Sequential Autoregressive Flow-Based Policies

Alex Guerra [1] **Joseph Marino** [1]

Abstract

Policy-based approaches to reinforcement learning are conventionally formulated as state-action mappings, however, this direct setup may be limiting in some contexts. We present a novel policy formulation composed of two components. A low-level dynamical component, parameterized by an autoregressive flow, models temporal dependencies between actions, and a high-level component, parameterized by a deep network, maps the current state to a base distribution for the flow. We provide preliminary experiments characterizing and evaluating this class of policies on MuJoCo continuous control tasks.[1]

1. Introduction

Interacting with complex environments to achieve goals requires learning sophisticated behavioral *policies*. In deep reinforcement learning (RL) for continuous control, these policies are parameterized by deep neural networks, mapping observed states to distributions over actions (Schulman et al., 2015; Lillicrap et al., 2015). In some cases, such policies may also contain latent variables, providing added flexibility (Haarnoja et al., 2018a; Tirumala et al., 2019). Indeed, in animal nervous systems, motor control is organized hierarchically (Merel et al., 2019). One key component of these systems are so-called central pattern generator circuits (Marder & Bucher, 2001), low-level dynamical primitives, which receive transient top-down signals from higher-level motor areas (Shalit et al., 2012).

In this work, we emulate and investigate a similar processing structure, separating control policies into a low-level dynamical component, parameterized by an autoregressive normalizing flow across time steps (Marino et al., 2020), and a higher-level component, parameterized as a standard policy network. With this setup, the autoregressive flow can learn a basis of dynamical behaviors, hopefully simplifying learning for the policy network. We empirically characterize these policies and investigate this approach on the MuJoCo (Todorov et al., 2012) continuous control environments from OpenAI gym (Brockman et al., 2016).

2. Background

2.1. Preliminaries

We consider a fully-observable Markov decision process (MDP) defined by $(\mathcal{S}, \mathcal{A}, p_{\text{env}}, r, \gamma)$. At time t, an agent receives a state observation $\mathbf{s}_t \in \mathcal{S}$ and takes action $\mathbf{a}_t \in \mathcal{A}$ by sampling from a policy distribution, π. The agent then receives reward $r(\mathbf{s}_t, \mathbf{a}_t)$ and the environment transitions to the next state $\mathbf{s}_{t+1} \sim p_{\text{env}}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. With a discount factor $\gamma \in [0, 1)$, the objective is to find a policy that maximizes the expected discounted sum of rewards, $\mathbb{E}_{p_{\text{env}},\pi}\left[\sum_t \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right]$.

2.2. Soft Actor Critic

We investigate our approach using soft actor critic (SAC) (Haarnoja et al., 2018b), a state-of-the-art model-free algorithm. SAC is formulated in the maximum entropy RL framework (Ziebart, 2010; Levine, 2018), learning a policy (actor) to maximize a parameterized Q-network (critic) (Mnih et al., 2015). This is performed using the entropy-augmented objective,

$$\mathcal{J}(\pi) = \mathbb{E}_{p_{\text{env}},\pi}\left[\sum_{t=1}^{T} \gamma^t (r(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log \pi(\mathbf{a}_t|\mathbf{s}_t))\right], \tag{1}$$

where α is a Lagrange multiplier controlling the entropy weight. Rather than attempting to optimize this objective directly, SAC approximates the expected future terms conditioned on the current action, i.e. the action-value, defined recursively as $Q_\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{p_{\text{env}},\pi}[Q_\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \alpha \log \pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})]$. The approximator, Q_π^ψ, is learned using temporal difference learning, with an ensemble of deep networks (Fujimoto et al., 2018), target value networks (Mnih et al., 2015), and an experience replay buffer (Lin, 1992).

SAC estimates the policy using a deep network, denoted π_ϕ,

[*]Equal contribution [1]California Institute of Technology (Caltech), CA, USA. Correspondence to: Joseph Marino <jmarino@caltech.edu>.

Second workshop on *Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models* (ICML 2020), Virtual Conference

[1]Code is available at https://github.com/aguerra7002/ARSAC.

which typically takes the form of a conditional Gaussian: $\pi_\phi(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(\mathbf{a}_t; \boldsymbol{\mu}_\phi(\mathbf{s}_t), \text{diag}(\boldsymbol{\sigma}^2_\phi(\mathbf{s}_t)))$. In MuJoCo's bounded action space, `tanh` is typically appplied to $\mathbf{a}_t$ (Haarnoja et al., 2018b). The policy parameters, ϕ, are optimized by backpropagating through the Q-network (and entropy) using the reparameterization estimator (Kingma & Welling, 2014; Rezende et al., 2014).

2.3. Normalizing Flows

Normalizing flows (Rippel & Adams, 2013; Rezende & Mohamed, 2015; Dinh et al., 2015) transform a *base distribution* into a more complex distribution using invertible transforms. Denoting the base variable as $\mathbf{u}$ and the transformed variable as $\mathbf{a}$, we can evaluate the transformed distribution using the change of variables formula:

$$p_\theta(\mathbf{a}) = p_\theta(\mathbf{u}) \left| \frac{\partial \mathbf{a}}{\partial \mathbf{u}} \right|^{-1}. \tag{2}$$

Many normalizing flows employ affine transforms (Dinh et al., 2017), i.e. $\mathbf{a} = \boldsymbol{\beta}_\theta \odot \mathbf{u} + \boldsymbol{\delta}_\theta$, where $\boldsymbol{\beta}_\theta$ and $\boldsymbol{\delta}_\theta$ are parameterized using deep networks and $\odot$ denotes element-wise multiplication. Affine autoregressive flows (Kingma et al., 2016; Papamakarios et al., 2017) are a class of affine flow that autoregressively transforms each dimension, e.g.

$$a_j = \beta_\theta(a_{<j}) \cdot u_j + \delta_\theta(a_{<j}). \tag{3}$$

Here, a_j denotes the j^{th} dimension of $\mathbf{a}$, which is a function of $a_{<j}$ and u_j. Previous works have applied autoregressive flows to variational inference (Kingma et al., 2016), generative modeling (Papamakarios et al., 2017; Huang et al., 2017), and model distillation (van den Oord et al., 2018). Recently, Marino et al. (2020) applied autoregressive flows across time steps to perform temporal normalization in sequential latent variable models. We adapt this approach to reinforcement learning policies.

Recent works have also investigated normalizing flow-based policies. Tang & Agrawal (2018) and Ward et al. (2019) utilize normalizing flows to improve policy distributions *within* time steps. Likewise, Haarnoja et al. (2018a) demonstrate the benefits of using normalizing flows to parameterize hierarchical policies. We explore an orthogonal approach, utilizing normalizing flows *across* time steps.

3. Autoregressive Flow-Based Policies

3.1. Motivation

In fully-observable MDPs, policy-based approaches to RL conventionally consider policies parameterized as a direct mapping from the current state to a distribution over actions. However, policy optimization does not explicitly require the current state, as any dependence is already implicitly captured by $Q_\pi(\mathbf{s}, \mathbf{a})$. That is, the Q-value (and entropy) already define the optimal policy. In this sense, policy networks are a form of *amortized* optimization (Gershman & Goodman, 2014; Marino et al., 2019). With this perspective, we can consider other forms of amortization.

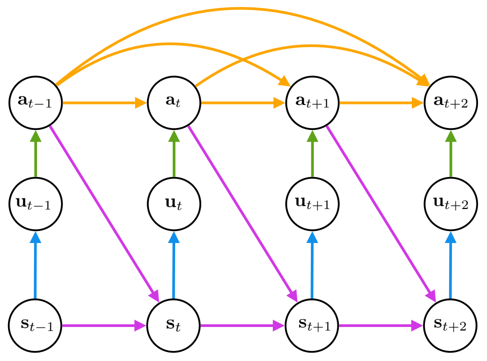


Figure 1. **Autoregressive Flow-Based Policy**. To interact with an environment (purple), p_{env}, an autoregressive flow-based policy converts a base distribution (blue), $\pi_\phi(\mathbf{u}_t|\mathbf{s}_t)$, using an affine transform (green), with affine parameters, $\boldsymbol{\beta}_\theta(\mathbf{a}_{<t})$ and $\boldsymbol{\delta}_\theta(\mathbf{a}_{<t})$, containing temporal dependencies (orange).

In this work, we consider policies that explicitly model temporal dependencies between actions. Thus, we condition $\mathbf{a}_t$ on $\mathbf{a}_{<t}$ in addition to $\mathbf{s}_t$. This setup can have two main benefits: temporal dependencies can **1)** provide a basis of dynamical motor primitives (Ijspeert et al., 2002), simplifying learning and control, and **2)** compensate if communication is limited, e.g. costly or delayed, between states and actions.

3.2. Formulation

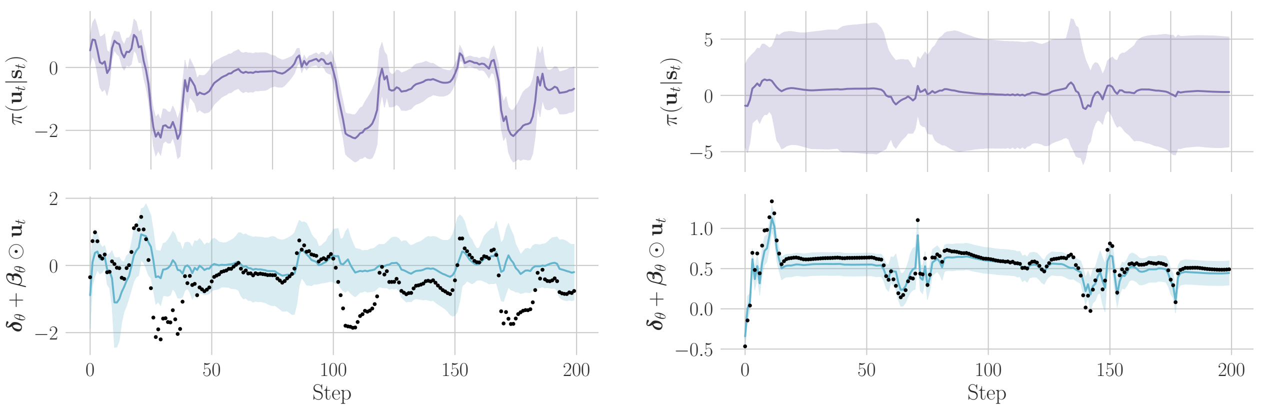
We consider policies composed of a state-dependent base distribution, $\pi_\phi(\mathbf{u}_t|\mathbf{s}_t)$, over a latent variable, $\mathbf{u}$, and a dynamical affine autoregressive component, defined by a shift, $\boldsymbol{\delta}_\theta(\mathbf{a}_{<t})$, and scale, $\boldsymbol{\beta}_\theta(\mathbf{a}_{<t})$. To generate action $\mathbf{a}_t$, we sample the latent variable, $\mathbf{u}_t \sim \pi_\theta(\mathbf{u}_t|\mathbf{s}_t)$, then apply the affine transform,

$$\mathbf{a}_t = \boldsymbol{\beta}_\theta(\mathbf{a}_{<t}) \odot \mathbf{u}_t + \boldsymbol{\delta}_\theta(\mathbf{a}_{<t}). \tag{4}$$

With the change of variables, the action probability is then

$$\pi_{\phi,\theta}(\mathbf{a}_t|\mathbf{s}_t, \mathbf{a}_{<t}) = \pi_\phi(\mathbf{u}_t|\mathbf{s}_t) \left| \prod_i \beta_{\theta,i}(\mathbf{a}_{<t}) \right|^{-1}, \tag{5}$$

where $\beta_{\theta,i}$ is the i^{th} dimension of $\boldsymbol{\beta}_\theta$. We can also apply a final `tanh` transform if the action space is bounded in $[-1, 1]$, as in Haarnoja et al. (2018b). In our implementation, the base distribution is a Gaussian, output by a deep network with parameters ϕ, and, similarly, the affine transform parameters are output by a deep network with parameters θ. The overall setup is shown in Figure 1.



(a) Action Dimension 6

(b) Action Dimension 2

Figure 2. **Flow-Based Policy Visualizations**. Plots visualize the base distribution (top) and the affine transform (bottom) for different dimensions of an affine autoregressive policy on `Walker2d-v2`. In the top plots, the solid line is the mean, and the shaded region is $\pm$ one standard deviation. In the bottom plot, the solid line is the shift, $\boldsymbol{\delta}_\theta$, and the shaded region is $\pm\boldsymbol{\beta}_\theta$. Black dots denote the actions, $\mathbf{a}_t$, after applying the affine transform. In (**a**), we see that the base distribution still contains substantial dynamical structure, whereas in (**b**), much of the dynamical structure is captured in the transform.

As noted by Kingma et al. (2016), if $\mathbf{u}_t \sim \mathcal{N}(\mathbf{u}_t; \mathbf{0}, \mathbf{I})$, then the affine flow is equivalent to an autoregressive Gaussian model, as Eq. 4 is the Gaussian reparameterization trick (Kingma & Welling, 2014; Rezende et al., 2014). For more general base distributions, this serves as a technique for adding dependencies to output sequences, $\mathbf{a}_{1:T}$. Conversely, given a sequence, $\mathbf{a}_{1:T}$, the affine transform provides a mechanism for removing dependencies, simplifying estimation in the space of $\mathbf{u}_{1:T}$ (Marino et al., 2020).

4. Experiments

We investigate sequential autoregressive flow-based policies on MuJoCo (Todorov et al., 2012) continuous control environments from OpenAI gym (Brockman et al., 2016). We are interested in determining 1) whether autoregressive flow-based policies capture temporal dependencies in actions, 2) how this changes during the course of training, and 3) whether this impacts task performance. To see how this is impacted by limitations on the base distribution policy network, we vary the number of hidden layers in $\{1, 2\}$ and units in $\{32, 128, 256\}$. In our experiments, autoregressive flows are conditioned on the previous 3 actions, and the shift and scale are calculated using a network with 2 layers of 256 units and ReLU non-linearities. We use the SAC hyperparameters from Haarnoja et al. (2018b) elsewhere. Section 4.1 qualitatively characterizes these policies and Section 4.2 compares performance metrics with baseline policies. Additional details are found in Appendix A and the accompanying code.

4.1. Qualitative Analyses

Visualizing Flow-Based Policies We visualize autoregressive flow-based policies in Figure 2, plotting subsequences of episodes in the `Walker2d-v2` environment. The base distribution (top) is shown alongside the affine transform (bottom), with black dots denoting the transformed actions, in this case, the base distribution mean. We see that in some action dimensions (Fig. 2a), the base distribution still contains substantial dynamical structure, modulating the actions at particular time steps. In other dimensions (Fig. 2b), however, the autoregressive flow captures much of the dynamical structure. In this latter case, the policy's base distribution is relatively constant throughout and effectively unused.

Affine Scale Magnitude The scale parameter of the affine transform, $\boldsymbol{\beta}_\theta$, effectively controls the weighting between the affine shift, $\boldsymbol{\delta}_\theta$, and the base distribution sample, $\mathbf{u}$. Thus, the scale parameter quantifies the degree to which the policy is relying on the autoregressive flow. In Figure 3, we plot the average log-scale throughout training on `HalfCheetah-v2` for various base distribution network architectures. Overall, we see an initial increase, through 150k steps, followed by a steady decrease. This signifies an initial reliance on the base distribution, followed by an increasing reliance on the autoregressive flow. Similar plots for other environments are shown in Figure 5 in the Appendix. Surprisingly, the reliance on the autoregressive flow is larger for larger base distribution network architectures. We are still interpreting this result.

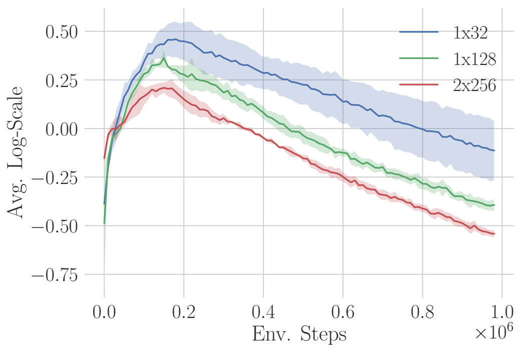


Figure 3. **Autoregressive Training**. The log-scale of the affine transform (β_θ) initially increases, then decreases during training, signifying more reliance on the autoregressive component. Each curve represents 4 seeds of a base policy architecture (number of layers × number of units per layer) on `HalfCheetah-v2`.

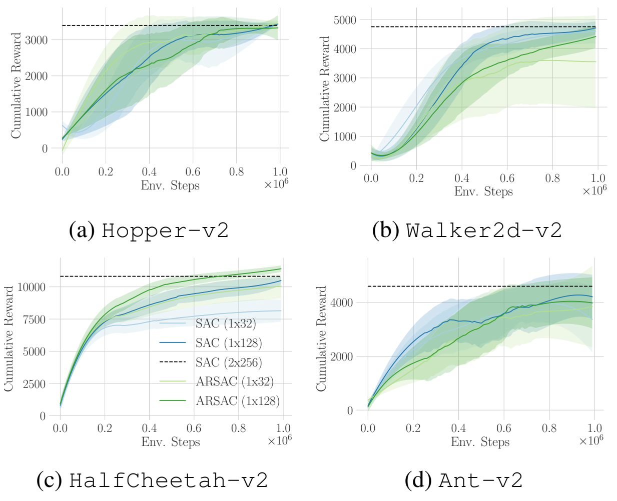


Figure 4. **Performance Curves**. Solid lines denote mean over 4 random seeds, and shaded regions denote one standard deviation. Each curve represents one policy network architecture, with (number of layers × number of units per layer). The dashed line is the average final performance of SAC with default policy network size (2 × 256).

4.2. Performance Comparison

In Figure 4, we plot the performance for each policy setup, varying the size of the base distribution network and whether or not the policy uses autoregressive flows. We refer to the autoregressive version of SAC as ARSAC. Each curve represents the mean and standard deviation of 4 random seeds. The dashed black line is the average final performance of SAC using the default (2×256) policy network architecture.

Overall, the results are mixed. On `Hopper-v2` and `Ant-v2`, autoregressive policies do not offer a clear benefit and are slightly worse on `Walker2d-v2`. On `HalfCheetah-v2`, we instead see a slight improvement for ARSAC over SAC, with larger base distribution networks (128 vs. 32) providing improved performance in both setups. Of particular interest, smaller policy networks ultimately achieve the same performance as the larger default SAC policy on `Hopper-v2`, `Walker2d-v2`, and `Ant-v2`. In contrast, on `HalfCheetah-v2`, the small (1×32) policy network is not able to reach the same asymptotic performance as the default policy architecture. This suggests that many of the MuJoCo continuous control tasks are too simple to benefit from autoregressive policies, as significantly smaller policies, i.e. base distributions, are already capable of matching the full-size baseline. We discuss future experiments to tease apart these policy classes in the following section.

5. Discussion

We have presented a policy formulation based on combining sequential autoregressive flows with base distribution policy networks. This hybrid technique allows the autoregressive flow to model dynamical aspects of the policy, hopefully simplifying the task for the state-action policy network. In our preliminary investigation, we observed that autoregressive flows do capture some policy dynamics, with the reliance on the flow increasing throughout training. However, we did not observe a clear performance improvement across environments. We largely attribute this to the simplicity of the MuJoCo tasks, which have deterministic Markov state dynamics and, rather surprisingly, can be solved with relatively small policy networks (1 hidden layer with 32 units). As noted above, the only environment where we observed a significant benefit from autoregressive policies, `HalfCheetah-v2`, is the same environment where smaller policy networks did not match the asymptotic performance of the larger default SAC policy architecture. Thus, while the autoregressive flows are utilized in all environments, they do not offer a benefit above the already-capable base distribution in most of the environments. Our formulation does not explicitly require that the policy rely on the autoregressive flow; it can be ignored ($\boldsymbol{\delta}_\theta = \mathbf{0}, \boldsymbol{\beta}_\theta = \mathbf{1}$) if it is unnecessary. In future experiments, we plan to investigate autoregressive flow-based policies in settings where they are more essential, i.e. more limited base policy networks and more complex environments. For instance, in partially observable environments, where estimating internal state dynamics is essential, autoregressive policies may simplify this task. Other possible directions include more expressive, non-affine forms of normalizing flows (Durkan et al., 2019), as well as multi-level flows.

References

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

Dinh, L., Krueger, D., and Bengio, Y. Nice: Non-linear independent components estimation. In *International Conference on Learning Representations*, 2015.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. In *International Conference on Learning Representations*, 2017.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural spline flows. *arXiv preprint arXiv:1906.04032*, 2019.

Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1587–1596, 2018.

Gershman, S. and Goodman, N. Amortized inference in probabilistic reasoning. In *Proceedings of the Cognitive Science Society*, volume 36, 2014.

Haarnoja, T., Hartikainen, K., Abbeel, P., and Levine, S. Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pp. 1846–1855, 2018a.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018b.

Huang, C.-W., Touati, A., Dinh, L., Drozdzal, M., Havaei, M., Charlin, L., and Courville, A. Learnable explicit density for continuous latent space and variational inference. *arXiv preprint arXiv:1710.02248*, 2017.

Ijspeert, A. J., Nakanishi, J., and Schaal, S. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 2, pp. 1398–1403. IEEE, 2002.

Kingma, D. P. and Welling, M. Stochastic gradient vb and the variational auto-encoder. In *Proceedings of the International Conference on Learning Representations*, 2014.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pp. 4743–4751, 2016.

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

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

Lin, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Marder, E. and Bucher, D. Central pattern generators and the control of rhythmic movements. *Current biology*, 2001.

Marino, J., Piché, A., and Yue, Y. On the design of variational rl algorithms. In *NeurIPS Workshop on Deep Reinforcement Learning*, 2019.

Marino, J., Chen, L., He, J., and Mandt, S. Improving sequential latent variable models with autoregressive flows. In *Symposium on Advances in Approximate Bayesian Inference*, pp. 1–16, 2020.

Merel, J., Botvinick, M., and Wayne, G. Hierarchical motor control in mammals and machines. *Nature Communications*, 10(1):1–12, 2019.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pp. 2338–2347, 2017.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pp. 1530–1538, 2015.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the International Conference on Machine Learning*, pp. 1278–1286, 2014.

Rippel, O. and Adams, R. P. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013.

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

Shalit, U., Zinger, N., Joshua, M., and Prut, Y. Descending systems translate transient cortical commands into a sustained muscle activation signal. *Cerebral cortex*, 22(8): 1904–1914, 2012.

Tang, Y. and Agrawal, S. Boosting trust region policy optimization by normalizing flows policy. *arXiv preprint arXiv:1809.10326*, 2018.

Tirumala, D., Noh, H., Galashov, A., Hasenclever, L., Ahuja, A., Wayne, G., Pascanu, R., Teh, Y. W., and Heess, N. Exploiting hierarchy for learning and transfer in kl-regularized rl. *arXiv preprint arXiv:1903.07438*, 2019.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

van den Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Driessche, G., Lockhart, E., Cobo, L., Stimberg, F., et al. Parallel wavenet: Fast high-fidelity speech synthesis. In *International Conference on Machine Learning*, pp. 3915–3923, 2018.

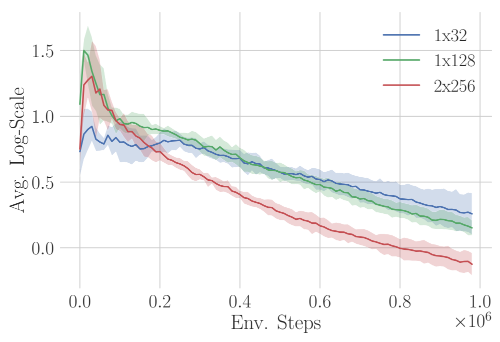
Ward, P. N., Smofsky, A., and Bose, A. J. Improving exploration in soft-actor-critic with normalizing flows policies. *ICML Workshop on Invertible Neural Nets and Normalizing Flows*, 2019.

Ziebart, B. D. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, CMU, 2010.

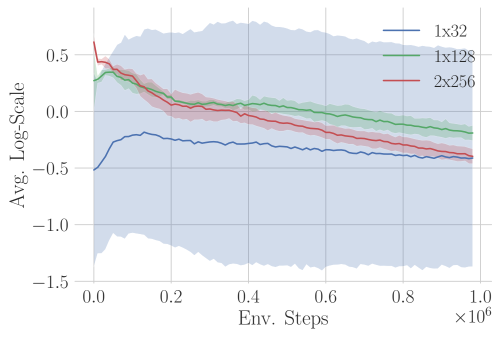
A. Experiment Details

Table 1. **Hyperparameters.**

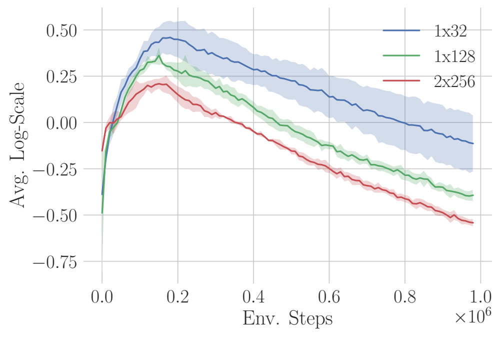
Hyperparameter	Value
Discount Factor (γ)	0.99
Entropy Weight (α)	0.2
Q-network Update Rate (τ)	$5 \cdot 10^{-3}$
Learning Rate	$3 \cdot 10^{-4}$
Batch Size	256
Initial Random Steps	10^4
Replay Buffer Size	10^6
Q-network Layers	2
Q-network Units/Layer	256
AR-network Layers	2
AR-network Units/Layer	256
Non-linearity (All Networks)	`ReLU`



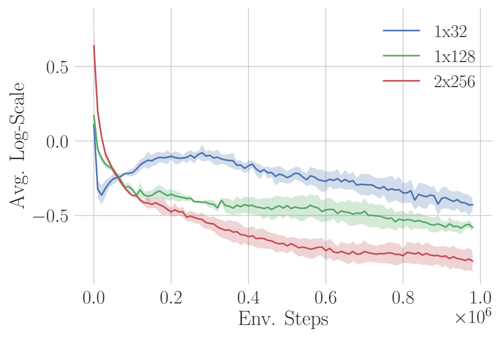
(a) `Hopper-v2`



(b) `Walker2d-v2`



(c) `HalfCheetah-v2`



(d) `Ant-v2`

Figure 5. **Affine Scale During Training for all Environments**.