Report on Benchmarking the Performance of Reinforcement Learning Techniques

Fangyuan Zhang, Xing Fan, Juntao Feng

Abstract

We implemented Augmented Random Search (ARS), Policy Search with Natural Gradient (NPG) and Trust Region Policy Optimization (TRPO) on five Open AI Gym tasks, which are Swimmer-v2, Hopper-v2, HalfCheetah-v2, Walker2d-v2 and InvertedPendulum-v2. We benchmarked these three algorithms' performance and obtained the training curve as well as highest rewards. Then we analyzed the experimental result and compared these reinforcement learning algorithms.

I. Background

Reinforcement learning (RL) refers to an agent in an environment, where it stays in one of many states and takes one of many actions. Once an action is taken, the environment delivers a reward to the agent as feedback. How the environment reacts to certain actions and how the environment reacts to certain actions is defined by a model. The model can be known or unknown to us. If the mode is unknown, it is categorized as model-free reinforcement learning. In our project, we adopted three algorithms: Augmented Random Search (ARS)(1), Policy Search with Natural Gradient (NPG)(2) and Trust Region Policy Optimization (TRPO)(3) to deal with model-free situations. Our goal is to find policies to maximize the average reward on given tasks and benchmark the performance of these three algorithms.

Here, we use an oracle model for reinforcement learning to quantify an algorithm's performance. Each time the algorithm queries the oracle with a proposed policy, the oracle returns to the algorithm a sequence of states, actions, and rewards which represent a trajectory generated from the system according to current policy. The number of oracle queries needed for solving a problem is defined as oracle complexity or sample complexity. We use it to measure the algorithms performance in the experiment. The complexity is calculated as the number of iterations multiplied by number of rollouts or trajectories during one iteration.

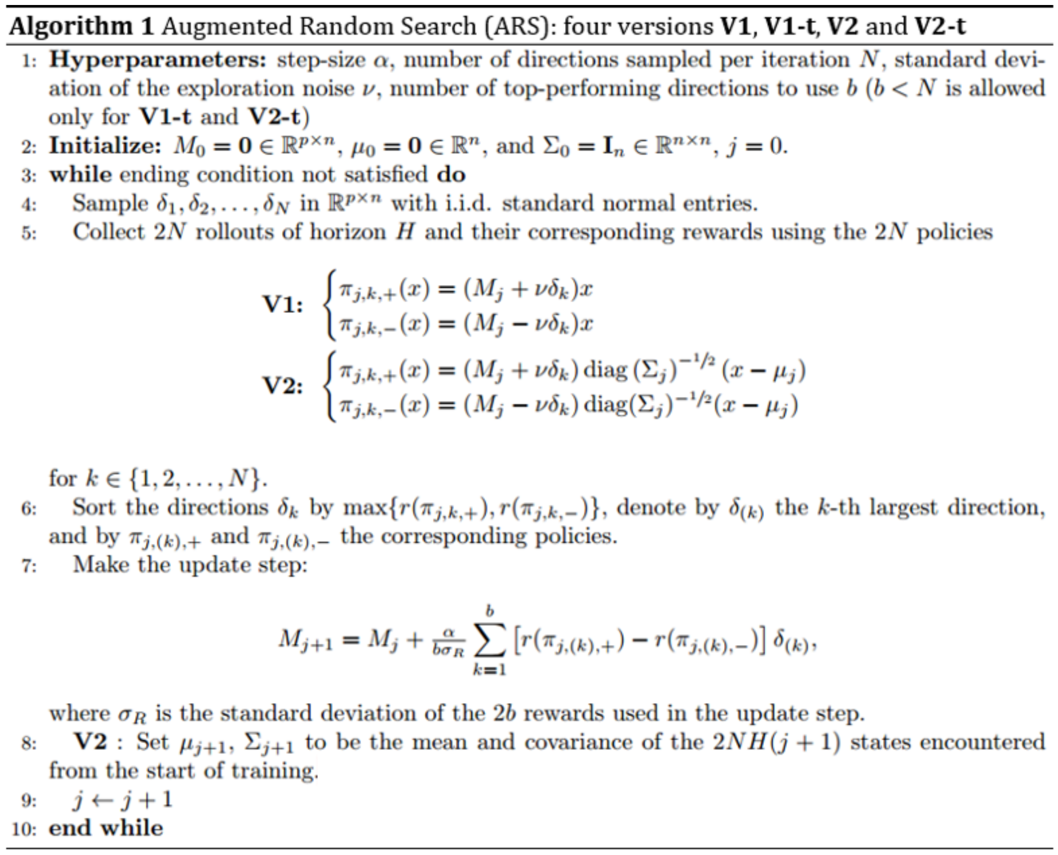
II. Algorithms

A. Augmented Random Search

Horia Mania and his colleagues proposed several variations of basic random search (BRS)(4) method to deal with model-free reinforcement learning problems.

The first variation, named v1, is obtained from BRS by scaling its update steps using the standard deviation of the rewards collected in each iteration (see Line 7 of Algorithm1). The intuition of this normalization step is that a fixed step size α could cause harmful changes between large and small steps. As training processing, the variance of rewards gained in each direction grows sharply. Therefore, a normalized step-size could help the agent pick actions more wisely.

Another variant called v2 trains policies in state space which have been normalized by a mean and standard deviation computed online (see Line 5 of Algorithm1). Different states may have a different range of coordinate values. Normalizing states guarantees the exploration in each direction of parameter space will have equal influence over various state components.



Algorithm 1 Augmented Random Search (ARS): four versions **V1**, **V1-t**, **V2** and **V2-t**

```
1:  Hyperparameters: step-size $\alpha$, number of directions sampled per iteration $N$, standard deviation of the exploration noise $\nu$, number of top-performing directions to use $b$ ($b < N$ is allowed only for V1-t and V2-t)
2:  Initialize: $M_0 = \mathbf{0} \in \mathbb{R}^{p\times n}$, $\mu_0 = \mathbf{0} \in \mathbb{R}^n$, and $\Sigma_0 = \mathbf{I}_n \in \mathbb{R}^{n\times n}$, $j = 0$.
3:  while ending condition not satisfied do
4:      Sample $\delta_1, \delta_2, \dots, \delta_N$ in $\mathbb{R}^{p\times n}$ with i.i.d. standard normal entries.
5:      Collect $2N$ rollouts of horizon $H$ and their corresponding rewards using the $2N$ policies
            V1: $\begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)x \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)x \end{cases}$
            V2: $\begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)\operatorname{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)\operatorname{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \end{cases}$
        for $k \in \{1, 2, \dots, N\}$.
6:      Sort the directions $\delta_k$ by $\max\{r(\pi_{j,k,+}), r(\pi_{j,k,-})\}$, denote by $\delta_{(k)}$ the $k$-th largest direction, and by $\pi_{j,(k),+}$ and $\pi_{j,(k),-}$ the corresponding policies.
7:      Make the update step:
            $M_{j+1} = M_j + \frac{\alpha}{b\sigma_R} \sum_{k=1}^{b} \left[ r(\pi_{j,(k),+}) - r(\pi_{j,(k),-}) \right] \delta_{(k)},$
        where $\sigma_R$ is the standard deviation of the $2b$ rewards used in the update step.
8:      V2 : Set $\mu_{j+1}$, $\Sigma_{j+1}$ to be the mean and covariance of the $2NH(j+1)$ states encountered from the start of training.
9:      $j \leftarrow j + 1$
10: end while
```

Furthermore, Mania proposed v1-t and v2-t methods which were corresponding to v1 and v2, respectively. These two methods update searching steps with an average over directions that obtain high rewards, among all the directions.

B. Policy Search with Natural Gradient and Trust Region Policy Optimization

Policy gradient (PG) approach is also a model-free method. Instead of randomly exploring parameter space, PG picks the steepest ascent direction in the metric of the parameter space as the best choice to approximate a stochastic policy directly. Since a fixed step size α is hard to be consistent with the scales of the reward, NPG and TRPO methods adapt normalized step size δ (see Line 7 of Algorithm2). Rajeswaran and his colleagues' experimental results illustrate that a normalized step size is numerically more stable and easier to tune.

NPG and TRPO use advantage function to evaluate the actions the agent takes. To reduce the variance of advantage function, the algorithm cooperates baseline, which is independent of the action space, in the advantage function. In our project, we use the quadratic baseline to balance the evaluation variance.

Other than the baseline, NPG and TRPO also adopt Fisher information matrix (5) to improve the performance of advantage function. Fisher information matrix is a measure of how much model predictions change with local parameter changes. Therefore, Fisher information matrix could help agent pick more reasonable directions.

Nevertheless, Generalized advantage estimation (GAE)(6) procedure is used for estimating advantage function. Initially, when the policy is very far from the correct answer, even if the movement direction is not along the gradient (biased), it is beneficial to make consistent progress and not bounce around due to high variance. Thus, high bias estimates of the policy gradient, corresponding to smaller λ values make fast initial progress. However, after this initial phase, it is important to follow an unbiased gradient, and consequently, the low-bias variants corresponding

Algorithm 2 Policy Search with Natural Gradient (NPG) & Trust Region Policy Optimization(TRPO)

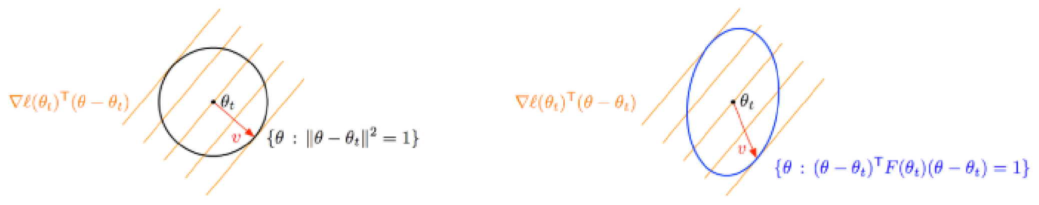
Notations: $M = \{S, A, R, T, \rho_0\}$. S, A, R are a (continuous) set of states, set of actions, and reward function respectively; length of rollouts T, probability distribution over initial states ρ_0, stochastic policy π; objective function $\eta(\pi)$, value function $V^\pi(s, t)$, Q function $Q^\pi(s, a, t)$, and advantage function $A^\pi(s, a, t)$ as:

$$\eta(\pi) = \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{\pi,\mathcal{M}}\left[\sum_{t=1}^{T} r_t\right] \qquad V^\pi(s,t) = \mathbb{E}_{\pi,\mathcal{M}}\left[\sum_{t'=t}^{T} r_{t'}\right]$$

$$Q^\pi(s,a,t) = \mathbb{E}_{\mathcal{M}}\Big[\mathcal{R}(s,a)\Big] + \mathbb{E}_{s'\sim\mathcal{T}(s,a)}\Big[V^\pi(s',t+1)\Big] \qquad A^\pi(s,a,t) = Q^\pi(s,a,t) - V^\pi(s,t)$$

Steps:

```
1: Initialize policy parameters to $\theta_0$
2: for $k = 1$ to $K$ do
3:     Collect trajectories $\{\tau^{(1)}, \dots \tau^{(N)}\}$ by rolling out the stochastic policy $\pi(\cdot;\theta_k)$.
4:     Compute $\nabla_\theta \log \pi(a_t|s_t;\theta_k)$ for each $(s,a)$ pair along trajectories sampled in iteration $k$.
5:     Compute advantages $A_k^\pi$ based on trajectories in iteration $k$ and approximate value function $V_{k-1}^\pi$.
6:     Compute policy gradient according to
           $\nabla_\theta \hat{\eta}(\theta) = g = \frac{1}{T}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}^\pi(s_t,a_t,t)$
7:     Compute the Fisher matrix $F_{\theta k}$ and perform gradient ascent
           $\hat{F}_{\theta_k} = \frac{1}{T}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \nabla_\theta \log \pi_\theta(a_t|s_t)^T \qquad \theta_{k+1} = \theta_k + \sqrt{\frac{\delta}{g^T \hat{F}_{\theta_k}^{-1} g}}\, \hat{F}_{\theta_k}^{-1} g$
       TRPO: Since the estimated improvement is computationally expensive, applying a line
       search to determine a more optimal step size before performing the update.
       *Since we have $N$ trajectories, we also apply average over all trajectories.
8:     Update parameters of value function in order to approximate $V_k^\pi(s_t^{(n)}) \approx R(s_t^{(n)})$, where
       $R(s_t^{(n)})$ is the empirical return computed as $R(s_t^{(n)}) = \sum_{t'=t}^{T} \gamma^{(t'-t)} r_t^{(n)}$. Here $n$ indexes
       over the trajectories.
9: end for
```


$\nabla\ell(\theta_t)^\top(\theta-\theta_t)$
θ_t
v
$\{\theta : \|\theta-\theta_t\|^2 = 1\}$
$\nabla\ell(\theta_t)^\top(\theta-\theta_t)$
θ_t
v
$\{\theta : (\theta-\theta_t)^\top F(\theta_t)(\theta-\theta_t) = 1\}$

(a) The Euclidean steepest descent direction. (b) The Fisher steepest descent direction.

Fig. 1. Euclidean and Fisher steepest descent direction.

to larger λ values show better asymptotic performance. Even without the use of GAE (i.e. $\lambda = 1$), we observe good asymptotic performance. But with GAE, it is possible to get faster initial learning due to reasons discussed above.

In a standard optimal control problem, according to the optimal control theory, the input of a linear quadratic regulator can be written as a linear function in the state. When the model is known, the linear coefficient, as known as the controller K, could be solved by Algebraic Riccati Equation (ARE). However, this approach is not suitable for a model-free setting. So in order to measure the performance of the policy gradient algorithm, we apply a Gaussian linear policy to interact with the environments.

C. Comparison of ARS and NPG/TRPO

- All the algorithms (ie. ARS, NPG and TRPO) explore the parameter space rather than the action space to update the policy for an agent. Exploring in parameter space allows agent dealing with continuous action space.
- ARS uses deterministic policies, while NPG/TRPO uses stochastic policies. Since in some states several actions could achieve the same reward, which is the same to the real world, the agent does not need to make a choice among these actions. Thus stochastic policies seem to be more reasonable than deterministic policies.
- ARS picks several random directions, and uses their rewards to update policy parameters; however, NPG/TRPO produces some trajectories and calculates the average of their natural gradient to update policy parameters.
- ARS scales step size by the covariance of rewards to obtain a more reasonable step size; NPG/TRPO uses a normalized step size with the same effect.
- ARS requires an improvement of policy with equal influence on all the states, thus v2 method uses normalized states; NPG/TRPO requires an unbiased advantage function estimation, thus it uses GAE procedure to set the tolerant level of bias.

III. Experimental result

As mentioned before, we train the (gaussian) linear policy with NPG, TRPO and ARS algorithms. The tasks aim on robots to learn locomotion in five gym environments.

For ARS, we use the v2-t version with an optimal set of parameters; for NPG and TRPO, we set learning rate to 0.95, the parameter λ for GAE to 0.97, the number of trajectories as 10 and normalized step size δ as 0.1 for Swimmer and Hopper, as 0.1 for HalfCheetah, and 0.01 for the rest environments; and for TRPO, we set KL distance as 0.01.

A. Performance versus iterations

First, we recorded the maximum rewards per iteration of each algorithm. The intuition is that we wanted to measure the efficiency of each iteration for these algorithms. In other words, assuming in each iteration, the same number of samples are taken, we attempted to figure out how much per iteration could contribute to updating policies. Fig. 2 are the performance curves for environments Swimmer-v2, Hopper-v2, HalfCheetah-v2 and Walker2d-v2. For InvertedPendulum-v2, we will discuss it in the latter part. It is obvious that for the simple environments, such as Swimmer-v2 and HalfCheetah-v2, ARS converges fast. But for Walker2d-v2, ARS trains extremely slow compared with the other two algorithms.

B. Performance versus oracle complexity

Then we record the maximum rewards per RL algorithms query. It comes from the definition of oracle complexity, which can be simply represented as the number of rollouts or samples. Since each query access the same information about the system: rollouts from fixed policies, and the associated states and rewards, oracle complexity could measure whether one algorithm is making better use of this information than the other. We adopt exactly the same parameters mentioned above, and the results are shown in Fig. 3.

For simple environments, such as Swimmer-v2 and InvertedPendulum-v2, ARS is more stable and converges faster as well. Especially for Hopper-v2, the performance of policy gradient algorithms grows with significant variance. Even for some complicated environments, such as Walker2d-v2, ARS still performs more steadily than NPG and TRPO. However, for Walker2d-v2, it seems that ARS fails in this task, but in fact, it is because ARS converges much slower than the other algorithms in this environment, with about 5000 iterations. This method learns almost nothing at the beginning of the training, but it does success with the highest performance after more exploration. The training curve is shown in Fig. 4.

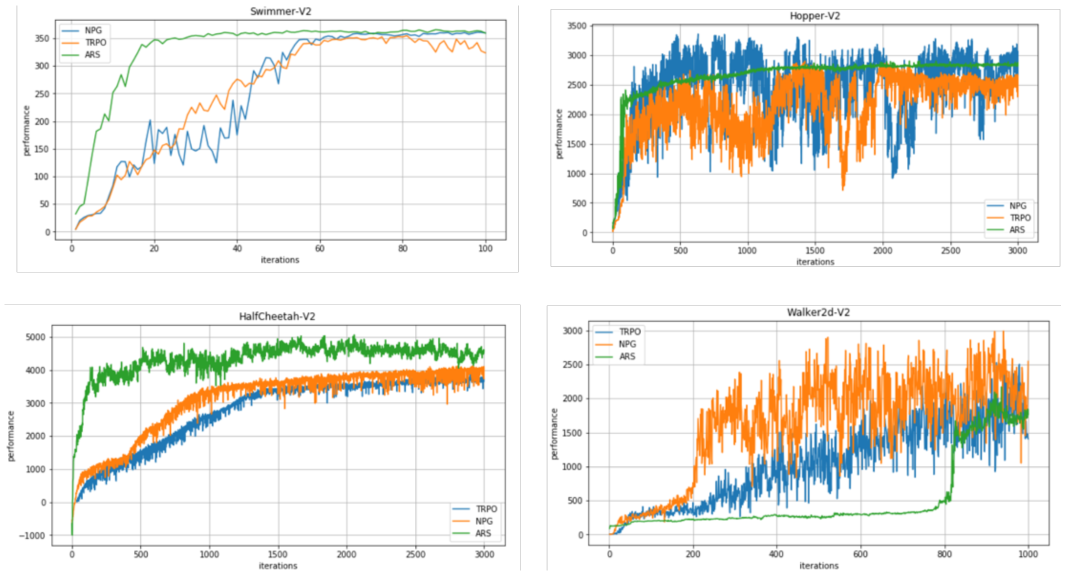


Fig. 2. Performance versus number of iterations in different environments. (a) Swimmer-v2; (b) Hopper-v2; (c) HalfCheetah-v2; (d) Walker2d-v2

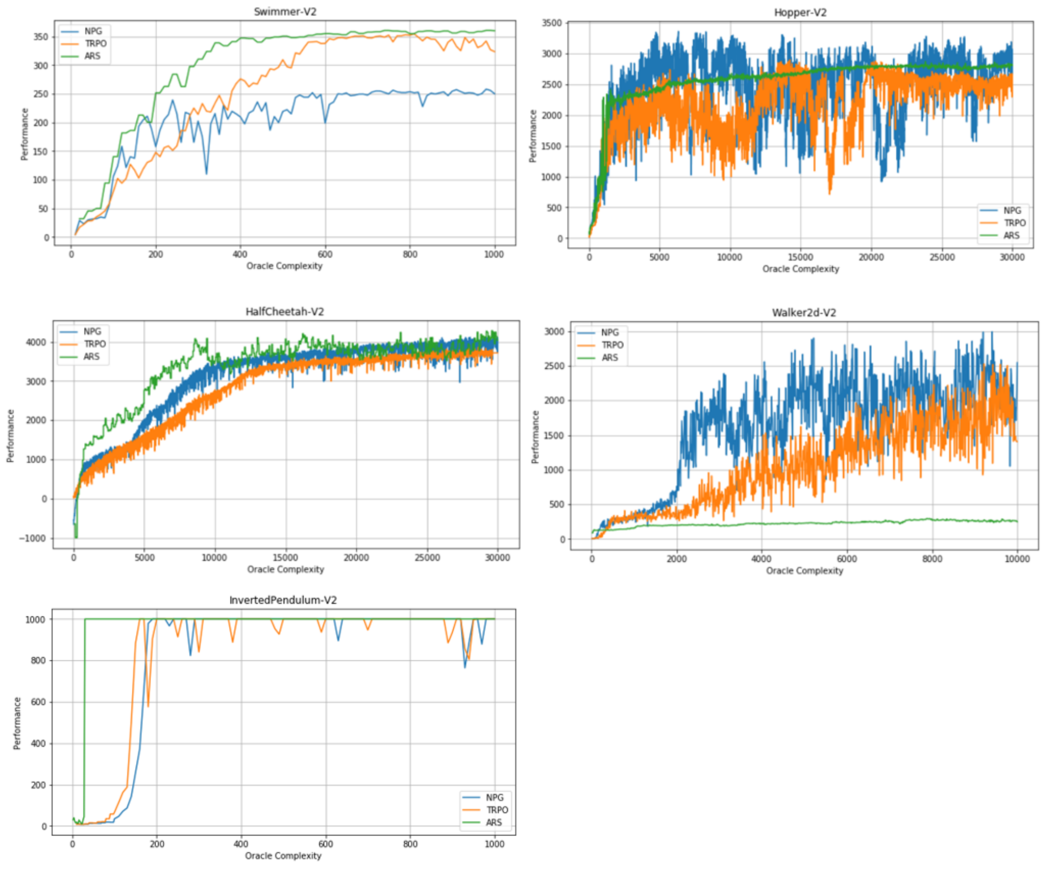


Fig. 3. Performance versus oracle complexity in different environments. (a) Swimmer-v2; (b) Hopper-v2; (c) HalfCheetah-v2; (d) Walker2d-v2; (e) InvertedPendulum-v2.

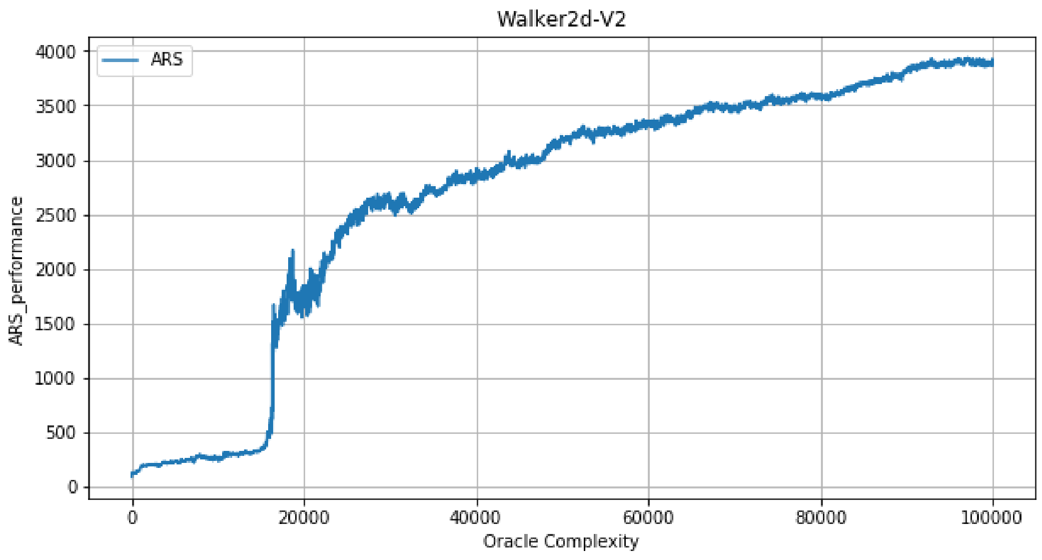


Fig. 4. Performance versus oracle complexity of ARS in Walker2d-v2.

C. Best performance in each environment

We also recorded the best performance of algorithms could achieve in Table. I. Notice that they could reach the best performance in the early stage. Especially for NPG, it may have decreasing performance as training processing after convergence. Furthermore, though NPG seems to win over TRPO on performance in several environments, the truth is that the variance of NPG's performance is extremely high, and the average performance of TRPO wins NPG's in general.

TABLE I
BEST PERFORMANCE OF EACH ALGORITHM

Task	ARS	NPG	TRPO
Swimmer-v2	366	270	355
Hopper-v2	2906	3362	2902
HalfCheetah-v2	5060	4117	3862
Walker2d-v2	3938	2992	2528
InvertedPendulum-v2	1000	1000	1000

D. Multiple random seeds evaluations of ARS

RL algorithms show large training variances, so we also explore the performance of ARS over many random seeds. We measured performance under two environments using the parameters mentioned above. Swimmer-v2: 50 seeds are chosen randomly from 1 to 500 for 50 experiments in the environment Swimmer-v2 with the mentioned parameters. Most of the trials converge to a reward value of around 360 after 500 iterations.

InvertedPendulum-v2: We randomly choose 50 seeds from 1 to 500 for experiments with different sets of parameters and the performance distribution of the best set of parameters is shown in Fig. 5. In each trial, after 500 iterations, some converges to 1000, while some does not converge.

IV. Conclusion

In general, ARS could achieve higher performance than NPG and TRPO, but it needs long exploration in complicated situations. The reason lies here is that in complicated environments, ARS updates policies almost blindly at the beginning of training, since little information has

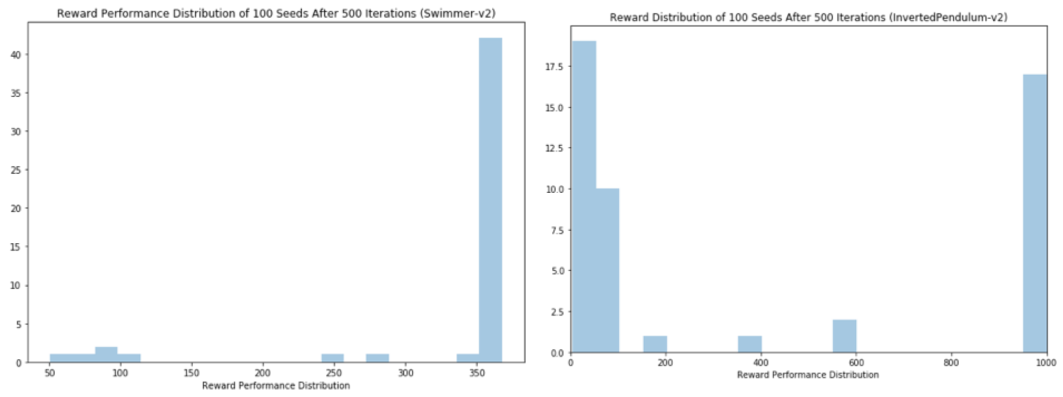


Fig. 5. Performance of 50 random seeds. (a) Swimmer-v2; (b) InvertedPendulum-v2

been discovered, while policy gradient provides a recommended direction related to advantage function and this direction could bring about wiser actions at the beginning, letting NPG and TRPO converge faster than ARS. While as training processing, policy gradient is more likely to lead the exploration in parameter space to a local optimal position, thus NPG and TRPO may converge at local optimal values. On the contrary, ARS explores much more directions than policy gradient, so that ARS is more possible to get out of the local optimal position and move to global optimal values. Thus we conclude, policy gradient methods (ie. NPG and TRPO) can make better use of the information than random search (ie.ARS) with the conduction of advantage function, especially in some complicated environment. While ARS could achieve higher performance and stability than NPG and TRPO. So if we have unbounded, or at least, adequate resources (ie. time, computation resources, etc.), ARS is the best choice; if we have limited resources, then we need to make a trade-off between performance and cost: if lower cost and shorter training period mean a lot to us, NPG and TRPO would be a better choice.

References

[1] H. Mania, A. Guy, and B. Recht, Simple random search provides a competitive approach to reinforcement learning, 2018.
[2] A. Rajeswaran, K. Lowrey, E. Todorov, and S. Kakade, Towards Generalization and Simplicity in Continuous Control, 2017.
[3] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, Trust Region Policy Optimization, 2015.
[4] A. D. Flaxman, A. T. Kalai, and H. B. McMahan. Online convex optimization in the bandit setting:gradient descent without a gradient. *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 385394, 2005.
[5] M. Fazel, R. Ge, S. M. Kakade, and M. Mesbahi, Global Convergence of Policy Gradient Methods for the Linear Quadratic Regulator, 2018.
[6] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, High-Dimensional Continuous Control Using Generalized Advantage Estimation, 2015.