Black-box Optimization with Actor-Critic Algorithm

Ahmet F. Budak and Venkata Suresh Rayudu
https://youtu.be/QH2z2MyfDRQ

I. INTRODUCTION

Many optimization problems in scientific literature is black box optimization problem. The objective is to maximize the non-linear mathematical expression $f(\mathbf{x})$ where $\mathbf{x}$ is a $\mathbf{d}$ dimensional variable. The final solution to the maximization problem is to find the global optimal of the variable $\mathbf{x}$, which can be expressed as: $\mathbf{x}^\star = \text{argmax}_\mathbf{x}\, f(\mathbf{x})$. In the optimization community globally optimizing black box functions have attracted much attention and various methods such as differential evolution [1], particle swarm optimization [2] and Bayesian optimization [3] are proposed to tackle that problem.

Many of the real world problems have expensive function evaluations which makes the efficiency of the optimization algorithm the bottleneck. Recent work on black-box optimization is focused on that problem. Traditionally used genetic algorithms mostly rely on a simple heuristic to balance the exploration-exploitation dilemma and they perform well when the time/simulation budged is large but they suffer poorly when the simulation budget is limited. Bayesian optimization methods, on the other hand, are more sample efficient. They make use of the previous evaluations to build a surrogate model, which is generally by utilizing Gaussian Processes(GP) [4]. However, Gaussian Processes have expensive modeling time and tuning effort. The complexity of the GP is due to the matrix inversion operation with complexity of $\mathcal{O}(N^3)$ where N is the number of samples in training set.

In this project, we aim to investigate the applicability of recent advances in Reinforcement Learning (RL) on black-box optimization problems. More specifically, we propose two RL approach on black-box optimization problem. The first approach, is a critic-only method embedded on a genetic algorithm framework and the second approach is an actor-critic based optimization framework. We modeled optimization problem as an $\mathbf{d}$-dimensional continuous control problem with continuous action space. The state representation for the black-box function is provided by the evaluation point in space, namely $state_i = \{\mathbf{x}_i\}$ for point i and action is the amount of deviation from given point, s.t., $action_i = \{\Delta\, \mathbf{x}_i\}$. A critic network is used to approximate state-action values, $Q(state_i, action_i)$, and an actor network is trained to approximate optimal action for given state observation.

The rest of the report is organized as follows. In Section II, a brief review to Differential Evolution (DE) optimization and only-critic involved optimization framework (DECN) is introduced. In section III, Black-Box Optimization with Actor-Critic Algorithm (BOAC) method is presented. In Section IV, the performance of DECN and BOAC algorithms are demonstrated by three benchmark problems used in optimization community and comparison with existing optimization techniques is given. We mention the future and ongoing work on Section V and the conclusions are provided in Section V.

II. Differential Evolution with Critic Network: DECN

Differential evolution (DE) is a type of genetic algorithm used for long-time by the optimization community. In this section we will first introduce a single-step DE optimization algorithm then propose a method to improve its performance. More specifically, we aim to have more accurate and faster convergence characteristic for the optimizer.

A. Single-Step Differential Evolution (SSDE)

DE [1] is a popular global optimization algorithm and is widely used in black-box optimization problems. DE starts with initializing each individual within the given search ranges. The i^{th} individual in the d-dimensional search space at generation t can be represented as:

$$X_i(t) = [x_{i,1}, x_{i,2}, ..., x_{i,d}], \quad i = 1, 2, ...N. \tag{1}$$

where N is the population size.

A mutant vector V_i is then generated by adding the weighted difference of randomly chosen individuals to the base vector.

$$V_i(t) = X_i + F(X_{best} - X_i) + F(X_{r1} - X_{r2}) \tag{2}$$

where F is the weight for the differential operation, X_i is the base vector. Indices $r1$ and $r2$ ($r1, r2 \in \{1, 2, ..., N\}$) are randomly chosen, which are mutually different from i. There are various kinds of DE mutations and the above DE/current-to-best/1 mutation strategy is used in our methods.

Crossover operator is carried out after the mutation. For each mutant vector, a trial vector $U_i(t) = [u_{i,1}, u_{i,2}, ..., u_{i,d}]$ is generated as follows:

$$\mathrm{u}_{i,j} = \begin{cases} v_{i,j}(t)\ if\ (rand(j) \leq CR\ or\ j = randn(i)) \\ x_{i,j}(t)\ otherwise \end{cases} \tag{3}$$

where CR is the crossover rate and $randn(i)$ is a randomly chosen index to guarantee at least one element of U_i is different from X_i.

Finally, randomly pick a member from trial vector $U(t)$ and its corresponding member from the parent population. Based on the objective evaluations the next survival is carried to the next generation.

B. Differential Evolution with Critic Network: DECN

For any algorithm using value approximation, the inference power of the approximation significantly affects the performance of the whole algorithm. Therefore, we start our investigation by measuring how good our critic's approximation

with respect to a random inference method. Most of the time beating a random inference is difficult to achieve. We present Differential Evolution with Critic Network: DECN where a critic network is trained to make inferences about not-yet taken actions.

DECN mostly follows the architecture proposed for single-step DE and becomes active when we need to select a member from trial vector $U(t)$. In simple terms, we obtain the state-action pairs that connects the parent population with the trial vector (child population) and use critic for approximating these pair's values. The pair with the highest prediction is selected as the member to be evaluated for the next iteration. We update the critic parameters in each iteration by using **pseudo-samples** generated from available evaluations. The detailed explanation on generating pseudo-samples is given at BOAC algorithm. The algorithm for DECN is given in Algorithm 1.

Algorithm 1 The DECN Framework

1: Randomly initialize critic network $Q\left(s,a|\theta^Q\right)$ with weights θ^Q
2: Use Latin Hypercube sampling [5] to sample α candidate designs from the design space.
3: Perform simulations to all of them and let them form the initial training set
4: Generate pseuod-samples and intiate replay buffer R;
5: **repeat**
6: Use pseudo-sample subset to train critic network by using the following loss for minimization. $L = \frac{1}{N}\sum_i \left(y_i - Q\left(s_i,a_i|\theta^Q\right)\right)^2$
7: Apply DE steps, namely mutation and crossing-over to generate trial vector/child population
8: Generate state-action pairs for each parent-child pair. $(s,a) = (x_{parent}, x_{child} - x_{parent})$
9: Use critic network and select the highest valued state-action pair to evaluate $x_{t+1} = argmax_i(Q(s_i,a_i))$ where i is the indice for each parent-child pair.
10: Evaluate the selected child and save its function value
11: Set $y_t = r\left(s_t,a_t\right) = R(s_{t+1}) = f(x_{t+1})$
12: Update pseudo-samples by using the latest evaluation.
13: **until** the number of required sampling iterations is reached.
14: Output the best design;

III. Black-box Optimization with Actor-Critic Algorithm: BOAC

A. Actor-Critic Algorithm

Many of the RL methods are fall into either actor-only or critic-only methods where actor-only methods make use of policy gradient and updates the parameters in the direction of improvement and critic-only methods rely on value-function approximation and uses Bellman equations to find optimal-policy. Actor-critic algorithms combines these two methods by using simulation result to update value-function approximation (critic) and then updating actor's policy parameters based on the updated critic approximation [6].

Recently, using deep-neural-networks as value and policy approximators in actor-critic algorithms shows promising results on complex tasks [7], [8]. Our work follows the ideas proposed in [8], where deep deterministic policy gradient algorithm (DDPG) is introduced. DDPG is a model-free, off-policy actor-critic algorithm tailored for problems with continuous action spaces.

B. Modeling Optimization Problem as MDP

In Markov Decision Processes (MDPs) an action a_t is taken at state s_t and the environment responds to that action by assigning a new state s' and a reward $R(s,a,s')$. In the optimization case, we have evaluated points denoted as $\mathbf{x}$ where each x_i is a d dimensional input to the optimization problem and represents the state of each agent. Then corresponding $\Delta\,\mathbf{x}_i$ vectors would be the actions generating potential query evaluation points in the optimization. Our desire in this framework to find optimal actions which would generate the highest improvement in the maximization problem.

In our problem, we used real function evaluations as the returns for state-action pairs.

$$Q(s_t,a_t) = Q(x_t,\Delta\,x_t) = f(x_t + \Delta\,x_t)$$

C. BOAC Framework

Since the fundamental ideas of this project is given in DDPG [8], we briefly mention the ideas proposed there and highlight the steps we modified.

Our problem has a real valued action space $a_t \in \mathbb{R}^N$ and its policy π is a probability mapping between states and actions $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$. In the original DDPG, the return of a state-action is defined as a discounted sum of all future rewards but we simply use the functions evaluation value in a given state as its return. Notice that, in this convention, practically infinitely many state-action pairs that result in the same state would have the same return.

$$Q^\pi\left(s_t,a_t\right) = \mathbb{E}_{r_{i\geq t},s_{i>t}\sim E,a_{i>t}\sim\pi}\left[R_t|s_t,a_t\right] \tag{4}$$

Greedy policy $\mu(s) = \arg\max_a Q(s,a)$ is used in Q-learning [9] and function approximators parameterized by θ^Q, which is optimized by minimizing the loss:

$$L\left(\theta^Q\right) = \mathbb{E}_{s_t\sim\rho^\beta,a_t\sim\beta,r_t\sim E}\left[\left(Q\left(s_t,a_t|\theta^Q\right) - y_t\right)^2\right] \tag{5}$$

where

$$y_t = r\left(s_t,a_t\right) = R(s_{t+1}) = f(x_{t+1}) \tag{6}$$

Compared to DDPG algorithm, our y_t does not include a $Q(.|\theta^Q)$ approximated value which eliminates the approximation error.

We also modify the actor minimization function due to lower/upper bound constraints of our solution space. We modify the loss function by adding violation terms for each dimension.

$$L(\theta^\mu) = -1 * \sum_i Q(x_i,a_i) - \lambda * viol_i^2$$

where

$$viol_{id} = min((x_{id} - lb_d), 0) + max((x_{id} - ub_d), 0)$$

We have seen during our experiments that choosing an adequate exploration noise is very crucial for the success of the algorithm. We defined a decaying noise term by taking advantage of our sample space. Our noise term is adaptive to the standard deviation (std) of the top results so far observed. It is a random Gaussian noise with zero mean and whose std in each dimension is equal to std of top solutions. Therefore, as we gather high quality solutions by converging to a particular region in the solution space, our exploration noise decays adaptively.

The algorithm for NOAC optimizer is given in Algorithm 2. Recall that NOAC is proposed for optimizing black-box functions in an online fashion. Therefore, we target our algorithm to be effective in a single episode so that we could have a fair comparison with other optimization techniques. This behaviour requires our algorithm to have a source of information from the beginning which is generally not the case for RL algorihtms in the first episode. However, the characteristics of our problem allow us to generate this information from the very beginning.

Optimizer is initiated by sampling α points randomly or via design of experiment from the solution space which is subject to the lower bound and upper bound requirements given for each dimensions. Then, **pseudo-samples** are generated to provide the initial source of information of the problem. In our algorithm pseudo-samples are particularly used for training critic parameters. Notice that, we have an action x_{ij} for each of the initial α points which can take us from point x_i and brings to point x_j whose function value is known to us. Although our agent has not really taken that action in the past, we have the exact information for return of that state-action pair and simply making use of this information by generating these pseudo state-action values. The following state,action,reward combinations are created for each pairings of samples and α^2 pseudo-samples are created.

$$\begin{aligned} a_{ij} &= \Delta\, x_{ij} = x_j - x_i \\ s_{ij} &= s_i = x_i \\ R_{ij} &= R_j = y_j = f(x_j) \end{aligned} \tag{7}$$

IV. Experimental Results

In this section, we present the experimental results with the best-performing version of NOAC algorithm. We have attempted couple improvements on this algorithm but could not make them work as intended. Those ideas are presented in the next section.

To give a comparison for the performance of our algorithm we use two different optimization algorithms. The first is a single step differential evolution (DE) algorithm and the other is Bayesian Optimization (BO) algorithm using Gaussian Processes. Python package gpflowopt [10] is used as the Bayesian Optimization framework.

Algorithm 2 The BOAC Framework

1: Randomly initialize critic network $Q\left(s, a|\theta^Q\right)$ and actor $\mu\left(s|\theta^\mu\right)$ with weights θ^Q and θ^μ
2: Use Latin Hypercube sampling [5] to sample α candidate designs from the design space.
3: Perform simulations to all of them and let them form the initial training set
4: Generate pseuod-samples and intiate replay buffer R;
5: **repeat**
6: Select action $a_t = \mu\left(s_t|\theta^\mu\right) + \mathcal{N}_t$ according to the current policy and exploration noise;
7: Execute action a_t and observe reward r_t and observe new state s_{t+1}
8: Store transition (s_t, a_t, r_t, s_{t+1}) in R_t
9: Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
10: Set $y_t = r\left(s_t, a_t\right) = R(s_{t+1}) = f(x_{t+1})$
11: Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i \left(y_i - Q\left(s_i, a_i|\theta^Q\right)\right)^2$
12: Update the actor policy using the sampled policy gradient: $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a \left(Q\left(s, a|\theta^Q\right) - \lambda * viol_i^2\right) \nabla_{\theta^\mu}\mu\left(s|\theta^\mu\right)$
13: **until** the number of required sampling iterations is reached.
14: Output the best design;

For testing our algorithm and comparing with other optimizers, we used three different unconstrained-optimization benchmark problems from the literature, namely, 3-dimensional Ackley function, 2-dimensional Shubert function and 6-dimensional Trid function. Their minimization values and lower/upper bound information are given in TABLE I and the equations and plots are shown in Fig. 1. Notice that these plots are belong to 2D version of the test problems but we use higher dimensional version for Ackley and Trid function for our experiments.

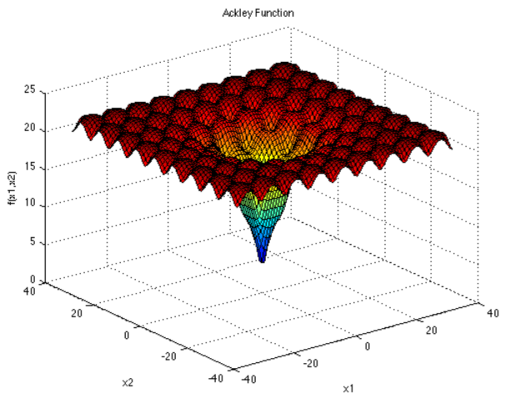
To compare the algorithm, we did 10 runs with each optimizer and averaged the convergence trends. Notice that although we limited the simulation budget with the number of evaluations, Bayesian Optimization has time complexity scaling with $\mathcal{O}(N^3)$ and time requirement for BO for single iteration is typically 20-50 times more expensive than both of our algorithms.

TABLE I: Benchmark Functions

Function(f)	Dimension	(LB, UB)	f(x*)
Ackley	3	(-15, 30)	0
Shubert	2	(-10, 10)	-186.7309
Trid	6	(-36, 36)	-50

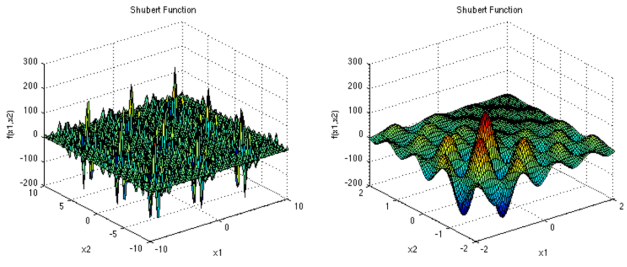
A. Experiment 1:Verifying DECN

Our initial experiment is to check whether critic network's inference is good enough to help for convergence and/or



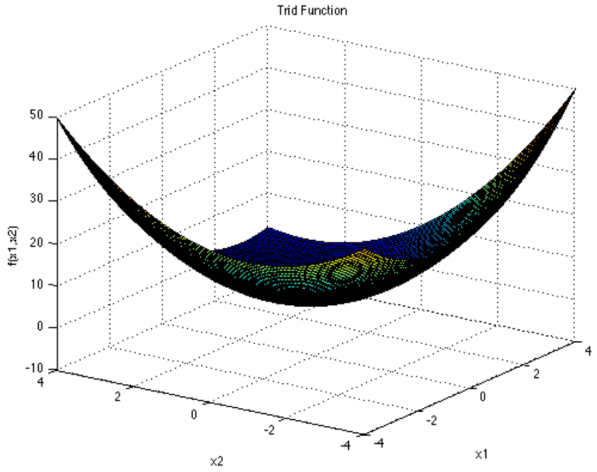
$$f(\mathbf{x}) = -a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$

(a) Ackley Function



$$f(\mathbf{x}) = \left(\sum_{i=1}^{5} i\cos((i+1)x_1 + i)\right)\left(\sum_{i=1}^{5} i\cos((i+1)x_2 + i)\right)$$

(b) Shubert Function



$$f(\mathbf{x}) = \sum_{i=1}^{d}(x_i - 1)^2 - \sum_{i=2}^{d} x_i x_{i-1}$$

(c) Trid Function

Fig. 1: Optimization Test Functions

accuracy of the optimizer. For that purpose, we demonstrate a performance comparison between the Single-Step Differential Evolution (SSDE) algorithm and Differential Evolution with Critic Network (DECN). This comparison will measure how good the critic's state-action approximation with respect to random child selection. Considering that even beating random can be hard for many cases, we find this comparison very critical in terms of measuring critic's inference power.

The experimental results for critic-only case are shown in Fig. 2. The show results are the average of 10 runs for 500 iterations. Using critic network for directing child selection improved the convergence trend of the optimization significantly. In 500 iterations DE_rand could not reach to global optima in none of the runs. On the other hand, all the runs for DE+Critic find the global optimal points in all testing cases.

This experiment has shown us that by employing a critic network and using pseudo-samples to train it, we could have meaningful inferences about space of the problem we are trying to optimize.

B. Experiment 2: Replacing DE Operations with Actor

Observing that critic network can help making inference on the solution space, our next aim is to find next query points with the help of an actor-network, in other words, we now want an actor for replacing child population phase of DE algorithm.

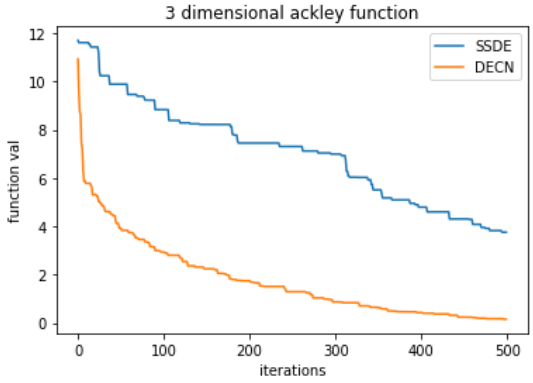
BOAC algorithm is indeed a modified version of DECN algorithm where we replaced the DE operations for generating child population with an actor network. In this part of the algorithm we are highly interested in the performance of BOAC vs DECN. BOAC framework mimics one of the state-of-the-art RL algorithm and we compare it with a critic empowered traditional optimization method. Notice that both algorithm uses the same network structure for critic so we will be able to make apples-to-apple comparison for actor's performance.

We present results for three optimization algorithms in this section: BOAC vs DECN vs Bayesian Optimization and average of the 10 runs are shown in Fig. 3 for each of the test problems.

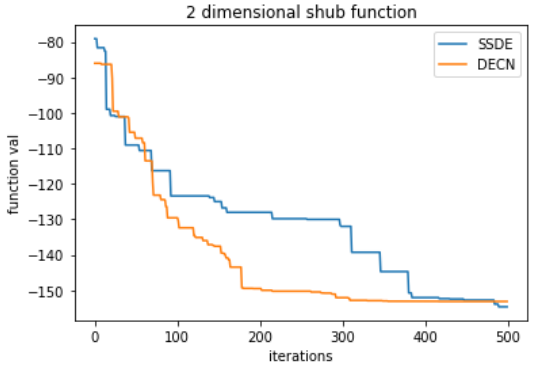
For the first test case, i.e., Ackley function, BO failed to find near-optima solutions for given budget in all runs. On the other hand, BOAC becomes the best-performing algorithm with clear margins. DECN has worse convergence than BOAC while maintaining a better performance than BO.

BO performed relatively better on Shubert function by hitting the global optima for 6 times out of 10 iterations within the given simulation budget but both BOAC and DECN outperforming BO. We notice that BOAC is the only algorithm hitting the optimal point at each run in the given simulation budget.

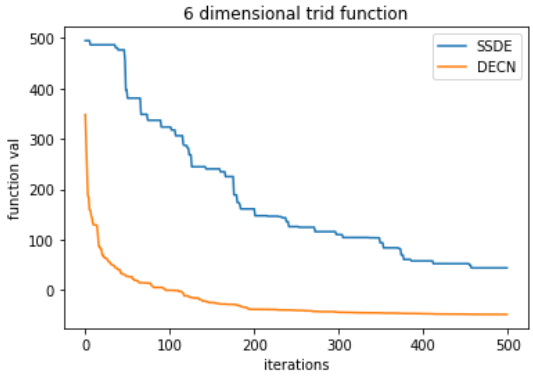
We see that BO performed very poorly for Trid function, however, both BOAC and DECN are able to find the global optima for this function and BOAC has comparable convergence trend with DECN, although it started from less advantegous point on average.



(a) Ackley Function

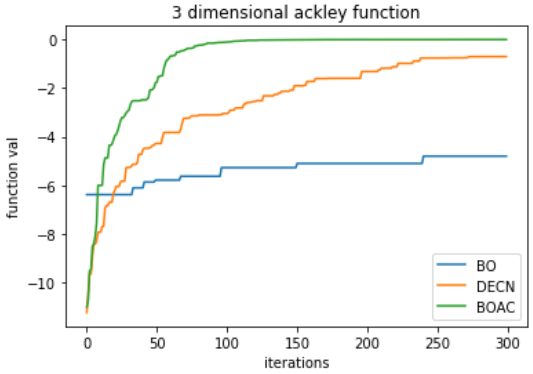


(b) Shubert Function

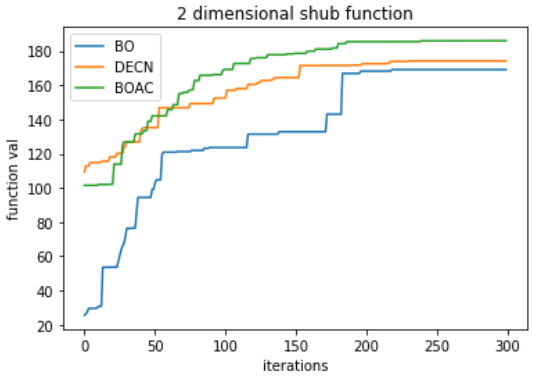


(c) Trid Function

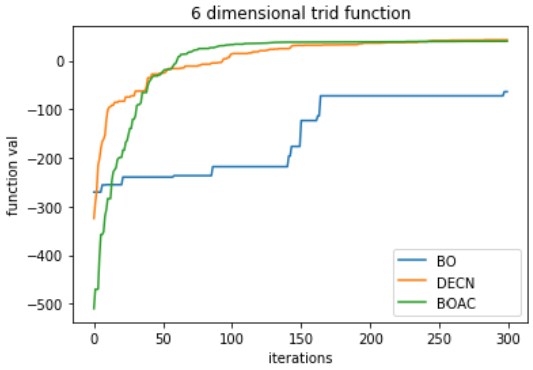
Fig. 2: Optimization Plots for DE-rand (SSDE) DE+Critic (DECN)



(a) Ackley Function



(b) Shubert Function



(c) Trid Function

Fig. 3: Optimization Plots for DE-rand DE+Critic

In overall, we are satisfied to see that actor did not fall behind the DE steps in terms of generating the candidate query points. Notice that DE steps are there for couple decays and saturated for improvement, our actor-critic model has a very large room for improvement since this is a vanilla version and very first implementation of both authors. Our guess is that we could have much improvement on the algorithm from 2 main factors. First is by improving the the Neural Network and training architecture and the second is by modifying the actor network by an exploration term.

V. Future Directions

A. *Embedding Exploration in Actor-Critic*

One of the bottlenecks in NOAC algorithm is that the algorithm collapses to the critic's optimal point very quickly. Its due to the fact that actor trains itself so that the action taken will result in a state with highest critic approximation. If critic's update is not significant then the actor suggests near query points at consecutive iterations which causes algorithm to collapse to that point. If that point is not optimal we stuck with a sub-optimal solution.

To tackle this problem, we spend significant time on embedding exploration term in actor's loss function. However, none of these attempts are resulted better than exploration noise method. Our methods are mainly based on modifying actor loss using the exploration bonus (EB) in the following way:

$$L(\theta^{\mu}) = -1 * \sum_{i} Q(x_i, a_i) + EB - \lambda * viol_i^2$$

In our experiment, using actor's actions directly without exploration noise can lead us to global optimal point very quickly if we can deal with the collapsing problem. We mainly tried two other methods for the exploration bonus in actor's loss in order to replace exploration noise added to actor's action.

1) Add exploration bonus based on how far the query state is from the last visited state.

$$EB = \lambda_{EB} * abs(x_{t+1} - x_t)$$

2) Define a count based exploration [11] in a discrete format. We tried to cluster our previously visited states and define a EB based on which cluster the candidate solution falls in.

B. Transfer Learning

One of the most exciting properties of RL for optimization is to transfer the learning in different domains [12]. To our best-knowledge none of the previous optimization algorithms have such property.

We initially intended to find similar functions in terms of shape and bounds and try transferring the learning from one domain to the other. However, we could not have investigated this aspect due to time constraints.

VI. Conclusion

In this project, we investigated the applicability of recent advances in Reinforcement Learning (RL) algorithms on black-box optimization problems. We mainly presented two performance analysis in this work. First analysis is to demonstrate inference power of a critic-network's approximation on state-action pairs. We showed that if we used the critic to approximate unseen state-action pairs and make a ranking based on that approximation, we have significant improvement on required number of simulations for finding global optimal points. The second analysis introduces an actor-critic algorithm for global optimization problems. Although, this algorithm needs and will have too much fine tuning and improvement it outperforms the Bayesian Optimization package and its critic-only counterpart.

References

[1] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Berlin, Heidelberg: Springer-Verlag, 2005.

[2] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, Nov 1995, pp. 1942–1948 vol.4.

[3] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 2951–2959.

[4] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning*. The MIT Press, 2005.

[5] M. D. McKay, R. J. Beckman, and W. J. Conover, "Comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[6] V. Konda, "Actor-critic algorithms," Ph.D. dissertation, Cambridge, MA, USA, 2002, aAI0804543.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *ICLR*, Y. Bengio and Y. LeCun, Eds., 2016.

[9] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992.

[10] N. Knudde, J. van der Herten, T. Dhaene, and I. Couckuyt, "GPflowOpt: A Bayesian Optimization Library using TensorFlow," *arXiv preprint – arXiv:1711.03845*, 2017. [Online]. Available: https://arxiv.org/abs/1711.03845

[11] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," *CoRR*, vol. abs/1606.01868, 2016. [Online]. Available: http://arxiv.org/abs/1606.01868

[12] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, pp. 1633–1685, Dec. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1577069.1755839