Boosting Trust Region Policy Optimization with Normalizing Flows Policy

Yunhao Tang [1]   Shipra Agrawal [1]

Abstract

We propose to improve trust region policy search with normalizing flows policy. From the *Control as Inference* perspective, we motivate using expressive policy classes to improve policy optimization procedure. We illustrate that when the trust region is constructed by KL divergence constraints, normalizing flows policy generates samples far from the 'center' of the previous policy iterate, which potentially enables better exploration. Through extensive comparisons, we show that the normalizing flows policy significantly improves upon baseline architectures especially on high-dimensional tasks with complex dynamics.

1. Introduction

In on-policy optimization, vanilla policy gradient algorithms suffer from occasional updates with large step size, which lead to collecting bad samples that the policy cannot recover from (Schulman et al., 2015). Motivated to overcome such instability, Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) constraints the KL divergence between consecutive policies to achieve much more stable updates. However, with factorized Gaussian policy, such KL divergence constraint can put a very stringent restriction on the new policy iterate, making it hard to bypass locally optimal solutions and slowing down the learning process.

Can we improve the learning process of trust region policy search by using a more expressive policy class? Intuitively, a more expressive policy class has more capacity to represent complex distributions and as a result, the KL constraint may not impose a very strict restriction on the sample space. Though prior works (Haarnoja et al., 2017; 2018b;a) have proposed to use expressive generative models as policies, their focus is on off-policy learning. In this work, we show how expressive distributions, in particular normalizing flows (Rezende & Mohamed, 2015; Dinh et al., 2016) can be combined with on-policy learning and boost the performance of

trust region policy optimization.

The structure of our paper is as follows. In Section 1 and 2, we provide backgrounds on policy optimization and normalizing flows. In Section 3, we introduce normalizing flows for control and provide a motivation & justification for expressive policies for control from the *Control as Inference* perspective. In Section 4, we show by comprehensive experiments that normalizing flows significantly outperforms baseline policy classes when combined with trust region policy search algorithms such as TRPO and ACKTR. We discuss related work in Section 5. Find our code at `https://github.com/robintyh1/onpolicybaselines`.

2. Background

In the standard formulation of Markov Decision Process (MDP), 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 $J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^\infty \gamma^t r_t \right]$, 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 $Q^\pi(s, a) = \mathbb{E}_\pi \left[J(\pi)|s_0 = s, a_0 = a \right]$ and value function $V^\pi(s) = \mathbb{E}_\pi \left[J(\pi)|s_0 = s, a_0 \sim \pi(\cdot|s_0) \right]$. We also define the advantage function $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$.

2.1. Policy Optimization

One way to approximately find π^* is through direct policy search within a given policy class $\pi_\theta, \theta \in \Theta$ where Θ is the parameter space for the policy parameter. We can update the paramter θ with policy gradient ascent, by computing $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^\infty A^{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$, then updating $\theta_{\text{new}} \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$ with some learning rate $\alpha > 0$. Alternatively, the update can be formulated by first considering a trust region optimization problem

$$\max_{\theta_{\text{new}}} \mathbb{E}_{\pi_\theta} \left[\frac{\pi_{\theta_{\text{new}}}(a_t|s_t)}{\pi_\theta(a_t|s_t)} A^{\pi_\theta}(s_t, a_t) \right],$$
$$||\theta_{\text{new}} - \theta||_2 \leq \epsilon, \tag{1}$$

[1]Columbia University, New York, NY, USA. Correspondence to: Yunhao Tang <yt2541@columbia.edu>.

for some $\epsilon > 0$. If we do a linear approximation of the objective in (1), $\mathbb{E}_{\pi_\theta}\left[\frac{\pi_{\theta_{\text{new}}}(a_t|s_t)}{\pi_\theta(a_t|s_t)}A^{\pi_\theta}(s_t, a_t)\right] \approx \nabla_\theta J(\pi_\theta)^T(\theta_{\text{new}} - \theta)$, we recover the policy gradient update by properly choosing ϵ given α.

2.2. Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) applies information theoretic constraints instead of Euclidean constraints (as in (1)) between θ_{new} and θ to better capture the geometry on the parameter space induced by the underlying distributions. In particular, consider the following trust region formulation

$$\max_{\theta_{\text{new}}} \mathbb{E}_{\pi_\theta}\left[\frac{\pi_{\theta_{\text{new}}}(a_t|s_t)}{\pi_\theta(a_t|s_t)}A^{\pi_\theta}(s_t, a_t)\right],$$
$$\mathbb{E}_s\left[\mathbb{KL}[\pi_\theta(\cdot|s)||\pi_{\theta_{\text{new}}}(\cdot|s)]\right] \leq \epsilon, \qquad (2)$$

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 (2) optimizes a lower bound of $J(\pi_\theta)$, so as to avoid accidentally taking large steps that irreversibly degrade the policy performance during training as with vanilla policy gradient (1) (Schulman et al., 2015). ACKTR Wu et al. propose to replace the CG descent of TRPO by Kronecker-factored approximation (Martens & Grosse, 2015) when computing matrix inversion. This approximation is more stable than CG descent and yields performance gain over conventional TRPO.

2.3. Normalizing flows

Normalizing flows (Rezende & Mohamed, 2015; Dinh et al., 2016) have been applied in variational inference and probabilistic modeling to represent complex distributions. In general, consider transforming a source noise $\epsilon \sim \rho_0(\cdot)$ by a series of invertible nonlinear functions $g_{\theta_i}(\cdot), 1 \leq i \leq K$ each with parameter θ_i, to output a target sample x,

$$x = g_{\theta_K} \circ g_{\theta_{K-1}} \circ ... \circ g_{\theta_2} \circ g_{\theta_1}(\epsilon). \qquad (3)$$

Let Σ_i be the inverse of the Jacobian matrix of $g_\theta(\cdot)$, then the log density of x is computed by the change of variables formula,

$$\log p(x) = \log p(\epsilon) + \sum_{i=1}^K \log \det(\Sigma_i). \qquad (4)$$

The distribution of x is determined by the noise ϵ and the transformations $g_{\theta_i}(\cdot)$. When the transformations $g_{\theta_i}(\cdot)$ are very complex (e.g. $g_{\theta_i}(\cdot)$ are neural networks) we expect $p(x)$ to be highly expressive as well. However, for a general invertible transformation $g_{\theta_i}(\cdot)$, computing the determinant $\det(\Sigma_i)$ is expensive. In this work, we follow the architecture of (Dinh et al., 2014) to ensure that $\det(\Sigma_i)$ is computed

cheaply while $g_{\theta_i}(\cdot)$ are expressive. We leave all details to Appendix B. We will henceforth also address the normalizing flows policy as the NF policy.

3. Normalizing flows Policy for On-Policy Optimization

3.1. Normalizing flows for control

We now construct a stochastic policy based on normalizing flows. By design, we require the source noise ϵ to have the same dimension as the action a. Recall that the normalizing flows distribution is implicitly defined by a sequence of invertible transformation (3). To define a proper policy $\pi(a|s)$, we first embed state s by another neural network $L_{\theta_s}(\cdot)$ with parameter θ_s and output a state vector $L_{\theta_s}(s)$ with the same dimension as ϵ. We can then insert the state vector between any two layers of (3) to make the distribution conditional on state s. In our implementation, we insert the state vector after the first transformation (for clarity of the presentation, we leave all details of the architectures of g_{θ_i} and L_{θ_s} in Appendix B).

$$a = g_{\theta_K} \circ g_{\theta_{K-1}} \circ ... \circ g_{\theta_2} \circ (L_{\theta_s}(s) + g_{\theta_1}(\epsilon)). \qquad (5)$$

Though the additive form of $L_{\theta_s}(s)$ and $g_{\theta_1}(\epsilon)$ may in theory limit the capacity of the model, in experiments below we show that the resulting policy is still very expressive. For simplicity, we denote the above transformation (5) as $a = f_\theta(s, \epsilon)$ with parameter $\theta = \{\theta_s, \theta_i, 1 \leq i \leq K\}$. It is obvious that the transformation $a = f_\theta(s, \epsilon)$ is still invertible between a and ϵ, which is critical for computing $\log \pi_\theta(a|s)$ according to the change of variables formula (4). This architecture builds complex action distributions with explicit probability density $\pi_\theta(\cdot|s)$, and hence entails training using score function gradient estimators.

Practical Considerations. In implementations, it is necessary to compute gradients of the entropy $\nabla_\theta \mathbb{H}[\pi_\theta(\cdot|s)]$, either for computing Hessian vector product for CG (Schulman et al., 2015) or for entropy regularization (Schulman et al., 2015; 2017a; Mnih et al., 2016). Unlike Gaussian, for normalizing flows there is no analytic form for entropy, instead we can use N samples to estimate entropy by re-parameterization, $\mathbb{H}[\pi_\theta(\cdot|s)] = \mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[-\log \pi_\theta(a|s)] = \mathbb{E}_{\epsilon\sim\rho_0(\cdot)}[-\log \pi_\theta(f_\theta(s, \epsilon)|s)] \approx \frac{1}{N}\sum_{i=1}^N -\log \pi_\theta(f_\theta(s, \epsilon_i)|s)$. The gradient of the entropy can be easily estimated with samples and implemented using back-propagation $\nabla_\theta \mathbb{H}[\pi_\theta(\cdot|s)] \approx \frac{1}{N}\sum_{i=1}^N -\nabla_\theta \log \pi_\theta(f_\theta(s, \epsilon_i)|s)$.

3.2. Motivating Expressive Policies for Control

We motivate the expressive policies for control from the framework of *Control as Variational Inference* (Levine,

2018). Here we briefly introduce the framework. Consider a MDP with horizon T and let $\tau = \{s_t, a_t\}_{t=0}^{T-1}$ denote a trajectory consisting of T state-action pairs. Let $r_t = r(s_t, a_t)$ be the instant reward and define an optimality variable O_t that follows a Bernoulli distribution $p(O_t = 1|s_t, a_t) = \exp(\frac{r_t}{c})$ for some $c > 0$ [1]. Given a prior distribution over trajectories $p(\tau)$, the inference objective is to compute $p(\tau|O_{0:T-1} = 1)$.

Now consider a variational distribution $q(\tau) = \Pi_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$ implicitly induced by the policy $\pi_\theta(a_t|s_t)$. Also assume that the prior $p(\tau)$ is a uniform distribution. The variational inference objective reduces to minimizing the KL-divergence (equivalently below we maximize the negative of the KL-divergence)

$$\max_\theta \; -\mathbb{KL}[q(\tau)||p(\tau|O_{0:T-1} = 1)]$$

$$\propto \max_\theta \; \mathbb{E}_{q(\tau)}\big[\sum_{t=0}^{T-1} r_t + c\mathbb{H}(\pi_\theta(\cdot|s_t))\big] \qquad (6)$$

where the $\propto$ results from scaling the RHS of (6) by a factor of c. In general, the RHS of (6) is the maximum entropy RL objective (Haarnoja et al., 2017; 2018b; Levine, 2018), which also reduces to the conventional RL objective $J(\pi_\theta)$ defined in Section 2 when $c \to 0$ [2].

In variational inference, improving the expressiveness of a variational distribution can improve its approximation to the usually intractable target distribution (Rezende & Mohamed, 2015). Similarly in (6), a more flexible variational distribution $q(\tau)$ should approximate $q(\tau|O_{0:T-1} = 1)$ better by finding a lower minimum of the KL-divergence, which in turn translates to a higher RL objective $J(\pi_\theta)$ in the RHS of (6). Since $q(\tau)$ is induced by $\pi_\theta(a_t|s_t)$, we could only improve the expressiveness of $q(\tau)$ by making $\pi_\theta(a_t|s_t)$ more flexible. This motivates using normalizing flows as policy distributions. To provide more intuitions as to why normalizing flows for control is beneficial to policy optimization, we construct illustrative examples in Appendix E where we show the capability of normalizing flows to capture correlation, multi-modality and potentially entail better exploration to bypass bad local optima.

4. Experiments

In experiments we aim to address the following questions: **(a)** Do NF policies outperform simple policies (e.g. factorized Gaussian baseline) as well as recent prior works (e.g. (Chou et al., 2017)) with trust region search algorithms on benchmark tasks, and especially on high-dimensional tasks

with complex dynamics? **(b)** How sensitive are NF policies to hyper-parameters compared to Gaussian policies?

To address **(a)**, we carry out comparison in three parts: (1) We implement and compare NF policy with Gaussian mixtures model (GMM) policy and factorized Gaussian policy, on a comprehensive suite of tasks in OpenAI gym MuJoCo (Brockman et al., 2016; Todorov, 2008), rllab (Duan et al., 2016), Roboschool Humanoid (Schulman et al., 2017b) and Box2D (Brockman et al., 2016) illustrated in Appendix F; (2) We implement and compare NF with straightforward architectural alternatives and prior work (Chou et al., 2017) on tasks with complex dynamics; (3) Lastly but importantly, we compare NF with Gaussian policy results directly reported in prior works (Schulman et al., 2017b; Wu et al.; Chou et al., 2017; Haarnoja et al., 2018b). For (1) we show results for both TRPO and ACKTR (see Appendix C), and for (2)(3) only for TRPO (see Appendix C). Here, we note that (3) is critical because the performance of the same algorithm across papers can be very different (Henderson et al., 2017), and here we aim to show that our proposed method achieves significant gains consistently over results reported in prior works. To address **(b)**, we perform an ablation study in Appendix C. Implementation details can be found in Appendix A.

TRPO - Comparison with Gaussian & GMM. In Figure 1 we show the results on benchmark control problems from MuJoCo and in Figure 4 we show the results on Box2D tasks. We compare four policy classes under TRPO: factorized Gaussian (blue curves), GMM with $K = 2$ (yellow curves), GMM with $K = 5$ (green curves) and normalizing flows (red curves). For GMM, each cluster has the same probability weight and each cluster is a factorized Gaussian with independent parameters. We find that though GMM policies with $K \geq 2$ outperform factorized Gaussian on relatively complex tasks such as Ant and HalfCheetah, they suffer from less stable learning in more high-dimensional Humanoid tasks. However, normalizing flows consistently outperforms GMM and factorized Gaussian policies on a wide range of tasks, especially tasks with highly complex dynamics such as Humanoid.

Since we make minimal changes to the algorithm and algorithmic hyper-parameters, we could attribute performance gains to the architectural difference. One critical question is whether the more effective policy optimization results from an increased number of parameters? To study the effect of network size, we train Gaussian policy with large networks (with more parameters than NF policy) and find that this does not lead to performance gains. We detail the results of Gaussian with bigger networks in Appendix C. Such comparison further validates our speculation that the performance gains result from an expressive policy distribution.

---

[1] Here we properly scale r_t to make the probability well defined.

[2] Here the discount factor should also be $\gamma = 1$. We can define an inference problem that incorporates the discount factor $\gamma < 1$, which is ommited here.
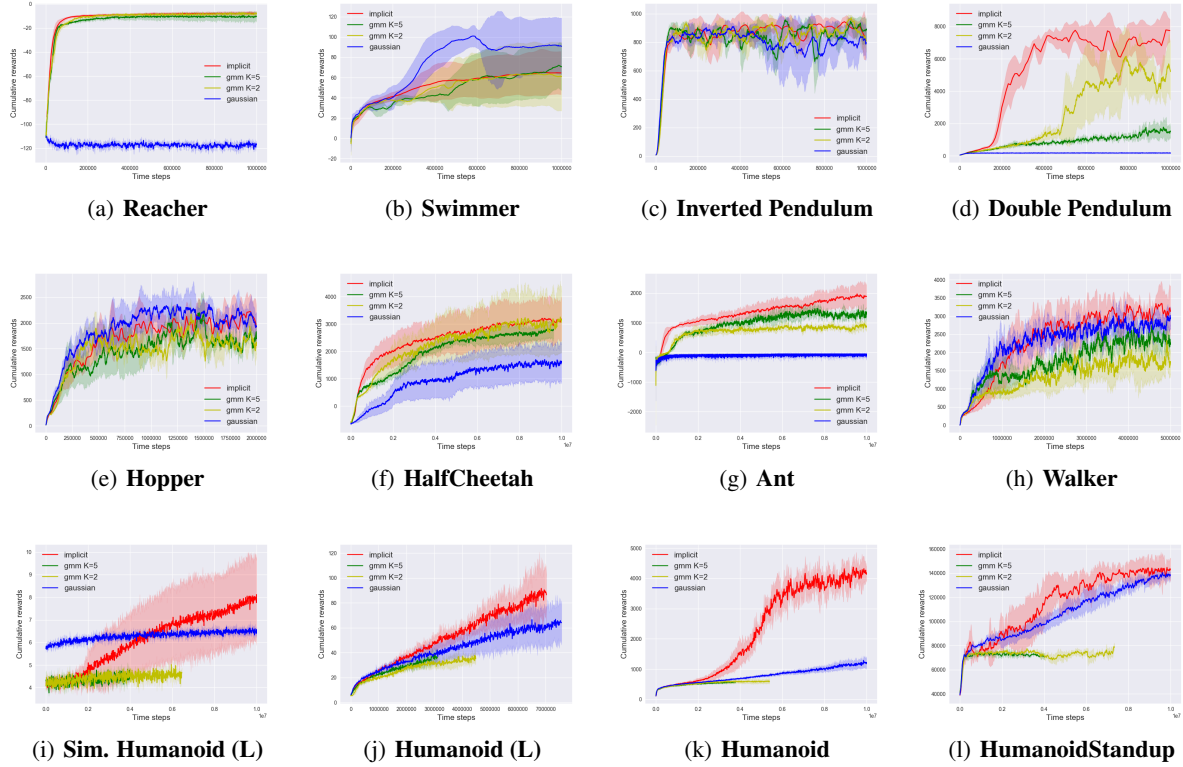
Figure 1. MuJoCo Benchmark: learning curves on MuJoCo locomotion tasks. Tasks with (L) are from rllab. Each curve corresponds to a different policy class (Red: NF (labelled as *implicit*), Green: GMM $K = 5$, Yellow: GMM $K = 2$, Blue: Gaussian). We observe that the NF policy consistently outperforms other baselines on high-dimensional complex tasks (Bottom two rows).

5. Related Work

Policy Classes. Several recent prior works have proposed to boost RL algorithms with expressive policy classes. Thus far, expressive policy classes have shown improvement over baselines in off-policy learning: Soft Q-learning (SQL) (Haarnoja et al., 2017) takes an implicit generative model as the policy and trains the policy by Stein variational gradients (Liu & Wang, 2016); Tang & Agrawal (2018) applies an implicit policy along with a discriminator to compute entropy regularized gradients. Latent space policy (Haarnoja et al., 2018a) applies normalizing flows as the policy and displays promising results on hierarchical tasks; Soft Actor Critic (SAC) (Haarnoja et al., 2018b) applies a mixture of Gaussian as the policy. However, aforementioned prior works do not disentangle the architectures from the algorithms, it is hence not clear whether the gains come from an expressive policy class or novel algorithmic procedures. In this work, we fix the trust region search algorithms and study the net effect of expressive policy classes. For on-policy optimization, Gaussian policy is the default baseline (Schulman et al., 2015; 2017a). Recently, (Chou et al., 2017) propose Beta distribution as an alternative to Gaussian and show improvements on TRPO. We make a full comparison and show that

expressive distributions achieves more consistent and stable gains than such bounded distributions for TRPO.

Normalizing flows. Normalizing flows is widely applied in probabilistic generative modeling and variational inference (Rezende & Mohamed, 2015; Dinh et al., 2014; Louizos & Welling, 2017; Dinh et al., 2016). Complement to prior works, we show that normalizing flows can significantly boost the performance of TRPO/ACKTR. We limit our attention to the architectures of (Dinh et al., 2016) while more recent flows structure might offer additional performance gains (Kingma & Dhariwal, 2018).

6. Conclusion

We propose normalizing flows as a novel on-policy architecture to boost the performance of trust region policy search. In particular, we observe that the empirical properties of the NF policy allows for better exploration in practice. We evaluate performance of NF policy combined with trust region algorithms (TRPO/ ACKTR) and show that they consistently and significantly outperform a large number of previous policy architectures.

References

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

Po-Wei Chou, Daniel Maturana, and Sebastian Scherer. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In *International Conference on Machine Learning*, pp. 834–843, 2017.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. https://github.com/openai/baselines, 2017.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. *arXiv preprint arXiv:1702.08165*, 2017.

Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018b.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances In Neural Information Processing Systems*, pp. 2378–2386, 2016.

Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961*, 2017.

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

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

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

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

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

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

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

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

Yuhuai Wu, Elman Mansimov, Roger B Gross, Shun Liao, and Jummy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *Advances in neural information processing systems*, pp. 5279–5288.

A. Hyper-parameters

All implementations of algorithms (TRPO and ACKTR) are based on OpenAI baselines (Dhariwal et al., 2017). We implement our own GMM policy and NF policy. Environments are based on OpenAI gym (Brockman et al., 2016), rllab (Duan et al., 2016) and Roboschool (Schulman et al., 2017b).

We remark that various policy classes have exactly the same interface to TRPO and ACKTR. In particular, TRPO and ACKTR only requires the computation of $\log \pi_\theta(a|s)$ (and its derivative). Different policy classes only differ in how they parameterize $\pi_\theta(a|s)$ and can be easily plugged into the algorithmic procedure originally designed for Gaussian (Dhariwal et al., 2017).

We present the details of each policy class as follows.

Factorized Gaussian Policy. A factorized Gaussian policy has the form $\pi_\theta(\cdot|s) = \mathbb{N}(\mu_\theta(s), \Sigma)$, where Σ is a diagonal matrix with $\Sigma_{ii} = \sigma_i^2$. We use the default hyper-parameters in baselines for factorized Gaussian policy. The mean $\mu_\theta(s)$ parameterized by a two-layer neural network with 64 hidden units per layer and tanh activation function. The standard deviation σ_i^2 is each a single variable shared across all states.

Factorized Gaussian+tanh Policy. The architecture is the same as above but the final layer is added a $\tanh$ transformation to ensure that the mean $\mu_\theta(s) \in [-1, 1]$.

GMM Policy. A GMM policy has the form $\pi_\theta(\cdot|s) = \sum_{i=1}^{K} p_i \mathbb{N}(\mu_\theta^{(i)}(s), \Sigma_i)$, where the cluster weight $p_i = \frac{1}{K}$ is fixed and $\mu_\theta^{(i)}(s), \Sigma_i$ are Gaussian parameters for the ith cluster. Each Gaussian has the same parameterization as the factorized Gaussian above.

Beta Policy. A Beta policy has the form $\pi(\alpha_\theta(s), \beta_\theta(s))$ where π is a Beta distribution with parameters $\alpha_\theta(s), \beta_\theta(s)$. Here, $\alpha_\theta(s)$ and $\beta_\theta(s)$ are shape/rate parameters parameterized by two-layer neural network $f_\theta(s)$ with a softplus at the end, i.e. $\alpha_\theta(s) = \log(\exp(f_\theta(s)) + 1) + 1$, following (Chou et al., 2017). Actions sampled from this distribution have a strictly finite support. We notice that this parameterization introduces potential instability during optimization: for example, when we want to converge on policies that sample actions at the boundary, we require $\alpha_\theta(s) \to \infty$ or $\beta_\theta(s) \to \infty$, which might be very unstable. We also observe such instability in practice: when the trust region size is large (e.g. $\epsilon = 0.01$) the training can easily terminate prematurely due to numerical errors. However, reducing the trust region size (e.g. $\epsilon = 0.001$) will stabilize the training but degrade the performance.

Normalizing flows Policy (NF Policy). A NF policy has a generative form: the sample $a \sim \pi_\theta(\cdot|s)$ can be generated via $a = f_\theta(s, \epsilon)$ with $\epsilon \sim \rho_0(\cdot)$. The detailed architecture of $f_\theta(s, \epsilon)$ is in Appendix B.

Other Hyper-parameters. Value functions are all parameterized as two-layer neural networks with 64 hidden units per layer and tanh activation function. Trust region sizes are enforced via a constraint parameter ϵ, where $\epsilon \in \{0.01, 0.001\}$ for TRPO and $\epsilon \in \{0.02, 0.002\}$ for ACKTR. All other hyper-parameters are default parameters from the baselines implementations.

B. Normalizing flows Policy Architecture

We design the neural network architecture following the idea of (Dinh et al., 2014; 2016). Recall that normalizing flows (Rezende & Mohamed, 2015) consists of layers of transformation as follows ,

$$x = g_{\theta_K} \circ g_{\theta_{K-1}} \circ ... \circ g_{\theta_2} \circ g_{\theta_1}(\epsilon),$$

where each $g_{\theta_i}(\cdot)$ is an invertible transformation. We focus on how to design each atomic transformation $g_{\theta_i}(\cdot)$. We overload the notations and let x, y be the input/output of a generic layer $g_\theta(\cdot)$,

$$y = g_\theta(x).$$

We design a generic transformation $g_\theta(\cdot)$ as follows. Let x_I be the components of x corresponding to subset indices $I \subset \{1, 2...m\}$. Then we propose as in (Dinh et al., 2016),

$$
\begin{aligned}
y_{1:d} &= x_{1:d} \\
y_{d+1:m} &= x_{d+1:m} \odot \exp(s(x_{1:d})) + t(x_{1:d}),
\end{aligned}
\tag{7}
$$

where $t(\cdot), s(\cdot)$ are two arbitrary functions $t, s : \mathbb{R}^d \mapsto \mathbb{R}^{m-d}$. It can be shown that such transformation entails a simple Jacobien matrix $|\frac{\partial y}{\partial x^T}| = \exp(\sum_{j=1}^{m-d} [s(x_{1:d})]_j)$ where $[s(x_{1:d})]_j$ refers to the jth component of $s(x_{1:d})$ for $1 \le j \le m - d$. For each layer, we can permute the input x before apply the simple transformation (7) so as to couple different components across layers. Such coupling entails a complex transformation when we stack multiple layers of (7). To define a policy, we need to incorporate state information. We propose to preprocess the state $s \in \mathbb{R}^n$ by a neural network $L_{\theta_s}(\cdot)$ with parameter θ_s, to get a state vector $L_{\theta_s}(s) \in \mathbb{R}^m$. Then combine the state vector into (7) as follows,

$$
\begin{aligned}
z_{1:d} &= x_{1:d} \\
z_{d+1:m} &= x_{d+1:m} \odot \exp(s(x_{1:d})) + t(x_{1:d}) \\
y &= z + L_{\theta_s}(s).
\end{aligned}
\tag{8}
$$

It is obvious that $x \leftrightarrow y$ is still bijective regardless of the form of $L_{\theta_s}(\cdot)$ and the Jacobien matrix is easy to compute accordingly.

In locomotion benchmark experiments, we implement s, t both as 4-layers neural networks with $l_1 = 3$ units per hidden layer. We stack $K = 4$ transformations: we implement (8) to inject state information only after the first transformation, and the rest is conventional coupling as in (7). $L_{\theta_s}(s)$ is implemented as a feedforward neural network with 2 hidden layers each with 64 hidden units. Value function critic is implemented as a feedforward neural network with 2 hidden layers each with 64 hidden units with rectified-linear between hidden layers.

C. Additional Experiments

C.1. Benchmark experiments

TRPO - Comparison with other Architecture Alternatives. In Table 1, we compare with some architectural alternatives and recently proposed policy classes: Gaussian distribution with tanh non-linearity at the output layer, and Beta distribution (Chou et al., 2017). The primary motivation for these architectures is that they either bound the Gaussian mean (Gaussian +tanh) or strictly bound the distribution support (Beta distribution), which has been claimed to remove the implicit bias of an unbounded action distribution (Chou et al., 2017). In Table 2, We compare the NF policy with these alternatives on benchmark tasks with complex dynamics. We find that the NF policy performs significantly better than other alternatives uniformly and consistently across all presented tasks.

Here we discuss the results for Beta distribution. In our implementation, we find training with Beta distribution tends to generate numerical errors when the trust region size is large (e.g. $\epsilon = 0.01$). Shrinking the trust region size (e.g. $\epsilon = 0.001$) reduces numerical errors but also greatly degrades the learning performance. We suspect that this is because the Beta distribution parameterization (Appendix A and (Chou et al., 2017)) is numerically unstable, and we discuss the potential causes in Appendix A. The results in Table 2 for Beta policy are obtained as the performance of the last 10 iterations before termination (potentially prematurely due to numerical error). We make further comparison with Beta policy in Appendix C and show that the NF policy achieves performance gains more consistently and stably.

Our findings suggest that for TRPO, an expressive policy brings more benefits than a policy with a bounded support. We speculate that this is because bounded distributions require warping the sampled actions in a way that makes the optimization more difficult. For example, consider 1-d action space when NF is combined with tanh nonlinearity for the final output [3]: the samples a are bounded but an extra factor $(1 - a^2)$ is introduced in the gradient w.r.t. θ. When $a \approx \pm 1$, the gradient can vanish quickly and make it hard to sample on the exact boundary $a = \pm 1$. In practice, we also find the performance of NF policy + tanh to be inferior.

---

[3] Similar to the Gaussian + tanh construct, we can bound the support of the NF policy by applying tanh at the output, the ending distribution still has tractable likelihood (Haarnoja et al., 2018b).

TRPO - Comparison with Gaussian on Roboschool Humanoid and Box2D. To further illustrate the strength of the NF policy on high-dimensional tasks, we evaluate normalizing flows vs. factorized Gaussian on Roboschool Humanoid tasks shown in Figure 3. We observe that ever since the early stage of learning (steps $\leq 10^7$) NF policy (red curves) already outperforms Gaussian (blue curves) by a large margin. In 3 (a)(b), Gaussian is stuck in a locally optimal gait and cannot progress, while the NF policy can bypass such gaits and makes consistent improvement. Surprisingly, BipedalWalker (Hardcore) Box2D task (Figure 4) is very hard for the Gaussian policy, while the NF policy can make consistent progress during training.

TRPO - Comparison with Gaussian results in prior works. Schulman et al. (2017b); Wu et al. report results on TRPO with Gaussian policy, represented as 2-layer neural network with 64 hidden units per layer. Since they report the performance after 10^6 steps of training, we record the performance of NF policy after 10^6 steps of training for fair comparison in Table 1. The results of (Schulman et al., 2017b; Wu et al.) in Table 1 are directly estimated from the plots in their paper, following the practice of (Mania et al., 2018). We see that our proposed method outperforms the Gaussian policy + TRPO reported in (Schulman et al., 2017b; Wu et al.) for most control tasks.

Some prior works (Haarnoja et al., 2018b) also report the results for Gaussian policy after training for more steps. We also offer such a comparison in Appendix C, where we show that the NF policy still achieves significantly better results on reported tasks.

Comprehensive comparison across results reported in prior works ensures that we compare with (approximate) state-of-the-art results. Despite implementation and hyper-parameter differences from various prior works, the NF policy can consistently achieve faster rate of learning and better asymptotic performance.

ACKTR - Comparison with Gaussian. We also evaluate different policy classes combined with ACKTR (Wu et al.). In Figure 2, we compare factorized Gaussian (red curves) against NF (blue curves) on a suite of MuJoCo and Roboschool control tasks. Though the NF policy does not uniformly outperform Gaussian on all tasks, we find that for tasks with relatively complex dynamics (e.g. Ant and Humanoid), the NF policy achieves significant performance gains. We find that the effect of an expressive policy class is fairly orthogonal to the additional training stability introduced by ACKTR over TRPO and the combined algorithm achieves even better performance.
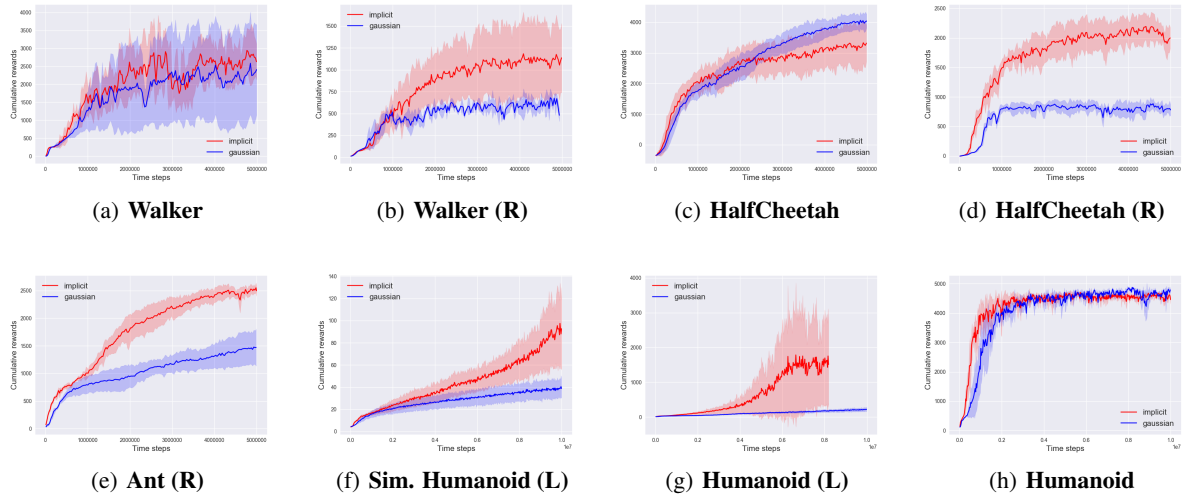

Figure 2. MuJoCo and Roboschool Benchmarks : learning curves on locomotion tasks for ACKTR. Each curve corresponds to a different policy class (Red: NF (labelled as *implicit*), Blue: Gaussian). Tasks with (R) are from Roboschool. The NF policy achieves consistent and significant performance gains across high-dimensional tasks with complex dynamics.

C.2. Sensitivity to Hyper-parameters and Ablation Study

We evaluate the policy classes' sensitivities to hyper-parameters in Appendix C, where we compare Gaussian vs. normalizing flows. Recall that ϵ is the constant for KL constraint. For each policy, we uniformly random sample $\log_{10} \epsilon \in [-3.0, -2.0]$

Table 1. Comparison with TRPO results reported in (Schulman et al., 2017b; Wu et al.). Since Schulman et al. (2017b); Wu et al. report the performance for 10^6 steps of training, we also record the performance of our method after training for 10^6 for fair comparison. Better results are highlighted in bold font. When results are not statistically different, both are highlighted.

TASK	NF (OURS)	GAUSSIAN (OURS)	SCHULMAN ET AL. 2017	WU ET AL. 2017
REACHER	$\approx -\mathbf{10}$	≈ -115	≈ -115	$\approx -\mathbf{10}$
SWIMMER	≈ 64	≈ 90	$\approx \mathbf{120}$	≈ 40
INVERTED PEND.	≈ 900	≈ 800	≈ 900	$\approx \mathbf{1000}$
DOUBLE PEND.	$\approx \mathbf{7800}$	< 1000	≈ 0	< 1000
HOPPER	$\approx \mathbf{2000}$	$\approx \mathbf{2000}$	$\approx \mathbf{2000}$	≈ 1400
HALFCHEETAH	$\approx \mathbf{1500}$	≈ 0	≈ 0	< 500
WALKER2D	≈ 1700	$\approx \mathbf{2000}$	≈ 1000	≈ 550
ANT	$\approx \mathbf{800}$	≈ 0	N/A	< -500

Table 2. Table 1: A comparison of various policy classes on complex benchmark tasks. For each task, we show the cumulative rewards (mean $\pm$ std) after training for 10^7 steps across 5 seeds (for Humanoid (L) it is $7 \cdot 10^6$ steps). For each task, the top two results are highlighted in bold font. The NF policy (labelled as *NF*) consistently achieves top two results.

	GAUSSIAN	GAUSSIAN+TANH	BETA	NF
ANT	-76 ± 14	-89 ± 13	$\mathbf{2362 \pm 305}$	$\mathbf{1982 \pm 407}$
HALFCHEETAH	1576 ± 782	386 ± 78	$\mathbf{1643 \pm 819}$	$\mathbf{2900 \pm 554}$
HUMANOID	1156 ± 153	$\mathbf{6350 \pm 486}$	3812 ± 1973	$\mathbf{4270 \pm 142}$
HUMANOID (L)	$\mathbf{64.7 \pm 7.6}$	38.2 ± 2.3	37.8 ± 3.4	$\mathbf{87.2 \pm 19.6}$
SIM. HUMANOID (L)	$\mathbf{6.5 \pm 0.2}$	4.4 ± 0.1	4.2 ± 0.2	$\mathbf{8.0 \pm 1.8}$
HUMANOID STANDUP	$\mathbf{137955 \pm 9238}$	133558 ± 9238	111497 ± 15031	$\mathbf{142568 \pm 9296}$

and one of five random seeds, and train policies with TRPO for a fixed number of time steps. The final performance (cumulative rewards) is recorded and Figure 5 in Appendix C shows the quantile plots of final rewards across multiple tasks. We observe that the NF policy is generally much more robust to such hyper-parameters, importantly to ϵ. We speculate that such additional robustness partially stems from the fact that for the NF policy, the KL constraint does not pose very stringent restriction on the sampled action space, which allows the system to efficiently explore even when ϵ is small.

We carry out a small ablation study that addresses how hyper-parameters inherent to normalizing flows can impact the results. Recall that the NF policy (Section 4) consists of K transformations, with the first transformation embedding the state s into a vector $L_{\theta_s}(s)$. Here we implement $L_{\theta_s}(s)$ as a two-layer neural networks with l_1 hidden units per layer. We evaluate on the policy performance as we vary $K \in \{2, 4, 6\}$ and $l_1 \in \{3, 5, 7\}$. We find that the performance of NF policies are fairly robust to such hyper-parameters (see Appendix C).

C.3. Sensitivity to Hyper-Parameters and Ablation Study

In Figure 6, we show the ablation study of the NF policy. We evaluate how the training curves change as we change the hyper-parameters of NF policy: number of transformation $K \in \{2, 4, 6\}$ and number of hidden units $l_1 \in \{3, 5, 7\}$ in the embedding function $L_{\theta_s}(\cdot)$. We find that the performance of the NF policy is fairly robust to changes in K and l_1. When K varies, l_1 is set to 3 by default. When l_1 varies, K is set to 4 by default.

C.4. Comparison with Gaussian Policy with Big Networks

In our implementation, the NF policy has more parameters than Gaussian policy. A natural question is whether the gains in policy optimization is due to a bigger network. To test this, we train Gaussian policy with large networks: 2-layer neural network with 128 hidden units per layer. In Table 3, we find that for Gaussian policy, bigger network does not perform as well as the smaller network (32 hidden units per layer). Since Gaussian policy with bigger network has more parameters than NF policy, this validates the claim that the performance gains of NF policy are not (only) due to increased parameters.
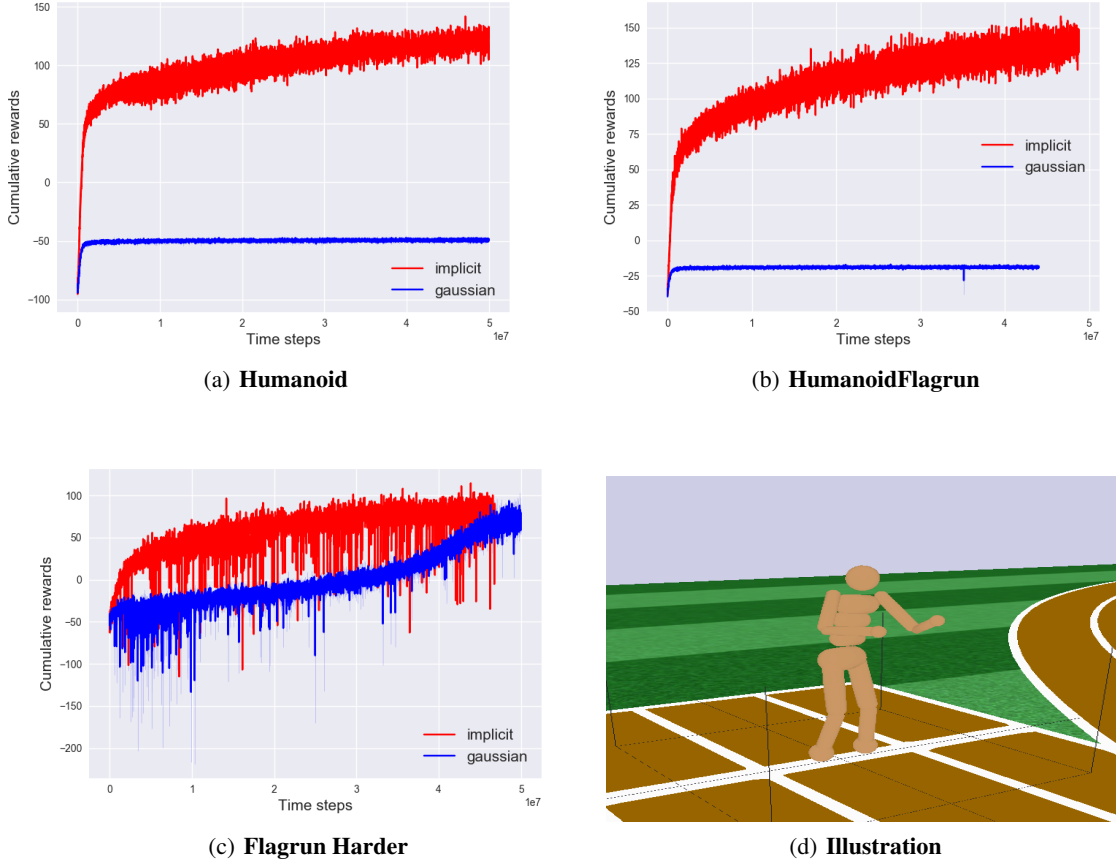
(a) **Humanoid**

(b) **HumanoidFlagrun**

(c) **Flagrun Harder**

(d) **Illustration**

Figure 3. Roboschool Humanoid benchmarks: (a)-(c) show learning curves on Roboschool Humanoid locomotion tasks. Each color corresponds to a different policy class (Red: NF (labelled as *implicit*), Blue: Gaussian). The NF policy significantly outperforms Gaussian since the early stage of training. (d) is an illustration of the Humanoid tasks.

C.5. Additional Comparison with Prior works

To ensure that we compare with state-of-the-art results of baseline TRPO, we also compare with prior works that report TRPO results.

In Table 4, we compare with Gaussian policy + TRPO reported in (Haarnoja et al., 2018b). In (Haarnoja et al., 2018b), policies are trained for sufficient steps before evaluation, and we estimate their final performance directly from plots in (Haarnoja et al., 2018b). For fair comparison, we record the performance of NF policy + TRPO. We find that NF policy significantly outperforms the results in (Haarnoja et al., 2018b) on most reported tasks.

Additional Comparison with Beta Distribution. Chou et al. (2017) show the performance gains of Beta policy for a limited number of benchmark tasks, most of which are relatively simple (with low dimensional observation space and action space). However, they show performance gains on Humanoid-v1. We compare the results of our Figure 1 with Figure 5(j) in (Chou et al., 2017) (assuming each training epoch takes ≈ 2000 steps): within 10M training steps, NF policy achieves faster progress, reaching ≈ 4000 at the end of training while Beta policy achieves ≈ 1000. According to (Chou et al., 2017), Beta distribution can have an asymptotically better performance with ≈ 6000, while we find that NF policy achieves asymptotically ≈ 5000.
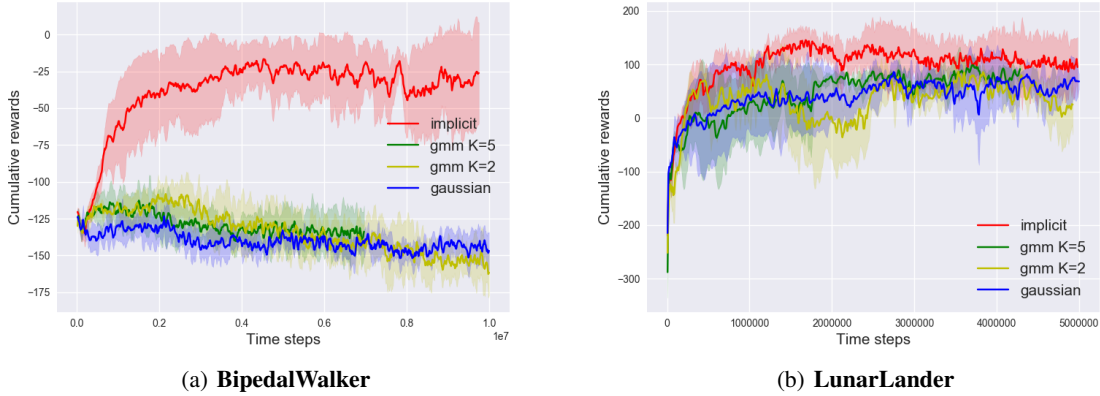
(a) **BipedalWalker** (b) **LunarLander**

Figure 4. Box2D benchmarks: (a)-(b) show learning curves on Box2D locomotion tasks. Each curve corresponds to a different policy class (Red: NF (labelled as *implicit*), Blue: Gaussian). The NF policy also achieves performance gains on Box2D environments.
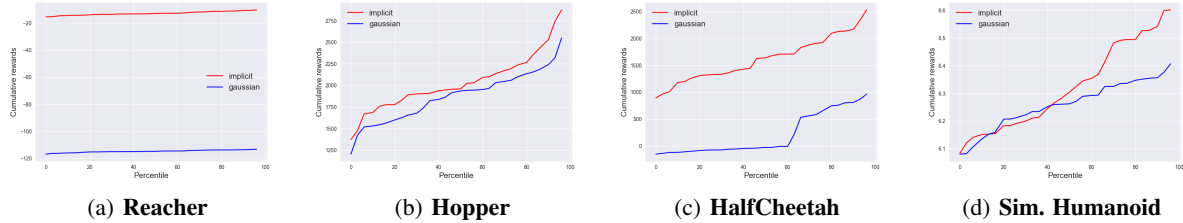

(a) **Reacher** (b) **Hopper** (c) **HalfCheetah** (d) **Sim. Humanoid**

Figure 5. Sensitivity to Hyper-parameters: quantile plots of policies' performance on MuJoCo benchmark tasks under various hyper-parameter settings. For each plot, we randomly generate 30 hyper-parameters for the policy and train for a fixed number of time steps. Reacher for 10^6 steps, Hopper and HalfCheetah for $2 \cdot 10^6$ steps and SimpleHumanoid for $\approx 5 \cdot 10^6$ steps. The NF policy is in general more robust than Gaussian policy.

D. Reward Structure of Ant Locomotion Task

For Ant locomotion task (Brockman et al., 2016), the state space $\mathcal{S} \subset \mathbb{R}^{116}$ and action space $\mathcal{A} \subset \mathbb{R}^8$. The state space consists of all the joint positions and joint velocities of the Ant robot, while the action space consists of the torques applied to joints. The reward function at time t is $r_t \propto v_x$ where v_x is the center-of-mass velocity of along the x-axis. In practice the reward function also includes terms that discourage large torques and encourages the Ant robot to stay alive (as defined by not flipping itself over).

Intuitively, the reward function encourages the robot to move along x-axis as fast as possible. This is reflected in Figure 8 (c) as the trajectories (red) generated by the NF policy is spreading along the x-axis. Occasionally, the robot also moves in the opposite direction.

E. Understanding Normalizing flows Policy for Control

In the following we illustrate the empirical properties of NF policy on toy examples.

Generating Samples under KL Constraints. We analyze the properties of NF policy vs. Gaussian policy under the KL constraints of TRPO. As a toy example, assume we have a factorized Gaussian in $\mathbb{R}^2$ with zero mean and diagonal covariance $\mathbb{I} \cdot \sigma^2$ where $\sigma^2 = 0.1^2$. Let $\hat{\pi}_o$ be the empirical distribution formed by samples drawn from this Gaussian. We can define a KL ball centered on $\hat{\pi}_o$ as all distributions such that a KL constraint is satisfied $\mathcal{B}(\hat{\pi}_o, \epsilon) = \{\pi : \mathbb{KL}[\hat{\pi}_o || \pi] \leq \epsilon\}$. We study a typical normalizing flows distribution and a factorized Gaussian distribution on the boundary of such a KL ball (such that $\mathbb{KL}[\hat{\pi}_o || \pi] = \epsilon$). We obtain such distributions by randomly initializing the distribution parameters and then running
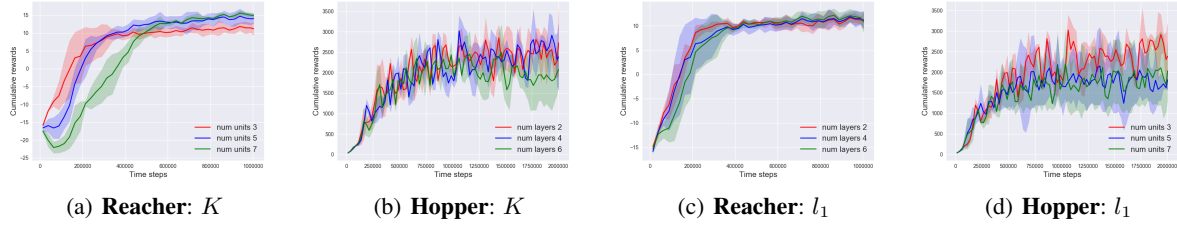
| (a) **Reacher**: K | (b) **Hopper**: K | (c) **Reacher**: l_1 | (d) **Hopper**: l_1 |

Figure 6. Sensitivity to normalizing flows Hyper-parameters: training curves of the NF policy under different hyper-parameter settings (number of hidden units l_1 and number of transformation K, on Reacher and Hopper task. Each training curve is averaged over 5 random seeds and we show the mean $\pm$ std performance. Vertical axis is the cumulative rewards and horizontal axis is the number of time steps.)

Table 3. Comparison of Gaussian policy with networks of different sizes. Big network has 128 hidden units per layer while small network has 32 hidden units per layer. Both networks have 2 layers. Small networks generally performs better than big networks. Below we show averageg $\pm$ std cumulative rewards after training for $5 \cdot 10^6$ steps.

TASK	GAUSSIAN (BIG)	GAUSSIAN (SMALL)
ANT	-104 ± 30	-94 ± 44
SIM. HUMAN. (L)	5.1 ± 0.7	6.4 ± 0.4
HUMANOID	501 ± 14	708 ± 43
HUMANOID (L)	20 ± 2	53 ± 9

gradient updates until $\mathbb{KL}[\hat{\pi}_o || \pi] \approx \epsilon$. In Figure 7 (a) we show the log probability contour of such a factorized Gaussian vs. normalizing flows, and in (b) we show their samples (blue are samples from the distributions on the boundary of the KL ball and red are empirical samples of $\hat{\pi}_o$). As seen from both plots, though both distributions satisfy the KL constraints, normalizing flows distribution has much larger variance than the factorized Gaussian, which also leads to a much larger effective support.

Such sample space properties have practical implications. For a factorized Gaussian distribution, enforcing KL constraints does not allow the new distribution to generate samples that are too far from the 'center' of the old distribution. On the other hand, for a normalizing flows distribution, the KL constraint does not hinder the new distribution to have a very distinct support from the reference distribution (or the old policy iterate), hence allowing for more efficient exploration that bypasses local optimal in practice.

Expressiveness of Normalizing flows Policy. We illustrate two potential strengths of the NF policy: learning correlated actions and learning multi-modal policy. First consider a 2D bandit problem where the action $a \in [-1, 1]^2$ and $r(a) = -a^T \Sigma^{-1} a$ for some positive semidefinite matrix Σ. In the context of conventional RL objective $J(\pi)$, the optimal policy is deterministic $\pi^* = [0, 0]^T$. However, in maximum entropy RL (Haarnoja et al., 2017; 2018b) where the objective is $J(\pi) + c\mathbb{H}[\pi]$, the optimal policy is $\pi^*_{\text{ent}} \propto \exp(\frac{r(a)}{c})$, a Gaussian with $\frac{\Sigma}{c}$ as the covariance matrix (red curves show the density contours). In Figure 8 (a), we show the samples generated by various trained policies to see whether they manage to learn the correlations between actions in the maximum entropy policy π^*_{ent}. We find that the factorized Gaussian policy cannot capture the correlations due to the factorization. Though Gaussian mixtures models (GMM) with $K \geq 2$ components are more expressive than factorized Gaussian, all the modes tend to collapse to the same location and suffer the same issue as factorized Gaussian. On the other hand, the NF policy is much more flexible and can fairly accurately capture the correlation structure of π^*_{ent}.

To illustrate multi-modality, consider again a 2D bandit problem (Figure 8 (b)) with reward $r(a) = \max_{i \in I} \{(a - \mu_i)^T \Lambda_i^- (a - \mu_i)\}$ where $\Lambda_i, i \in I$ are diagonal matrices and $\mu_i, i \in I$ are modes of the reward landscape. In our example we set $|I| = 2$ two modes and the reward contours are plotted as red curves. Notice that GMM with varying K can still easily collapse to one of the two modes while the NF policy generates samples that cover both modes.

To summarize the above two cases, since the maximum entropy objective $J(\pi) + c\mathbb{H}[\pi] = -\mathbb{KL}[\pi || \pi^*_{\text{ent}}]$, the policy search problem is equivalent to a variational inference problem where the variational distribution is π and the target distribution

Table 4. Comparison with TRPO results reported in (Haarnoja et al., 2018b). Haarnoja et al. (2018b) train on different tasks for different time steps, we compare the performance of our method with their results with the same training steps. Better results are highlighted in bold font. When results are not statistically different, both are highlighted.

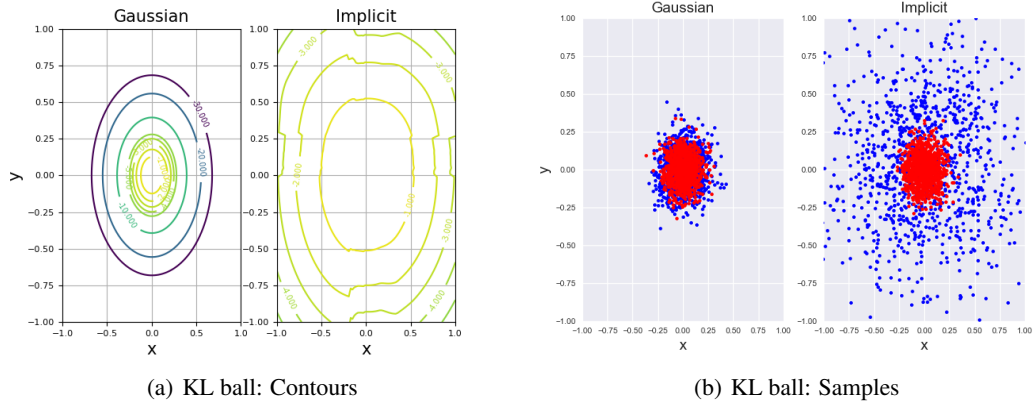| TASK	NF (OURS)	GAUSSIAN (OURS)	HARRNOJA ET AL. 2018
HOPPER (2M)	$\approx$ **2000**	$\approx$ **2000**	≈ 1300
HALFCHEETAH (10M)	$\approx$ **2900**	≈ 1600	≈ 1500
WALKER2D (5M)	$\approx$ **3200**	$\approx$ **3000**	≈ 800
ANT (10M)	$\approx$ **2000**	≈ 0	≈ 1250
HUMANOIDRLLAB (7M)	$\approx$ **90**	≈ 60	$\approx$ **90**

(a) KL ball: Contours

(b) KL ball: Samples

Figure 7. Analyzing normalizing flows vs. Gaussian: Consider a 2D Gaussian distribution with zero mean and factorized variance $\sigma^2 = 0.1^2$. Samples from the Gaussian form an empirical distribution $\hat{\pi}_o$ (red dots in (b)) and define the KL ball $\mathcal{B}(\hat{\pi}_o, \epsilon) = \{\pi : \mathbb{KL}[\hat{\pi}_o || \pi] \leq \epsilon\}$ centered at $\hat{\pi}_o$. We then find a NF distribution and a Gaussian distribution at the boundary of $\mathcal{B}(\hat{\pi}, 0.01)$ such that the constraint is tight. (a) Contour of log probability of a normalizing flows distribution (right) vs. Gaussian distribution (left); (b) Samples (blue dots) generated from NF distribution (right) and Gaussian distribution (left).

is $\pi_{\text{ent}}^* \propto \exp(\frac{r(a)}{c})$. Since NF policy is a more expressive class of distribution than GMM and factorized Gaussian, we also expect the approximation to the target distribution to be much better (Rezende & Mohamed, 2015). This *control as inference* perspective provides partial justification as to why an expressive policy such as NF policy achieves better practical performance.

The properties of NF policy illustrated above potentially allow for better exploration during training and help bypass bad local optima. For a more realistic example, we illustrate such benefits with the locomotion task of Ant robot (Figure 8 (d)) (Brockman et al., 2016). In Figure 8 (c) we show the robot's 2D center-of-mass trajectories generated by NF policy (red) vs. Gaussian policy (blue) after training for $2 \cdot 10^6$ time steps. We observe that the trajectories by NF policy are much more widespread, while trajectories of Gaussian policy are quite concentrated at the initial position (the origin $[0.0, 0.0]$). Behaviorally, the Gaussian policy gets the robot to jump forward quickly, which achieves high immediate rewards but terminates the episode prematurely (due to a termination condition of the task). On the other hand, the NF policy bypasses such locally optimal behavior by getting the robot to move forward in a fairly slow but steady manner, even occasionally move in the opposite direction to the high reward region (Task details in Appendix D).
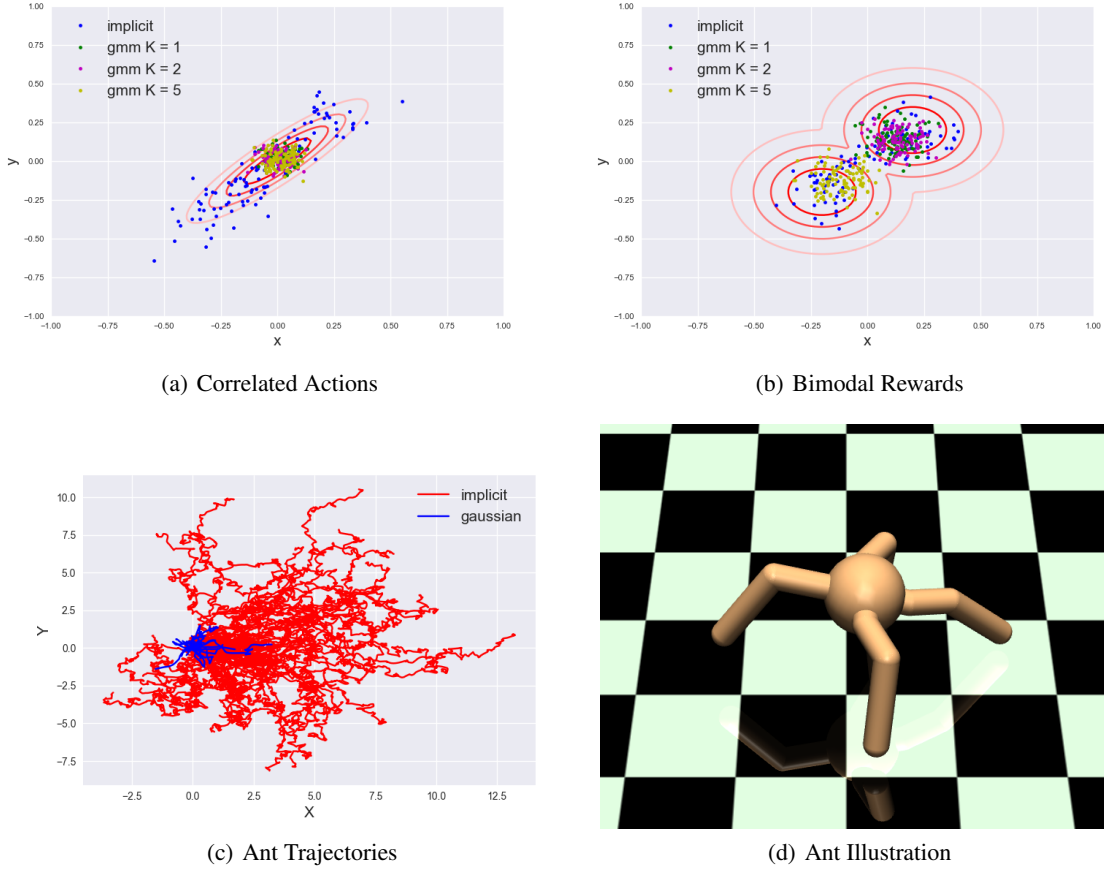
F. Illustration of Locomotion tasks

(a) Correlated Actions

(b) Bimodal Rewards

(c) Ant Trajectories

(d) Ant Illustration

Figure 8. Expressiveness of NF policy: (a) Bandit problem with reward $r(a) = -a^T \Sigma^{-1} a$. The maximum entropy optimal policy is a Gaussian distribution with Σ as its covariance (red contours). The NF policy (blue) can capture such covariance while Gaussian cannot (green). (b) Bandit problem with multi-modal reward (red contours the reward landscape). normalizing flows policy can capture the multi-modality (blue) while Gaussian cannot (green). (c) Trajectories of Ant robot. The trajectories of Gaussian policy center at the initial position (the origin $[0.0, 0.0]$), while trajectories of NF policy are much more widespread. (d) Illustration of the Ant locomotion task.
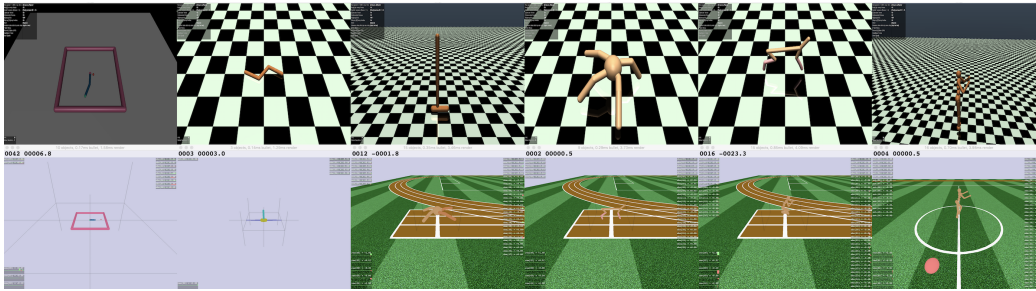

Figure 9. Illustration of benchmark tasks in OpenAI MuJoCo (Brockman et al., 2016; Todorov, 2008), rllab (top row) (Duan et al., 2016) and Roboschool (bottom row) (Schulman et al., 2017b).