Stochastic Sampling of Parametric Policies

Leonid Keselman
lkeselma@cs.cmu.edu

Abstract

We present experimental results which demonstrate that some standard reinforcement learning problems can be solved with very simple methods. Namely, we investigate how policies that are parameterized as linear functions of observations are able to solve this problems. As opposed to recent work which focuses on gradient-descent strategies, our optimization methods for these policies are based on stochastic sampling of the parameter space.

1. Introduction

Research in neural networks has seen extended success, achieving human-level on image classification [8]. With the addition of reinforcement learning into the framework, these neural networks have also demonstrated human-level performance on old video games [17]. Many of recent papers focus on either playing 2D Atari games [17] or control for various forms of locomotion [13, 23]. However, many of these methods focus on proving their results on fairly simple environments [9], such as controlling a classic inverted pendulum, a swing-up task for the inverted pendulum, and the classic *Mountain Car* environment [18]. The first of these is known to have an optimal linear controller, while the latter two present a challenge where the system lacks the control authority to immediately pursuing the goal, requiring moving opposite the direction of the goal to gain potential energy and then moving towards the goal.

As interest in reinforcement learning grows, it seems sensible to ask the question of *how hard are these problems?*. Some recent work tackles this question through brute-force exploration of different model sizes with a given optimizer [12]. In contrast, our work tries to address this empirically, by asking *can very simple models and methods solve these problems?*

2. Related Work

2.1. "Deep" Methods

Many modern uses of reinforcement learning methods make use of parametric function approximators. Many of these model-free methods can broadly be categorized into two categories [19]. The first category are those that, given state (and perhaps additional information), approximate the value of that state. A classic example would be Q-learning [27]. The second category is those that given state directly learn to output the optimal action, such as the classic REINFORCE method [28]. When full state can be observed, such as Atari games with screen capture input, Deep Q-Learning variants offer state-of-the-art performance [10]. However, more classic control problems, where state a set of numerical measurements about the system, policy-gradient methods such as REINFORCE, DDPG [13] and TRPO [23] tend to perform better across a wide variety of benchmarks [4].

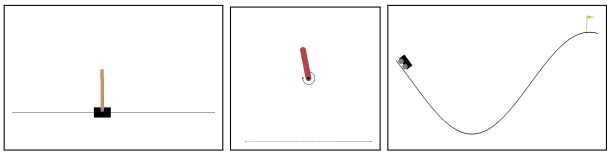

Figure 1. A visual example of the three sample environments from OpenAI gym [2]. For further details about them see section 3.1.

In this work, we only consider a parametric policy, without explicitly trying to estimate policy gradient. Recent work has shown that simple linear policies [21] with natural gradient methods can provide competitive performance with deep models, while being much easier to train.

2.2. Stochastic sampling

Many policy-gradient methods such as REINFORCE, have large variance in estimating the optimal policy due to the the sampling nature of the algorithm. There are many ways to reduce this variance [20], often by improving baselines or by implementing better assumptions about temporal difference learning. However, these methods have added complexity for minimal performance improvement [4].

In performing black-box optimization of unknown objective functions, the *No Free Lunch* theorems [29] demonstrate that random search cannot be beat over the space of all objective functions. There exist empirical results that demonstrate random sampling to be effective baseline

for general optimization [6]. Additionally,for performing hyper-parameter optimization for neural networks [1], random search can be difficult to outperform.

One interesting method is to use random sampling of parameters. If one is careful about the choice of distribution and updates the distribution parameters with respect to measured costs (or rewards), there exist multiple methods with promising theoretical and empirical performance. One interesting method which we re-implement is Policy Exploring Parameter Gradients (PGPE) [25]. Here, parameters for the policy are sampled from a distribution (such as a multivariate Gaussian), and the parameters of the distribution are updated through gradient ascent. This allows one to use non-differentiable policies as one only computes differential updates from rewards into distribution parameters. Theoretical analysis of PGPE has shown it to be a lower variance estimator than REINFORCE [30]. Additionally, there are extensions of PGPE to multimodal distributions [24], which we also re-implement and evaluate against random restarts.

A recent paper has claimed very effective results by using random search in place of reinforcement learning [15]. However, in our analysis, we see this a numerical gradient method as opposed to one which considers optimization via utilizing a stochastic parameter space. Similar to our work, they demonstrate a very simple method can also be competitive in solving control problems. They also demonstrate the value in whitening one's reward spaces and being careful to weight common and rare experiences correctly. We will consider applications of these insights to our proposed methods as future work.

3. Methods

3.1. Environment

For our experiments, we will be using the OpenAI gym environment designed for reinforcement learning [2]. For this work, we focus on three of the simple environments, which are still often used as standard evaluation tasks in research papers [9] and are the basis of reportedly challenging homework in graduate courses [5]. For a visual example of these three environments, see figure 1.

The first's environment, called CartPole, as the goal of maintaining an upright inverted pendulum using discrete applications of force to the cart, either a fixed size force to the left or a fixed size force to the right for every iteration; the initial condition is a random, almost upright configuration. The second environment, called Pendulum, has the goal of swinging up and controlling an inverted pendulum by applying a continuous valued torque (with a torque limit) at the base of an arm joint; the initial condition is a random joint angle for the arm. The third environment, called MountainCar [18], has the goal of driving a cart over the mountain on the right by discrete applications of force, similar to the first environment; the initial condition in this environment is always in a slightly perturbed starting location in the middle basin.

In the Pendulum and MountainCar environments, the system doesn't have the control authority to directly accomplish the task, and requires going in the opposite direction of the reward, gaining potential energy, and then solving the task by going the other direction. MountainCar is generally more challenging than Pendulum, and is more challenging for all known methods [4] (see Table 1). We suspect this because Pendulum is able to sample all random initial conditions (thus not requiring exploration, only samples), while MountainCar always has an initial condition in the basin. Additionally, we evaluate on the discrete action form of MoutainCar, which we found to be more difficult than the one where the action space consists of continuous values torques; this is unsurprising as the former has a more discontinuous objective landscape.

3.2. Formulation

We consider a parametric policy which takes some state $\vec{s}$, has parameters $\vec{\theta}$ and produces an appropriate action.

$$\pi(\vec{s}) = f_\theta(\vec{s}) \tag{1}$$

In our case, we will consider a policy that is a linear combination of some $\vec{x}$, with an activation funcion $g(y)$.

$$f_\theta(s) = g(\sum_i \theta_i x_i) \tag{2}$$

The activation function is designed for each specific environment. For example, to handle actions which are binary, integral and numerical respectively, we use the following:

$$\begin{aligned} g(y) &= y > 0 \\ g(y) &= \max(-1, \min([x], 1)) \\ g(y) &= \max(-1, \min(y, 1)) \end{aligned} \tag{3}$$

To handle pairwise dependencies in our observation vector ($\vec{s}$) with a linear a linear classifier, we expand our feature vector to include a bias term and all pairwise features. Although the below notation is over-complete, we use a minimal representation where $\vec{x}$ is dimension $d + \frac{d^2+d}{2} + 1$ for a some $s \in \mathbb{R}^d$

$$x_{ij} = s_i \cdot s_j \mid \forall s_i \forall s_j \in [s_1, s_2, ..., s_n, 1] \tag{4}$$

We consider the case where each element in the parameter vector $\vec{\theta}$ is sampled from probability distribution.

$$\theta_i \sim P(z) \tag{5}$$

For example, we consider the uniform distribution of parameter values. Although one could also consider a normal

distribution, and the PGPE [25] method uses a normal distribution with a gradient-updated μ and σ.

$$\begin{aligned} P(z) &= U(-1,1) \\ P(z) &= N(0,1) \\ P(z) &= N(\mu,\sigma) \end{aligned} \tag{6}$$

3.3. Tested methods

- **Uniform Random Sampling (URS)** Our primary hypothesis, that random sampling in parameter space provides adequate solutions. In practice, we sample each dimension independently from the distribution $P(z) = U(-10, 10)$
- **PGPE** [25] Our implementation of PGPE. We perform ablation studies on various components of implementations in section 4.3, including symmetric [25] and multimodal [24] variants.
- **CMA-ES** [7] A commonly used baseline in black box optimization and continuous control. It iterative fits a multivariate Gaussian distribution to the returned rewards. PGPE and CMA-ES have been shown to be comparable methods [22]. Compared to PGPE, CMA-ES can handle modeling off-diagonal terms in the σ matrix, and includes a weighting term where lower costs get higher weight in the gradient update.

4. Results

4.1. Uniform Random Sampling

To see our statistical results for uniform random sampling, see figure 2. For these experiments, we evalatute *how long does it take URS to find a valid solution*, as, over time, URS will explore all possible configurations. In short, URS is able to find valid solution for all three environments, in roughly a distribution that resembles a decaying exponential.

For inverted pendulum, the environment fails if arm begins to fall or the cart moves too far to the left or right. However, in most cases, a valid solution is found in a few hundred trials.

For swing-up, because the environment has random starting angles, the OpenAI simulation of swing-up often finds itself in a "good" random configuration and finds a solution even faster in expectation. Although, not all of these configurations would work over multiple random restarts of the simulation. As seen in table 2, a stable solution can again be found in a few hundred trials.

Lastly, for the MountainCar task, random search often finds a solution in a few thousand iterations, taking much longer to find a valid solution. As explained in section 3.1, the increased challenge in solving MountainCar is due to initial condition sampling and the control authority of the configured environment. This is the same result seen in other reported benchmarks of reinforcement learning methods [4].

Method	**Trials** μ	**Trials** σ	**Time (s)** μ	**Time (s)** σ
DQN	83	10	38.8	21.9
URS	679	539	0.4	0.3
PGPE	400	348	2.1	2.0
CMA-ES	127	95	1.3	1.0

Table 1. Cartpole Results over 10 trials. Comparing Deep Q Networks, Uniform Random Sampling and Policy Gradient Parameter Exploration. URS is the least sample efficient, but has nearly no overhead. DQN is the most sample efficient but has such large overhead that it takes an order of magnitude longer. PGPE has roughly the same overhead as URS but is more sample efficient.

Method	**Trials** μ	**Trials** σ	**Time (s)** μ	**Time (s)** σ
DDPG	70	6	296.2	50.4
URS	320	225	3.4	2.5
PGPE	180	118	1.8	1.1
CMA-ES	1022	1438	10.3	14.5

Table 2. Pendulum Results over 10 trials. Comparing Deterministic Deep Policy Gradients, Uniform Random Search, PGPE and CMA-ES in solving the swing-up task. The task is considered complete when the method gets a stable upright solution. However, since the initial conditions are random, sometimes it is very easy to find a solution. With this criteria, the probabilistic sampling methods perform very well. Ideally we'd test over a larger set of random initial conditions to conclude that the controller actually works, which is how results are reported in sections 4.3 and 4.4.

4.2. Deep learning baselines

We perform an evaluation against deep learning baselines in for CartPole and Pendulum environments and report the results in tables 1 and 2. In CartPole (table 1), the reference Deep Q Network [17] is the most sample efficient, but takes the longest due to overhead. On the other hand, random search is the fastest, despite being the least sample efficient. This is true even on a single-core instance of random evaluation. In the Pendulum case (table 2), we see similar results, except we compare against DDPG [13], which we find has fairly low variance, but orders of magnitude longer running time.

4.3. PGPE variants

To validate PGPE's performance, we report the results of several ablation studies on the algorithm's performance. All of these results are on swing-up inverted Pendulum and,

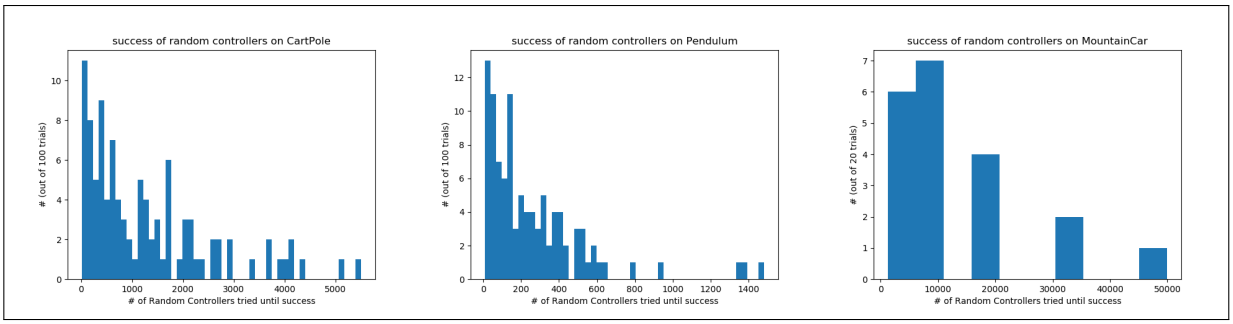

Figure 2. Histograms demonstrating the performance of Uniform Random Search on solving three different problems (left-to-right): controlling an inverted pendulum, performing a swing-up and control of an inverted pendulum, and a MountainCar task. More details are in section 3.1. Success in this case is defined as having a trial which solves the problem within the OpenAI timestep limit (200 iterations)

unlike the above experiments, the algorithm has to work on 5 random configurations to be deemed a success.

4.3.1 Sample number

We experimented with changing the number of samples given to each PGPE iteration. See table 3 for results, where we found 15 samples worked best. We also experimented with adding uniformly randomly sampled datapoints into optimization but this decreased performance.

Samples	Average Runtime (s)
5	25
10	24
15	12
20	76

Table 3. Results of different sampling numbers in PGPE for the Pendulum task. These are the means over 10 runs.

4.3.2 Baseline design

In many policy gradient methods, advantage models are key in obtaining good performance. In an advantage model, rewards are compared against a baseline expected reward and this greatly improves convergence speed [19]. While PGPE has a several ways of establishing baselines [30], we implement a moving average of the form

$$b_{t+1} = (1 - \alpha)R + \alpha b_t$$

Here, $\alpha \in [0, 1]$ while R is either the mean or minimum reward over the observed iteration. We experimented with different α values See figure 3 for empirical results. With the current numbers of samples, the results are inconclusive.

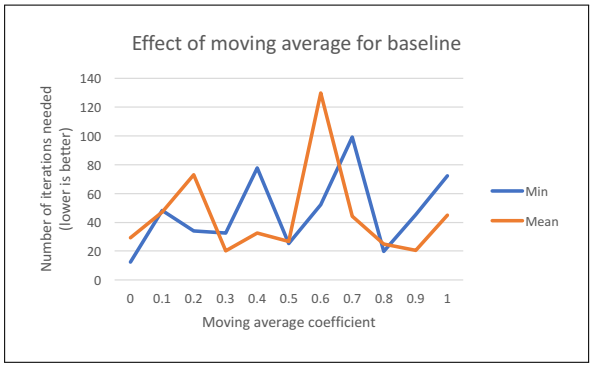

Figure 3. Experimental results of varying α for design of a baseline. Results are inconclusive with our sample size of 10 trials.

Method	**Trials**	**Time (seconds)**
SymPGPE	4179 ± 3975	273 ± 253
PGPE	470 ± 330	23 ± 15

Table 4. In contrast to what the published paper claimed, in our testing, the symmetric variant performed much worse on Pendulum swing-up

4.3.3 Symmetric samples

The original PGPE paper presents a symmetric variant, which uses twice the number of samples per timestep but reportedly has converges faster even in real-time. In the symmetric variant, noise vectors are sampled and then symmetrically added and subtracted from the mean, giving an estimate of numerical gradient. This is similar to the optimization method that recent was recently published and shown to outperform reinforcement learning [15]. We did not find this to be the case. Instead, the symmetric variant often struggled to get significant variation, often getting stuck in the local minima of only trying to swing-up directly, taking much longer to discover the trick. See figure 4 for results.

Method	Modes	Trials	Time (seconds)
Standard	1	470 ± 330	23 ± 15
Multimode	3	1024 ± 401	80 ± 37
	5	1451 ± 1372	97 ± 83
Random inits	3	792 ± 557	43 ± 32
	5	770 ± 289	40 ± 15

Table 5. We compare the multi-modal variant of PGPE against simply doing multiple random initialization in parallel. Our baseline method using multiple random initializations outperforms the Gaussian Mixture Model proposed in the paper.

4.3.4 Multimodal distributions

Later variants of PGPE [24] implement a Gaussian mixture model for the distribution sampling parameter instead of the single Gaussian used in the original paper. We replicate this method and compare it against trying multiple random initializations computed in parallel. See figure 5 for numerical results. We find that using multiple random intializations and updating them in parallel leads to a correct answer faster than using a Gaussian mixture model. However, it performs worse than the unimodal variant overall, but each individual random seed requires less time than in the unimodal case. Thus, multiple modal PGPE still offers some promise in other optimization environments.

4.4. Comparison

In table 6 we show results on our challenging situation where multiple suceed in random settings are required to pass the task. We can see that CMA and PGPE are comparable in both run-time and iteration count, while random search takes only 4 or 5 times longer.

Method	Trials	Time (second)
URS	2335 ± 1568	81 ± 49
PGPE	470 ± 330	23 ± 15
CMA-ES	538 ± 651	16 ± 19

Table 6. Results on the challenging environment where multiple trials of the simulation have to succeed. PGPE and CMA are competitive. Other methods such as MaxLIPO[14] did not converge in under an hour and results are excluded.

5. Conclusion

Due to its simplicity of implementation and minimal overhead, random sampling can be very efficient in solving simple control problems. While it requires an order of magnitude more iterations than more sophisticated methods, it can sometimes still win in runtime cost, even on a single-core implementation. In a distributed computing context, such non-sequential evaluation methods would have even more value. There are methods for scaling deep reinforcement learning methods to distributed compute settings [16] but they don't scale perfectly, and require sophisticated parameter tuning.

Simple variants of probabilistic sampling in optimization, such as PGPE, have been demonstrated to be effective in robustly solving control problems. Variations in the algorithm have been testing and implemented and presented.

Additionally, there is a large literature on analyzing connections between black box optimization and reinforcement learning. They provided a useful resource and reference for the preparation of this project, and contain further insights and methods which we'd like to explore in the future [3, 11, 19, 26].

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[3] Konstantinos Chatzilygeroudis, Roberto Rama, Rituraj Kaushik, Dorian Goepp, Vassilis Vassiliades, and Jean-Baptiste Mouret. Black-box data-efficient policy search for robotics. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 51–58. IEEE, 2017.

[4] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *CoRR*, abs/1604.06778, 2016.

[5] Katerina Fragkiadaki and Ruslan Satakhutdinov. Deep Reinforcement Learning and Control Spring 2017, CMU 10703. `https://katefvision.github.io/`.

[6] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM, 2017.

[7] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[9] Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

[10] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

[11] Jemin Hwangbo, Christian Gehring, Hannes Sommer, Roland Siegwart, and Jonas Buchli. ROCK - Efficient black-box optimization for policy learning. In *IEEE-RAS International Conference on Humanoid Robots*, volume 2015-Febru, pages 535–540, 2015.

[12] C. Li, H. Farkhoor, R. Liu, and J. Yosinski. Measuring the Intrinsic Dimension of Objective Landscapes. *ArXiv e-prints*, April 2018.

[13] 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.

[14] Cédric Malherbe and Nicolas Vayatis. Global optimization of Lipschitz functions. 2017.

[15] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *CoRR*, abs/1803.07055, 2018.

[16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[18] Andrew William Moore. Efficient memory-based learning for robot control. (UCAM-CL-TR-209), November 1990.

[19] Jan Peters and Gerhard Neumann. Policy Search: Methods and Applications. *International Conference on Machine Learning Tutorials* `https://icml.cc/2015/tutorials/PolicySearch.pdf`, 2015.

[20] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2219–2225. IEEE, 2006.

[21] Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham Kakade. Towards generalization and simplicity in continuous control. *CoRR*, abs/1703.02660, 2017.

[22] Thomas Rückstiess, Frank Sehnke, Tom Schaul, Daan Wierstra, Yi Sun, and Jürgen Schmidhuber. Exploring parameter space in reinforcement learning. *Paladyn*, 1(1):14–24, 2010.

[23] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[24] Frank Sehnke, Alex Graves, Christian Osendorfer, and Jurgen Schmidhuber. Multimodal parameter-exploring policy gradients. In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, pages 113–118. IEEE, 2010.

[25] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

[26] Freek Stulp and Olivier Sigaud. Policy improvement methods: Between black-box optimization and episodic reinforcement learning. *Jfpda 2013*, 2, 2012.

[27] Christopher Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[28] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.

[29] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[30] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. Analysis and improvement of policy gradient estimation. In *Advances in Neural Information Processing Systems*, pages 262–270, 2011.