Bayesian Policy Gradients via Alpha Divergence Dropout Inference

Peter Henderson* Thang Doan* Riashat Islam David Meger
* Equal contributors
McGill University
peter.henderson@mail.mcgill.ca thang.doan@mail.mcgill.ca
riashat.islam@mail.mcgill.ca david.meger@mcgill.ca

Abstract

Policy gradient methods have had great success in solving continuous control tasks, yet the stochastic nature of such problems makes deterministic value estimation difficult. We propose an approach which instead estimates a distribution by fitting the value function with a Bayesian Neural Network. We optimize an α-divergence objective with Bayesian dropout approximation to learn and estimate this distribution. We show that using the Monte Carlo posterior mean of the Bayesian value function distribution, rather than a deterministic network, improves stability and performance of policy gradient methods in continuous control MuJoCo simulations.

1 Introduction

Reinforcement learning (RL) has recently had great success in solving complex tasks with continuous control [1, 2, 3]. However, as these methods can be high variance and often deal with unstable environments, distributional perspectives of function approximations in RL have begun to gain popularity [4, 5]. Currently, such methods in RL typically use many approximators (usually with shared hidden layers) to fit a distribution. However, in other fields, recent advances in Bayesian inference using deep neural networks have had notable success in providing predictive uncertainty estimates [6, 7, 8, 9]. Particularly, it has been shown that dropout can be used as a variational Bayesian approximation [6, 10]. Such dropout approximations of uncertainty estimates have demonstrated improved performance from domains such as simple classification tasks to active learning [11].

In this work, we develop an approach to using Bayesian neural networks (BNNs) as value function approximators in model-free policy gradient methods for continuous control tasks. We use a technique for dropout inference in BNNs using α-divergences [10] that provides accurate approximation of uncertainty and a Monte Carlo objective which can simulate a Gaussian distribution. We show that by using the posterior mean of this uncertainty distribution during value estimation, we achieve more stable learning and significantly better results versus a standard deterministic function approximator. We demonstrate the significance of using Monte Carlo dropout approximation across a range of policy gradient methods including Trust Region Policy Optimization (TRPO) [1] and Proximal Policy Optimization (PPO) [2], and Deep Deterministic Policy Gradients (DDPG), on a variety of benchmark continuous control tasks from OpenAI Gym [12] using the MuJoCo simulator [13]).

2 Background and Related Work

2.1 Policy Gradient Methods for Continuous Control

Policy gradient (PG) methods [14] are a form of reinforcement learning which can utilize stochastic gradient descent to optimize a parameterized policy π_θ. Such methods optimize the discounted

Workshop on Bayesian Deep Learning (NIPS 2017), Long Beach, CA, USA.

return: $\rho(\theta, s_0) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)|s_0\right]$, such that the overall policy gradient theorem results in: $\frac{\delta\rho(\theta,s_0)}{\delta\theta} = \sum_s \mu_{\pi_\theta}(s|s_0) \sum_a \frac{\delta\pi_\theta(a|s)}{\delta\theta} Q_{\pi_\theta}(s,a)$, where $\mu_{\pi_\theta}(s|s_0) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s|s_0)$ [14]. Trust Region Policy Optimization (TRPO) [1] and Proximal Policy Optimization (PPO) [2] constrain these updates via trust regions. Furthermore, these methods leverage advantage estimation to reduce variance in updates. Generally, for these trust region methods, updates are as follows: $\max_\theta \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t(s_t, a_t)\right]$ subject to $\mathbb{E}_t\left[\mathrm{KL}\left[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right] \leq \delta$. $A_t(s_t, a_t)$ is the advantage function. TRPO uses constrained conjugate gradient descent to solve the constrained optimization problem. PPO transforms the constraint into a penalty or clipping objective, depending on the implementation. Here, we look exclusively at the clipping objective. Finally, DDPG [3] leverages the policy gradient theorem in an actor-critic format such that a deterministic policy can be used and off-policy sampling is utilized. This yields faster training, but often at the cost of higher variance and instability [15].

2.2 Dropout as Bayesian Inference

Several works demonstrate different ways to approximate an uncertainty distribution (as with a Bayesian Neural Network) via simple dropout [6, 10]. In [6], it is shown that fitting a dropout objective (and running dropout at test time) can approximate a variational Bayesian approximation (and thus an uncertainty estimate). This work was further expanded in [10], where it was shown that by fitting a Monte Carlo variational inference objective utilizing the α-divergence, an improved uncertainty estimate can be achieved. Such dropout Bayesian approximation has shown success in *model-based* RL already. When fitting a dynamics model with a dropout uncertainty estimate, typical Gaussian processes can be replaced with deep function approximations which tend to improve model – and in turn policy – performance [11, 16].

2.3 Distributional Perspectives in RL

Several recent works have investigated various methods for modeling distributions in the context of reinforcement learning. In [17], a Bayesian framework was applied to the actor-critic architecture by fitting a Gaussian Process (GP) for the critic which allowed for a closed-form derivation of update rules. In [6], dropout uncertainty estimates of a Q-value function were used for Thompson sampling and optimistic exploration. [4] uses value function update rules according to a distributional perspectives. Similarly, in [5], the authors use k-heads on the Q-value to model a distribution rather than the dropout approach of [6]. Here, we build on these works and successfully provide a simple replacement for fitting a BNN value function using a dropout α-divergence objective, that, without significant modification to the core policy gradient approach, can generally improve performance in continuous control tasks.

3 Bayesian Value Functions in Policy Gradient Reinforcement Learning

3.1 Variational Inference for Value Functions

As in [6, 10], we estimate uncertainty through approximate dropout variational inference. This involves using Monte Carlo dropout sampling to approximate a posterior distribution $q_\theta(\omega)$ where ω corresponds to a set of random weight matrices in a neural network $\omega = (W_i)_{i=1}^L$ where L is the number of hidden layers. This distribution is typically fit by minimizing the KL divergence of the estimated (dropout) distribution with the true distribution. However, in [10], it is found that rather than minimizing the KL divergence in variational inference, a better uncertainty estimate can found by using the generalized α-divergence distance metric. Hence, following this process, we minimize an α energy, which with MC Dropout sampling becomes [10] :

$$\mathcal{L}_\alpha^{MC} = -\frac{1}{\alpha}\sum_{n=1}^{N} log\, sum\, exp[-\frac{\alpha\tau}{2} \parallel y_n - V^{\hat{\omega_k}}(x_n) \parallel_2^2] + \frac{ND}{2} log\tau + p_i \sum_{i=1}^{K} \parallel M_i \parallel_2^2 \quad (1)$$

where τ is the precision of the model, $\hat{\omega}_k$ are the sampled dropout weights, $\{V^{\hat{\omega_k}}(x_n)\}_{k=1}^K$ are a set of K stochastic forward passes through the neural network, and M_i are the neural network weights without dropout. We will refer to the BNN fit with this objective as an α-BNN.

3.2 TRPO and PPO

We first examine policy gradient methods which use advantage estimation, where the value function is used as a baseline. We focus on two such methods: PPO [2] and TRPO [1]. To learn a policy, these optimize $\mathbb{E}_{s\sim\rho_{old}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t^{GAE(\gamma,\lambda)}\right]$ where $\hat{A}_t^{GAE(\gamma,\lambda)}$ is the generalized advantage estimate [18] given as:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}\delta_{t+l} = -V(s_t) + r_t + \gamma r_{t+1} + .. + \gamma^k V(s_{t+k}) \tag{2}$$

Although the use of the advantage function reduces prediction variance, this step is by definition an uncertain approximation made using limited data. As uncertain estimates accumulate through Bellman updates over time, this has been shown to yield over estimation [19]. Using an α-BNN value function, we can account for this to some extent by modeling the uncertainty distribution. Hence, we can define a Bayesian GAE function as:

$$\mathbb{E}[\hat{A}_t^{GAE(\gamma,\lambda)}|\mathcal{D}] = \mathbb{E}_{\omega\sim q(\omega)}[\hat{A}_t^{GAE(\gamma,\lambda)}(s,a;\omega)] = \sum_{l=0}^{\infty}(\gamma\lambda)^l\mathbb{E}_{\omega\sim q(\omega)}[\delta_{t+l}(\omega)] \tag{3}$$

where $\mathbb{E}_{\omega\sim q(\omega)}[\delta_{t+l}(\omega)] = \int_\omega q(\omega)[-V(s_t;\omega)+r_t+\gamma r_{t+1}+..+\gamma^k V(s_{t+k};\omega)]d\omega$. Here, the value functions can be approximated by performing K stochastic forward passes and taking the average of the posterior distribution, similarly to [10]. This posterior mean models the uncertainty the agent has about the value of a given state. To fit the α-BNN value function ($V^\alpha(s)$), we simply use the objective from Equation 1, where the target $y_i = r_t + \gamma r_{t+1} + .. + \gamma^k V(s_{t+k})$, as in [2].

3.3 DDPG

Next, we investigate using an α-BNN Q-value function in the off-policy actor-critic method, DDPG [3]. Again, we use the same loss as in Equation 1 to fit an uncertainty estimate of the action-value function. As with $V^\alpha(s)$, the α-BNN Q function ($Q^\alpha(s,a)$) is fit such that the target corresponds to $y_i = r_t + \gamma Q'(s_{t+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}$. Where μ' is the target policy and Q' in this case corresponds to a target network updated with soft updates as in [3].

With this Bayesian approch the DDPG update is given by (see appendix for proof):

$$\begin{aligned}\nabla_\theta\mathbb{E}[J(\mu_\theta)|\mathcal{D}] &= \mathbb{E}_{s\sim\rho^\mu,\omega\sim q(\omega)}[\nabla_\theta\mu_\theta(s)\nabla_a Q^\pi(s,a;\omega)]|_{a=\mu_\theta(s)} &(4)\\ &= [\int_S \rho^\mu(s)\nabla_\theta\mu_\theta(s)\int_\omega q(\omega)Q^\pi(s,a;\omega)d\omega ds]|_{a=\mu_\theta(s)} &(5)\end{aligned}$$

and the gradient is approximated using the Monte Carlo samples, where where $\hat{\omega}_{j,k}$ is the sampled weight (single forward pass) of the layer j, K is the number of samples and M the batch size.

$$\nabla_\theta\mathbb{E}[J(\mu_\theta)|\mathcal{D}] \simeq \frac{1}{M}\sum_{i=1}^{M}\nabla_\theta\mu_\theta(s_i)[\frac{1}{K}\sum_{k=1}^{K}\nabla_a Q^\pi(s_i,a_i;\hat{\omega}_{1,k},..\hat{\omega}_{L,k})\ |_{a_i=\mu(s_i)}] \tag{6}$$

4 Experiments and Results

To evaluate our use of α-BNN value functions, we use MuJoCo locomotion tasks [13] provided by OpenAI Gym [12]. We use hyperparameters based on analysis in [15] (see Appendix for detailed setup). For each algorithm we run 10 trials with 10 different random seeds[1]. We compare our α-BNN value function methods against the baseline implementation as of PPO and TRPO provided with [2] and DDPG provided with [20]. We additionally compare against a modified version of those

[1]Due to the long runtime of Q^α DDPG, we limit our experiments to HalfCheetah-v1 and 5 random seeds.

algorithms which only uses *L2* regularization on the value function. We do so to determine the effect of the regularization term from Equation 1.

Table 1 summarizes our results for using the α-BNN value function in PPO and TRPO respectively. Experimental results with α-BNN value functions are demonstrated across Hopper-v1, HalfCheetah-v1, and Walker2d-v1 environments. Figure 1 shows learning curves for the HalfCheetah-v1 environment on all algorithms. Detailed results can also found in the Appendix. We present here optimal hyperparameters specific to the α-divergence objective *only* found via a grid search with ablation analysis on effects of individual parameters in the appendix. Shared hyperparameters between the baseline are all held constant at the default levels as provided in the OpenAI baselines implementation (this includes learning rate, maximum KL, clip params, network architecture, etc.). The only difference through all experiments from the original baselines implementation is that we use *relu* activations, instead of *tanh*.

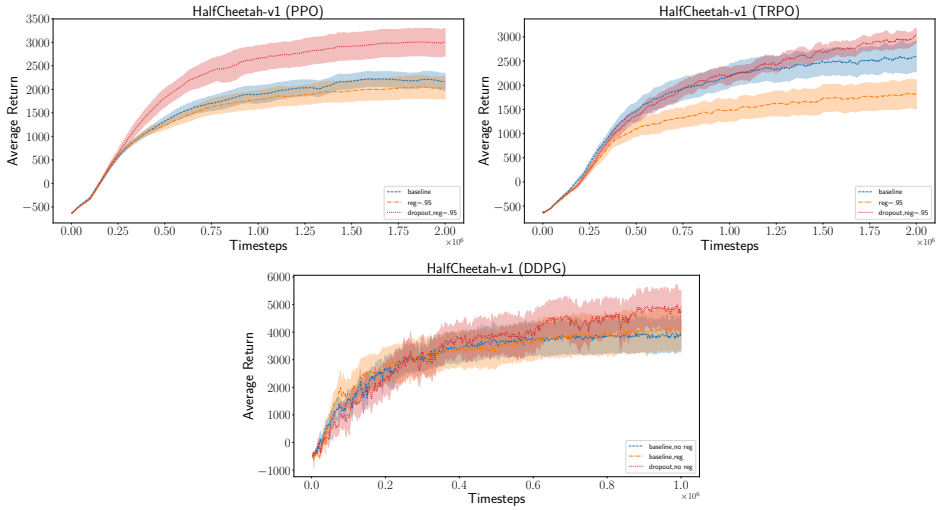

Figure 1: Performance on HalfCheetah-v1 across all three algorithms.

	$V(s)$	$V_{reg}(s)$	$V^{\alpha}_{reg}(s)$
PPO			
Hopper-v1	2342 ± 124	2228 ± 220	$\mathbf{2608 \pm 79}$
HalfCheetah-v1	2155 ± 177	2030 ± 234	$\mathbf{2790 \pm 284}^{*}$
Walker2d-v1	3165 ± 344	3219 ± 137	$\mathbf{3593 \pm 228}^{*}$
TRPO			
Hopper-v1	1989 ± 213	$\mathbf{2400 \pm 137}$	2237 ± 193
HalfCheetah-v1	2605 ± 313	1814 ± 300	$\mathbf{3026 \pm 144}^{*}$
Walker2d-v1	2974 ± 171	2328 ± 172	$\mathbf{3406 \pm 134}^{*}$
DDPG			
HalfCheetah-v1	4159 ± 762	3854 ± 575	$\mathbf{4772 \pm 736}$

Table 1: Final average return $\pm$ standard error across all random seeds (10 for TRPO/PPO, 5 for DDPG). Note, we do not use regularization for DDPG ($reg = 0$). Significant improvements over both L2 regularization and the baseline according to 2-sample t-test method [15] ($p < 0.05$) marked with an asterisk ($*$).

5 Discussion

Our results show significant improvements in many cases simply by fitting an α-BNN value function and using the posterior mean during policy updates. In particular, we find large improvements in Proximal Policy Optimization from using the posterior mean. We suspect that these improvements are partially due to the added exploration, aiding the on-policy and adaptive nature of PPO, during early stages of learning where the uncertainty distribution is large. This is backed by ablation analysis

(see Appendix) where we find that PPO in the Half-Cheetah environment sees larger improvements from a higher dropout rate (presumably yielding slightly more exploratory value estimates). In the overall ablation analysis (all results in Appendix), we found that the number of Monte Carlo samples, the dropout rate, and the τ factor generally yielded the largest difference in results. The τ parameter can be thought of as the trade-off between optimization of the objective loss and regularization. As such, we can see large variations in performance due to emphasis on these different components and too small a τ can hurt performance by over-emphasizing regularization.

We also find that in many cases, the α-BNN version held more consistent results across random seeds (lower standard error across trials in Table 1). While the baseline results see much larger deviations across random seeds, using the α-BNN yields fairly consistent results with more stable learning curves (see Appendix for all learning curves).

Finally, we found that the Q-value estimates in DDPG were much lower than in the baseline version (see Appendix for more details). We believe this is due to a variance reduction property as in Double-DQN [21]. While in Double-DQN, two function approximators are used for the Q-value function, using dropout effectively creates a parallel to Double-DQN where many different approximators are used from the same set of network weights. This also aligns with [5, 4]. Thus, as expected, we see a reduction in Q-value estimates as in Double-DQN.

6 Conclusion

We build off of work in Atari domains [5], Gaussian Processes [17], and distributional perspectives [4] in uncertain value functions to demonstrate an extension which can successfully be leveraged in continuous control domains with dropout uncertainty estimates. Overall, by providing a simple replacement for value functions, we demonstrate an increase in performance across policy gradient algorithms in continuous control tasks. The significant performance increases in PPO further suggest the importance of a stable baseline which the posterior mean provides.

Our work demonstrates the potential of using Bayesian approximation methods in policy gradient algorithms. This work provides a method which can be used to not only improve performance of existing algorithms by simple replacement of the value function, but be leveraged for more complex uses of value function uncertainty estimates in continuous control. By modelling the uncertainty over value functions in continuous control domains, our work opens up possibilities to use this uncertainty information for other applications such as in safe reinforcement learning.

7 Acknowledgments

We would like to thank the AWS Cloud Credits for Research program for compute resources. We'd also like to thank Juan Camilo Gamboa Higuera for helpful discussions.

References

[1] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[3] 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. *CoRR*, abs/1509.02971, 2015.

[4] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.

[5] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. *CoRR*, abs/1602.04621, 2016.

[6] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[7] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *CoRR*, abs/1505.05424, 2015.

[8] José Miguel Hernández-Lobato and Ryan P. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1861–1869, 2015.

[9] David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron C. Courville. Bayesian hypernetworks. *CoRR*, abs/1710.04759, 2017.

[10] Yingzhen Li and Yarin Gal. Dropout inference in bayesian neural networks with alpha-divergences. *arXiv preprint arXiv:1703.02914*, 2017.

[11] Yarin Gal, Rowan Thomas McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop*, volume 951, page 2016, 2016.

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

[13] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[14] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[15] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning that Matters. *arXiv preprint arXiv:1709.06560*, 2017.

[16] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. *arXiv preprint arXiv:1705.07832*, 2017.

[17] Mohammad Ghavamzadeh, Yaakov Engel, and Michal Valko. Bayesian policy gradient and actor-critic algorithms. *Journal of Machine Learning Research*, 17(66):1–53, 2016.

[18] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

[19] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

[20] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[21] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

8 Appendix

8.1 Further Experimental Setup and Results

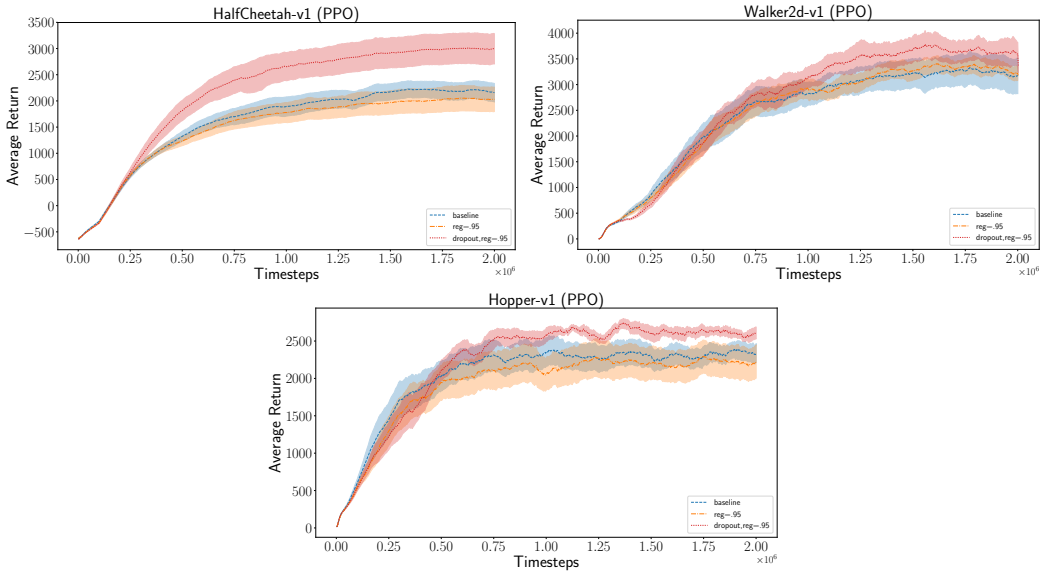

Figure 2: Proximal Policy Optimization with a dropout value function estimator.

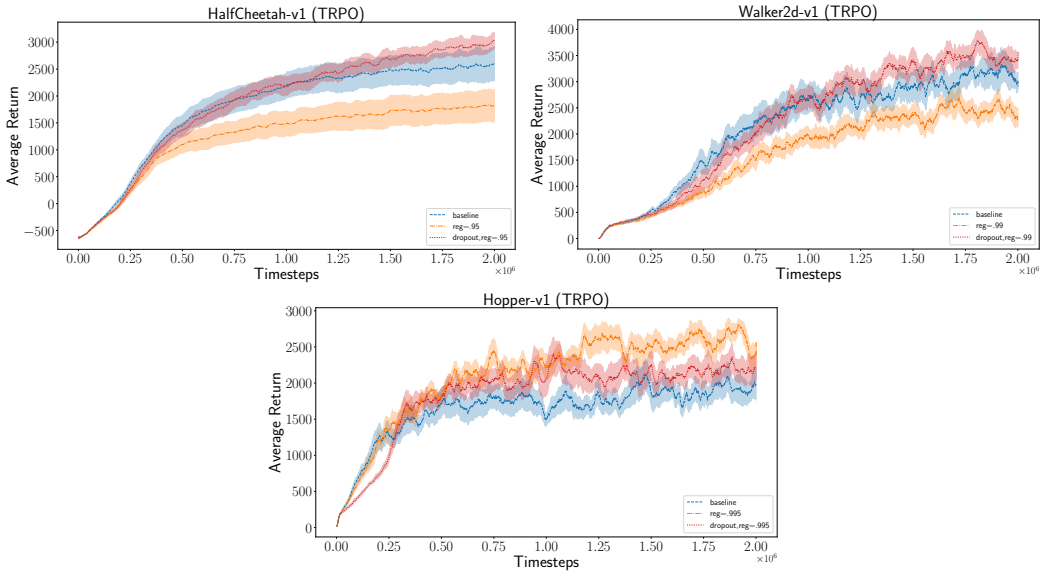

Figure 3: Trust Region Policy Optimization with a dropout value function estimator.

For TRPO and PPO experiments we use random seeds $1-10$ inclusive and for DDPG we use seeds $1-5$. We keep all hyperparameters which are shared between the α-BNN version and the baseline version for each algorithm constant. We do not modify hyperparameters from the baselines implementation except that we use *relu* activations instead of *tanh* in the policy and value functions. These can be found in our repository as the default settings in the individual run scripts: `https://github.com/Breakend/BayesianPolicyGradients`. For DDPG we use the adaptive exploration strategy from [20]. We use 50 Monte Carlo samples in parallel for all experiments. For all optimal experiments we set $\tau = .85$ and as per [10] we set the L2 regularizer to be equivalent to the keep probability on the dropout layers. For PPO we set this keep probability to .95 for dropping out

network weights and for TRPO and DDPG we set this to .99. We use a high dropout keep probability due to the small size of the networks as found via a grid search. See [6]. We do not use a regularizer (L2) for DDPG as it deteriorated performance. For the α-value, we use find that it is only beneficial in TRPO to modify it and set this to 1.0 in the case of Hopper experiments. All other experiments held α constant at 0.5. This is according to [10]. This provides a beneficial trade-off between mass covering ($\alpha = 1$, KL divergence) and zero forcing ($\alpha = 0$,free energy). Overall, the optimal hyperparameters used were as follows.

For TRPO:

- $\alpha = .5$ (Walker,HalfCheeta) $\alpha = 1.0$ (Hopper)
- $\tau = .85$
- $KeepProb = .95$ (HalfCheetah), $KeepProb = .99$ (Walker), $KeepProb = .995$ (Hopper)
- $MCSamples = 50$

For PPO:

- $\alpha = .5$
- $\tau = .85$
- $KeepProb = .75$ (HalfCheetah), $KeepProb = .95$ (Hopper, Walker)
- $MCSamples = 25$ (Hopper,Walker), $MCSamples = 50$ (Hopper,Walker)

8.2 Expanded Analysis

Here, we investigate the effects of various parameters and properties of our previously described methods.

8.2.1 DDPG Q-Value Approximation

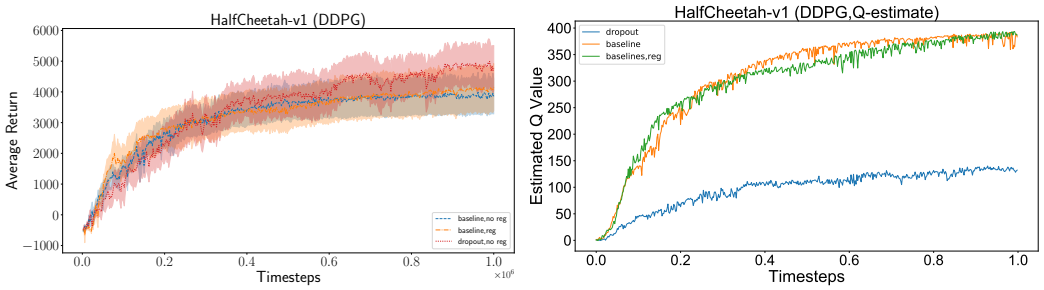

Figure 4: Comparison of Q value estimates during the learning process.

Figure 4 shows the Q-value estimate of the evaluation trials versus the training steps. Similarly to [21] (Figure 3 in that work), we notice that the DDPG value estimates are increasing much faster than that of the α-BNN value function over time. We suspect that the regularizing property of dropout inference provides a similar variance reduction benefit as in DDQN – and hence a lower Q-estimate overtime – tackling the overestimation problem in a similar fashion.

8.2.2 Ablation Analysis

To determine the effect of various hyperparameters, we run ablation analysis on the selection of Hopper and Half-Cheetah environments for TRPO and PPO. We investigate τ, the number of Monte Carlo dropout samples, the α parameter, and the keep probability of the dropout layers. We hold all hyperparameters constant at the default set of $\tau = .85, \alpha = .5, MCSamples = 50, KeepProb = .95$. Note that our optimal set of hyperparameters used in the presented main results was found after a subsequent gridsearch across the parameter space.

Figures 5 shows ablation analysis where we vary the τ hyperparameter. Overall, we find that HalfCheetah-v1 is more susceptible to variation due to hyperparameter changes. This is likely due

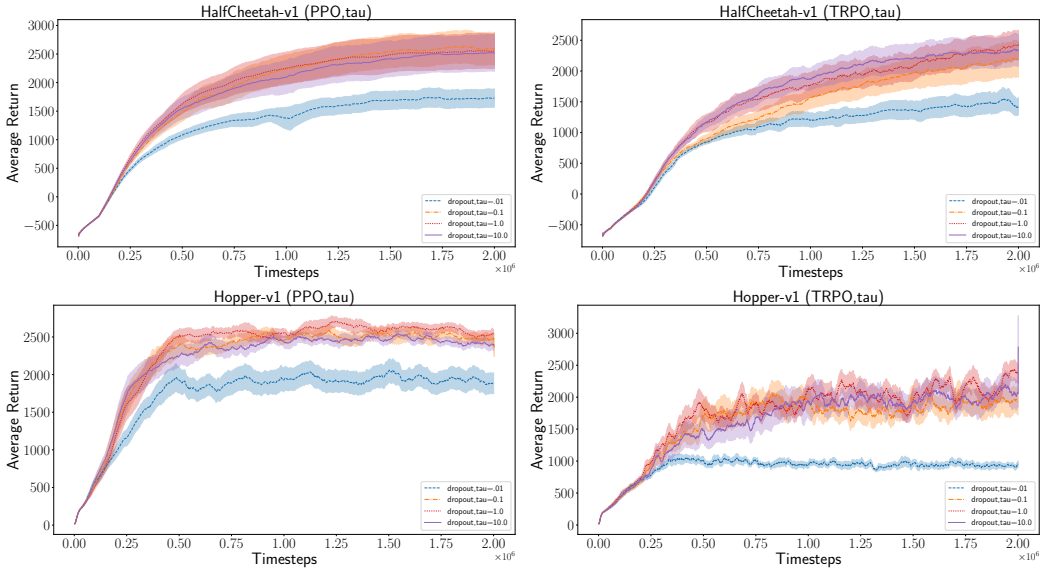

Figure 5: Ablation investigation into the τ parameter.

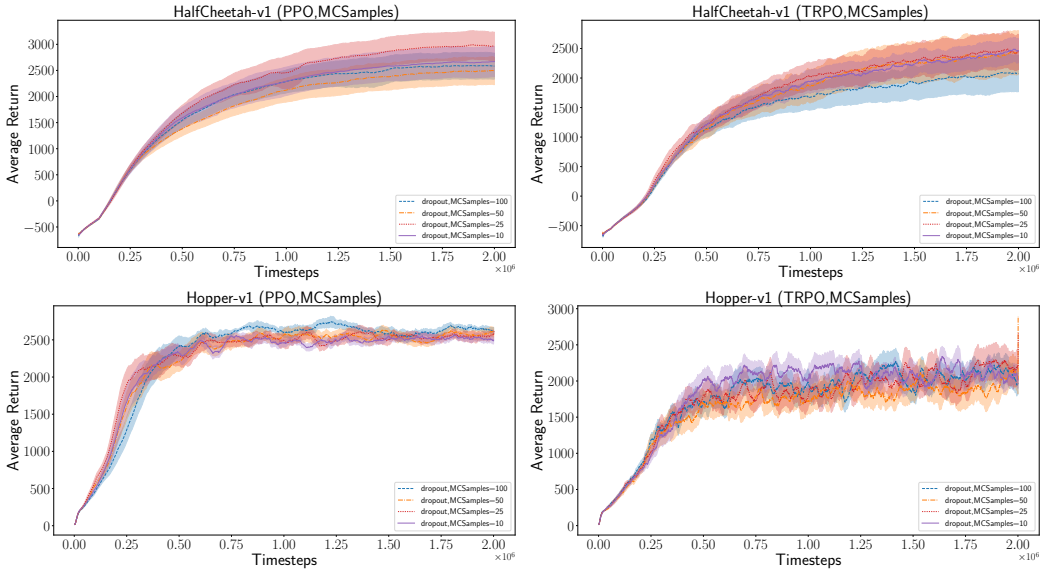

Figure 6: Ablation investigation into the number of MC samples parameter.

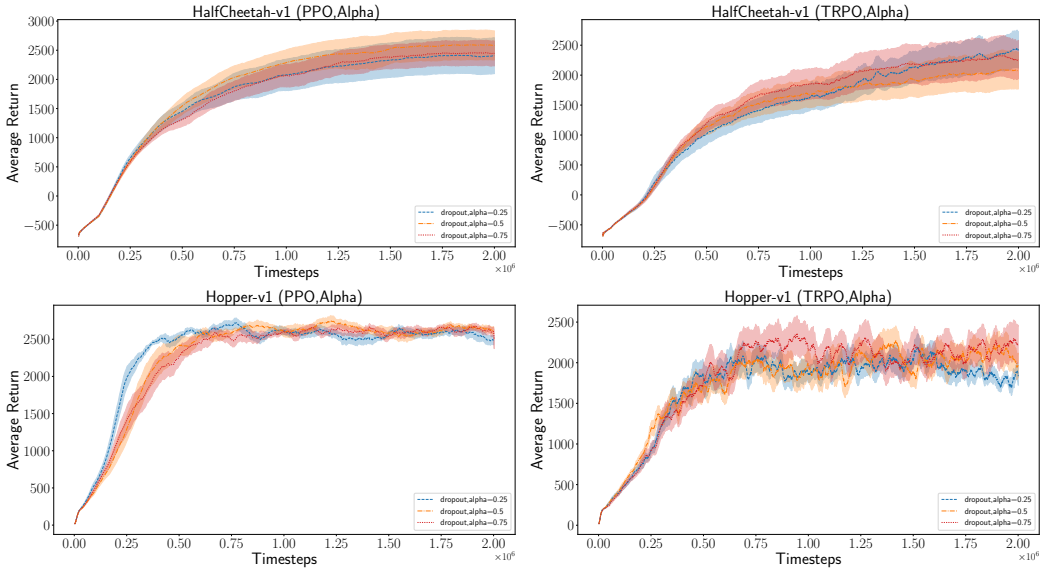

Figure 7: Ablation investigation into the α parameter in the α-divergence objective.

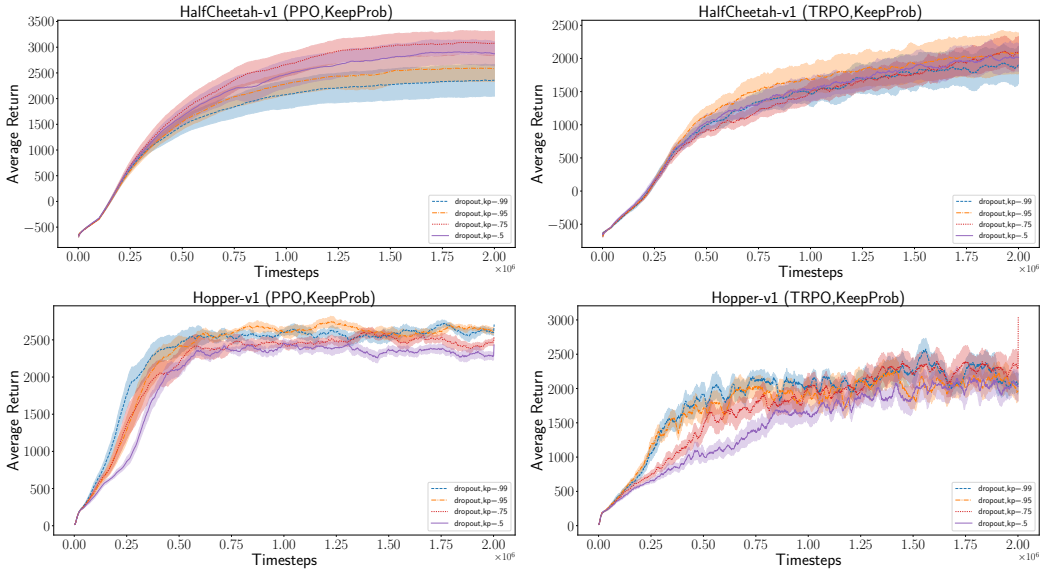

Figure 8: Ablation investigation into the keep probability for the dropout function.

to the stable nature of this environment over others like Hopper-v1. This is discussed in [15]. The τ parameter can be thought of as the trade-off between optimization of the objective loss and regularization. As such we can see large variations in performance due to emphasis on the different components. Furthermore, Figures 8, 6, 7 investigate the $KeepProbability$, $MCSamples$, and α paramater of the α-divergence objective. We find that in some cases (e.g. as with the $KeepProbability$ in PPO HalfCheetah) effects vary significantly and can cause performance gains, while in others generally do not affect results (e.g. with α in PPO). We find also that effects of hyperparameters vary between PPO and TRPO, showing that despite them both using trust regions, these two algorithms have different properties and dynamics.

8.3 Extended Proof for Alpha Divergence DDPG Updates

To demonstrate how the Monte Carlo dropout distributional expectation fits into the DDPG update, we can examine the following proof as per the original DDPG derivation. We want to prove that by maximizing the posterior reward $\mathbb{E}[J(\mu_\theta)|\mathcal{D}]$, we end up with the following Bayesian DDPG updates:

$$\nabla_\theta \mathbb{E}[J(\mu_\theta)|\mathcal{D}] = \mathbb{E}_{s\sim\rho,\omega\sim q(\omega)}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\pi(s,a;\omega)|_{a=\mu_\theta(s)}]$$

where

$$\mathbb{E}[J(\mu_\theta)|\mathcal{D}] = \int_S p_1(s) \int_\omega q(\omega) V^{\mu_\theta}(s) d\omega ds = \int_S p_1(s) \int_\omega q(\omega) Q^{\mu_\theta}(s, \mu_\theta(s)) d\omega ds$$

Proof, notations and framework are largely adapted from [3], please refer to it for more details.
$Q^{\mu_\theta}(s,\mu_\theta(s)) = Q^{\mu_\theta}(s,\mu_\theta(s);\omega)$ where ω are the weight of $Q(s,a)$

$$\begin{aligned}
\mathbb{E}_{\omega\sim q(\omega)}[\nabla_\theta V^{\mu_\theta}(s)] &= \nabla_\theta \mathbb{E}_{\omega\sim q(\omega)}[Q^{\mu_\theta}(s,\mu_\theta(s))] \\
&= \nabla_\theta (r(s,\mu_\theta(s) + \int_S \gamma p(s'|s,\mu_\theta(s)) \int_\omega q(\omega) Q^{\mu_\theta}(s,\mu_\theta(s')) d\omega ds') \\
&= \nabla_\theta \mu_\theta(s) \nabla_a r(s,a)|_{a=\mu_\theta(s)} + \int_S \gamma \nabla_\theta \mu_\theta(s) \nabla_a p(s'|s,a)|_{a=\mu_\theta(s)} \int_\omega q(\omega) Q^{\mu_\theta}(s',\mu_\theta(s')) d\omega ds' \\
&+ \int_S \gamma p(s'|s,\mu_\theta(s)) \int_\omega q(\omega) \nabla_\theta V^{\mu_\theta}(s') d\omega ds' \\
&= \nabla_\theta \mu_\theta(s) \nabla_a (r(s,a) + \int_S \gamma p(s'|s,a) \int_\omega q(\omega) Q^{\mu_\theta}(s,\mu_\theta(s')) d\omega ds')|_{a=\mu_\theta(s)} \\
&+ \int_S \gamma p(s'|s,\mu_\theta(s)) \nabla_\theta \int_\omega q(\omega) Q^{\mu_\theta}(s,\mu_\theta(s')) d\omega ds' \\
&= \underbrace{\nabla_\theta \mu_\theta(s) \nabla_a \mathbb{E}_{\omega\sim q(\omega)}[Q^{\mu_\theta}(s,\mu_\theta(s))]|_{a=\mu_\theta(s)}}_{1} \\
&+ \underbrace{\int_S \gamma p(s\to s',1,\mu_\theta) \nabla_\theta \mathbb{E}_{\omega\sim q(\omega)}[Q^{\mu_\theta}(s',\mu_\theta(s'))] ds}_{2}
\end{aligned}$$

Expanding $Q^{\mu_\theta}(s, \mu_\theta(s'))$ in the 2nd term and iterating gives:

$$\begin{aligned}
\mathbb{E}_{\omega\sim q(\omega)}[\nabla_\theta V^{\mu_\theta}(s)] &= 1 + \int_S \gamma p(s \to s', 1, \mu_\theta)\nabla_\theta \mu_\theta(s)\nabla_a \mathbb{E}_{\omega\sim q(\omega)}[Q^{\mu_\theta}(s', \mu_\theta(s'))]|_{a=\mu_\theta(s')} ds' \\
&\quad + \int_S \gamma^2 p(s \to s', 1, \mu_\theta) \int_S p(s' \to s'', 1, \mu_\theta)\nabla_\theta \mathbb{E}_{\omega\sim q(\omega)}[Q^{\mu_\theta}(s, \mu_\theta(s''))] ds'' ds' \\
&\quad \ldots \\
&= \int_S \sum_{t=0}^{\infty} \gamma^t p(s \to s', t, \mu_\theta)\nabla_\theta \mu_\theta(s')\nabla_a \mathbb{E}_{\omega\sim q(\omega)}[Q^{\mu_\theta}(s, \mu_\theta(s'))]|_{a=\mu_\theta(s')} ds'
\end{aligned}$$

Finally, we get:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{s\sim\rho, a\sim\mu_\theta, \omega\sim q(\omega)}[J(\mu_\theta)] =& \nabla_\theta \int_S p_1(s) \int_\omega q(\omega) Q^{\mu_\theta}(s, \mu_\theta(s)) d\omega ds \\
=& \mathbb{E}_{s\sim\rho, \omega\sim q(\omega)}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\pi(s, a; \omega)|_{a=\mu_\theta(s)}]
\end{aligned}$$

with $\int_S \sum_{t=0}^{\infty} \gamma^t p_1(s) p(s \to s', t, \mu_\theta) ds = \rho^{\mu_\theta}(s')$