Towards Generalization and Simplicity in Continuous Control

Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, Sham Kakade
Paul Allen School of Computer Science
University of Washington Seattle
{ aravraj, klowrey, sham, todorov } @cs.washington.edu

***Abstract*—This work shows that policies with simple linear and RBF parameterizations can be trained to solve a variety of continuous control tasks studied in recent literature. The performance of these trained policies are competitive with state of the art results obtained with more elaborate parametrizations like multi-layer neural networks.**

We not only study the performance of various representations with regard to the reward function at hand, but also with regard to robustness towards external perturbations. We find that existing training and testing scenarios are limited and produce fragile trajectory centric policies, regardless of the representation used. Finally, we directly modify the training scenario to favor robust and generalizable policies, which allows for interactive control scenarios where the system recovers from large on-line perturbations; as shown in the video [1].

Overall, this study suggests that multi-layer architectures should not be the default choice, unless a side-by-side comparison to simpler architectures shows otherwise. Such comparisons are unfortunately lacking, and this work is among the first to study this issue. More broadly, we hope that these results lead to greater interest in studying architectural choices and associated trade-offs for training generalizable and robust policies.

I. Introduction

Deep learning has recently demonstrated remarkable success in computer vision and speech recognition. In parallel, deep learning in conjunction with reinforcement learning (deepRL) has achieved impressive results in sequential decision making problems like games [9, 8]. This success has motivated efforts to adapt deepRL methods for robotic control. The complexity of systems solvable with RL methods is not yet at the level of what can be achieved with trajectory optimization in simulation [28, 19, 1], or with hand-crafted controllers on physical robots (e.g. Boston Dynamics). However, RL approaches are exciting because they are generic, model-free, and highly automated.

Recent success of RL [8, 16, 15, 14, 21] has been enabled largely due to engineering efforts such as large scale data collection [8, 21] or careful systems design [16, 15] with well behaved robots. When advances in a field are largely empirical in nature, it is important to understand the relative contributions of representations, optimization methods, and task design or modeling: both as a sanity check and to scale up to harder tasks. Furthermore, in line with Occam's razor, the simplest reasonable approaches should be tried and understood first. A thorough understanding of these factors is unfortunately lacking in the community.

In this backdrop, we ask the pertinent question: "What are the simplest set of ingredients needed to succeed in some of the popular benchmarks?" To attempt this question, we use the Gym-v1 [4] continuous control benchmarks, which have accelerated research and enabled objective comparisons. Since the tasks involve under-actuation, contact dynamics, and are high dimensional (continuous space), they have been accepted as benchmarks in the deepRL community. Recent works test their algorithms either exclusively or primarily on these tasks [26, 17, 11], and success on these tasks have been regarded as demonstrating a "proof of concept".

Our contributions in this work are as follows:

1) Simple policies based on linear and RBF parameterizations are able to achieve state of the art results on widely studied tasks. Furthermore, such policies, particularly the linear ones, can be trained significantly faster due to orders of magnitude fewer parameters. This could indicate that even for tasks with elaborate dynamics, there could exist relatively simple policies. This opens the door for studying a wide range of representations in addition to deep neural networks, and understand trade-offs including computational time, theoretical justification, robustness, sample complexity etc.
2) We study various representations not only with regard to the performance metric at hand but we also take the further step in examining their robustness towards external perturbations. Our results show that under the standard training scenarios, the trained policies are extremely brittle regardless of the representation used. For example, the agent is able to successfully learn a limit cycle for walking, but cannot recover from any perturbations that are delivered to it. Studying methods to train robust policies is crucial for transferring success of RL to robotics.
3) Finally, we modify the training conditions to favor robust policies by using a diverse set of initializations. This is similar in spirit to model ensemble approaches [18, 23] and domain randomization approaches [24, 29], which have successfully demonstrated improved robustness and simulation to real world transfer. Under the modified training scenario, we again find that there is no compelling evidence to favor the use of multi-layer architectures. The RBF representations are sufficient to solve the tasks, and at times out perform the neural network.

II. Problem Formulation and Methods

We consider Markov Decision Processes (MDPs) in the average reward setting, which is defined using the tuple: $\mathcal{M} =$

[1]Project site: https://sites.google.com/view/simple-pol

$\{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \rho_0\}$. $\mathcal{S} \subseteq \mathbb{R}^n$, $\mathcal{A} \subseteq \mathbb{R}^m$, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ are (continuous) set of states, set of actions, and reward function respectively, and have the usual meaning. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the stochastic transition function and ρ_0 is the probability distribution over initial states. We wish to solve for a stochastic policy of the form $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which optimizes the objective function:

$$\eta(\pi) = \lim_{T \to \infty} \frac{1}{T} \, \mathbb{E}_{\pi, \mathcal{M}} \left[\sum_{t=1}^{T} r_t \right]. \tag{1}$$

Since we use simulations with finite length rollouts to estimate the objective and gradient, we approximate $\eta(\pi)$ using a finite T. In this finite horizon rollout setting, we define the value, Q, and advantage functions as follows:

$$V^\pi(s,t) = \mathbb{E}_{\pi, \mathcal{M}} \left[\sum_{t'=t}^{T} r_{t'} \right]$$

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

$$A^\pi(s,a,t) = Q^\pi(s,a,t) - V^\pi(s,t)$$

Note that even though the value functions are time-varying, we still optimize for a stationary policy. We consider parametrized policies π_θ, and hence wish to optimize for the parameters (θ). Thus, we overload notation and use $\eta(\pi)$ and $\eta(\theta)$ interchangeably.

A. Algorithm

Ideally, a controlled scientific study would seek to isolate the challenges related to architecture, task design, and training methods for separate study. In practice, this is not entirely feasible as the results are partly coupled with the training methods. Here, we utilize a straightforward natural policy gradient method for training. The work in [5] suggests that this method is competitive with most state of the art methods. We now discuss the training procedure.

Using the likelihood ratio approach and Markov property of the problem, the sample based estimate of the policy gradient is derived to be [31]:

$$\nabla_\theta \hat{\eta}(\theta) = g = \frac{1}{T} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}^\pi(s_t, a_t, t) \tag{2}$$

Gradient ascent using this "vanilla" gradient is sub-optimal since it is not the steepest ascent direction in the metric of the parameter space [13, 2]. The steepest ascent direction is obtained by solving the following local optimization problem around iterate θ_k:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & g^T(\theta - \theta_k) \\ \text{subject to} \quad & (\theta - \theta_k)^T F_{\theta_k} (\theta - \theta_k) \leq \delta, \end{aligned} \tag{3}$$

where F_{θ_k} is the Fisher Information Metric at the current iterate θ_k. We estimate F_{θ_k} as

$$\hat{F}_{\theta_k} = \frac{1}{T} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \nabla_\theta \log \pi_\theta(a_t|s_t)^T, \tag{4}$$

as originally suggested by Kakade [13]. This yields the steepest ascent direction to be $\hat{F}_{\theta_k}^{-1} g$ and corresponding update rule: $\theta_{k+1} = \theta_k + \alpha \hat{F}_{\theta_k}^{-1} g$. Here α is the step-size or learning rate parameter. Empirically, we observed that choosing a fixed value for α or an appropriate schedule is difficult [20]. Thus, we use the normalized gradient ascent procedure, where the normalization is under the Fisher metric. This procedure can be viewed as picking a normalized step size δ as opposed to α, and solving the optimization problem in (3). This results in the following update rule:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{\delta}{g^T \hat{F}_{\theta_k}^{-1} g}} \; \hat{F}_{\theta_k}^{-1} g. \tag{5}$$

A dimensional analysis of these quantities reveal that α has the unit of return^{-1} whereas δ is dimensionless. Though units of α are consistent with a general optimization setting where step-size has units of objective^{-1}, in these problems, picking a good α that is consistent with the scales of the reward was difficult. On the other hand, a constant normalized step size was numerically more stable and easier to tune: for all the results reported in this paper, the same $\delta = 0.05$ was used. When more than one trajectory rollout is used per update, the above estimators can be used with an additional averaging over the trajectories.

For estimating the advantage function, we use the GAE procedure [26]. This requires learning a function that approximates V_k^π, which is used to compute A_k^π along trajectories for the update in (5). GAE helps with variance reduction at the cost of introducing bias, and requires tuning hyperparameters like a discount factor and an exponential averaging term. Good heuristics for these parameters have been suggested in prior work. The same batch of trajectories cannot be used for both fitting the value function baseline, and also to estimate g using (2), since it will lead to overfitting and a biased estimate. Thus, we use the trajectories from iteration $k-1$ to fit the value function, essentially approximating V_{k-1}^π, and use trajectories

Algorithm 1 Policy Search with Natural Gradient

1: Initialize policy parameters to θ_0
2: **for** $k = 1$ **to** K **do**
3: Collect trajectories $\{\tau^{(1)}, \ldots \tau^{(N)}\}$ 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^π based on trajectories in iteration k and approximate value function V_{k-1}^π.
6: Compute policy gradient according to (2).
7: Compute the Fisher matrix (4) and perform gradient ascent (5).
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**

from iteration k for computing A_k^π and g. Similar procedures have been adopted in prior work [5].

B. Policy Architecture

We first consider a linear policy that directly maps from the observations to the motor torques. We use the same observations as used in prior work which includes joint positions, joint velocities, and for some tasks, information related to contacts. Thus, the policy mapping is $a_t \sim \mathcal{N}(Ws_t + b, \sigma)$, and the goal is to learn W, b, and σ. For most of these tasks, the observations correspond to the state of the problem (in relative coordinates). Thus, we use the term states and observations interchangeably. In general, the policy is defined with observations as the input, and hence is trying to solve a POMDP.

Secondly, we consider a representation that enriches the representational capacity using random Fourier features of the observations. Since these features approximate the RKHS features under an RBF Kernel [22], we call this policy parametrization the RBF policy. Features are constructed as:

$$y_t^{(i)} = \sin\left(\frac{\sum_j P_{ij} s_t^{(j)}}{\nu} + \phi^{(i)}\right), \tag{6}$$

where each element P_{ij} is drawn from $\mathcal{N}(0,1)$, ν is a bandwidth parameter chosen approximately as the average pairwise distances between different observation vectors, and ϕ is a random phase shift drawn from $U[-\pi, \pi)$. Thus the policy is $a_t \sim \mathcal{N}(Wy_t + b, \sigma)$, where W, b, and σ are trainable parameters. This architecture can also be interpreted as a two layer neural network: the bottom layer is clamped with random weights, a sinusoidal activation function is used, and the top layer is finetuned. The principal purpose for this representation is to slightly enhance the capacity of a linear policy, and the choice of activation function is not very significant.

III. Results on OpenAI Gym benchmarks

As indicated before, we train linear and RBF policies with the natural policy gradient on the popular OpenAI gym-v1 benchmark tasks simulated in MuJoCo [30]. The tasks primarily consist of learning locomotion gaits for simulated robots ranging from a swimmer to a 3D humanoid (23 dof).

Figure 1 presents the learning curves along with the performance levels reported in prior work using TRPO and fully connected neural network policies. Table 1 also summarizes the final scores, where "stoc" refers to the stochastic policy with actions sampled as $a_t \sim \pi_\theta(s_t)$, while "mean" refers to using mean of the Gaussian policy, with actions computed as $a_t = \mathbb{E}[\pi_\theta(s_t)]$. We see that the linear policy is competitive on most tasks, while the RBF policy can outperform previous results on five of the six considered tasks. Though we were able to train neural network policies that match the results reported in literature, we have used publicly available prior results for an objective comparison. Visualizations of the trained linear and RBF policies are presented in the supplementary video. Given the simplicity of these policies, it is surprising that they can produce such elaborate behaviors.

Table 1: Final performances of the policies

Task	**Linear**		**RBF**		**NN**
	stoc	mean	stoc	mean	TRPO
Swimmer	362	**366**	361	365	131
Hopper	3466	3651	3590	**3810**	3668
Cheetah	3810	4149	6477	**6620**	4800
Walker	4881	5234	5631	**5867**	5594
Ant	3980	4607	4297	4816	**5007**
Humanoid	5873	6440	6237	**6849**	6482

Table 2: Number of episodes to achieve threshold

Task	**Th.**	**Linear**	**RBF**	**TRPO+NN**
Swimmer	325	**1450**	1550	N-A
Hopper	3120	13920	**8640**	10000
Cheetah	3430	11250	6000	**4250**
Walker	4390	36840	25680	**14250**
Ant	3580	39240	**30000**	73500
Humanoid	5280	**79800**	96720	87000

Table 2 presents the number of samples needed for the policy performance to reach a threshold value for reward. The threshold value is computed as 90% of the final score achieved by the stochastic linear policy. We visually verified that policies with these scores are proficient at the task, and hence the chosen values correspond to meaningful performance thresholds. We see that linear and RBF policies are able to learn faster on four of the six tasks.

All the simulated robots we considered are under-actuated, have contact discontinuities, and continuous action spaces making them challenging benchmarks. When adapted from model-based control [7, 28, 6] to RL, however, the notion of "success" established was not appropriate. To shape the behavior, a very narrow initial state distribution and termination conditions are used in the benchmarks. As a consequence, the learned policies become highly trajectory centric – i.e. they are good only where they tend to visit during training, which is a very narrow region. For example, the walker can walk very well when initialized upright and close to the walking limit cycle. Even small perturbations, as shown in the supplementary video, alters the visitation distribution and dramatically degrades the policy performance. This makes the agent fall down at which point it is unable to get up. Similarly, the swimmer is unable to turn when its heading direction is altered. For control applications, this is undesirable. In the real world, there will always be perturbations – stochasticity in the environment, modeling errors, or wear and tear. Thus, the specific task design and notion of success used for the simulated characters are not adequate. The simulated robots themselves are rather complex and harder tasks could be designed with them, as partly illustrated in Section 4.

IV. Modified Tasks and Results

Using the same set of simulated robot characters outlined in Section 3, we designed new tasks with two goals in mind:

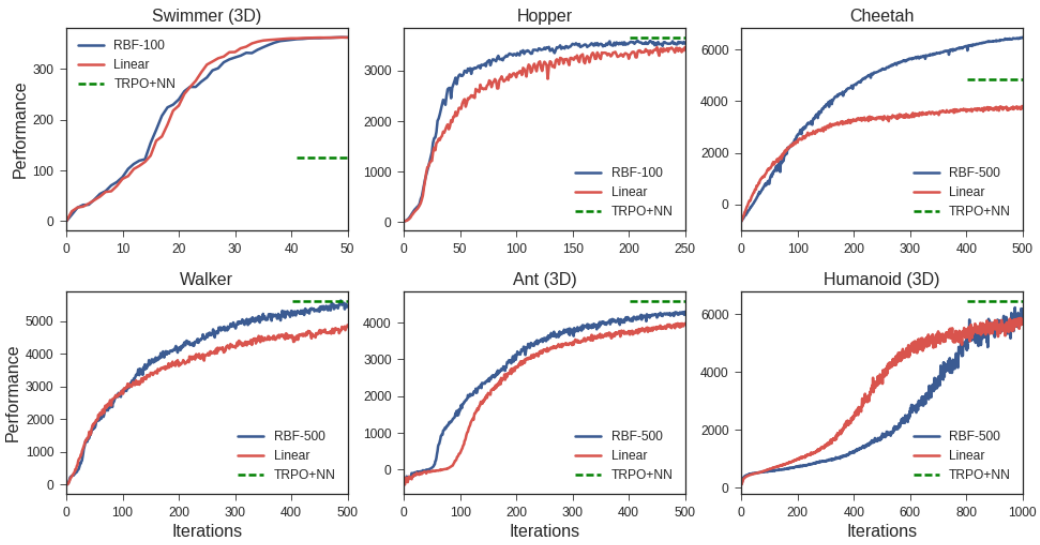


Fig. 1: Learning curves for the Linear and RBF policy architectures. The green line corresponding to the reward achieved by neural network policies on the OpenAI Gym website, as of 02/24/2017 (trained with TRPO). It is observed that for all the tasks, linear and RBF parameterizations are competitive with state of the art results. The learning curves depicted are for the stochastic policies, where the actions are sampled as $a_t \sim \pi_\theta(s_t)$. The learning curves have been averaged across three runs with different random seeds.

(a) to push the representational capabilities and test the limits of simple policies; (b) to enable training of "global" policies that are robust to perturbations and work from a diverse set of states. To this end, we make the following broad changes, also summarized in Table 3 (appendix):

1) Wider initial state distribution to force generalization. For example, in the walker task, some fraction of trajectories have the walker initialized prone on the ground. This forces the agent to simultaneously learn a get-up skill and a walk skill, and not forget them as the learning progresses. Similarly, the heading angle for the swimmer and ant are randomized, which encourages learning of a turn skill.
2) Reward shaping to generate the required motion. Since the modified swimmer starts with a randomized heading angle, we include a small reward for adjusting its heading towards the correct direction. Similarly, for the walker, we include a small reward for increasing the height of the torso, so that the agent walks and not crawls forward. In conjunction, we also remove all termination conditions used in the Gym-v1 benchmarks.
3) Changes to environment's physics parameters, such as mass and joint torque. If the agent has sufficient power, most tasks are easily solved. By reducing an agent's action ability and/or increase it's mass, the agent is more under-actuated. These changes also produce more realistic looking motion.

Combined, these modifications require that the learned policies not only make progress towards maximizing the reward, but also recover from adverse conditions and resist perturbations. An example of this is illustrated in Figure 2, where the hopper executes a get-up sequence before hopping to make forward progress. Furthermore, at test time, a user can interactively apply pushing and rotating perturbations to better understand the failure modes. We note that these interactive perturbations may not be the ultimate test for robustness, but a step towards this direction.

A. *Results and Analysis*

The supplementary video demonstrates successful policies on the modified tasks. We concentrate on the results of the walker task in the main paper. Figure 3 studies the performance as we vary the representational capacity. Increasing the Fourier features allows for more expressive policies and consequently allow for achieving a higher score. The policy with 500 RBF features performs the best, followed by the fully connected neural network. The linear policy also makes forward progress and can get up from the ground, but is unable to learn as efficient a walking gait.

Next, we test the robustness of our policies by perturbing the system with an external force. This external force represents an unforeseen change which the agent has to resist or overcome, thus enabling us to understand push and fall recoveries. Fall recoveries of the trained policies are demonstrated in the supplementary video. In these tasks, perturbations are not applied to the system during the training phase. Thus, the

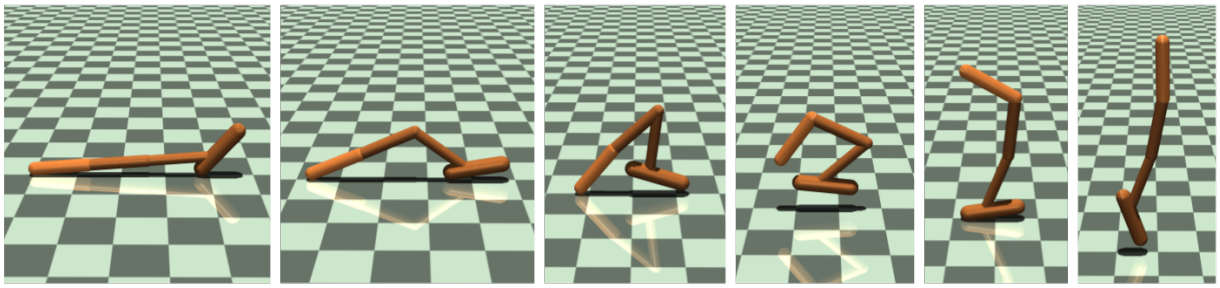


Fig. 2: Hopper completes a get-up sequence before moving to its normal forward walking behavior. The getup sequence is learned along side the forward hopping in the modified task setting.

ability to generalize and resist perturbations come entirely out of the states visited by the agent during training. Figure 4 indicates that the RBF policy is more robust, and also that diverse initializations are important to obtain the best results. This indicates that careful design of initial state distributions are crucial for generalization, and to enable the agent to learn a wide range of skills.

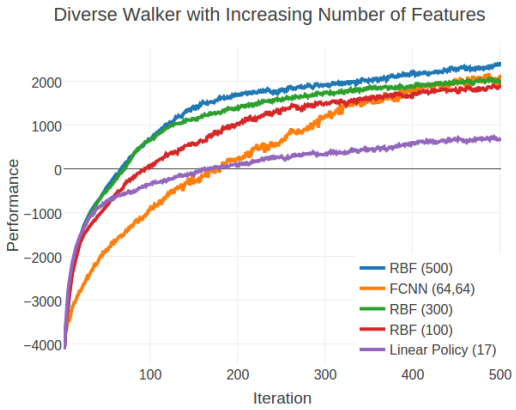
V. Summary and Discussion

The experiments in this paper were aimed at trying to understand the effects of (a) representation; (b) task modeling; and (c) optimization. We summarize the results with regard to each aforementioned factor and discuss their implications.

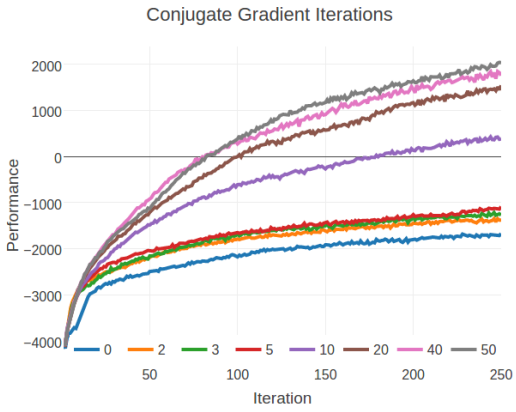
A. Representation

The finding that linear and RBF policies can be trained to solve continuous control tasks is very surprising. Recently, a number of algorithms have been shown to successfully solve these tasks [25, 12, 17, 11], but all of these works use multi-layer neural networks. This suggests a widespread belief that expressive function approximators are needed to capture intricate details necessary for movements like running. The results in this work conclusively demonstrates that this is not the case, at least for the limited set of popular testbeds. This raises an interesting question: what are the capability limits of shallow policy architectures? The linear policies were not exemplary in the modified tasks, but it must be noted that they were not terrible either. The RBF policy using random Fourier features was able to successfully solve the modified tasks producing global policies, suggesting that we do not yet have a sense of its limits.

One of the primary motivations to use multi-layer architectures in problems like vision is to extract features that transfer across tasks. Does this make sense for control too? Our understanding of physics allows representing states using joint angles and velocities, which unlike vision, are compact. Are more compact, task agnostic representations possible in general? It is certainly possible that non-linear functions of the states are required for controlling the system well, but it is not clear if this is tied to learning better representations of the state. Is it possible to get away with simple nonlinearities applied to random linear projections for harder tasks? Answering this is beyond the scope of current work, but provides for interesting future work.



(a)



(b)

Fig. 3: (a) Learning curve on modified walker (diverse initialization) for different policy architectures. The curves are averaged over three runs with different random seeds. (b) Learning curves when using different number of conjugate gradient iterations to compute $\hat{F}_{\theta_k}^{-1} g$ in (5). A policy with 300 Fourier features has been used to generate these results.

B. Modeling

When using RL methods to solve practical problems, the world provides us with neither the initial state distribution

Fig. 4: We test policy robustness by measuring distanced traveled in the swimmer, walker, and hopper tasks for three training configurations: (a) with termination conditions; (b) no termination, and peaked initial state distribution; and (c) with diverse initialization. Swimmer does not have a termination option, so we consider only two configurations. For the case of swimmer, the perturbation is changing the heading angle between $-\pi/2.0$ and $\pi/2.0$, and in the case of walker and hopper, an external force for 0.5 seconds along its axis of movement. All agents are initialized with the same positions and velocities.

nor the reward. Both of these must be designed by the researcher and must be treated as assumptions about the world or prescriptions about the required behavior. The quality of assumptions will invariably affect the quality of solutions, and thus care must be taken in this process. Here, we show that starting the system from a narrow initial state distribution produces elaborate behaviors, but the trained policies are very brittle to perturbations. Using a more diverse state distribution, in these cases, is sufficient to train robust policies. More broadly, diverse training conditions using multitude of models [18, 23] and textures [24, 29] have demonstrated improved robustness. Finding ways to automatically generate appropriate distributions or a training curriculum to this effect seems a promising venue to speed up learning of robust policies.

C. Optimization

In line with the theme of simplicity, we first tried REINFORCE [31], which we found to be very sensitive to hyperparameter choices. There are a class of policy gradient methods which use pre-conditioning to help navigate the warped parameter space of probability distributions and also possibly for step size selection. Many variants of pre-conditioned policy gradient methods have been reported to achieve near state of the art performance [5]. We feel that the used natural policy gradient method is the most straightforward pre-conditioned method. To demonstrate that the pre-conditioning helps, Figure 3 depicts the learning curve for different number of CG iterations used to compute the update in (5). The curve corresponding to $CG = 0$ is the REINFORCE method. As can be seen, pre-conditioning helps with the learning process. However, there is a trade-off with computation, and hence using an intermediate number of CG steps like 20 could lead to best results in wall-clock sense for large scale problems.

We chose to compare with neural network policies trained with TRPO, since it has demonstrated impressive results and is closest to the algorithm used in this work. Are function approximators linear with respect to free parameters sufficient for other methods is an interesting open question (in this sense, RBFs are linear but NNs are not). For a large class of methods based on dynamic programming (including Q-learning, SARSA, approximate policy and value iteration), linear function approximation has guaranteed convergence and error bounds, while non-linear function approximation is known to diverge in many cases [10, 27, 3]. It may of course be possible to avoid divergence in specific applications, or at least slow it down long enough, for example with target networks or replay buffers. Nevertheless, guaranteed convergence has indisputable advantages. Similar to recent work using policy gradient methods, recent work using dynamic programming methods have adopted multi-layer networks without careful side-by-side comparisons to simpler architectures. Could a global quadratic approximation to the optimal value function (which is linear in the set of quadratic features) be sufficient to solve most of the continuous control tasks currently studied in RL? Given that quadratic value functions correspond to linear policies, and good linear policies exist as shown here, this might make for interesting future work.

The results presented in this work highlight the need for simplicity and generalization in RL. We hope that this would encourage future work with a variety of architectures and understanding various trade-offs associated with computation, robustness, and sample complexity.

References

[1] M. Al Borno, M. de Lasa, and A. Hertzmann. Trajectory Optimization for Full-Body Movements with Complex Contacts. *IEEE Transactions on Visualization and Computer Graphics*, 2013.

[2] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10:251–276, 1998.

[3] D. Bertsekas. Approximate dynamic programming. 2008.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym, 2016.

[5] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *ICML*, 2016.

[6] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov. An integrated system for real-time model predictive control of humanoid robots. In *Humanoids*, pages 292–299, 2013.

[7] T. Erez, Y. Tassa, and E. Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. In *RSS*, 2011.

[8] D. Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529, 2016.

[9] V. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518, 2015.

[10] A. Geramifard, T. Walsh, S. Tellex, G. Chowdhary, N. Roy, and J. How. A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Foundations and Trends® in Machine Learning*, 6(4):375–451, 2013.

[11] S. Gu, T. Lillicrap, Z. Ghahramani, R. Turner, and S. Levine. Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic. In *ICLR*, 2017.

[12] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning continuous control policies by stochastic value gradients. In *NIPS*, 2015.

[13] S. Kakade. A natural policy gradient. In *NIPS*, 2001.

[14] V. Kumar, A. Gupta, E. Todorov, and S. Levine. Learning dexterous manipulation policies from experience and imitation. *ArXiv e-prints*, 2016.

[15] V. Kumar, E. Todorov, and S. Levine. Optimal control with learned local models: Application to dexterous manipulation. In *ICRA*, 2016.

[16] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 17(39):1–40, 2016.

[17] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ArXiv e-prints*, September 2015.

[18] I. Mordatch, K. Lowrey, and E.Todorov. Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids. In *IROS*, 2015.

[19] I. Mordatch, E. Todorov, and Z. Popovic. Discovery of complex behaviors through contact-invariant optimization. *ACM SIGGRAPH*, 2012.

[20] J. Peters. Machine learning of motor skills for robotics. *PhD Dissertation, University of Southern California*, 2007.

[21] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *ICRA*, 2016.

[22] A. Rahimi and B. Recht. Random Features for Large-Scale Kernel Machines. In *NIPS*, 2007.

[23] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine. EPOpt: Learning Robust Neural Network Policies Using Model Ensembles. In *ICLR*, 2017.

[24] F. Sadeghi and S. Levine. (CAD)2RL: Real Single-Image Flight without a Single Real Image. *ArXiv e-prints*, 2016.

[25] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel. Trust region policy optimization. In *ICML*, 2015.

[26] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

[27] J. Si. *Handbook of learning and approximate dynamic programming*, volume 2. John Wiley & Sons, 2004.

[28] Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. *International Conference on Intelligent Robots and Systems*, 2012.

[29] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *ArXiv e-prints*, 2017.

[30] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, 2012.

[31] R. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

APPENDIX

Table 3: Modified Task Description
v_x is forward velocity; θ is the heading angle; p_z is the height of torso; and a is the action (e.g. torques).

Task	Description	Reward (des = desired value)
Swimmer (3D)	Agent swims in the desired direction. Should recover (turn) if rotated around.	$v_x - 0.1\|\theta - \theta^{\mathrm{des}}\| - 0.0001\|\|a\|\|^2$
Hopper (2D)	Agent hops forward as fast as possible. Should recover (get up) if pushed down.	$v_x - 3\|\|p_z - p_z^{\mathrm{des}}\|\|^2 - 0.1\|\|a\|\|^2$
Walker (2D)	Agent walks forward as fast as possible. Should recover (get up) if pushed down.	$v_x - 3\|\|p_z - p_z^{\mathrm{des}}\|\|^2 - 0.1\|\|a\|\|^2$
Ant (3D)	Agent moves in the desired direction. Should recover (turn) if rotated around.	$v_x - 3\|\|p_z - p_z^{\mathrm{des}}\|\|^2 - 0.01\|\|a\|\|^2$