Exploration in Action Space

Anirudh Vemula
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: vemula@cmu.edu

Wen Sun
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: wensun@cs.cmu.edu

J. Andrew Bagnell
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: dbagnell@ri.cmu.edu

Abstract—**Parameter space exploration methods with black-box optimization have recently been shown to outperform state-of-the-art approaches in continuous control reinforcement learning domains. In this paper, we examine reasons why these methods work better and the situations in which they are worse than traditional action space exploration methods. Through a simple theoretical analysis, we show that when the parametric complexity required to solve the reinforcement learning problem is greater than the product of action space dimensionality and horizon length, exploration in action space is preferred. This is also shown empirically by comparing simple exploration methods on several toy problems.**

I. Introduction

Recently, in a series of blog posts[1] and in [12], Ben Recht and colleagues reached the following conclusion: "Our findings contradict the common belief that policy gradient techniques, which rely on exploration in the action space, are more sample efficient than methods based on finite-differences."

That's a conclusion that we have often felt has much merit. In a survey (with Jens Kober and Jan Peters) [10], we wrote:

"Black box methods are general stochastic optimization algorithms (Spall, 2003) using only the expected return of policies, estimated by sampling, and do not leverage any of the internal structure of the RL problem. These may be very sophisticated techniques (Tesch et al., 2011) that use response surface estimates and bandit-like strategies to achieve good performance. White box methods take advantage of some of additional structure within the reinforcement learning domain, including, for instance, the (approximate) Markov structure of problems, developing approximate models, value-function estimates when available (Peters and Schaal, 2008c), or even simply the causal ordering of actions and rewards. A major open issue within the field is the relative merits of the these two approaches: in principle, white box methods leverage more information, but with the exception of models (which have been demonstrated repeatedly to often make tremendous performance improvements, see Section 6), the performance gains are traded-off with additional assumptions that may be violated and less mature optimization algorithms. Some recent work including (Stulp and Sigaud, 2012; Tesch et al., 2011) suggest that much of the benefit of policy search is achieved by black-box methods."

Many empirical examples–the classic Tetris [17] for instance–demonstrate that Cross-Entropy or other (fast) black-box heuristic methods are generally far superior to any policy gradient method and that policy gradient methods often achieve orders of magnitude better performance than methods demanding still more structure like, *e.g.* temporal difference learning [16]. In this paper, we set out to study why black-box parameter space exploration methods work better and in what situations can we expect them to perform worse than traditional action space exploration methods.

II. The Structure of Policies

Action space exploration methods, like REINFORCE [18], SEARN [6], PSDP [3], AGGREVATE [14, 15], LOLS [5], leverage more structure than parameter space exploration methods. More specifically, they understand the relationship (*e.g.*, the Jacobian) between a policy's parameters and its outputs. We could ask: *Does this matter?* In the regime of large parameter space and small output space problems as explored often by our colleagues, like atari games [13], it might. (Typical implementations also leverage causality of reward structure as well, although one might expect that is relatively minor.)

In particular, the intuition behind the use of action space exploration techniques is that they should perform well when the action space is quite small compared to the parametric complexity required to solve the Reinforcement Learning problem.

III. Experiments

We test this intuition across three experiments: MNIST, Linear Regression and LQR. The code for all these experiments is published at https://github.com/LAIRLAB/ARS-experiments.

A. MNIST

To investigate this we, like [1], consider some toy RL problems, beginning with a single time step MDP. In particular, we start with a classic problem: MNIST digit recognition. To put in a bandit/RL framework, we consider a $+1$ reward for getting the digit correct and a -1 reward for getting it wrong. We use a LeNet-style architecture[2], [11]. The total number of

---

[1]http://www.argmin.net/2018/03/20/mujocoloco/

[2]Two convolution layers each with 5×5 kernels with 10 and 20 output channels, followed by two fully connected layers of 320 and 50 units and a output softmax layer resulting in 1-hot encoding of dimensionality 10
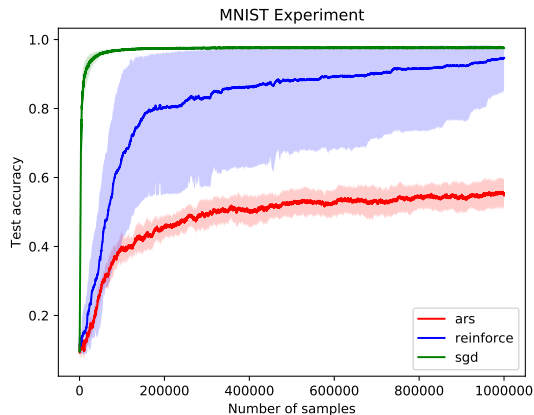
Fig. 1: Test accuracy of different approaches against number of samples

trainable parameters in this architecture is $d = 21840$. The experimental setup is described in greater detail in Appendix C2.

We then compare the learning curves for vanilla REIN-FORCE with supervised learning and with ARS **V2-t**, an augmented random search in parameter space procedure indicated in Mania et al. [12]. Figure 1 demonstrates the results where solid lines represent mean test accuracy over 10 random seeds and the shaded region corresponds to ± 1 standard deviation. Clearly, in this setting where the parameter space dimensionality significantly exceeds the action space dimensionality, we can see that action space exploration methods such as REINFORCE outperform parameter space exploration methods like ARS.

B. Linear Regression

Following the heuristic of the *Linearization Principle* introduced in [1], we attempt to get a handle on the trade-off between sample complexity in parameter space and complexity in action space by considering another simple, one step problem: linear regression with a *single* output variable and d input dimensionality, and thus d parameters. We consider learning a random linear function.

Before empirically studying REINFORCE and ARS, we first perform a regret analysis (Appendix A) on online linear regression for three learning strategies: (1) online gradient descent, (2) exploration in action space, and (3) exploration in parameter space, which correspond to full information setting, linear contextual bandit setting, and bandit setting respectively. The online gradient descent approach is simply OGD from Zinkevich [19] applied to full information online linear regression setting, and for exploration in parameter space, we simply used the BGD algorithm from Flaxman et al. [7], which completely ignores the properties of linear regression setting and works in a bandit setting. The algorithm for random exploration in action space—possibly the simplest linear contextual bandit algorithm, shown in Alg. 3, operates in the middle: it has access to feature vectors and performs

random search in prediction space to form estimations of gradients. The analysis of all three algorithms is performed in the online setting: no statistical assumptions on the sequence of linear loss functions, and multi-point query per loss function [2] is not allowed.[3]

The detailed algorithms and analysis are provided in Appendix C3. The main difference between exploration in action space and exploration in parameter space is that exploration in action space can take advantage of the fact that the predictor it is learning is *linear* and it has access to the linear feature vector (i.e., the Jacobian of the predictor). The key advantage of exploration in action space over exploration in parameter space is that *exploration in action space is input-dimension free*.[4] More specifically, one can show that in order to achieve ϵ average regret, the algorithm (Alg. 2) performing exploration in parameter space requires $O(\frac{d^2}{\epsilon^4})$ samples (we ignore problem specific parameters such as the maximum norm of feature vector, the maximum norm of the linear predictor, and the maximum value of prediction, which we assume are constants), while the algorithm (Alg. 3) performs exploration in action space requires $O(\frac{1}{\epsilon^4})$, which is not explicitly dependent on d.

We empirically compare the test squared loss of REIN-FORCE, natural REINFORCE (which simply amounts to whitening of input features) [8] and the ARS **V2-t** method discussed in Mania et al. [12] with classic follow-the-regularized-leader (Supervised Learning). The results are shown in Figures 2a, 2b and 2c, where solid lines represent mean test squared loss over 10 random seeds and the shaded region corresponds to ± 1 standard deviation. The learning curves match our expectations, and moreover show that this bandit style RE-INFORCE lies between the curves of supervised learning and parameter space exploration: that is action space exploration takes advantage of the Jacobian of the policy itself and can learn much more quickly.

C. LQR

Finally, we consider what happens as we extend the time horizon. In Appendix B, we consider finite horizon ($H > 1$) optimal control task with deterministic dynamics, fixed initial state and a linear stationary policy. We show that we can estimate the policy gradient via a random search in parameter space as ARS did (Eq. 21 in Appendix B), or we can do a random search in action space across all time steps independently (Eq. 20 in Appendix B). Comparing the norm of both gradient estimates, we can see that the major difference is that the norm of the gradient estimate from random exploration in parameter space (Eq. 21) linearly scales with the dimensionality of state space (i.e., dimensionality of parameter space as we assume linear policy), while the norm of the gradient estimate from random search in action space (Eq. 20) linearly scales with the

---

[3]Note that the ARS algorithms presented in Mania et al. [12] actually take advantage of the reset property of episodic RL setting and perform two-point feedback query to reduce the variance of gradient estimations

[4]It will dependent on the output dimension, if one considers multivariate linear regression.

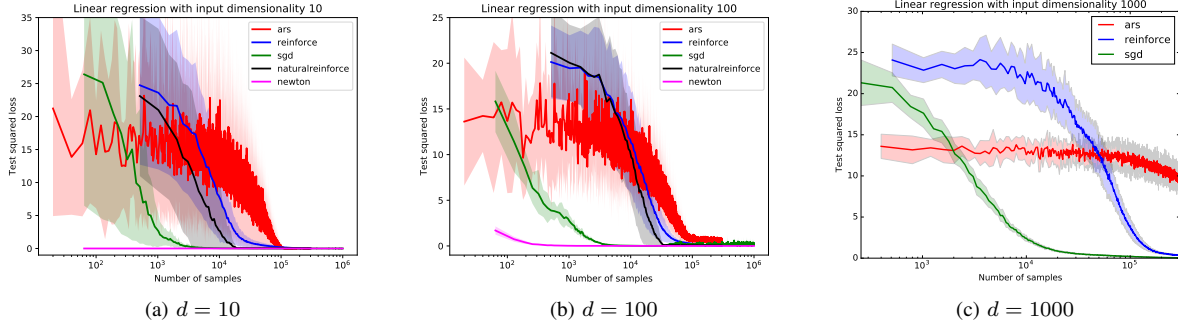(a) $d = 10$ (b) $d = 100$ (c) $d = 1000$

Fig. 2: Linear Regression Experiments with varying input dimensionality

product of horizon and action space dimensionality. Hence, when the dimensionality of the state space is smaller than the product of horizon and action space dimensionality, one may prefer random search in parameter space, otherwise random search in action space is preferable. Note that for most of the continuous control tasks in OpenAI gym [4], the horizon is significantly larger than the state space dimensionality.[5] This explains why ARS [12] outperforms most of the action space exploration methods in these tasks.

The simplest setting to empirically evaluate this is where we'd all likely agree that a model based method would be the preferred approach: a finite-horizon Linear Quadratic Regulator problem with 1-d control space and a 100-d state space. We then compare random search (ARS **V1-t** from Mania et al. [12]) versus REINFORCE (with ADAM [9] as the underlying optimizer), in terms of the number of samples they need to train a stationary policy that reaches within 5% error of the non-stationary optimal policy's performance with respect to the horizon H ranging from 10 to 160. Fig. 3 shows the comparison where the statistics are averaged over 10 random seeds (mean $\pm$ standard error).

From Fig. 3 we see that as H increases, both algorithms require larger number of samples.[6] While ARS is stable across different random seeds, we found reinforce becomes more and more sensitive to random seeds when H is larger, and performance becomes less stable. However notice that when H is small (e.g., $H \leq 40$), we see that REINFORCE has lower variance as well and can consistently outperform ARS. Though we would expect that REINFORCE would require more samples than ARS when $H \geq 100$, which is the dimension of state space, in this experiment we only notice this phenomenon at $H = 160$, though the variance of REINFORCE is already terrible at that point.

---

[5]Take Walker2d-v2 as an example, H is usually equal to 1000. The action space dimension is 6, and the dimension of the state space is 17. Hence random exploration in action space is actually randomly searching in 6000 dimension space, while random search in parameter space is searching in 17 dimension space.

[6]Our simple analysis on finite horizon optimal control with deterministic dynamic shows that both gradient estimators' norm linearly depends on the objective value measured at the current policy. As H increases, the total cost increases.
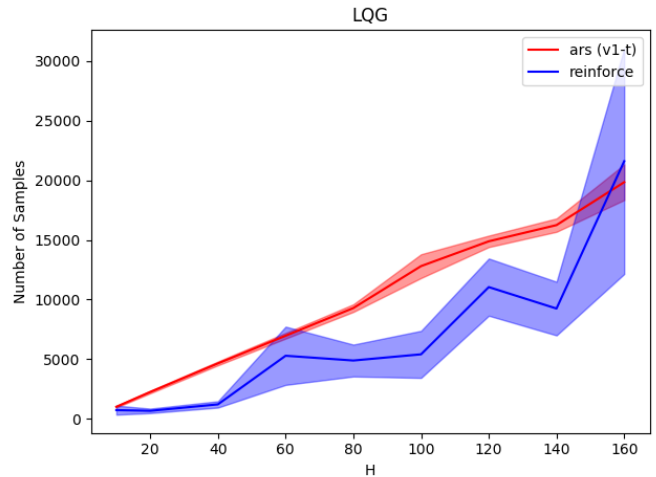

Fig. 3: LQG with state dimensionality $d = 100$ with varying horizon H

IV. CONCLUSION

Mania et al. [12] have shown that simple random search in parameter space is a competitive alternative to traditional action space exploration methods. In our work, we show that this is only true for domains where the parameter space dimensionality is significantly smaller than the product of action space dimensionality and horizon length. For domains where this does not hold true, action space exploration methods such as REINFORCE [18] are more sample efficient as they do not have an explicit dependence on parameter space dimensionality.

V. ACKNOWLEDGEMENTS

REFERENCES

[1] An outsider's tour of reinforcement learning. http://www.argmin.net/2018/05/11/outsider-rl/. Accessed: 2019-05-26.

[2] Alekh Agarwal, Ofer Dekel, and Lin Xiao. Optimal algorithms for online convex optimization with multi-point bandit feedback. In *COLT*, pages 28–40. Citeseer, 2010.

[3] J Andrew Bagnell, Sham M Kakade, Jeff G Schneider, and Andrew Y Ng. Policy search by dynamic programming. In *Advances in neural information processing systems*, pages 831–838, 2004.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] Kai-wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. Learning to search better than your teacher. In *ICML*, 2015.

[6] Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 2009.

[7] Abraham D Flaxman, Adam Tauman Kalai, and H Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 385–394. Society for Industrial and Applied Mathematics, 2005.

[8] Sham Kakade. A natural policy gradient. *NIPS*, 2002.

[9] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[12] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

[13] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[14] Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

[15] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *ICML*, 2017.

[16] RichardS. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[17] Christophe Thiery and Bruno Scherrer. Building controllers for tetris. *ICGA Journal*, 32:3–11, 2009.

[18] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

[19] Martin Zinkevich. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *ICML*, 2003.

APPENDIX

A. Detailed Algorithms and Analysis for Linear Regression

We present our algorithms in the online learning setting with square loss function at time step t as $\ell_t(w) = (w^T x_t - y_t)^2$, where $x_t \in \mathbb{R}^d, y_t \in \mathbb{R}$. We design three different levels of feedback: (1) For classic supervised learning, at each round the learner receives feature x_t and then the learner makes a prediction $w_t^T x_t$; it will receives the loss $\ell_t(w_t) = (\hat{y}_t - y_t)^2$ together with y_t; (2) For random search in parameter space (e.g., ARS), we assume we operate in a full bandit setting: once the learner proposes some predictor w_t, it will only receives the loss measured at w_t: $\ell_t(w_t) = (w_t^T x_t - y_t)^2$ without explicit information on x_t or y_t; (3) For random search in action space (e.g., REINFORCE), we operate in a linear contextual bandit setting: the learner receives feature x_t, and then the learner makes prediction $\hat{y}_t = w_t^T x_t$, it receives the loss measured at $\hat{y}_t$, i.e., $(y_t - \hat{y}_t)^2$ but no y_t. For simplicity in analysis, we assume the learner always chooses prediction w from a pre-defined convex set $\mathbb{W}$: $w \in \mathbb{W} \subseteq \mathbb{R}^d$.

Below we present three algorithms. The first algorithm, OGD from [19], shown in Alg. 1, simply operates in the full information setting; the second algorithm, Random Search in Parameter Space, adopted from [7],[7] presented in Alg. 2, operates in a full bandit setting; the third algorithm, Random Search in Action Space, presented in Alg. 3, operates in a classic linear contextual bandit setting. Below we present the regret analysis for these three algorithms.

Algorithm 1 Online Gradient Descent [19]

1: **Input:** Learning rate $\mu_t \in \mathbb{R}^+$
2: Learner initializes $w_0 \in \mathbb{W}$.
3: **for** $t = 0$ to T **do**
4:    Learner receives x_t
5:    Learner makes prediction $\hat{y}_t = w_t^T x_t$.
6:    Learner receives loss $(\hat{y}_t - y_t)^2$ and y_t.
7:    Learner update: $w'_{t+1} = w_t - \mu_t(\hat{y}_t - y_t)x_t$.
8:    Projection $w_{t+1} = \arg\min_{w \in \mathbb{W}} \|w'_{t+1} - w\|_2^2$
9: **end for**

We assume w is bounded as $\|w\|_2 \leq \mathcal{W} \in \mathbb{R}^+$, x is bounded $\|x\|_2 \leq \mathcal{X} \in \mathbb{R}^+$, regression target y is bounded: $\|y\| \leq \mathcal{Y} \in \mathbb{R}^+$, to make sure the loss is bounded $|w^T x - y| \leq C \in \mathbb{R}^+$, and the gradient is bounded $\|\nabla_w \ell_t(w)\|_2 \leq C\mathcal{X}$. Note that with these assumptions, the loss function $\ell_t(w)$ is Lipschitz continuous with Lipschitz constant $L \leq (\mathcal{W}\mathcal{X} + \mathcal{Y})\mathcal{X}$. We assume that these constants, $\mathcal{W}, \mathcal{X}, \mathcal{Y}$, are independent of the feature dimension d. For the following analysis, we may omit constants that are independent of d and T, but will keep the dependency on d or T explicitly.

---

[7]The algorithm presented in [7] deals with projection more rigorously: to make sure that the randomly perturbed predictor $w_t + \delta u$ also belongs to $\mathbb{W}$, we have to project w'_t to a slightly shrunk convex set $(1 - \alpha)\mathbb{W}$ with $\alpha \in (0, 1)$. Here we simply assume it is acceptable to step outside $\mathbb{W}$ a little bit since δ will be extremely small when T is large.

Algorithm 2 Random Search in Parameter Space (BGD [7])

1: **Input:** Learning rate $\mu_t \in \mathbb{R}^+$, finite difference parameter $\delta \in \mathbb{R}^+$.
2: Learner initializes $w_0 \in \mathbb{W}$.
3: **for** $t = 0$ to T **do**
4:     Learner samples u uniformly from the surface of d-dim sphere.
5:     Learner chooses predictor $w_t + \delta u$.
6:     Learner only receives loss signal ℓ_t, which is $((w_t + \delta u)^T x_t - y_t)^2$.
7:     Learner update: $w'_{t+1} = w_t - \mu_t \frac{\ell_t d}{\delta} u$.
8:     Projection $w_{t+1} = \arg\min_{w \in \mathbb{W}} \|w'_{t+1} - w\|_2^2$.
9: **end for**

Algorithm 3 Random Search in Action Space

1: **Input:** Learning rate $\mu_t \in \mathbb{R}^+$, finite difference parameter $\delta \in \mathbb{R}^+$.
2: Learner initializes $w_0 \in \mathbb{W}$.
3: **for** $t = 0$ to T **do**
4:     Learner receives feature x_t
5:     Learner samples e uniformly from $\{-1, 1\}$
6:     Learner makes a prediction $\hat{y}_t = w_t^T x_t + \delta e$
7:     Learner only receives loss signal ℓ_t, which is $(\hat{y}_t - y_t)^2$
8:     Learner update: $w'_{t+1} = w_t - \mu_t \frac{\ell_t e}{\delta} x_t$.
9:     Projection $w_{t+1} = \arg\min_{w \in \mathbb{W}} \|w'_{t+1} - w\|_2^2$.
10: **end for**

Theorem A.1. *After T rounds, with $\mu_t = \frac{\mathcal{W}}{C\mathcal{X}\sqrt{t}}$ in Alg. 1, Alg. 1 has regret:*

$$\sum_{t=0}^{T} \ell_t(w_t) - \min_{x^* \in \mathbb{W}} \ell_t(w^*) \leq \mathcal{W}C\mathcal{X}\sqrt{T}. \tag{1}$$

With $\mu_t = \frac{\mathcal{W}\delta}{d(C^2+\mathcal{X}^2)\sqrt{T}}$ and $\delta = T^{-0.25}\sqrt{\frac{\mathcal{W}d(C^2+\mathcal{X}^2)}{2L}}$ in Alg. 2, Alg. 2 incurs regret:

$$\mathbb{E}\left[\sum_{t=0}^{T} \ell_t(w_t)\right] - \min_{w^* \in \mathbb{W}} \sum_{t=0}^{T} \ell_t(w^*) \leq \sqrt{\mathcal{W}d(C^2+\mathcal{X}^2)L}T^{3/4} \tag{2}$$

With $\mu_t = \frac{\mathcal{W}\delta}{(C^2+1)\mathcal{X}\sqrt{T}}$ and $\delta = T^{-0.25}\sqrt{\frac{\mathcal{W}(C^2+1)\mathcal{X}}{2C}}$, Alg. 3, Alg. 3 incurs regret:

$$\mathbb{E}\left[\sum_{t=0}^{T} \ell_t(w_t)\right] - \min_{w^* \in \mathbb{W}} \sum_{t=0}^{T} \ell_t(w^*) \leq \sqrt{\mathcal{W}(C^2+1)\mathcal{X}C}T^{3/4} \tag{3}$$

Proof: Result in Eq 1 is directly from [19] with the fact that $\|w\|_2 \leq \mathcal{W}$ and $\|\nabla_w \ell_t(w)\|_2 \leq C\mathcal{X}$ to any w and t.

To prove Eq. 2 for Alg. 2, we use the proof techniques from [7]. The proof is more simpler than the one in [7] as we do not have to deal with shrinking and reshaping the predictor set $\mathbb{W}$.

Denote $u \sim \mathbb{B}$ as uniformly sampling u from a d-dim unit ball, $u \sim \mathbb{S}$ as uniformly sampling u from the d-dim unit sphere, and $\delta \in (0, 1)$. Consider the loss function $\hat{\ell}_t(w_t) = \mathbb{E}_{v \sim \mathbb{B}}[\ell_t(w_t + \delta v)]$, which is a smoothed version of $\ell_t(w_t)$. It is shown in [7] that the gradient of $\hat{\ell}_t$ with respect to w is:

$$\nabla_w \hat{\ell}_t(w)|_{w=w_t} = \frac{d}{\delta}\mathbb{E}_{u \sim \mathbb{S}}[\ell_t(w_t + \delta u)u] \tag{4}$$

$$= \frac{d}{\delta}\mathbb{E}_{u \sim \mathbb{S}}[((w_t + \delta u)^T x_t - y_t)^2 u] \tag{5}$$

Hence, the descent direction we take in Alg. 2 is actually an unbiased estimate of $\nabla_w \hat{\ell}_t(w)|_{w=w_t}$. So Alg. 2 can be considered as running OGD with an unbiased estimate of gradient on the sequence of loss $\hat{\ell}_t(w_t)$. It is not hard to show that for an unbiased estimate of $\nabla_w \hat{\ell}_t(w)|_{w=w_t} = \frac{d}{\delta}(w_t + \delta u)^T x_t - y_t)^2 u$, the norm is bounded as $d(C^2+\mathcal{X}^2)/\delta$. Now we can directly applying Lemma 3.1 from [7], to get:

$$\mathbb{E}\left[\sum_{t=0}^{T} \hat{\ell}_t(w_t)\right] - \min_{w^* \in \mathbb{W}} \sum_{t=0}^{T} \hat{\ell}_t(w^*) \leq \frac{\mathcal{W}d(C^2+\mathcal{X}^2)}{\delta}\sqrt{T} \tag{6}$$

We can bound the difference between $\hat{\ell}_t(w)$ and $\ell_t(w)$ using the Lipschitiz continuous property of ℓ_t:

$$|\hat{\ell}_t(w) - \ell_t(w)| = |\mathbb{E}_{v \sim \mathbb{B}}[\ell_t(w + \delta v) - \ell_t(w)]|$$
$$\leq \mathbb{E}_{v \sim \mathbb{B}}[|\ell_t(w + \delta v) - \ell_t(w)|] \leq L\delta \tag{7}$$

Substitute the above inequality back to Eq. 6, rearrange terms, we get:

$$\mathbb{E}\left[\sum_{t=0}^{T} \ell_t(w_t)\right] - \min_{w^* \in \mathbb{W}} \sum_{t=0}^{T} \ell_t(w^*) \leq \frac{\mathcal{W}d(C^2+\mathcal{X}^2)}{\delta}\sqrt{T} + 2LT\delta. \tag{8}$$

By setting $\delta = T^{-0.25}\sqrt{\frac{\mathcal{W}d(C^2+\mathcal{X}^2)}{2L}}$, we get:

$$\mathbb{E}\left[\sum_{t=0}^{T} \ell_t(w_t)\right] - \min_{w^* \in \mathbb{W}} \sum_{t=0}^{T} \ell_t(w^*) \leq \sqrt{\mathcal{W}d(C^2+\mathcal{X}^2)L}T^{3/4} \tag{9}$$

To prove Eq. 3 for Alg. 3, we follow the similar strategy in the proof of Alg. 2.

Denote $\epsilon \sim [-1, 1]$ as uniformly sampling ϵ from the interval $[-1, 1]$, $e \sim \{-1, 1\}$ as uniformly sampling e from the set containing -1 and 1. Consider the loss function $\tilde{\ell}_t(w) = \mathbb{E}_{\epsilon \sim [-1,1]}[(w^T x_t + \delta\epsilon - y_t)^2]$. One can show that the gradient of $\tilde{\ell}_t(w)$ with respect to w is:

$$\nabla_w \tilde{\ell}_t(w) = \frac{1}{\delta}\mathbb{E}_{e \sim \{-1,1\}}[e(w^T x_t + \delta e - y_t)^2 x_t] \tag{10}$$

As we can see that the descent direction we take in Alg. 3 is actually an unbiased estimate of $\nabla_w \tilde{\ell}_t(w)|_{w=w_t}$. Hence Alg. 3 can be considered as running OGD with unbiased estimates of gradients on the sequence of loss functions $\tilde{\ell}_t(w)$. For an unbiased estimate of the gradient, $\frac{1}{\delta}e(w_t^T x_t + \delta e - y_t)^2 x_t$, its norm is bounded as $(C^2+1)\mathcal{X}/\delta$. Note that different from Alg. 2, here the maximum norm of the unbiased gradient *is*

independent of feature dimension d. Now we apply Lemma 3.1 from [7] on $\tilde{\ell}_t$, to get:

$$\mathbb{E}\left[\sum_{t=0}^{T} \tilde{\ell}_t(w_t)\right] - \min_{w^* \in \mathbb{W}} \sum_{t=0}^{T} \tilde{\ell}_t(w^*) \leq \frac{\mathcal{W}(C^2+1)\mathcal{X}}{\delta}\sqrt{T} \tag{11}$$

Again we can bound the difference between $\tilde{\ell}_t(w)$ and $\ell_t(w)$ for any w using the fact that $(\hat{y}_t - y_t)^2$ is Lipschitz continuous with respect to prediction $\hat{y}_t$ with Lipschitz constant C:

$$\begin{aligned}
|\tilde{\ell}_t(w) &- \ell_t(w)| \\
&= |\mathbb{E}_{\epsilon\sim[-1,1]}[(w^T x_t + \delta\epsilon - y_t)^2 - (w^T x_t - y_t)^2]| \\
&\leq \mathbb{E}_{\epsilon\sim[-1,-1]}[C\delta|\epsilon|] \leq C\delta
\end{aligned} \tag{12}$$

Substitute the above inequality back to Eq. 11, rearrange terms:

$$\begin{aligned}
\mathbb{E}\left[\sum_{t=0}^{T} \tilde{\ell}_t(w_t)\right] - \min_{w^* \in \mathbb{W}} \sum_{t=0}^{T} \tilde{\ell}_t(w^*) \leq &\frac{\mathcal{W}(C^2+1)\mathcal{X}}{\delta}\sqrt{T} \\
&+ 2C\delta T
\end{aligned} \tag{13}$$

Set $\delta = T^{-0.25}\sqrt{\frac{\mathcal{W}(C^2+1)\mathcal{X}}{2C}}$, we get:

$$\mathbb{E}\left[\sum_{t=0}^{T} \tilde{\ell}_t(w_t)\right] - \min_{w^* \in \mathbb{W}} \sum_{t=0}^{T} \tilde{\ell}_t(w^*) \leq \sqrt{\mathcal{W}(C^2+1)\mathcal{X}C}T^{3/4}. \tag{14}$$

∎

In summary, we showed that in general online learning, the advantage of random exploration in action space (i.e., Alg. 3 in the linear contextual bandit setting) compared to random exploration in parameter space (i.e., Alg. 2 in pure bandit setting) comes from the fact that the regret bound of Alg. 3 does not explicitly depend on the feature dimension d.[8]

B. Analysis on RL

We now consider the general RL setting. For simplicity, we just focus on finite horizon control problems with deterministic dynamics $x_{t+1} = f(x_t, a_t)$, fixed initial position x_1, one-step cost function $c(x, a)$ and horizon H. We assume $x \in \mathbb{X} \subseteq \mathbb{R}^d$ and $a \in \mathbb{A} \subseteq \mathbb{R}$, where d is large. Namely we consider control problem with high feature dimension d and low action dimension ($1D$). Given any sequence of actions $\mathbf{a} = a_1, ..., a_H$, the total cost is fully determined by the sequence of actions:

$$J(\mathbf{a}) = \sum_{t=1}^{H} c(x_t, a_t) \tag{15}$$

$$s.t., x_{t+1} = f(x_t, a_t), t \in [1, H-1]. \tag{16}$$

Given any $\mathbf{a} \in \mathbb{R}^H$, with the knowledge of dynamics f, one can easily compute the gradient of J with respect $\mathbf{a}$, denoted as $\nabla_{\mathbf{a}} J(\mathbf{a})$. With $\nabla_{\mathbf{a}} J(\mathbf{a})$, one can perform gradient descent on the action sequence, which has been used in the trajectory optimization literature.

In the model-free setting, we are no longer able to exactly compute $\nabla_{\mathbf{a}} J(\mathbf{a})$. However, we can again use *exploration in the action space* to form an estimation of $\nabla_{\mathbf{a}} J(\mathbf{a})$. Given $\mathbf{a}$, we sample $u_H \sim \mathbb{S}_H$ (i.e., u_H is uniformly sampled from a H-dim unit sphere),[9] with a small $\delta \in \mathbb{R}^+$, we can formulate an estimation of $\nabla_{\mathbf{a}} J(\mathbf{a})$ as:

$$\tilde{\nabla}_{\mathbf{a}} J(\mathbf{a}) = \frac{H}{\delta} J(\mathbf{a} + \delta u_H) u_H, \tag{17}$$

where $\tilde{\nabla}_{\mathbf{a}} J(\mathbf{a}) \in \mathbb{R}^H$. Again, we can show that $\tilde{\nabla}_{\mathbf{a}} J(\mathbf{a})$ is an unbiased estimate of the gradient of a smoothed version of J: $\mathbb{E}_{v \sim \mathbb{B}_H}[J(\mathbf{a} + \delta v)]$, where $\mathbb{B}_H$ is a H-dim unit ball.

Now let us take policy into consideration. We assume parameterized deterministic linear policy $w^T x$ that takes state x as input and outputs an action deterministically. At any state x, the Jacobian of the policy with respect to parameter w is simply x. A given w fully determines the total cost:

$$J(w) = \sum_{t=1}^{H} c(x_t, a_t) \tag{18}$$

$$s.t., a_t = w^T x_t, x_{t+1} = f(x_t, a_t), t \in [1, H-1]. \tag{19}$$

If we know the model and cost function, we can exactly compute the gradient of $J(w)$ with respect to w. In model-free setting, we have two ways to estimate the $\nabla_w J(w)$.

The first approach uses random exploration in action space. Given w, we first execute $\pi(x) = w^T x$ on the real system to generate a trajectory $\tau = x_1, a_1, ..., x_H, a_H$. Denote $\mathbf{a} = a_1, ..., a_H$, we can compute $\tilde{\mathbf{a}} = \mathbf{a} + \delta u_H$, and execute $\tilde{\mathbf{a}}$ in a open-loop manner and receive $J(a + \delta u_H)$ at the end of the simulation, from which we compute $\tilde{\nabla}_{\mathbf{a}} J(\mathbf{a})$ as shown in Eq. 17. Now by chain rule and using the Jacobians of the policy, we can estimate the gradient $\nabla_w J(w)$ as:

$$\tilde{\nabla}_w J(w) = X \tilde{\nabla}_{\mathbf{a}} J(\mathbf{a}) = \frac{H J(\tilde{\mathbf{a}})}{\delta} X u_H, \tag{20}$$

where $X \in \mathbb{R}^{d \times H}$ and the i'th column of X is the state x_i along the trajectory τ, and $u_H \sim \mathbb{S}_H$.

The second approach estimates $\nabla_w J(w)$ by *exploration in parameter space*. Denote $\tilde{w} = w + \delta u_d$, where $u_d \sim \mathbb{S}_d$, we can estimate $\nabla_a J(w)$ as:

$$\hat{\nabla}_w J(w) = \frac{d J(\tilde{w})}{\delta} u_d, \tag{21}$$

where we can show that $\hat{\nabla}_w J(w)$ is an unbiased estimate of the gradient of a smoothed version of $J(w)$: $\mathbb{E}_{v \sim \mathbb{B}_d}[J(w + \delta v_d)]$.

Unlike our previous setting of linear regression, the optimization problem in RL is non-convex and it is difficult to analyze its convergence. However, the norm of the gradient estimator still plays an important role and the regret is dependent on the maximum value of the norm. In both $\tilde{\nabla}_w J(w)$ from Eq. 20 and $\hat{\nabla}_w J(w)$ from Eq. 21, the terms $J(\tilde{\mathbf{a}})$ and

---

[8]Note that if we consider multivariate regression problem, than the regret bound will explicitly depend on the dimension of the action space.

[9]Uniformly sampling from a $H-$dim unit sphere can be implemented by first sampling u_t from normal distribution for each time step t, and then forming the perturbation vector as $[u_1, ..., u_H]^T / \sqrt{\sum_t u_t^2}$.

Stepsize	$0.001, 0.005, 0.01, 0.02, 0.03$
# Directions	$10, 50, 100, 200, 500$
# Top Directions	$5, 10, 50, 100, 200$
Perturbation	$0.001, 0.005, 0.01, 0.02, 0.03$

TABLE I: Candidate hyperparameters used for tuning in ARS **V2-t** experiments

Experiment	Stepsize	# Dir.	# Top Dir.	Perturbation
MNIST	0.02	50	20	0.03
LR $d = 10$	0.03	10	10	0.03
LR $d = 100$	0.03	10	10	0.02
LR $d = 1000$	0.03	200	200	0.03

TABLE II: Hyperparameters chosen for ARS **V2-t** in each experiment. LR is short-hand for Linear Regression.

Experiment	Learning Rate	Batch size
MNIST	0.001	512
LR $d = 10$	0.08	512
LR $d = 100$	0.03	512
LR $d = 1000$	0.01	512

TABLE III: Learning rate and batch size used for REIN-FORCE experiments. We use an ADAM [9] optimizer for these experiments.

Experiment	Learning Rate	Batch size
LR $d = 10$	2.0	512
LR $d = 100$	2.0	512

TABLE IV: Learning rate and batch size used for Natural REINFORCE experiments. Note that we decay the learning rate after each batch by $\sqrt{T}$ where T is the number of batches seen.

$J(\tilde{w})$ are at the similar scale, $\|u_d\| = \|u_H\| = 1$ as they are sampled from the corresponding unit sphere, and δ is a small number close to zero. When comparing the norm of both estimators $\hat{\nabla}_w J(w)$ and $\tilde{\nabla}_w J(w)$, we can see that the terms that really matter are d, H, and $\|X\|_F$. Again we see that the norm of the estimator $\hat{\nabla}_w J(w)$ explicitly depends on d–the feature dimension, while the norm of the estimator $\tilde{\nabla}_w J(w)$ does not explicitly depend on d, but instead the horizon length H and the norm of the state $\|x\|$ which is problem dependent.

So under what situations we may encounter that the estimator $\hat{\nabla}_w J(w)$ has smaller norm than the estimator $\tilde{\nabla}_w J(w)$ (i.e., random exploration in parameter space is preferred)? We can see that for problems where $H > d$, which in fact is the case in most of continuous control tasks in OpenAI Gym, we could expect that random exploration in parameter space performs better than random exploration in action space. Another possible situation is that the dynamics under the current policy w (i.e., $x_{t+1} = f(x_t, w^T x_t)$) is unstable: the norm of the state x_t grows exponentially with respect to t along the trajectory, which corresponds to the LQR example that Mania et al. [12] demonstrated in their work. Such exponential dependency on horizon H could lead to an estimator $\tilde{\nabla}_w J(w)$ with extremely large norm.

C. Implementation Details

1) Tuning Hyperparameters for ARS: We tune the hyperparameters for ARS [12] in both MNIST and linear regression experiments, by choosing a candidate set of values for each hyperparameter: stepsize, number of directions sampled, number of top directions chosen and the perturbation length along each direction. The candidate hyperparameter values are shown in Table I.

We use the hyperparameters shown in Table II chosen through this tuning for each of the experiments in this work. The hyperparameters are chosen by averaging the test squared loss across three random seeds (different from the 10 random seeds used in actual experiments) and chosing the setting that has the least mean test squared loss after 100000 samples.

2) MNIST Experiments: The CNN architecture used is as shown in Figure 4[10]. The total number of parameters in this model is $d = 21840$. For supervised learning, we use a cross-entropy loss on the softmax output with respect to the true label. To train this model, we use a batch size of 64 and a stochastic gradient descent (SGD) optimizer with learning rate of 0.01 and a momentum factor of 0.5. We evaluate the test accuracy of the model over all the 10000 images in the MNIST test dataset.

For REINFORCE, we use the same architecture as before. We train the model by sampling from the categorical distribution parameterized by the softmax output of the model and then computing a ± 1 reward based on whether the model predicted the correct label. The loss function is the REINFORCE loss function given by,

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} r_i \log(\mathbb{P}(\hat{y}_i | x_i, \theta)) \quad (22)$$

where θ is the parameters of the model, r_i is the reward obtained for example i, $\hat{y}_i$ is the predicted label for example i and x_i is the input feature vector for example i. The reward r_i is given by $r_i = 2 * \mathbb{I}[\hat{y}_i = y_i] - 1$, where $\mathbb{I}$ is the $0 - 1$ indicator function and y_i is the true label for example i.

For ARS **V2-t**, we use the same architecture and reward function as before. The hyperparameters used are shown in Table II and we closely follow the algorithm outlined in [12].

3) Linear Regression Experiments: We generate training and test data for the linear regression experiments as follows: we sampled a random $d + 1$ dimensional vector w where d is the input dimensionality. We also sampled a random $d \times d$ covariance matrix C. The training and test dataset consists of $d + 1$ vectors x whose first element is always 1 (for the bias term) and the rest of the d terms are sampled from a multivariate normal distribution with mean **0** and covariance matrix C. The target vectors y are computed as $y = w^T x + \epsilon$

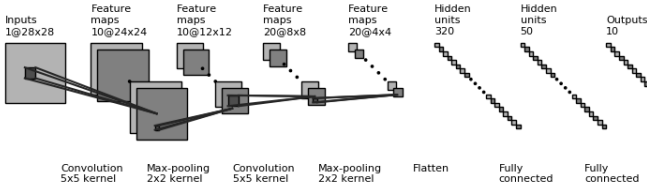[10]This figure is generated by adapting the code from https://github.com/gwding/draw_convnet

Fig. 4: CNN architecture used for the MNIST experiments

Stepsize	$0.005, 0.01, 0.02, 0.03$
# Directions	$20, 50, 100$
# Top Directions	$10, 25, 50$
Perturbation	$0.01, 0.02, 0.03, 0.04$

TABLE V: Candidate hyperparameters used for tuning in ARS **V1-t** in LQG at $H = 50$ and $H = 100$

where ϵ is sampled from a univariate normal distribution with mean 0 and standard deviation 0.001.

We implemented both SGD and Newton Descent on the mean squared loss, for the supervised learning experiments. For SGD, we used a learning rate of 0.1 for $d = 10, 100$ and a learning rate of 0.01 for $d = 1000$, and a batch size of 64. For Newton Descent, we also used a batch size of 64. To frame it as a one-step MDP, we define a reward function r which is equal to the negative of mean squared loss. Both REINFORCE and ARS **V2-t** use this reward function. To compute the REINFORCE loss, we take the prediction of the model $\hat{w}^T x$, add a mean 0 standard deviation $\beta = 0.5$ Gaussian noise to it, and compute the reward (negative mean squared loss) for the noise added prediction. The REINFORCE loss function is then given by

$$J(w) = \frac{1}{N} \sum_{i=1}^{N} r_i \frac{-(y_i - \hat{w}^T x_i)^2}{2\beta^2} \quad (23)$$

where $r_i = -(y_i - \hat{y}_i)^2$, $\hat{y}_i$ is the noise added prediction and $\hat{w}^T x_i$ is the prediction by the model. We use an Adam optimizer with learning rate and batch size as shown in Table III. For the natural REINFORCE experiments, we estimate the fisher information matrix and compute the descent direction by solving the linear system of equations $Fx = g$ where F is the fisher information matrix and g is the REINFORCE gradient. We use SGD with a $O(1/\sqrt{T})$ learning rate, where T is the number of batches seen, and batch size as shown in Table IV.

For ARS **V2-t**, we closely follow the algorithm outlined in [12].

4) LQR Experiments: In the LQR experiments, we randomly generate linear dynamical systems $x_{t+1} = Ax_t + Ba_t + \xi_t$, where $A \in \mathbb{R}^{100 \times 100}$, $B \in \mathbb{R}^{100}$, $x \in \mathbb{R}^{100}$, $a \in \mathbb{R}^1$, and the noise $\xi_t \sim \mathcal{N}(0_{100}, cI_{100 \times 100})$ with a small constant $c \in \mathbb{R}^+$. We explicitly make sure that the maximum eigenvalue of A is less than 1 to avoid instability. We fix a quadratic cost function $c(x, a) = x^T Q x + a R a$, where $Q = I_{100 \times 100}$, and $R = 1e - 3$. The stochastic linear policy is defined as a Gaussian distribution $\pi(\cdot|x; w, \sigma) = \mathcal{N}\left(w^T x, (\exp(\sigma))^2\right)$ where w and σ are parameters to learn. We initialize w such that the resulting markov chain $x_{t+1} = (A + Bw^T)x_t$ is unstable ($(A + Bw^T)$ has eigenvalues larger than one). We simple set $\alpha = 0$, resulting the initial standard deviation to 1.

We perform hyperparameter grid search for ARS V1-t at the specific horizon $H = 50$ [11] using two random seeds (different from the test random seeds). The candidate parameters are listed in Table V and then use the best set of hyperparameters for all H.

---

[11]We tuned hyperparameter for $H = 100$ but found the same set of hyperparameters as the one for $H = 50$.