Discretizing Continuous Action Space for On-Policy Optimization

Yunhao Tang,[1] **Shipra Agrawal**[1]*
[1]Columbia University, New York, NY, USA
{yt2541, sa3305}@columbia.edu

Abstract

In this work, we show that discretizing action space for continuous control is a simple yet powerful technique for on-policy optimization. The explosion in the number of discrete actions can be efficiently addressed by a policy with factorized distribution across action dimensions. We show that the discrete policy achieves significant performance gains with state-of-the-art on-policy optimization algorithms (PPO, TRPO, ACKTR) especially on high-dimensional tasks with complex dynamics. Additionally, we show that an ordinal parameterization of the discrete distribution can introduce the inductive bias that encodes the natural ordering between discrete actions. This ordinal architecture further significantly improves the performance of PPO/TRPO.

Introduction

In reinforcement learning (RL), the action space of conventional control tasks are usually dichotomized into either discrete or continuous (Brockman et al., 2016). While discrete action space is conducive to theoretical analysis, in the context of deep reinforcement learning, their application is limited to video game playing or board game (Mnih et al., 2013; Silver et al., 2016). On the other hand, in simulated or real life robotics control (Levine et al., 2016; Schulman et al., 2015a), the action space is by design continuous. Continuous control typically requires more subtle treatments, since a continuous range of control contains an infinite number of feasible actions and one must resort to parametric functions for a compact representation of distributions over actions.

Can we retain the simplicity of discrete actions when solving continuous control tasks? A straightforward solution is to discretize the continuous action space, i.e. we discretize the continuous range of action into a finite set of atomic actions and reduce the original task into a new task with a discrete action space. A common argument against this approach is that for an action space with M dimensions, discretizing K atomic actions per dimension leads to M^K combinations of joint atomic actions, which quickly becomes intractable when M increases. However, a simple fix is to represent the joint distribution over discrete actions as factorized across dimensions, so that the joint policy is still tractable. As prior works have applied such discretization method in practice (OpenAI, 2018; Jaśkowski et al., 2018), we aim to carry out a systemic study of such straightforward discretization method in simulated environments, and show how they improve upon on-policy optimization baselines.

The paper proceeds as follows. In Section 2, we introduce backgrounds on on-policy optimization baselines (e.g. TRPO and PPO) and related work. In Section 3, we introduce the straightforward method of discretizing action space for continuous control, and analyze the properties of the resulting policies as the number atomic actions K changes. In Section 4, we introduce *stick-breaking* parameterization (Khan et al., 2012), an architecture that parameterizes the discrete distributions while encoding the natural ordering between discrete actions. In Section 5, through extensive experiments we show how the discrete/ordinal policy improves upon current on-policy optimization baselines and related prior works, especially on high-dimensional tasks with complex dynamics.

Background

In the standard formulation of Reinforcement Learning (RL), at time step $t \geq 0$, an agent is in state $s_t \in \mathcal{S}$, takes an action $a_t \in \mathcal{A}$, receives an instant reward $r_t = r(s_t, a_t) \in \mathbb{R}$ and transitions to a next state $s_{t+1} \sim p(\cdot|s_t, a_t) \in \mathcal{S}$. Let $\pi : \mathcal{S} \mapsto P(\mathcal{A})$ be a policy, where $P(\mathcal{A})$ is the set of distributions over the action space $\mathcal{A}$. The discounted cumulative reward under policy π is $J(\pi) = \mathbb{E}_\pi\big[\sum_{t=0}^{\infty} \gamma^t r_t\big]$, where $\gamma \in [0, 1)$ is a discount factor. The objective of RL is to search for a policy π that achieves the maximum cumulative reward $\pi^* = \arg\max_\pi J(\pi)$. For convenience, under policy π we define action value function $Q^\pi(s, a) = \mathbb{E}_\pi\big[J(\pi)|s_0 = s, a_0 = a\big]$ and value function $V^\pi(s) = \mathbb{E}_\pi\big[J(\pi)|s_0 = s, a_0 \sim \pi(\cdot|s_0)\big]$. We also define the advantage function $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$. One common way to approximately find π^* is through direct policy search within a given policy class $\pi_\theta, \theta \in \Theta$. The algorithm proceeds by iteratively updating the policy parameter to search for higher performing policy.

*This research was supported in part by an Amazon Research Award (ARA) and a Google Faculty Research Award.

Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) formulates the policy search problem as a trust region optimization problem (Schulman et al., 2015a). In particular, consider the following trust region formulation

$$\max_{\theta_{\text{new}}} \mathbb{E}_{\pi_\theta}\big[\frac{\pi_{\theta_{\text{new}}}(a_t|s_t)}{\pi_\theta(a_t|s_t)} A^{\pi_\theta}(s_t,a_t)\big], \quad \mathbb{E}_s\big[\mathbb{KL}[\pi_\theta(\cdot|s)||\pi_{\theta_{\text{new}}}(\cdot|s)]\big] \leq \epsilon, \tag{1}$$

where $\mathbb{E}_s[\cdot]$ is w.r.t. the state visitation distribution induced by π_θ. The trust region enforced by the KL-divergence entails that the update according to (1) optimizes a lower bound of $J(\pi_\theta)$, so as to avoid accidentally taking large steps that irreversibly degrade the policy performance during training. More recently, ACKTR Wu et al. (2017) propose to replace the CG descent of TRPO by Kronecker-factored approximation (Martens and Grosse, 2015) when computing the matrix inversion. This approximation is more stable than CG descent and yields performance gain over conventional TRPO.

Proximal Policy Optimization (PPO) For a practical algorithm, TRPO requires approximately inverting the Fisher matrices by conjugate gradient iterations. Proximal Policy Optimization (PPO) Schulman et al. (2017a) propose to approximate a trust-region method by clipping the likelihood ratios $\rho_t = \frac{\pi_{\theta_{\text{new}}}(a_t|s_t)}{\pi_\theta(a_t|s_t)}$ as $\bar{\rho}_t = \text{clip}(\rho_t, 1-\eta, 1+\eta)$ where $\text{clip}(x,a,b)$ clips the argument x between a and b. Consider the following objective

$$\max_{\theta_{\text{new}}} \mathbb{E}_{\pi_\theta}\big[\min\{\rho_t A^{\pi_\theta}(s_t,a_t), \bar{\rho}_t A^{\pi_\theta}(s_t,a_t)\}\big], \quad ||\theta_{\text{new}} - \theta||_2 \leq \epsilon. \tag{2}$$

The intuition behind the objective (2) is when $A^{\pi_\theta}(s_t,a_t) > 0$, the clipping removes the incentives for the ratio ρ_t to go above $1+\eta$, with similar effects for the case when $A^{\pi_\theta}(s_t,a_t) < 0$. PPO achieves more stable empirical performance than TRPO and involves only relatively cheap first-order optimization.

Related Work

Policy Classes. Orthogonal to the algorithmic procedures for policy updates, one is free to choose any policy classes. In discrete action space, the only choice is a categorical distribution (or a discrete distribution). In continuous action space, the default baseline is factorized Gaussian (Schulman et al., 2015a, 2017a). Gaussian mixtures, implicit generative models or even Normalizing flows (Rezende and Mohamed, 2015) can be used for more expressive and flexible policy classes (Tang and Agrawal, 2018; Haarnoja et al., 2017, 2018b,a), which achieve performance gains primarily for off-policy learning. One issue with aforementioned prior works is that they do not disentangle algorithms from distributions, it is therefore unclear whether the benefits result from a better algorithm or an expressive policy. To make the contributions clear, we make no changes to the on-policy algorithms and show the net effect of how the policy classes improve the performance. Motivated by the fact that unbounded distributions can generate infeasible actions, Chou, Maturana, and Scherer (2017) propose to use Beta distribution and also show improvement on TRPO. Early prior work Shariff and Dick also propose truncated Gaussian distribution but such idea is not tested on deep RL tasks. Complement to prior works, we propose discrete/ordinal policy as simple yet powerful alternates to baseline policy classes.

Discrete and Continuous Action Space. Prior works have exploited the connection between discrete and continuous action space. For example, to solve discrete control tasks, Van Hasselt and Wiering (2009); Dulac-Arnold et al. (2015) leverage the continuity in the underlying continuous action space for generalization across discrete actions. Prior works have also converted continuous control problems into discrete ones, e.g. Pazis and Lagoudakis (2009) convert low-dimensional control problems into discrete problems with binary actions. Surprisingly, few prior works have considered a discrete policy and apply off-the-shelf policy optimization algorithms directly. Recently, OpenAI (2018); Jaśkowski et al. (2018) apply discrete policies to challenging hand manipulation and humanoid walking respectively. As a more comprehensive study, we carry out a full evaluation of discrete/ordinal policy on continuous control benchmarks and validate their performance gains.

To overcome the explosion of action space, Metz et al. (2018) overcome the explosion by sequence prediction, but so far their strategy is only shown effective on relatively low-dimensional problems (e.g. HalfCheetah). Tavakoli, Pardo, and Kormushev (2018) propose to avoid taking $\arg\max$ across all actions in Q-learning, by applying $\arg\max$ independently across dimensions. Their method is also only tested on a very limited number of tasks. As an alternative, we consider distributions that factorize across dimensions and we show that this simple technique yields consistent performance gains.

Ordinal Architecture. When discrete variables have an internal ordering, it is beneficial to account for such ordering when modeling the categorical distributions. In statistics, such problems are tackled as ordinal regression or classification (Winship and Mare, 1984; Chu and Ghahramani, 2005; Chu and Keerthi, 2007). Few prior works aim to combine ideas of ordinal regression with neural networks. Though Cheng, Wang, and Pollastri (2008) propose to introduce ordering as part of the loss function, they do not introduce a proper probabilistic model and need additional techniques during inference. More recently, Khan et al. (2012) motivate the *stick-breaking* parameterization, a proper probabilistic model which does not introduce additional parameters compared to the original categorical distribution. In our original derivation, we motivate the architecture of (Khan et al., 2012) by transforming the loss function of (Cheng, Wang, and Pollastri, 2008). We also show that such additional inductive bias greatly boosts the performance for PPO/TRPO.

Discretizing Action Space for Continuous Control

Without loss of generality, we assume the action space $\mathcal{A} = [-1, 1]^m$. We discretize each dimension of the action space into K equally spaced atomic actions. The set of atomic action for any dimension i is $\mathcal{A}_i = \{\frac{2j}{K-1} - 1\}_{j=0}^{K-1}$. To overcome the curse of dimensionality, we represent the distribution as factorized across dimensions. In particular, in state s, we specify a categorical distribution $\pi_{\theta_j}(a_j|s)$ over actions $a_j \in \mathcal{A}_j$ for each dimension j, with θ_j as parameters for this marginal distribution. Then we define the joint discrete policy $\pi(a|s) \coloneqq \Pi_{j=1}^{m} \pi_{\theta_j}(a_j|s)$ where $a = [a_0, a_1, ...a_{K-1}]^T$. The factorization allows us to maintain a tractable distribution over joint actions, making it easy to do both sampling and training.

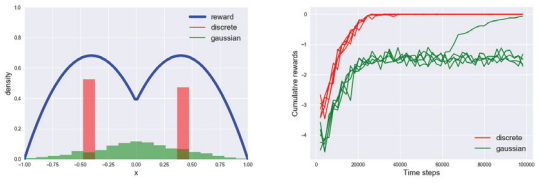
Network Architecture

The discrete policy is parameterized as follows. As in prior works (Schulman et al., 2015b, 2017b), the policy π_θ is a neural network that takes state s as input, through multiple layers of transformation it will encode the state into a hidden vector $h(s) = f_\theta(s)$. For the jth action in the ith dimension of the action space, we output a logit $L_{ij} = w_{ij}^T h(s) + b_{ij} \in \mathbb{R}$ with parameters w_{ij}, b_{ij}. For any dimension i, the K logits $L_{ij}, 1 \leq j \leq K$ are combined by soft-max to compute the probability of choosing action j, $p_{ij} = \text{softmax}(L_{ij})(\coloneqq \frac{\exp(L_{ij})}{\sum_{j=0}^{K-1} L_{ij}})$. By construction, the network has a fixed-size low-level parameter θ, while the output layer has parameters w_{ij}, b_{ij} whose size scales linearly with K.

Understanding Discrete Policy for Continuous Control

Here we briefly analyze the empirical properties of the discrete policy.

Discrete Policy is more expressive than Gaussian. Though discrete policy is limited on taking atomic actions, in practice it can represent much more flexible distributions than Gaussian when there are sufficient number of atomic actions (e.g. $K \geq 11$). Intuitively, discrete policy can represent multi-modal action distribution while Gaussian is by design unimodal. We illustrate this practical difference by a bandit example in Figure 1. Consider a one-step bandit problem with $\mathcal{A} = [-1, 1]$. The reward function for action a is $r(a)$ illustrated as the Figure 1 (a) blue curve. We train a discrete policy with $K = 11$ and a Gaussian policy on the environment for 10^5 steps and show their training curves in (b), with five different random seeds per policy. We see that 4 out 5 five Gaussian policies are stuck at a suboptimal policy while all discrete policies achieve the optimal rewards. Figure 1 (a) illustrates the density of a trained discrete policy (red) and a suboptimal Gaussian policy (green). The trained discrete policy is bi-modal and automatically captures the bi-modality of the reward function (notice that we did not add entropy regularization to encourage high entropy). The only Gaussian policy that achieves optimal rewards in (b) captures only one mode of the reward function.



(a) **Bandit: Density** (b) **Bandit: curves**

Figure 1: Analyzing discrete policy: (a) Bandit example: comparison of normalized reward (blue), trained discrete policy density (red) and trained Gaussian policy density (green). (b) Bandit example: learning curves of discrete policy vs. Gaussian policy, we show five random seeds. Most seeds for Gaussian policy get stuck at a suboptimal policy displayed in (a) and all discrete policies reach a bi-modal optimal policy as in (a).

For general high-dimensional problems, the reward landscape becomes much more complex. However, this simple example illustrates that the discrete policy can potentially capture the multi-modality of the landscape and achieve better exploration (Haarnoja et al., 2017) to bypass suboptimal policies.

Effects of the Number of Atomic Actions K. Choosing a proper number of atomic actions per dimension K is critical for learning. The trade-off comes in many aspects: **(a)** Control capacity. When K is small the discretization is too coarse and the policy does not have enough capacity to achieve good performance. **(b)** Training difficulty. When K increases, the variance of policy gradients also increases. We detail the analysis of policy gradient variance in Appendix B. We also present the combined effects of (a) and (b) in Appendix B, where we find that the best performance is obtained when $7 \leq K \leq 15$, and setting K either too large or too small will degrade the performance. **(c)** Model parameters and computational Costs. Both the number of model parameters increases linearly and computational costs grow linearly in K. We present detailed computational results in Appendix B.

Discrete Policy with Ordinal Architecture

Motivation

When the continuous action space is discretized, we treat continuous variables as discrete and discard important information about the underlying continuous space. It is therefore desirable to incorporate the notion of continuity when paramterizing distributions over discrete actions.

Ordinal Distribution Network Architecture

For simplicity, we discuss the discrete distribution over only one action dimension. Recall previously that a typical feed-forward architecture that produces discrete distribution over K classes produces K logits L_i at the last layer and derives the probability via a softmax $p_i = \text{softmax}(L_i), 1 \leq i \leq K$. In the ordinal architecture, we retain these logits L_i and first

transform them via a sigmoid function $s_i = \sigma(L_i)$. Then we compute the final logits as

$$L'_i = \sum_{j \le i} \log s_i + \sum_{j > i} \log(1 - s_i), \forall 1 \le i \le K, \qquad (3)$$

and derive the final output probability via a softmax $p'_i = \text{softmax}(L'_i)$. The actions are sampled according to this discrete distribution $a_j \sim p'_j$.

This architecture is very similar to the *stick-breaking* parameterization introduced in Khan et al. (2012), where they argue that such parameterization is beneficial when the samples drawn from class k can be easily separated from samples drawn from all the classes $j > k$. In our original derivation, we motivate the ordinal architecture from the loss function of (Cheng, Wang, and Pollastri, 2008) and we show that the intuition behind (3) is more clear from this perspective. We show the intuition below with a K-way classification problem, where the classes are internally ordered as $\{1, 2, ...K\}$.

Intuition behind (3). For clarity, let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^K$ such that $0 \le x_i, y_i \le 1$ and define the stable cross entropy $\text{CE}(\mathbf{x}, \mathbf{y}) \coloneqq -\sum_{i=1}^{K} x_i \log \max\{y_i, \epsilon\})$ with a very small $\epsilon > 0$ to avoid numerical singularity. For a sample from class k, the $K-$way classification loss based on (3) is $-L'_k = \text{CE}(\mathbf{t}_k, \mathbf{s})$, where the predicted vector $\mathbf{s} = [s_1, s_2...s_K]$ and a target vector $\mathbf{t}_k = [1, 1...0]$ with first k entries to be 1s and others 0s. The intuition becomes clear when we interpret $\mathbf{t}_k$ as a continuous encoding of the class k (instead of the one-hot vector) and $\mathbf{s}$ as a intermediate vector from which we draw the final prediction. We see that the continuity between classes is introduced through the loss function, for example $\text{CE}(\mathbf{t}_k, \mathbf{t}_{k+1}) < \text{CE}(\mathbf{t}_k, \mathbf{t}_{k+2})$, i.e. the discrepancy between class k and $k+1$ is strictly smaller than that between k and $k+1$. On the contrary, such information cannot be introduced by one-hot encoding: let $\mathbf{e}_k$ be the one-hot vector for class k, we always have e.g. $\text{CE}(\mathbf{e}_k, \mathbf{e}_{k+1}) = \text{CE}(\mathbf{e}_k, \mathbf{e}_{k+2})$, i.e. we introduce no discrepancy relationship between classes. While Cheng, Wang, and Pollastri (2008) introduce such continuous encoding techniques, they do not propose proper probabilistic models and require additional techniques at inference time to make predictions. Here, the ordinal architecture (3) defines a proper probabilistic model that implicitly introduces internal ordering between classes through the parameterization, while maintaining all the probabilistic properties of discrete distributions.

In summary, the oridinal architecture (3) introduces additional dependencies between logits L_i which implicitly inject the information about the class ordering. In practice, we find that this generally brings significant performance gains during policy optimization.

Experiments

Our experiments aim to address the following questions: **(a)** Does discrete policy improve the performance of baseline algorithms on benchmark continuous control tasks? **(b)** Does the ordinal architecture further improve upon discrete policy? **(c)** How sensitive is the performance to hyper-parameters, particularly to the number of bins per action dimension?

For clarity, we henceforth refer to *discrete policy* as with the discrete distribution, and *ordinal policy* as with the ordinal architecture. To address **(a)**, we carry out comparisons in two parts: (1) We compare discrete policy (with varying K) with Gaussian policy over baseline algorithms (PPO, TRPO and ACKTR), evaluated on benchmark tasks in gym MuJoCo (Brockman et al., 2016; Todorov, 2008), rllab (Duan et al., 2016), roboschool (Schulman et al., 2015a) and Box2D. Here we pay special attention to Gaussian policy because it is the default policy class implemented in popular code bases (Dhariwal et al., 2017); (2) We compare with other architectural alternatives, either straihghtforward architectural variants or those suggested in prior works (Chou, Maturana, and Scherer, 2017). We evaluate their performance on high-dimensional tasks with complex dynamics (e.g. Walker, Ant and Humanoid). All the above results are reported in Section 5.1. To address **(b)**, we compare discrete policy with ordinal policy with PPO in Section 5.2 (results for TRPO are also in Section 5.1). To address (c), we randomly sample hyper-parameters for Gaussian policy and discrete policy and compare their quantiles plots in Section 5.3 and Appendix C.

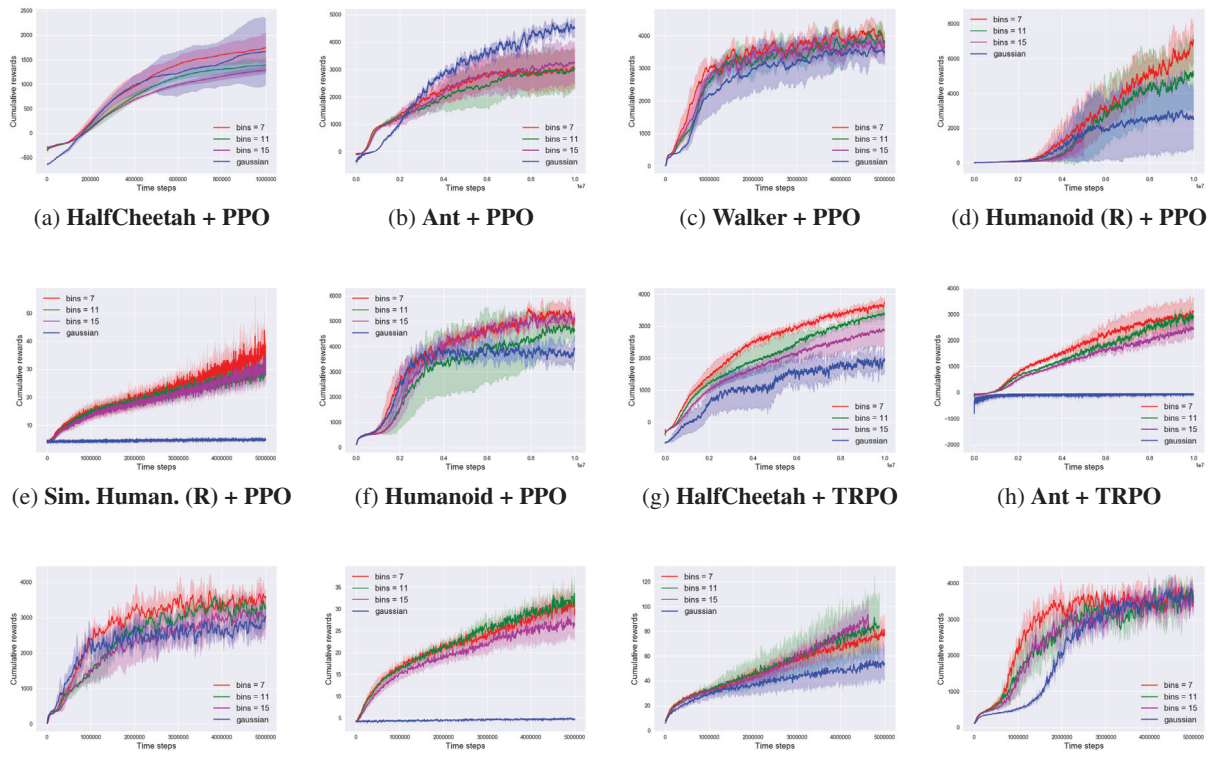
Implementation Details. As we aim to study the net effect of the discrete/ordinal policy with on-policy optimization algorithms, we make minimal modification to the original PPO/TRPO/and ACKTR algorithms originally implemented in OpenAI baselines (Dhariwal et al., 2017). We leave all hyper-parameter settings in Appendix A.

Benchmark performance

All benchmark comparison results are presented in plots (Figure 2,3) or tables (Table 1,2). For plots, we show the learning curves of different policy classes trained for a fixed number of time steps. The x-axis shows the time steps while the y-axis shows the cumulative rewards. Each curve shows the average performance with standard deviation shown in shaded areas. Results in Figure 2,4 are averaged over 5 random seeds and Figure 3 over 2 random seeds. In Table 1,2 we train all policies for a fixed number of time steps and we show the average $\pm$ standard deviation of the cumulative rewards obtained in the last 10 training iterations.

PPO/TRPO - Comparison with Gaussian Baselines. We evaluate PPO/TRPO with Gaussian against PPO/TRPO with discrete policy on the full suite of MuJoCo control tasks and display all results in Figure 2. For PPO, on tasks with relatively simple dynamics, discrete policy does not necessarily enjoy significant advantages over Gaussian policy. For example, the rate of learning of discrete policy is comparable to Gaussian for HalfCheetah (Figure 2(a)) and even slightly lower on Ant 2(b)). However, on high-dimensional tasks with very complex dynamics (e.g. Humanoid, Figure 2(d)-(f)), discrete policy significantly outperforms Gaussian policy. For TRPO, the performance gains by discrete policy are also very consistent and significant.

We also evaluate the algorithms on Roboschool Humanoid tasks as shown in Figure 3. We see that discrete policy achieves better results than Gaussian across all tasks and



(a) **HalfCheetah + PPO** (b) **Ant + PPO** (c) **Walker + PPO** (d) **Humanoid (R) + PPO**

(e) **Sim. Human. (R) + PPO** (f) **Humanoid + PPO** (g) **HalfCheetah + TRPO** (h) **Ant + TRPO**

(i) **Walker + TRPO** (j) **Sim. Human. (R) + TRPO** (k) **Humanoid (R) + TRPO** (l) **Humanoid + TRPO**

Figure 2: MuJoCo Benchmarks: learning curves of PPO on OpenAI gym MuJoCo locomotion tasks. Each curve is averaged over 5 random seeds and shows mean $\pm$ std performance. Each curve corresponds to a different policy architecture (Gaussian or discrete actions with varying number of bins $K = 7, 11, 15$). Vertical axis is the cumulative rewards and horizontal axis is the number of time steps. Discrete actions significantly outperform Gaussian on Humanoid tasks. Tasks with (R) are from rllab.

both algorithms. The performance gains are most significant with TRPO (Figure 3(b)(d)), where we see Gaussian policy barely makes progress during training while discrete policy has very stable learning curves. We provide additional Roboschool Humanoid results in the Appendix C due to space limit. For completeness, we also evaluate PPO/TRPO with discrete policy vs. Gaussian policy on Box2D tasks and see that the performance gains are significant. Due to space limit, We present Box2D results in Appendix C.

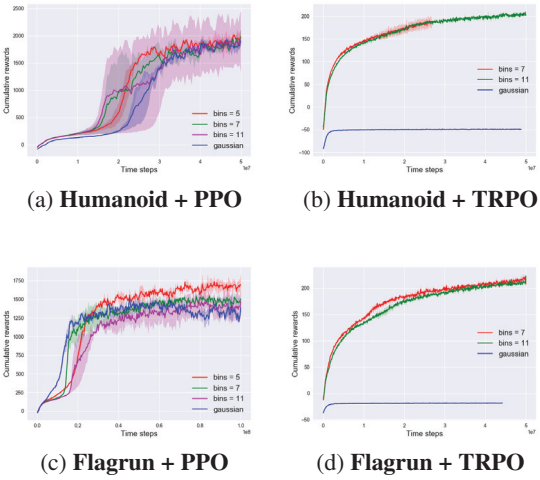
By construction, when discrete policy and Gaussian policy have the same encoding architecture $f_\theta(s)$ shown in Section 3, discrete policy has many more parameters than Gaussian policy. A critical question is whether we can achieve performance gains by simply increasing the number of parameters? We show that when we train a Gaussian policy with many more parameters (e.g. 128 hidden units per layer), the policy does not perform as well. This validates our speculation that the performance gains result from a more carefully designed distribution class rather than larger networks.

PPO - Comparison with Off-Policy Baselines. To further illustrate the strength of PPO with discrete policy on high-dimensional tasks with very complex dynamics, we compare PPO with discrete policy against state-of-the-art off-policy algorithms on Humanoid tasks (Humanoid-v1 and Humanoid rllab) [1]. Such algorithms include DDPG (Lillicrap et al., 2015), SQL (Haarnoja et al., 2017), SAC (Haarnoja et al., 2018b) and TD3 (Fujimoto, van Hoof, and Meger, 2018), among which SAC and TD3 are known to achieve significantly better performance on MuJoCo benchmark tasks over other algorithms. Off-policy algorithms reuse samples and can potentially achieve orders of magnitude better sample efficiency than on-policy algorithms. For example, it has been commonly observed in prior works (Haarnoja et al., 2018b,a; Fujimoto, van Hoof, and Meger, 2018) that SAC/TD3 can achieve state-of-the-art performance on most benchmark control tasks for only 10^6 steps of training, on condition that off-policy samples are heavily replayed. In general, on-policy algorithms cannot match such level of fast convergence because samples are quickly discarded. However, for highly complex tasks such as Humanoid even off-policy algorithms take many more samples to learn, potentially because off-policy learning becomes more unstable and off-policy sam-

[1] Humanoid-v1 has $|\mathcal{S}| = 376, |\mathcal{A}| = 17$ and Humanoid rllab has $|\mathcal{S}| = 142, |\mathcal{A}| = 21$. Both tasks have very high-dimensional observation space and action space.

ples are less informative. In Table 1, we record the final performance of off-policy algorithms directly from the figures in (Haarnoja et al., 2018b) following the practice of (Mania, Guy, and Recht, 2018). The final performance of PPO algorithms are computed as the average $\pm$ std of the returns in the last 10 training iterations across 5 random seeds. All algorithms are trained for 10^7 steps. We observe in Table 1 that PPO + discrete (ordinal) actions achieve comparable or even better results than off-policy baselines. This shows that for general complex applications, PPO + discrete/ordinal is still as competitive as the state-of-the-art off-policy methods.

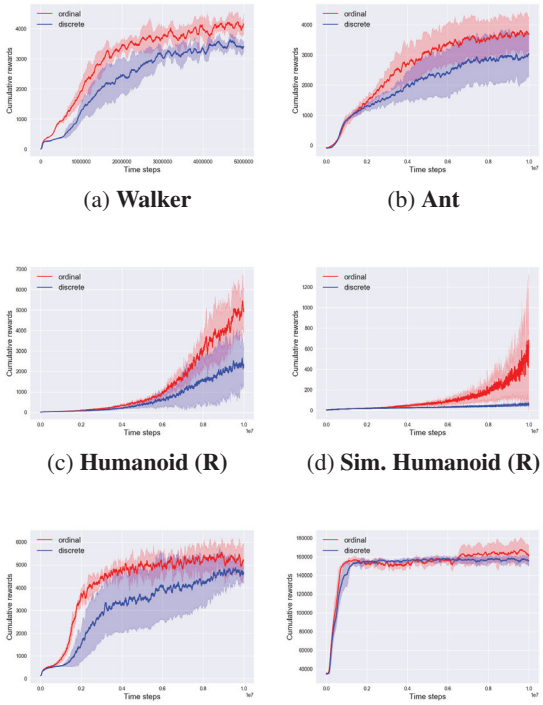
PPO/TRPO - Comparison with Alternative Architectures. We also compare with straightforward architectural alternatives: Gaussian with tanh non-linearity as the output layer, and Beta distribution (Chou, Maturana, and Scherer, 2017). The primary motivation for these architectures is that they naturally bound the sampled actions to the feasible range ($[-1, 1]$ for gym tasks). By construction, our proposed discrete/ordinal policy also bound the sampled actions within the feasible range. In Table 2, we show results for PPO/TRPO where we select the best result from $K \in \{7, 11, 15\}$ for discrete/ordinal policy. We make several observations from results in Table 2: (1) Bounding actions (or action means) to feasible range does not consistently bring performance gains, because we observe that Gaussian + tanh and Beta distribution do not consistently outperform Gaussian. This is potentially because the parameterizations that bound the actions (or action means) also introduce challenges for optimization. For example, Gaussian + tanh bounds the action means $\mu_\theta(s) \in [-1, 1]$, this implies that in order for $\mu_\theta(s) \approx \pm 1$ the parameter θ must reach extreme values, which is hard to achieve using SGD based methods. (2) Discrete/Ordinal policy achieve significantly better results consistently across most tasks. Combining (1) and (2), we argue that the performance gains of discrete/ordinal policies are due to reasons beyond a bounded action distribution.



(a) **Humanoid + PPO** (b) **Humanoid + TRPO**

(c) **Flagrun + PPO** (d) **Flagrun + TRPO**

Figure 3: Roboschool Humanoid Benchmarks: learning curves of PPO/TRPO on Roboschool Humanoid locomotion tasks. Each curve corresponds to a different policy architecture (Gaussian or discrete actions with varying number of bins $K = 5, 7, 11$). Discrete policies outperform Gaussian policy on all Humanoid tasks and the performance gains are more significant with TRPO.



(a) **Walker** (b) **Ant**

(c) **Humanoid (R)** (d) **Sim. Humanoid (R)**

(e) **Humanoid** (f) **Humanoid Standup**

Figure 4: MuJoCo Benchmarks: learning curves of PPO + discrete policy vs. PPO + ordinal policy on OpenAI gym MuJoCo locomotion tasks. All policies have $K = 11$. We see that for each task, ordinal policy outperforms discrete policy.

Here we discuss the results for Beta distribution. In our implementation we find training with Beta distribution tends to generate numerical errors when the update is more aggressive (e.g. PPO learning rate $3 \cdot 10^{-5}$ or TRPO trust region size is 0.01). More conservative updates (e.g. e.g. PPO learning rate $3 \cdot 10^{-6}$ or TRPO trust region size is 0.001) reduce numerical errors but also greatly degrade the learning performance. We suspect that this is because the Beta distribution parameterization (Appendix A and (Chou, Maturana, and Scherer, 2017)) is numerically unstable and we discuss the potential reason in Appendix A. In Table 2, the results for Beta distribution is recorded as the performance of the last 10 iterations before the training terminates (potentially prematurely due to numerical errors). The potential advantages of Beta distribution are largely offset by the unstable training. We show more results in Appendix C.

Table 1: A comparison of PPO with discrete/ordinal policy with state-of-the-art baseline algorithms on Humanoid benchmark tasks from OpenAI gym and rllab. For each task, we show the average rewards achieved after training the agent for a fixed number of time steps. The results for PPO + discrete/ordinal/Gaussian policy are mean performance averaged over 5 random seeds (see Figure 2. The results for DDPG, SQL, SAC and TD3 are approximated based on the figures in (Haarnoja et al., 2018b). The results for PPO is consistent with results in (Haarnoja et al., 2018b). Even compared to off-policy algorithms, PPO + ordinal policy achieves state of the art performance across both tasks.

Tasks	DDPG	SQL	SAC	TD3	PPO + Gaussian	PPO + discrete	PPO + ordinal
HUMAOID-V1	≈ 500	≈ 5500	$\approx \mathbf{6000}$	$\approx \mathbf{6000}$	≈ 4000	5119 ± 151	$\mathbf{6018 \pm 239}$
HUMANOID(RLLAB)	< 500	≈ 2000	$\approx \mathbf{5500}$	< 500	≈ 2500	4084 ± 1312	$\mathbf{4884 \pm 1562}$

Table 2: Comparison across a range of policy alternatives (Gaussian, Gaussian +tanh, and Beta distribution (Chou, Maturana, and Scherer, 2017)). All policies are optimized with **PPO/TRPO**. All tasks are training for 10M steps. Results are the average $\pm$ std performance for the last 10 training iterations. Top two results (with highest average) are highlighted in bold font. Tasks with (R) are from rllab.

PPO	Gaussian	Gaussian+tanh	Beta	Discrete	Ordinal
WALKER2D	3500 ± 360	3274 ± 251	274 ± 6	$\mathbf{3390 \pm 190}$	$\mathbf{4249 \pm 239}$
ANT	$\mathbf{4445 \pm 194}$	$\mathbf{4622 \pm 171}$	3112 ± 173	3256 ± 778	3690 ± 557
HALFCHEETAH	1598 ± 23	1566 ± 26	1193 ± 24	$\mathbf{4824 \pm 199}$	$\mathbf{3477 \pm 1497}$
HUMANOID	3905 ± 502	4007 ± 698	2680 ± 2493	$\mathbf{5119 \pm 151}$	$\mathbf{6018 \pm 403}$
HUMANOIDSTANDUP	$\mathbf{166446 \pm 18348}$	160983 ± 3842	155362 ± 8657	161618 ± 10224	$\mathbf{170275 \pm 19316}$
HUMANOID (R)	2522 ± 1684	5863 ± 1288	2680 ± 2493	$\mathbf{4084 \pm 1312}$	$\mathbf{4884 \pm 1562}$
SIM. HUMANOID (R)	5.1 ± 0.4	4.3 ± 0.5	4.4 ± 0.6	$\mathbf{214 \pm 136}$	$\mathbf{801 \pm 569}$
TRPO					
ANT	-76 ± 14	-89 ± 13	2362 ± 305	$\mathbf{2687 \pm 556}$	$\mathbf{2977 \pm 266}$
HALFCHEETAH	1576 ± 782	386 ± 78	1643 ± 819	$\mathbf{3081 \pm 766}$	$\mathbf{3352 \pm 1196}$
HUMANOID	1156 ± 163	$\mathbf{6350 \pm 486}$	3812 ± 1973	$\mathbf{3908 \pm 117}$	3577 ± 272
HUMANOID STANDUP	137955 ± 9238	133558 ± 9238	111497 ± 15031	$\mathbf{142640 \pm 2343}$	$\mathbf{143418 \pm 8638}$
HUMANOID (R)	65 ± 8	38 ± 2	38 ± 3	$\mathbf{84 \pm 24}$	$\mathbf{161 \pm 26}$
SIM. HUMANOID (R)	6.5 ± 0.2	4.4 ± 0.1	4.2 ± 0.2	$\mathbf{42 \pm 6}$	$\mathbf{93 \pm 28}$

ACKTR - Comparison with Gaussian Baselines. We show results for ACKTR in Appendix C. We observe that for tasks with complex dynamics, discrete policy still achieves performance gains over its Gaussian policy counterpart.

Discrete Policy vs. Ordinal Policy

In Figure 4, we evaluate PPO + discrete policy and PPO + ordinal policy on high-dimensional tasks. Across all presented tasks, ordinal policy achieves significantly better performance than discrete policy both in terms of asymptotic performance and speed of convergence. Similar results are also presented in table 2 where we show that PPO + ordinal policy achieves comparable performance as efficient off-policy algorithms on Humanoid tasks. We also compare these two architectures when trained with TRPO. The comparison of the trained policies can be found in table 1. For most tasks, we find that ordinal policy still significantly improves upon discrete policy.

Summarizing the results for PPO/TRPO, we conclude that the ordinal architecture introduces useful inductive bias that improves policy optimization. We note that *sticky-breaking* parameterization (3) is not the only parameterization that leverages natural orderings between discrete classes. We leave as promising future work how to better exploit task specific ordering between classes.

Sensitivity to Hyper-parameters

Here we evaluate the policy classes' sensitivity to more general hyper-parameters, such as learning rate α, number of bins per dimension K and random seeds. We present the results of PPO in Appendix C. For PPO with Gaussian, we uniformly sample $\log_{10} \alpha \in [-6.0, -3.0]$ and one of 5 random seeds. For PPO with discrete actions, we further uniformly sample $K \in \{7, 11, 15\}$. For each benchmark task, we sample 30 hyper-parameters and show the quantile plot of the final performance. As seen in Appendix C, PPO with discrete actions is generally more robust to such hyper-parameters than Gaussian.

Conclusion

We have carried out a systemic evaluation of action discretization for continuous control across baseline on-policy algorithms and baseline tasks. Though the idea is straightforward, we find that it greatly improves the performance of baseline algorithms, especially on high-dimensional tasks with complex dynamics. We also show that the ordinal architecture which encodes the natural ordering of the discretized actions

into the discrete distribution, can further boost the performance of baseline algorithms. We believe that these effective techniques can serve as convenient and practical plug-ins for many applications of interest.

References

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Cheng, J.; Wang, Z.; and Pollastri, G. 2008. A neural network approach to ordinal regression. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 1279–1284. IEEE.

Chou, P.-W.; Maturana, D.; and Scherer, S. 2017. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In *International Conference on Machine Learning*, 834–843.

Chu, W., and Ghahramani, Z. 2005. Gaussian processes for ordinal regression. *Journal of machine learning research* 6(Jul):1019–1041.

Chu, W., and Keerthi, S. S. 2007. Support vector ordinal regression. *Neural computation* 19(3):792–815.

Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; and Wu, Y. 2017. Openai baselines. https://github.com/openai/baselines.

Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; and Abbeel, P. 2016. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, 1329–1338.

Dulac-Arnold, G.; Evans, R.; van Hasselt, H.; Sunehag, P.; Lillicrap, T.; Hunt, J.; Mann, T.; Weber, T.; Degris, T.; and Coppin, B. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*.

Fujimoto, S.; van Hoof, H.; and Meger, D. 2018. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*.

Haarnoja, T.; Tang, H.; Abbeel, P.; and Levine, S. 2017. Reinforcement learning with deep energy-based policies. *arXiv preprint arXiv:1702.08165*.

Haarnoja, T.; Hartikainen, K.; Abbeel, P.; and Levine, S. 2018a. Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018b. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

Jaśkowski, W.; Lykkebø, O. R.; Toklu, N. E.; Trifterer, F.; Buk, Z.; Koutník, J.; and Gomez, F. 2018. Reinforcement learning to run fast. In *The NIPS'17 Competition: Building Intelligent Systems*. Springer. 155–167.

Khan, M.; Mohamed, S.; Marlin, B.; and Murphy, K. 2012. A stick-breaking likelihood for categorical data analysis with latent gaussian models. In *Artificial Intelligence and Statistics*, 610–618.

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2016. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research* 17(1):1334–1373.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mania, H.; Guy, A.; and Recht, B. 2018. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*.

Martens, J., and Grosse, R. 2015. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, 2408–2417.

Metz, L.; Ibarz, J.; Jaitly, N.; and Davidson, J. 2018. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

OpenAI. 2018. Leaning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*.

Pazis, J., and Lagoudakis, M. G. 2009. Binary action search for learning continuous-action control policies. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 793–800. ACM.

Rezende, D. J., and Mohamed, S. 2015. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015a. Trust region policy optimization. In *International Conference on Machine Learning*, 1889–1897.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015b. Trust region policy optimization. In *International Conference on Machine Learning*, 1889–1897.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017a. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017b. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shariff, R., and Dick, T. Lunar lander: A continous-action case study for policy-gradient actor-critic algorithms.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature* 529(7587):484–489.

Tang, Y., and Agrawal, S. 2018. Implicit policy for reinforcement learning. *arXiv preprint arXiv:1806.06798*.

Tavakoli, A.; Pardo, F.; and Kormushev, P. 2018. Action branching architectures for deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Todorov, E. 2008. General duality between optimal control and estimation. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, 4286–4292. IEEE.

Van Hasselt, H., and Wiering, M. A. 2009. Using continuous action spaces to solve discrete problems. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, 1149–1156. IEEE.

Winship, C., and Mare, R. D. 1984. Regression models with ordinal variables. *American sociological review* 512–525.

Wu, Y.; Mansimov, E.; Liao, S.; Gross, R.; and Ba, J. 2017. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *arXiv preprint arXiv:1708.05144*.