate it as a zeroth-order optimization problem. The targeted DNN image classifier $F(x) =$



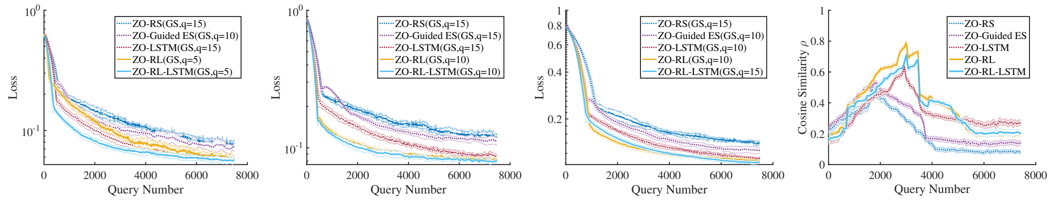
(a) colon-cancer (b) german (c) a9a (d) a9a

Figure 8: Non-convex optimization problems in the SGD setting.



(a) colon-cancer (b) german (c) a9a (d) a9a

Figure 9: Non-convex optimization problems in the signSGD setting.



(a) colon-cancer (b) german (c) a9a (d) a9a

Figure 10: Hyperparameter optimization problems in the SGD setting.

$[F_1, F_2, \cdots, F_K]$ takes as input an image $x \in [0,1]^d$ and outputs the prediction scores of K classes. Given an image $x_0 \in [0,1]^d$ and its corresponding true label $t_0 \in [1, 2, \cdots, K]$, an adversarial sample x is visually similar to the original image x_0 but leads the targeted model F to make wrong prediction other than t_0. The black-box attack problem is normally formulated as follows.

$$\max_x \{F_{t_0}(x) - \max_{j \neq t_0} F_j(x), 0\} + c\|x - x_0\|_p \tag{8}$$

where the first term is the attack loss which measures how successful the adversarial attack is and penalizes correct prediction by the targeted model. The second term is the distortion loss (p-norm of added perturbation) which enforces the perturbation added to be small and c is the regularization coefficient. In our experiment, we use ℓ_1 norm (*i.e.*, $p = 1$), and set $c = 0.1$ for MNIST attack task. Due to the black-box setting, one can only compute the function values of the above objective, which leads to ZOO problems Chen et al. (2017). Note that attacking each sample x_0 in the dataset corresponds to a particular ZOO problem instance, which motivates us to train a ZO optimizer offline with a small subset, and apply it to online attack to other samples with faster convergence (which means lower query complexity) and lower final loss (which means less distortion). We randomly select 50 images that are correctly classified by the targeted model in each test set to train the optimizer and select another 50 images to test the learned optimizer. All ZO optimizers use the same initial points for finding adversarial examples. The dimension of the optimizer function is $d = 28 \times 28$ for MNIST.

In this subsection, we compare the performance of different ZOO algorithms under Gaussian distribution. We provide the results under the spherical distribution in the appendix. Fig. 5$\sim$7.(a)$\sim$(c) show the black-box attack loss versus query number using different ZOO algorithms under Gaussian distribution in different settings. The loss curves are averaged over 10 independent random trails and the shaded areas indicate the standard deviation. The q in ZO-RL represents the number of sampled query directions in each iteration of random sampling. In ZOO problem where we can only access the function output, we more case the query complexity instead of running time. In the extreme case, we could satisfy the running time to reduce the query complexity. The results clearly show that our ZO-RL algorithm can effectively reduce the query complexity of ZOO algorithms especially in the later stage of the optimization process, and our ZO-RL-LSTM can always obtain the best results by combining learned sampling policy and sampling distribution. This is due to the fact that our ZO-RL algorithm learn a smarter sampling policy though RL instead of random sampling. Fig. 5$\sim$7.(d) plot the cosine similarities between ZO gradient estimator and ground-truth gradient. The direction of the ZO estimator generated by our ZO-RL algorithm is closer to the one of the ground-truth gradient compared to other ZO estimators. In the later stage of the optimization process, the convergence of ZO gradient leads to the reduction of the reward obtained by our ZO-RL algorithm in exploring action space. Thus, the cosine similarity decreases after the convergence of the ZOO algorithm.

4.2 Non-Convex Binary Classification Problems

We consider a binary classification problem with a non-convex least squared loss function $\min_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n}(y_i - 1/(1+e^{-w^T x_i}))^2$. Here (x_i, y_i) is the ith data sample containing feature $x_i \in \mathbb{R}^d$ and label $y_i \in \{-1, 1\}$. We compare the algorithms on benchmark datasets (colon-cancer ($d = 2000$), german ($d = 24$) and a9a ($d = 123$) [3]). All the algorithms only access to the ZO oracle of function value evaluations. For each dataset, we repeat the experiment 10 times and report the average and the standard deviation.

We compare the performance of different ZOO algorithms under Gaussian distribution. We provide the results under the spherical distribution in the appendix. Fig. 8$\sim$9.(a)$\sim$(c) shows the non-convex least squared loss versus query number using different ZOO algorithms under Gaussian distribution in different settings. The loss curves are averaged over 10 independent random trails and the shaded areas indicate the standard deviation. The results clearly demonstrate that our ZO-RL leads to much faster convergence and lower final loss under different parameter update settings compared to existing ZOO algorithms. Fig. 8$\sim$9.(d) plot the cosine similarities between ZO gradient estimator and ground-truth gradient. These results demonstrate that our proposed ZO-RL algorithm can obtain better performance than random sampling and learning sampling distribution, and can be well extended to different categories of ZOO problems.

4.3 Hyperparameter Optimization Problems

We consider a hyperparameter optimization problem in the l_2-regularized logistic regression model. We use the logistic loss $l(t) = \log(1 + e^-t)$ as the loss function. The hyperparameter optimization problem for l_2-regularized logistic regression is formulated as follows.

$$\arg\min_{\lambda\in[-10,10]} \sum_{i=1}^{n} l(y_i\langle x_i, w(\lambda)\rangle), \quad s.t. \quad w(\lambda) \in \arg\min_{w\in\mathbb{R}^d} \sum_{i=1}^{m} l(y_i\langle x_i, w(\lambda) + e^{\lambda}||w||^2$$

The solver used for solving the inner objective is L-BFGS Liu & Nocedal (1989). We compare the algorithms on benchmark datasets (colon-cancer ($d = 2000$), german ($d = 24$) and a9a ($d = 123$)). For each dataset, we repeat the experiments 10 times and report the average and the standard deviation.

We compare the performance of different ZOO algorithms under Gaussian distribution. In the appendix, we provide the additional results under the sphere distribution. Fig. 10.(a)$\sim$(c) shows the loss versus query number using different ZOO algorithms under Gaussian distribution in SGD settings. The loss curves are averaged over 10 independent random trails and the shaded areas indicate the standard deviation. Fig. 10.(d) plot the cosine similarities between ZO gradient estimator and ground-truth gradient. These results demonstrate that our ZO optimizer has learned a rather general ZOO algorithm which can generalize to different classes of ZOO problems well.

5 Conclusion

We proposed a RL based sampling policy for generating the perturbations in ZOO. Since our method only affects the generation of perturbed vectors, it can be used with different ZOO algorithms by substituting ZO estimator to further improve the efficiency of ZOO algorithms. Importantly, the existing ZOO algorithms of learning a distribution can be plugged in to improve the exploration of our ZO-RL. Experimental results on different ZOO algorithms show that our ZO-RL algorithm can effectively reduce the query complexity of ZOO algorithms especially in the later stage of the optimization process, and converge faster than existing ZOO algorithms in different scenarios.

[3]http://archive.ics.uci.edu/ml/datasets.html

REFERENCES

Kai Arulkumaran, Antoine Cully, and Julian Togelius. Alphastar: An evolutionary computation perspective. In *Proceedings of the genetic and evolutionary computation conference companion*, pp. 314–315, 2019.

Albert S Berahas, Liyuan Cao, Krzysztof Choromanski, and Katya Scheinberg. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *Foundations of Computational Mathematics*, pp. 1–54, 2021.

S. Bubeck and N. Cesa-Bianchi. *Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems*. 2012.

Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 15–26, 2017.

T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin. Learning to optimize: A primer and a benchmark. 2021.

John C Duchi, Peter L Bartlett, and Martin J Wainwright. Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 22(2):674–701, 2012.

Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

B. Gu, G. Liu, Y. Zhang, X. Geng, and H. Huang. Optimizing large-scale hyperparameters via automated learning algorithm. 2021.

Nikolaus Hansen. The cma evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pp. 75–102. Springer, 2006.

Patrick Koch, Oleg Golovidov, Steven Gardner, Brett Wujek, Joshua Griffin, and Yan Xu. Autotune: A derivative-free optimization framework for hyperparameter tuning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 443–452, 2018.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pp. 1008–1014, 2000.

T. Lattimore and A. Gyorgy. Improved regret for zeroth-order stochastic convex bandits. In *Conference on Learning Theory*, 2021.

K. Li and J. Malik. Learning to optimize. 2016.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

Liu Liu, Minhao Cheng, Cho-Jui Hsieh, and Dacheng Tao. Stochastic zeroth-order optimization via variance reduction method. *arXiv preprint arXiv:1805.11811*, 2018a.

Sijia Liu, Pin-Yu Chen, Xiangyi Chen, and Mingyi Hong. signsgd via zeroth-order oracle. In *International Conference on Learning Representations*, 2018b.

Niru Maheswaranathan, Luke Metz, George Tucker, Dami Choi, and Jascha Sohl-Dickstein. Guided evolutionary strategies: Augmenting random search with surrogate gradients. In *International Conference on Machine Learning*, pp. 4264–4273. PMLR, 2019.

Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pp. 506–519, 2017.

Yangjun Ruan, Yuanhao Xiong, Sashank Reddi, Sanjiv Kumar, and Cho-Jui Hsieh. Learning to learn by zeroth-order oracle. *arXiv preprint arXiv:1910.09464*, 2019.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *arXiv preprint arXiv:1805.11604*, 2018.

Omer Berat Sezer and Ahmet Murat Ozbayoglu. Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach. *Applied Soft Computing*, 70:525–538, 2018.

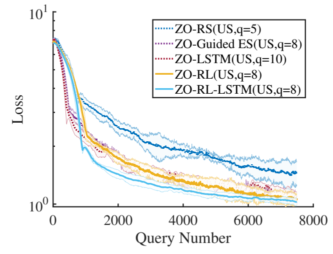
Fei-Yue Wang, Jun Jason Zhang, Xinhu Zheng, Xiao Wang, Yong Yuan, Xiaoxiao Dai, Jie Zhang, and Liuqing Yang. Where does alphago go: From church-turing thesis to alphago thesis and beyond. *IEEE/CAA Journal of Automatica Sinica*, 3(2):113–120, 2016.

Jun-Kun Wang, Xiaoyun Li, and Ping Li. Zeroth order optimization by a mixture of evolution strategies. 2019.

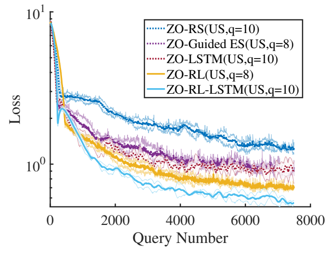
Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 3381–3387. IEEE, 2008.

A Adversarial Attack to Black-box Models

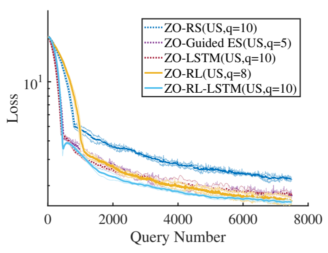
Fig. 11∼13 show the black-box attack loss versus query number using different ZOO algorithms under unit sphere distribution in different settings. The "US" is the abbreviation of unit sphere distribution. When the overall number of queries of function evaluations is fixed, the optimal number q of sampled perturbed vectors of the comparison algorithm is different. For fairness, we compare the results of each algorithm under their best number of sampled perturbed vectors. The results clearly show that our ZO-RL algorithm can effectively reduce the query complexity of ZOO algorithms especially in the later stage of the optimization process, and our ZO-RL-LSTM can always obtain the best results by combining learned sampling policy and sampling distribution.



(a) MNIST TEST 1

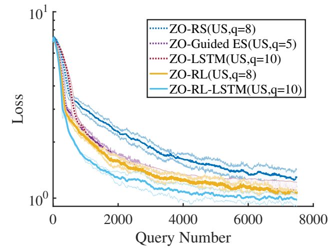


(b) MNIST TEST 2

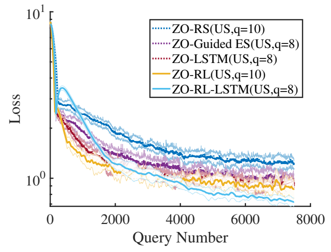


(c) MNIST TEST 3

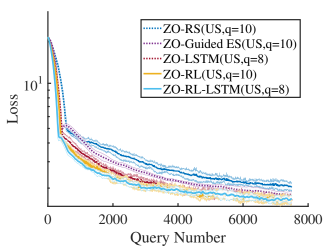
Figure 11: Adversarial attack to black-box models in the signSGD setting.



(a) MNIST TEST 1

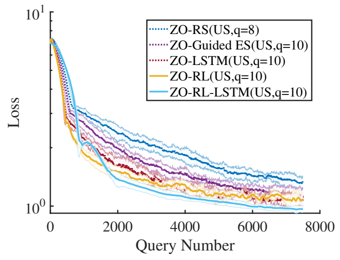


(b) MNIST TEST 2

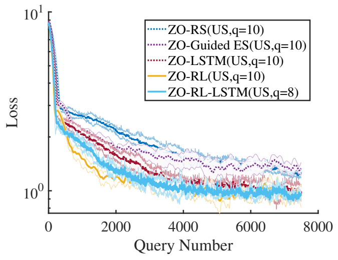


(c) MNIST TEST 3

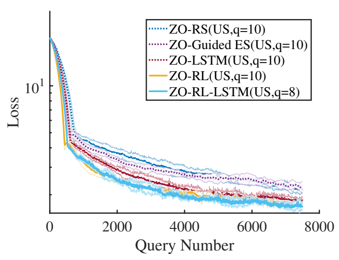
Figure 12: Adversarial attack to black-box models in the SGD setting.



(a) MNIST TEST 1



(b) MNIST TEST 2

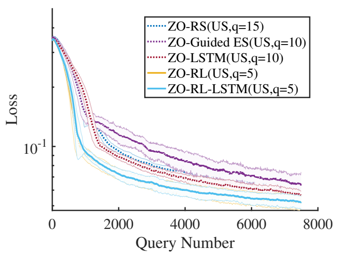


(c) MNIST TEST 3

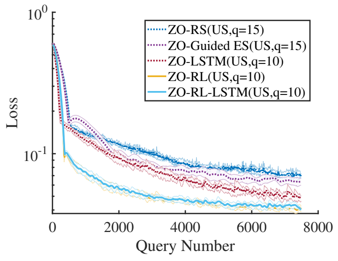
Figure 13: Adversarial attack to black-box models in the ADAM setting.

B Non-Convex Binary Classification Problems

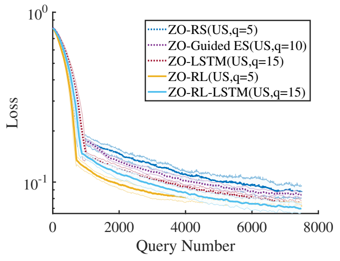
Fig. 14∼16 show the non-convex least squared loss versus query number using different ZOO algorithms under unit sphere distribution in different settings. The "US" is the abbreviation of unit sphere distribution. These results demonstrate that our proposed ZO-RL algorithm can obtain better performance than random sampling and learning sampling distribution, and can be well extended to different categories of ZOO problems.



(a) colon-cancer

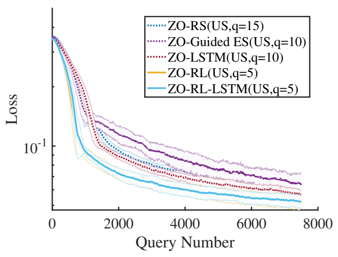


(b) german

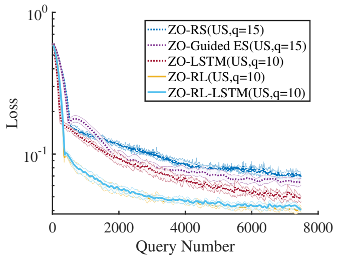


(c) a9a

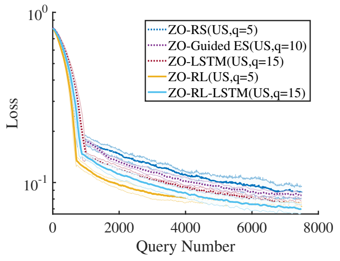
Figure 14: Non-convex optimization problems in the signSGD setting.



(a) colon-cancer

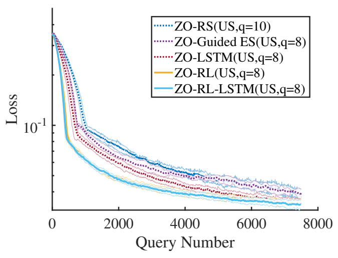


(b) german

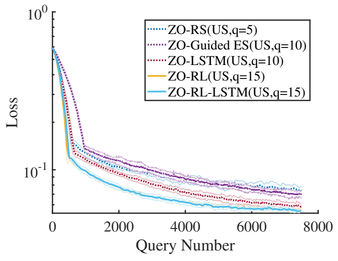


(c) a9a

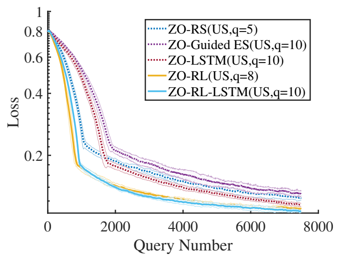
Figure 15: Non-convex optimization problems in the SGD setting.



(a) colon-cancer



(b) german

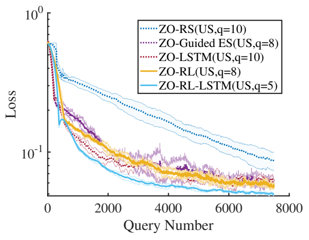


(c) a9a

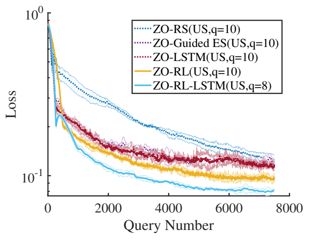
Figure 16: Non-convex optimization problems in the ADAM setting.

C Hyperparameter Optimization Problems

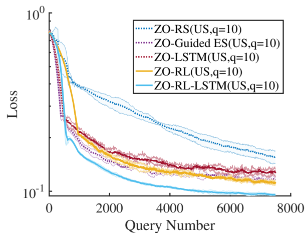
Fig. 17∼19 show the loss versus query number using different ZOO algorithms under unit sphere distribution in different settings. The "US" is the abbreviation of unit sphere distribution. These results demonstrate that our ZO optimizer has learned a rather general ZOO algorithm which can generalize to different classes of ZOO problems well.



(a) hcolon-cancer

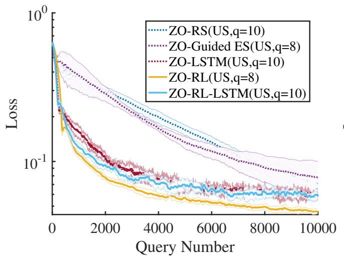


(b) german

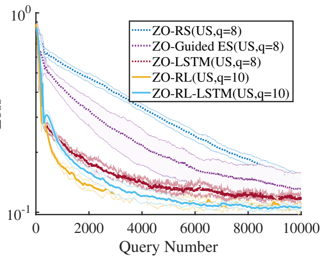


(c) a9a

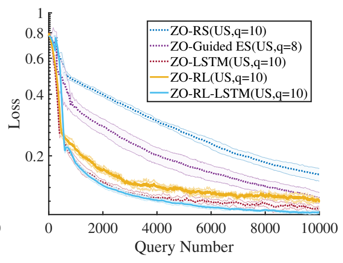
Figure 17: Hyperparameter optimization problems in the signSGD setting.



(a) colon-cancer

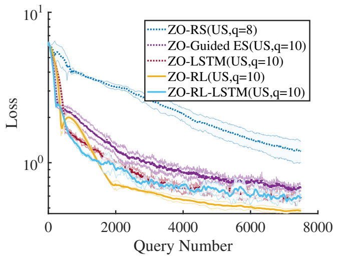


(b) german

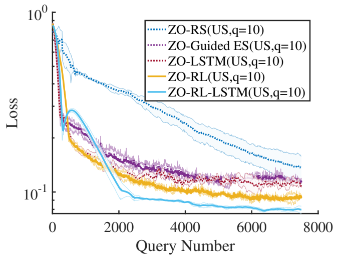


(c) a9a

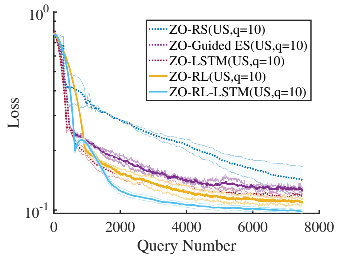
Figure 18: Hyperparameter optimization problems in the SGD setting.



(a) colon-cancer



(b) german



(c) a9a

Figure 19: Hyperparameter optimization problems in the ADAM setting.

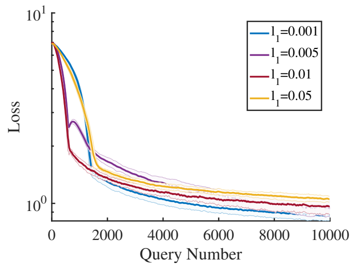
D Algorithm Sensitivity Experiment

D.1 Hyperparameter Tuning

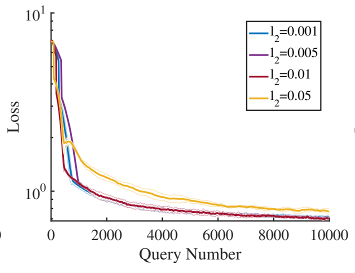
Reinforcement learning algorithm performances are known to be influenced by hyperparameters. To show the stability of our proposed algorithm, we test the sensitivity of our algorithm to reinforcement learning hyperparameters. We carry out hyperparametric tuning experiments for MNIST attack task. For each hyperparameter, we calculate the final average loss for 10 independent random trails for each hyperparameter candidate, and choose the optimal hyperparameters according to the final average loss minimization criterion. Specifically, we select the optimal hyperparameters from the following candidate set:

1. Learning rate for actor $l_1 = \{0.01, 0.05, 0.001, 0.005\}$.
2. Learning rate for critic $l_2 = \{0.01, 0.05, 0.001, 0.005\}$.
3. Mesh update parameter $\tau = \{0.001, 0.005, 0.01, 0.05\}$.
4. Cosine similarity threshold $\epsilon = \{0.3, 0.4, 0.5, 0.6\}$.
5. Query number $q = \{5, 10, 15, 20, 30\}$.
6. Smoothing parameter $\mu = \{0.001, 0.0001, 0.00001, 0.000001\}$.
7. Exploration paremeter $\alpha = \{0.66, 0.5, 0.33\}$

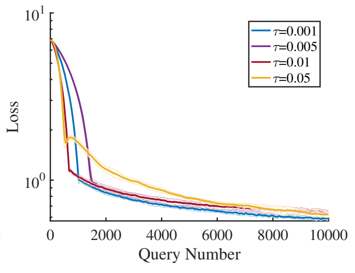
Fig. 20 show the loss versus query number using different hyperparameters under Gaussian distribution for MNIST attack task. The loss curves are averaged over 10 independent random trails and the shaded areas indicate the standard deviation. The results show that our ZO-RL algorithm is less affected by the hyperparameters l_2 and τ, and more affected by the hyperparameters l_1, ϵ, q, μ and α. The learning rate of actors has a much higher impact on the results than that of critics. That is because the output of actor network directly determines the quality of sampling policy. For mesh update parameter τ, cosine similarity threshold ϵ and smoothing parameter μ, choosing a smaller value can often get better results. For query number q and exploration paremeter α, too large or too small will lead to a bad result, and it takes more time to find an optimal value.



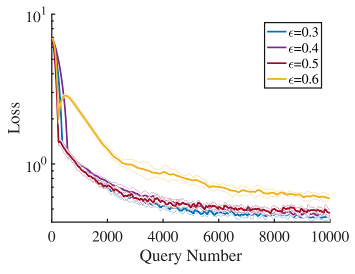
(a) Learning rate for actor



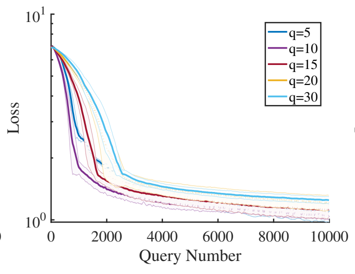
(b) Learning rate for critic



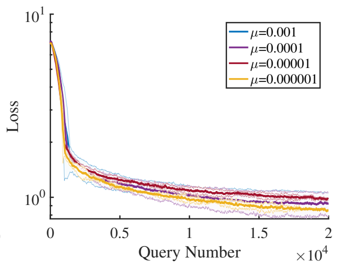
(c) Mesh update parameter



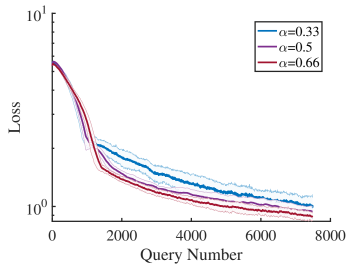
(d) Cosine similarity threshold



(e) Query number



(f) Smoothing parameter



(g) Exploration paremeter

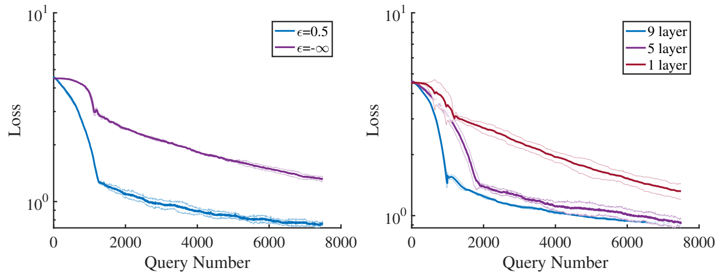
Figure 20: Hyperparameter tuning on MNIST attack task.

D.2 Ablation Experiments

We set up ablation experiments to test the impact of each component of ZO-RL on the performance of the algorithm in different problems. Specifically, we set up two ablation studies as follows:

1. To test whether pre-training can accelerate the learning rate of ZO-RL, we conducte experiments using random sampling to sample from the standard Gaussian distribution as the pre-training of reinforcement learning policy, i.e. using cosine similarity threshold $\epsilon = 0.5$, and directly using reinforcement learning policy to sample from the standard Gaussian distribution, i.e. using cosine similarity threshold $\epsilon = -\infty$.
2. To test the influence of convolutional layers on ZO-RL performance, we conducte experiments using 9-layer neural network structure (4 convolution layers, 4 pooling layers and 1 full connection layer), 5-layer neural network (2 convolution layers, 2 pooling layers and 1 full connection layer) and 1-layer neural network (1 full connection layer).

Fig. 21 shows the loss versus query number for two ablation studies. The results show that pre-training and deeper reinforcement learning network can effectively enhance the performance of our ZO-RL algorithm.

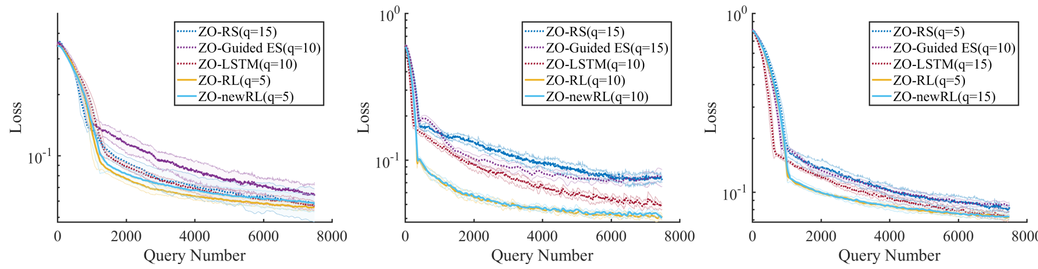


(a) Pre-training (b) Convolutional layers

Figure 21: Ablation Study.

E Additional Settings for States

We use the function values from the previous m iterations as states, since this state can reflect the effectiveness of the previous estimated ZO gradient. Fig. 22 show the black-box attack loss versus query number using different ZOO algorithms under Gaussian distribution on MNIST attack task in the SGD setting. The "ZO-newRL" represents the algorithm using the function values from the previous m iterations as the states. We set $m = 20$ for all the experiments in Fig. 22. The results show that our ZO-newRL and ZO-RL have similar results, which can effectively reduce the query complexity of ZOO algorithms. This further verifies that learning an effective sampling policy is effective in the ZOO algorithms. Fig. 23 show the the black-box attack loss versus query number for hyperparameter m. The results show that the more function values from previous iterations are used as states, the better the algorithm performs.



(a) MNIST TEST 1 (b) MNIST TEST 2 (c) MNIST TEST 3

Figure 22: Adversarial attack to black-box models in the SGD setting.

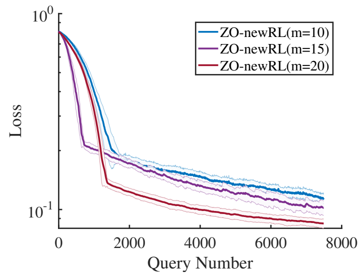


Figure 23: Hyperparameter m.