2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

NFPDE: Normalizing Flow-based Parameter Distribution Estimation for Offline Adaptive Domain Randomization

Rin Takano [1], Kei Takaya [1], and Hiroyuki Oyama [1]

***Abstract*— Reinforcement learning with domain randomization (DR) has been proposed as a promising approach for learning robust policies to environmental changes. However, for DR to work well in real-world environments, it is necessary to design appropriate DR distributions for model parameters. This paper proposes Normalizing Flow-based Parameter Distribution Estimation (NFPDE), a new estimation method for DR distributions. NFPDE models the target distribution by a flow-based generative model using normalizing flow and estimates the target distribution based on an offline dataset collected a priori in the target environment. Through numerical experiments on the OpenAI gym environment, we show that NFPDE can estimate the target distribution more accurately and efficiently than the previous estimation methods. We also show that the estimated DR distributions can improve the robustness of trained policies.**

I. INTRODUCTION

A. Background

In recent years, reinforcement learning (RL) has been utilized in various fields [1], [2] and has gained attention in robotics. However, applications of RL in robotics are lagging behind due to the need for extensive trial and error to train policies with generalization capabilities for diverse environments and the potential for hardware damage during data collection. A promising way to solve this issue is to use simulations for collecting data and training policies, and the learned policies are then applied to the actual devices. It enables us to gather enormous amounts of data quickly and safely, but this often fails in real-world scenarios due to the reality gap between simulations and the real world.

Domain randomization (DR) [3], [4] has been proposed as an effective way to address this reality gap by running RL on different environments while randomly changing simulation model parameters and training robust policies for environmental changes. However, it is necessary to appropriately design parameter distributions to make DR work well in real environments. If assuming a too broad parameter distribution compared to the real environment, the training difficulty increases, and the optimality of the trained policy is compromised. Conversely, if it is too narrow, it results in low generalization policies. A challenge lies in bringing the parameter distributions used in DR closer to those in real environments.

For solving this challenge, adaptive domain randomization (ADR) has been proposed to estimate parameter distributions for DR based on data acquired in real environments [5]–[14]. ADR consists of two main categories: Online ADR, which iteratively collects data and estimates parameter distributions [6]–[10], and Offline ADR, which uses pre-collected offline data for parameter distribution estimation [11]–[14]. In general, offline ADR methods are easier to implement than online ADR methods since online ADR methods have more implementation constraints. For example, online ADR methods cannot be fully parallelized, requiring that RL agents can easily access target environments and collect data during distribution estimation.

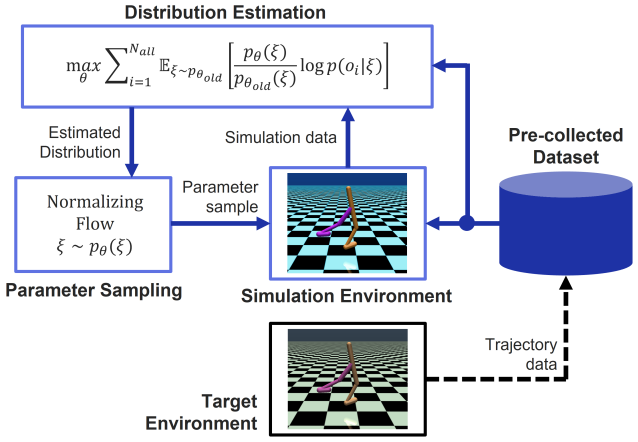


Fig. 1. Algorithm overview of NFPDE. NFPDE uses normalizing flow to model a parameter distribution of the target environment and fit the model by maximizing the likelihood of a pre-collected trajectory dataset.

B. Contributions

This paper proposes Normalizing Flow-based Parameter Distribution Estimation (NFPDE), a new estimation method of parameter distributions for offline ADR. NFPDE models the target distribution by a flow-based generative model using normalizing flow [15], and it performs model fitting based on the transition dataset collected a priori in the target environment. The model fitting is performed by maximizing the likelihood of the pre-collected dataset through gradient descent algorithms like policy gradient RL algorithms [16], [17]. The likelihood function is designed to be calculated from distances between the pre-collected data and data generated from simulators, but NFPDE does not require the simulator to be differentiable.

The main differences between NFPDE and previous offline ADR methods are parameter distribution estimation's accuracy and computational efficiency. While previous methods

[1]R. Takano, K. Takaya and H. Oyama are with Data Science Laboratories, NEC Corporation, 1753, Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa, 211-8666, Japan. {rin_takano, kei.takaya, h.oyama}@nec.com

approximate target distributions using simple distribution models (e.g., Gaussian distribution), NFPDE approximates the target distribution using a normalizing flow and improves model fitness for the target distributions. Especially when multiple environments with different values of model parameters are considered simultaneously, the target distribution can have a complex shape with numerous peaks. Still, NFPDE can provide flexible distribution estimation by utilizing the expressive power of normalizing flow. Furthermore, the gradient descent-based model fitting algorithm of NFPDE enables each update to be computed faster than the previous methods.

Contributions of this paper are as follows:

- Proposing NFPDE that estimates a parameter distribution of a target environment by using a normalizing flow from a pre-collected offline dataset.
- Showing a practical design likelihood function used in NFPDE.
- Demonstrating the estimation performances of NFPDE and comparing it with those of previous offline ADR methods in numerical experiments.
- Evaluating the performances of RL training with estimated distributions in numerical experiments.

II. Related Works

Parameter Estimation Methods Parameter Estimation (PE) is a common approach to deal with reality gap as well as DR. PE methods try to reduce the reality gap by identifying real-world parameters and making simulation models match the real world closely. The well-known PE approach uses the property that rigid body parameters appear linearly in the rigid body dynamics equations, and it estimates model parameters from real-world data by linear regression [18], [19]. Recent works use differentiable simulators [20]–[23] or neural network-based simulation model [24] to identify real-world parameters. They define a loss function including errors between the simulator outputs and real-world data, and use its analytic gradients to search parameters minimizing the loss function. However, PE methods cannot be directly applied to training robust policies because PE methods mainly focus on identifying not the distribution of parameters but only the value of parameters. [25] estimates model parameters by using PE methods and uses them to define uniform parameter distribution for DR, but its randomized range needs to be designed manually. On the other hand, NFPDE can automatically estimate the target parameter distribution from a pre-collected dataset, even in complex cases where multiple environments with different values of model parameters are considered simultaneously and the target distribution may have a complex shape with numerous peaks. Furthermore, while the above PE methods require an analytical model or differentiable simulators that precisely simulate target environments, NFPDE does not require them and can use general physics simulators (e.g., MuJoCo [26], PyBullet [27]).

Online ADR Methods Online ADR is one of the main categories of ADR, and it iteratively collects data and estimates parameter distributions from a collected dataset. There are two types of online ADR methods: one estimates parameter distributions that maximize the reward obtained when the intermediate policy is executed in the real world, and the other estimates parameter distributions that are similar to real-world distributions. The former does not directly estimate the real-world parameter distributions, but it implicitly estimates the parameter distributions through reward maximization. The papers [9], [10] proposed this type of online ADR method. The latter uses trajectory data collected in the real world and estimates the real-world parameter distributions. For example, SimOpt [6] proposed an online ADR method that iteratively estimates the parameter distributions by minimizing errors between real and simulated trajectory data. Online BayesSim and NPDR [7], [8] iteratively collect trajectory data and use Likelihood-Free Inference (LFI) [28] to approximate the real-world parameter distributions. Online ADR methods can iteratively improve the accuracy of the estimated parameter distribution. Still, their processes of repeatedly collecting data in the real world are expensive due to requiring careful expert supervision, e.g., safety checks and resetting environments. On the other hand, NFPDE makes use of previously collected data and does not require additional data collection in the real world. Hence, NFPDE can be implemented more easily than online ADR methods.

Offline ADR Methods Offline ADR is one of the main categories of ADR, and it estimates the real-world parameter distributions by using previously collected datasets. In contrast to some online ADR methods, offline ADR methods including NFPDE do not make any assumption on how data has been collected, and many kinds of data collected by human demonstration, random policy, and policy trained by RL can be used.

There are several variations of offline ADR methods. BayesSim [12] collects a dataset of trajectory and parameter pairs in simulators and trains mixture density random feature networks to model a posterior $p(\xi|\varphi)$ approximating a parameter distribution, where ξ is a parameter vector, and φ is a feature vector calculated from trajectory data. BayesSim finally uses the posterior conditioned by a feature vector calculated by one real-world trajectory data and approximates the real-world parameter distribution through LFI formulation. However, the trained posterior is insufficient to approximate complex real-world parameter distributions because the feature vector cannot directly contain information from multiple environments, and the formulation of the posterior is limited to a mixed Gaussian distribution. In contrast, the formulation of NFPDE can naturally handle trajectory data collected by multiple environments and well approximate complex parameter distributions by using normalizing flow. DROID [13] and DROPO [11] use a Gaussian distribution to approximate the real-world parameter distributions. DROID defines a loss function calculating distances between real and simulation trajectories, and DROPO defines a loss function calculating log-likelihoods of the real trajectories. Finally, they estimate the parameters of the Gaussian distribution

by minimizing their loss function through sampling-based optimization methods (e.g., CMA-ES [29], [30]). While the estimated distribution models of these two methods are also limited to simple Gaussian distributions, NFPDE using a more expressive distribution model can estimate complex real-world parameter distributions in more detail. Furthermore, NFPDE uses a gradient descent-based algorithm that can efficiently calculate updates of the model parameters, not sampling-based optimization algorithms.

III. Preliminaries

A. Markov Decision Process and Reinforcement Learning

We consider a parameterized Markov Decision Process (MDP) $\mathcal{M}_\xi$ that is defined by $\mathcal{M}_\xi = (\mathcal{S}_\xi, \mathcal{A}_\xi, P_\xi, R_\xi, \rho_\xi, \gamma_\xi)$, where $\mathcal{S}_\xi$ and $\mathcal{A}_\xi$ are state and action spaces, $P_\xi : \mathcal{S}_\xi \times \mathcal{A}_\xi \times \mathcal{S}_\xi \to \mathbb{R}_+$ is a state-transition probability function, $R_\xi : \mathcal{S}_\xi \times \mathcal{A}_\xi \to \mathbb{R}$ is a reward function, $\rho_\xi : \mathcal{S}_\xi \to \mathbb{R}_+$ is an initial state probability function and $\gamma_\xi \in [0, 1]$ is a reward discount factor. The domain of the parameterized MDP is characterized by a parameter vector $\xi \in \Xi$, and the vector ξ is a random variable distributed according to an unknown probability distribution $p(\xi) : \Xi \to \mathbb{R}_+$.

Let $\tau = (s_0, a_0, \cdots, s_T)$ be trajectory of the MDP $\mathcal{M}_\xi$. The goal of an RL agent in this paper is to learn a policy $\pi_\theta(a_t|s_t) : \mathcal{S}_\xi \times \mathcal{A}_\xi \to \mathbb{R}_+$ maximizes the following expected return:

$$J(\theta) = \mathbb{E}_{\xi\sim p(\xi)}\left[\mathbb{E}_{\tau\sim p(\tau)}\left[\sum_{t=0}^{T}\gamma^t R(s_t, a_t) \mid \theta, \xi, s_0\right]\right] \quad (1)$$

where $p(\tau)$ is a trajectory probability function defined by the MDP $\mathcal{M}_\xi$ and an output of policy $a_t \sim \pi_\theta(a_t|s_t)$. Unlike a standard RL formulation, the expected return (1) has an outer expectation with respect to the parameter vector ξ. It makes the RL agent learn a robust policy that can work in a large domain characterized by $p(\xi)$.

B. Normalizing Flow

Normalizing flow is one of the generative models, which can explicitly model a probability distribution expressing given datasets. It uses bijective mappings $f_i(z)$ to convert a sample z_0 from a simple distribution $p_Z(z_0)$ to data x,

$$\begin{aligned} z_i &= f_i(z_{i-1}) \quad i = 1, ..., K, \\ x &= z_K. \end{aligned} \quad (2)$$

The most important feature of normalizing flow is that it can explicitly calculate a probability of the model $p_X(x)$:

$$p_X(x) = p_Z(z_0)\prod_{i=1}^{K}\left|\det\left(\frac{\partial f_i(z_{i-1})}{\partial z_{i-1}}\right)\right|. \quad (3)$$

Hence, unlike the variational autoencoder (VAE) [31] and diffusion models [32] that are trained by maximizing evidence lower bound (ELBO), normalizing flow can be trained by directly maximizing log-likelihood of (3).

Normalizing flow has many variations depending on how the bijective mappings f_i are designed [33]–[35]. In this paper, we use Real NVP [15] for its ease of implementation and light computational complexity. Real NVP defines a flexible and tractable bijective function by stacking a sequence of simple bijections called an affine coupling layer. Given an input vector $x = [x_1, x_2] \in \mathbb{R}^D$, $x_1 \in \mathbb{R}^d$, $x_2 \in \mathbb{R}^{D-d}$ the affine coupling layer calculates the output $y = [y_1, y_2]$, $y_1 \in \mathbb{R}^d$, $y_2 \in \mathbb{R}^{D-d}$ as follows,

$$\begin{aligned} y_1 &= x_1, \\ y_2 &= x_2 \odot \exp\left(s\left(x_1\right)\right) + t\left(x_1\right), \end{aligned} \quad (4)$$

where $s, t : \mathbb{R}^d \to \mathbb{R}^{D-d}$ stand for scale and transformation, and they are modeled by neural networks. Note that each output of the layers f_i is reordered before being input to the next layer since all elements of the original input should be converted.

IV. Our Method: NFPDE

A. Distribution Estimation Through Maximizing Likelihood

Our method, NFPDE, estimates an unknown parameter distribution $p(\xi)$ by using offline state-action trajectory data $\mathcal{D} = \{\tau_1, ..., \tau_N\}$ consisting of N trajectories collected by any policy. We focus on transition tuples $o_t = (s_t, a_t, s_{t+1})$ obtained from the data $\mathcal{D}$ and let N_{all} be number of elements of all transition tuples. To approximate the unknown parameter distribution $p(\xi)$, we define a parameter distribution model $p_\theta(\xi)$ by using the normalizing flow with a parameter θ. Then, the log-likelihood of all transition tuples and its lower bound can be calculated by

$$\begin{aligned} \log p(o_{1:N_{all}}) &= \log\left(\prod_{i=1}^{N_{all}} p(o_i)\right) \\ &= \sum_{i=1}^{N_{all}} \log\left(\int p(o_i|\xi)p_\theta(\xi)d\xi\right) \quad (5) \\ &\geq \sum_{i=1}^{N_{all}} \int \log p(o_i|\xi)p_\theta(\xi)d\xi, \quad (6) \end{aligned}$$

where $p(o_i|\xi)$ is a conditional probability that user can design, and its design candidates are explained in the next subsection. Eq. (6) is derived from Eq. (5) by using Jensen's inequality [36]. The model $p_\theta(\xi)$ can be trained by looking for the optimal parameter θ^* maximizing the following function $L(\theta, o_{1:N_{all}})$ derived from Eq. (6),

$$L(\theta, o_{1:N_{all}}) = \sum_{i=1}^{N_{all}} \mathbb{E}_{\xi\sim p_\theta(\xi)}\left[\log p(o_i|\xi)\right]. \quad (7)$$

However, gradient descent methods like Adam [37] cannot directly be used to search the optimal parameter θ^* since the gradient of Eq. (7) cannot be approximated with naive Monte Carlo methods. Therefore, we consider sequentially maximizing the following loss instead of directly maximizing

Eq. (7),

$$L(\theta, \theta_{old}, o_{1:N_{all}}) = \sum_{i=1}^{N_{all}} \mathbb{E}_{\xi \sim p_{\theta_{old}}(\xi)} \left[\frac{p_\theta(\xi)}{p_{\theta_{old}}(\xi)} \log p(o_i|\xi) \right], \quad (8)$$

where θ_{old} is a parameter calculated in a previous training step. Importance sampling technique [38] is used to derive Eq. (8) from Eq. (7). This formulation enables us to calculate the gradients with naive Monte Carlo methods since it does not need sampling from the model $p_\theta(\xi)$ having the parameter θ to be optimized. Note that the loss function (8) is different from loss functions used in LFI-based algorithms [8], [12], and the formulation of Eq. (8) is similar to the loss functions used in policy gradient algorithms, e.g., TRPO [16], PPO [17]. Hence the loss function (8) can also be extended to a PPO styled surrogate loss functions for improving optimization stability and performance.

An algorithm of NFPDE for estimating an unknown parameter distribution utilizing Eq. (8) is presented in Algorithm 1.

Algorithm 1 Normalizing Flow-based Parameter Distribution Estimation (NFPDE)

Output: Optimized parameter distribution model $p_{\theta^*}(\xi)$

1: Initialize model parameter θ, θ_{old} and empty dataset $\mathcal{D}$
2: Collect N trajectories in the target domain and fill $\mathcal{D}$
3: **for** $i = 1$ to I **do**
4: Sample a set Ξ_s of M parameters from $p_{\theta_{old}}(\xi)$
5: **for all** $\xi_j \in \Xi_s$ **do**
6: Set simulator parameters to ξ_j
7: **for all** $o_t = (s_t, a_t, s_{t+1}) \in \mathcal{D}$ **do**
8: Set the simulator state to s_t
9: Simulate with action a_t and calculate the conditional probability $p(o_t|\xi_j)$
10: **end for**
11: **end for**
12: Optimize θ maximizing $L(\theta, \theta_{old}, o_{1:N_{all}})$ (Eq. 8)
13: Update $\theta_{old} \leftarrow \theta$
14: **end for**
15: $\theta^* \leftarrow \theta_{old}$
16: **return** $p_{\theta^*}(\xi)$

B. Design of Conditional Probability $p(o|\xi)$

One of the most important things to estimate the parameter distribution model $p_\theta(\xi)$ by using the algorithm 1 is how to design the conditional observation probability function $p(o|\xi)$. It must be defined such that the closer a given parameter ξ is to the true parameter that generated the observed data o, the higher the probability is obtained. To evaluate this, we use a simulator $f_{sim} : \mathcal{S} \times \mathcal{A} \times \Xi \to \mathcal{S}$ that approximates a state-transition probability of the parameterized MDP, and we define the following dynamics loss function:

$$l_{pred}(o_t, \xi) = \|s_{t+1} - f_{sim}(s_t, a_t, \xi)\|^2. \quad (9)$$

Note that the simulator f_{sim} can be any simulator, e.g., a physics simulator (MuJoCo [26], PyBullet [27]), a black box function, and they can be non-differentiable. Based on the dynamics loss function (9), we define two candidates of the probability function $p(o|\xi)$ satisfying the above requirement as follows.

1) Candidate 1 (CAI style): The first candidate is defined as

$$p(o_t|\xi) = \exp\left(-l_{pred}(o_t, \xi)\right). \quad (10)$$

It is inspired by an optimal probability of the control as inference (CAI) framework [39], and it simply evaluates the value of the probability with respect to $l_{pred}(o_t, \xi)$.

2) Candidate 2 (Softmax style): By using an assumption that the Monte Carlo method is used to approximate Eq. (8) in the algorithm 1, the second candidate is defined as

$$p(o_t|\xi) = \exp\left(\frac{\exp\left(-\frac{1}{\beta} l_{pred}(o_t, \xi)\right)}{\sum_{j=1}^{M} \exp\left(-\frac{1}{\beta} l_{pred}(o_t, \xi_j)\right)} \right), \quad (11)$$

where $\beta \in (0, 1]$ is a temperature parameter. This candidate calculates the value by using a softmax function with temperature [40], and the samples $\xi \in \Xi_s$ generated in line 4 of the algorithm 1 are used to evaluate the denominator of Eq. (11). Therefore, this candidate evaluates relative goodness for a given sample (o_t, ξ) with respect to the other samples $(o_t, \xi_j), \xi_j \in \Xi_s$.

C. Bounded Parameter Distributions

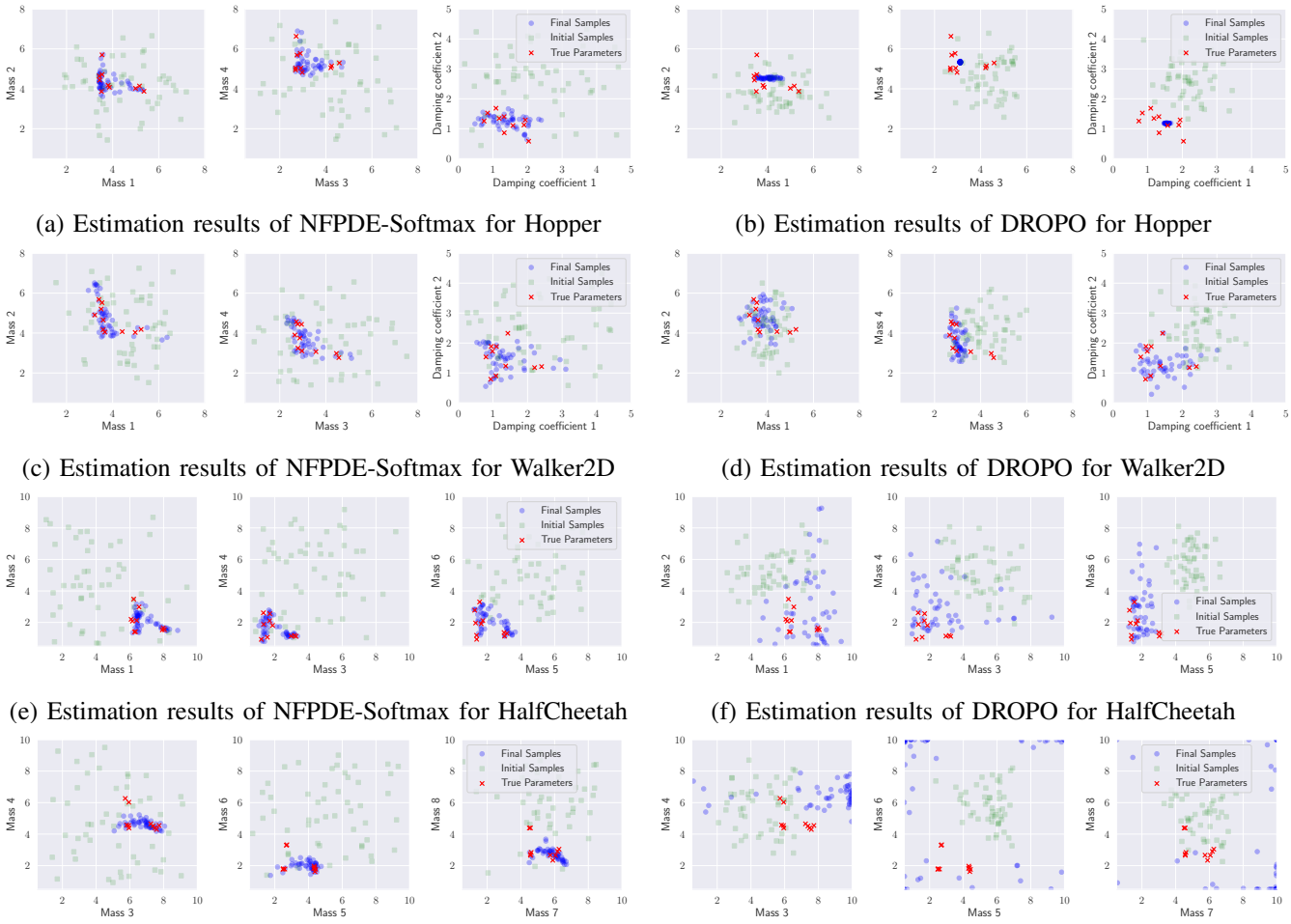
When an unknown parameter ξ is bounded (e.g., ξ is masses of objects), samples generated by the learned distribution model $p_\theta(\xi)$ must also satisfy this property. To achieve it, a modified parameter vector ξ' transformed by the sigmoid function can be used instead of ξ,

$$\xi' = \frac{\xi_u - \xi_l}{1 + \exp(-\xi)} + \xi_l, \quad (12)$$

where ξ_l, ξ_u are lower and upper bounds of the parameter ξ, respectively. The above operation is executed elementwise. In the algorithm 1, we replace the value of ξ to calculate conditional probability $p(o_i|\xi)$ with ξ'. Note that it is necessary to transform the generated samples when sampling using the distribution model learned with (12).

V. Experiments

In this section, we present numerical experiments to demonstrate the performance of NFPDE for parameter distribution estimation. We evaluate the performance of NFPDE on two measures: the accuracy of distribution estimation for unknown parameter distributions and the robustness of the policies trained by reinforcement learning with domain randomization using the estimated distributions.



(a) Estimation results of NFPDE-Softmax for Hopper

(b) Estimation results of DROPO for Hopper

(c) Estimation results of NFPDE-Softmax for Walker2D

(d) Estimation results of DROPO for Walker2D

(e) Estimation results of NFPDE-Softmax for HalfCheetah

(f) Estimation results of DROPO for HalfCheetah

(g) Estimation results of NFPDE-Softmax for Humanoid

(h) Estimation results of DROPO for Humanoid

Fig. 2. Results of parameter distribution estimation for the target environments with Mixed Gaussian Distribution. Red crosses are true parameters corresponding to the given pre-collected trajectory dataset. The green squares are samples from the initial estimated distribution, and the blue points are samples from the final estimated distribution.

A. *Benchmark Environments & Baseline Methods*

Using four OpenAI Gym environments [41], Hopper, Walker2D, HalfCheetah and Humanoid, we evaluated the performances of NFPDE. We assumed that the model parameters of the four models are unknown and generated from unknown distributions. We collected trajectories for ten episodes in each environment using uniform random control input and estimated parameter distribution corresponding to the collected data. We call these environments target environments.

We also compared the performances of our method with those of baselines of offline ADR methods, DROPO [11] and BayesSim [12] under the same settings. In implementations of all methods, the sigmoid transformation defined by Eq. (12) was used to make values of samples generated from estimated distributions be bounded within a given region. In DROPO implementation, we implemented the CMA-ES algorithm as an optimization solver based on the paper [30]. In BayesSim implementation, we used a mixture density random feature network with neural network features. The mixture density random network was modeled by a mixture of 3 Gaussians whose mean and covariance are calculated by each neural network. Since BayesSim models a parameter distribution as a conditional probability of the parameter given one feature vector, we used an average of feature vectors calculated from ten trajectories collected in the target environments. In NFPDE implementation, we used each

TABLE I

SIMULATION PARAMETERS OF TARGET ENVIRONMENTS

Environment	Dim of ξ	Unknown Parameters ξ
Hopper	6	Link masses (4 elements), joint damping coefficients (2 elements)
Walker2D	7	Link masses (4 elements), joint damping coefficients (2 elements), friction coefficient of floor (1 element)
HalfCheetah	14	Link masses (7 elements), joint damping coefficients (6 elements), friction coefficient of floor (1 element)
Humanoid	30	Link masses (13 elements), joint damping coefficients (17 elements)

candidate of the conditional probability $p(o|\xi)$: the CAI style probability and the softmax style probability to compare their performances. We call the former NFPDE-CAI and the latter NFPDE-Softmax. All neural networks were implemented by using PyTorch [42], and all computations were performed on a desktop computer with an Intel Core i9-13900F CPU running at 5.60GHz and a NVIDIA GeForce RTX 4080 GPU.

B. Performances of Parameter Distribution Estimation

We first evaluated the distribution estimation performances of NFPDE-Softmax, NFPDE-CAI, and two baselines. In this experiment, we assumed that unknown model parameters are the model's body masses, joint damping coefficients, and friction coefficients for friction between foot and floor. The number of unknown model parameters for each model and their details are summarized in the table I. We also assumed that they are generated from two parameter distributions: Gaussian distribution $p(\xi) = \mathcal{N}(\mu_\xi, \Sigma_\xi)$ and mixed Gaussian distribution $p(\xi) = \sum_{k=1}^{5} \alpha_k \mathcal{N}(\mu_{\xi,k}, \Sigma_{\xi,k})$. For the mixed Gaussian distribution, the mean of each Gaussian distribution was arranged to be L-shaped in the parameter space. It is intended for cases where distribution estimation using Gaussian distributions would result in a conservative approximation.

To evaluate the accuracy of the estimated distributions, we used the optimal transport (OT, earth mover's distance) [43] between parameter samples generated from the estimated distribution and the true parameters corresponding to the collected trajectories as an evaluation measure. It approximates calculating a statistical distance between the estimated and true distributions. The problem of calculating optimal transport costs between two sample sets is formulated as a linear programming problem. We used a Python Optimal Transport (POT) [44] that is an open-source library to calculate optimal transports in the experiments.

Table II shows the average optimal transport costs of the distributions estimated by NFPDEs and two baselines, and Fig. 2 shows generated samples from the estimated distributions. These results show that NFPDE-Softmax outperformed the other methods and provided the best estimation results in all environments. In particular, NFPDE-Softmax outperformed DROPO, the second-best performer, in the case of complex mixed Gaussian distributions. DROPO could estimate the distribution to encompass the true parameters in the Walker2D environment. Still, in the Hopper environment, the estimated distribution of DROPO converged around the mean of the true parameters and did not capture the true parameter distribution. Furthermore, in the HalfCheetah and the Humanoid environments, the estimated parameter distributions of DROPO failed to capture the true parameter distributions and almost diverged toward the parameter bounds. During the parameter distribution estimation, there was a tendency for the estimated distribution to become unstable when the parameter samples from the distribution approaches the parameter boundaries. Since generated samples were bounded by using the sigmoid transformation defined by Eq. (12) and DROPO uses a Gaussian distribution to model the parameter distribution, DROPO cannot update the estimated parameter distribution appropriately. On the other hand, NFPDE-Softmax took advantage of the expressive power of normalizing flow, and it could flexibly estimate parameter distributions in all environments. However, NFPDE-CAI could not estimate the distributions reasonably even though it uses the same NFPDE framework. Using the softmax style conditional probability (11) is important when estimating distributions using NFPDE. The convergence of the parameter distribution estimation by each method can be seen in the supplemental video.

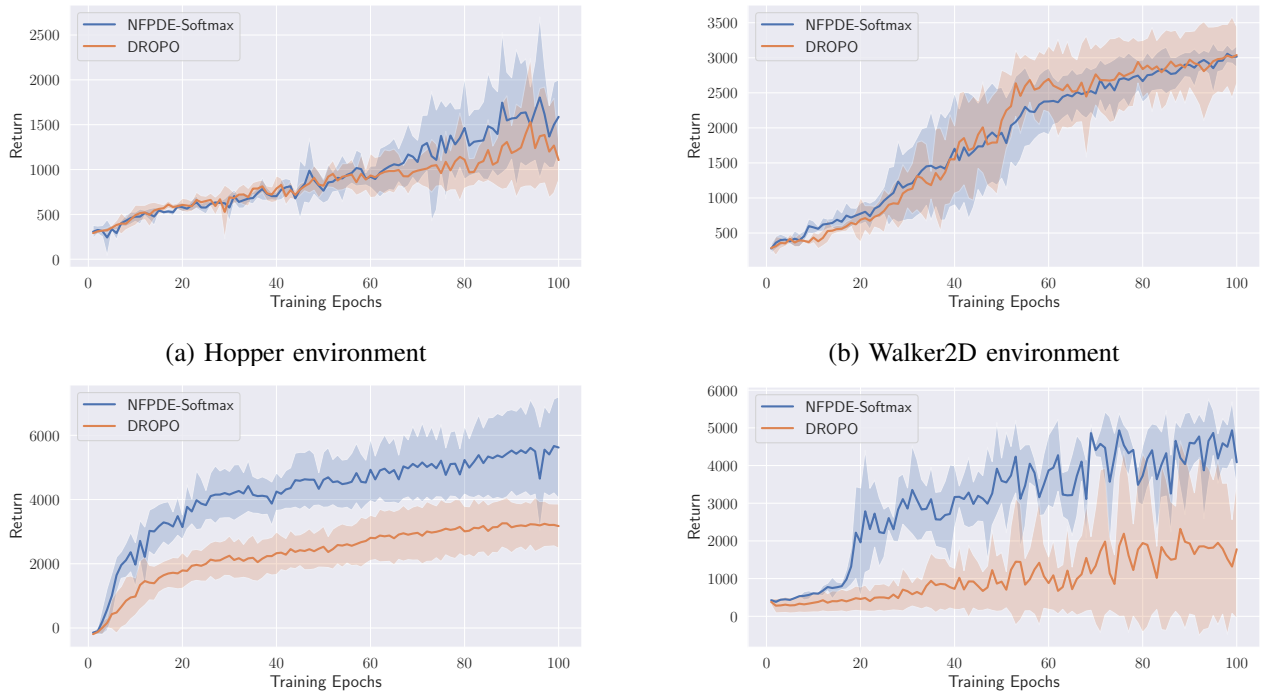
TABLE II
OPTIMAL TRANSPORT (OT) COSTS OF ESTIMATED PARAMETER DISTRIBUTIONS FOR FOUR ENVIRONMENTS

Hopper environment		
	Gaussian	Mixed Gaussian
NFPDE-Softmax	$\mathbf{0.336 \pm 0.0159}$	$\mathbf{0.793 \pm 0.111}$
NFPDE-CAI	4.53 ± 0.288	3.74 ± 0.428
DROPO	0.382 ± 0.0534	1.25 ± 0.130
BayesSim	3.42 ± 0.174	2.87 ± 0.0971
Walker2D environment		
	Gaussian	Mixed Gaussian
NFPDE-Softmax	$\mathbf{0.338 \pm 0.0231}$	$\mathbf{0.828 \pm 0.0497}$
NFPDE-CAI	1.44 ± 0.171	1.69 ± 0.264
DROPO	0.730 ± 0.210	1.51 ± 0.174
BayesSim	4.99 ± 0.825	4.36 ± 1.02
HalfCheetah environment		
	Gaussian	Mixed Gaussian
NFPDE-Softmax	$\mathbf{0.656 \pm 0.0702}$	$\mathbf{0.996 \pm 0.0920}$
NFPDE-CAI	1.63 ± 0.359	3.77 ± 0.687
DROPO	8.86 ± 2.04	8.84 ± 3.25
BayesSim	5.19 ± 0.0979	6.15 ± 1.17
Humanoid environment		
	Gaussian	Mixed Gaussian
NFPDE-Softmax	$\mathbf{1.69 \pm 0.418}$	$\mathbf{2.84 \pm 0.255}$
NFPDE-CAI	6.61 ± 1.16	10.23 ± 1.17
DROPO	15.7 ± 2.22	16.8 ± 3.04
BayesSim	19.0 ± 1.48	19.13 ± 0.336

TABLE III
AVERAGE COMPUTATIONAL TIMES TO ESTIMATE DISTRIBUTIONS FOR FOUR ENVIRONMENTS

	Hopper	Walker2D
NFPDE-Softmax	$\mathbf{23.5 \pm 4.60}$	$\mathbf{28.2 \pm 3.09}$
NFPDE-CAI	$\mathbf{22.5 \pm 3.17}$	$\mathbf{22.9 \pm 1.96}$
DROPO	518 ± 83.0	698 ± 64.0
BayesSim	834 ± 25.6	867 ± 12.4
	HalfCheetah	Humanoid
NFPDE-Softmax	$\mathbf{148 \pm 13.2}$	$\mathbf{135 \pm 17.0}$
NFPDE-CAI	$\mathbf{140 \pm 10.6}$	$\mathbf{129 \pm 14.3}$
DROPO	4726 ± 149	3143 ± 345
BayesSim	942 ± 37.1	934 ± 15.6

Table III shows the average computational time required for distribution estimation using each method, and it shows that NFPDE also outperformed other methods in computation time. By taking advantage of gradient-based optimization, NFPDE estimated the distributions about 20 times faster than the other methods. Note that the most important parameter of NFPDE-Softmax affecting convergence speed is the temperature parameter β used in the probability function (11). Empirically, the smaller the value of β is, the faster the



(a) Hopper environment

(b) Walker2D environment

(c) HalfCheetah environment

(d) Humanoid environment

Fig. 3. Average policy returns in the target environments during policy training using the estimated parameter distributions. Mixed Gaussian distributions are used to be the unknown parameter distributions. Each average return is calculated by using four trained policies.

convergence speed is.

These results show that our proposed method, NFPDE-Softmax, outperforms other offline ADR methods in both parameter estimation accuracy and computational time, and it can work appropriately even in environments having high-dimensional unknown parameters.

C. Performances of RL with Domain Randomization

Finally, we trained policies by using domain randomization with the estimated distributions and evaluated their robustness performances in the target environments, Hopper, Walker2D, HalfCheetah, and Humanoid. Based on the comparison results of the parameter distribution estimation accuracy in the last subsection, we evaluated the best-performing NFPDE-Softmax and the second-best-performing DROPO. To evaluate the robustness of the trained policies for complex parameter distributions, we also focus on the target environments whose parameters are generated from mixed Gaussian distributions.

By using Stable-baselines3 [45], we trained the policies with Soft Actor-Critic (SAC) [46]. Note that a trajectory dataset collected in the target environments is not used for policy training. Instead, policy training uses data collected in environments with parameters derived from the estimated distributions. We obtained four parameter distribution models by using each method and trained four policies corresponding to each estimated distribution model, and we evaluated their average result.

Fig. 3 shows the average policy returns for the target environments during policy training. In the three environments, Hopper, HalfCheetah, and Humanoid, where DROPO could not estimate parameter distribution appropriately, the average returns of policy training with domain randomization using NFPDE-Softmax are higher than DROPO. The difference in the performance of the learned policies is particularly large in the two environments, HalfCheetah and Humanoid, where NFPDE-Softmax outperformed in the distribution estimation. Therefore, these results are attributed to the differences in distribution estimation accuracy.

VI. Conclusions

In this paper, we proposed NFPDE, a new parameter distribution estimation method for offline ADR, which estimates a parameter distribution of a target environment by using normalizing flow from a pre-collected offline dataset. We demonstrated the estimation performances of NFPDE in OpenAI gym environments and compared the performances of NFPDE with those of previous offline ADR methods. The results show that NFPDE outperforms the previous methods regarding distribution estimation accuracy and computational time for estimation. Finally, we also trained policies by using domain randomization with the estimated parameter distributions and demonstrated their robustness performance.

Since we demonstrated NFPDE in only numerical environments, we will apply NFPDE to real environments and evaluate its estimation performances and sim-to-real performances of policies trained by using domain randomization with the estimated parameter distributions as future work. Furthermore, parameter distributions estimated by NFPDE were used for training only robust policies in

this paper, but they can also be used for training universal policies having parameters as input. We also plan to apply NFPDE to such cases and evaluate performances.

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M, Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, Vol. 529, pp. 484-489, 2016.

[2] M. M. Afsar, T. Crump, and B. Far, "Reinforcement Learning based Recommender Systems: A Survey," *ACM Computing Surveys*, Vol. 55, Issue 7, pp. 1–38, 2022.

[3] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng and W. Zaremba, "Learning Dexterous In-Hand Manipulation," *The International Journal of Robotics Research*, Vol. 39, Issue 1, pp.3–20, 2020.

[4] I. Mordatch, K. Lowrey, E. Todorov, "Ensemble-CIO: Full-Body Dynamic Motion Planning that Transfers to Physical Humanoids," in *IROS*, 2015.

[5] G. Tiboni, K. Arndt, G. Averta, V. Kyrki and T. Tommasi, "Online vs. Offline Adaptive Domain Randomization Benchmark," in *Human-Friendly Robotics*, 2022.

[6] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff and D. Fox, "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience," in *ICRA*, 2019.

[7] R. Possas, L. Barcelos, R. Oliveira, D. Fox and F. Ramos, "Online BayesSim for Combined Simulator Parameter Inference and Policy Improvement," in *IROS*, 2020.

[8] F. Muratore, T. Gruner, F. Wiese, B. Belousov, M. Gienger and J. Peters, "Neural Posterior Domain Randomization," in *CoRL*, 2022.

[9] F. Muratore, C. Eilers, M. Gienger and J. Peters, "Data-efficient Domain Randomization with Bayesian Optimization," *IEEE Robotics and Automation Letters*, Vol. 6, pp.911–918, 2020.

[10] M. Mozifian, J. C. G. Higuera, D. Meger and G. Dudek, "Learning Domain Randomization Distributions for Training Robust Locomotion Policies," in *IROS*, 2020.

[11] G. Tiboni, K. Arndt and V. Kyrki, "DROPO: Sim-to-real transfer with offline domain randomization," *Robotics and Autonomous Systems*, Vol. 166, 2023.

[12] F. Ramos, R. C. Possas and D. Fox, "BayesSim: adaptive domain randomization via probabilistic inference for robotics simulators," in *RSS*, 2019.

[13] Y. Tsai, H. Zu, Z. Ding, C. Zhang, D. Jonhs and B. Huang, "DROID: Minimizing the Reality Gap using Single-Shot Human Demonstration," *IEEE Robotics and Automation Letters*, Vol. 6, pp. 3168–3175, 2021.

[14] R. Antonova, F. Ramos, R. Possas and D. Fox, "BayesSimIG: Scalable Parameter Inference for Adaptive Domain Randomization with IsaacGym," arXiv preprint arXiv:2107.04527 , 2021.

[15] L. Dinh, J. Sohl-Dickstein and S. Bengio, "Density Estimation Using Real NVP," in *ICLR*, 2017.

[16] J. Schulman, S. Levine, P. Abbeel, M. Jordan and P. Moritz, "Trust Region Policy Optimization," in *ICML*, 2015.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal Policy Optimization Algorithms," arXiv preprint arXiv:1707.06347, 2017.

[18] C. G. Atkeson, C. H. An, and J. M. Hollerbach, "Estimation of inertial parameters of manipulator loads and links," *The International Journal of Robotics Research*, Vol. 5, Issue 3, pp.101–119, 1986.

[19] J. Ting, M. Mistry, J. Peters, S. Schaal and J. Nakanishi, "A Bayesian Approach to Nonlinear Parameter Identification for Rigid Body Dynamics," in *RSS*, 2006.

[20] K. M. Jatavallabhula, M. Macklin, F. Golemo, V. Voleti, L. Petrini, M. Weiss, B. Considine, J. Parent-Lévesque, K. Xie, K. Erleben, L. Paull, F. Shkurti, D. Nowrouzezahrai and S. Fidler, "gradSim: DIFFERENTIABLE SIMULATION FOR SYSTEM IDENTIFICATION AND VISUOMOTOR CONTROL," in *ICLR*, 2021.

[21] E. Heiden, M. Macklin, Y. Narang, D. Fox, A. Garg and F. Ramos, "DiSECt: A Differentiable Simulation Engine for Autonomous Robotic Cutting," in *RSS*, 2021.

[22] F. A. Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum and J. Z. Kolter, "End-to-End Differentiable Physics for Learning and Control," in *NeurIPS*, 2018.

[23] T. A. Howell, S. L. Cleac'h, J. Brüdigam, J. Z. Kolter, M. Schwager and Z. Manchester, "Dojo: A Differentiable Physics Engine for Robotics," arXiv preprint arXiv:2203.00806, 2022.

[24] T. Murooka, M. Hamaya, F. V. Drigalski, K. Tanaka and Y. Ijiri, "EXI-Net: EXplicitly/Implicitly Conditioned Network for Multiple Environment Sim-to-Real Transfer," in *CoRL*, 2020.

[25] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez and V. Vanhoucke, "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots," in *RSS*, 2018.

[26] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*, 2012.

[27] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation, games, robotics and machine learning," http://pybullet.org, 2016.

[28] G. Papamakarios, I. Murray, "Fast ϵ-free Inference of Simulation Models with Bayesian Conditional Density Estimation," in *NeurIPS*, 2016.

[29] N. Hansen, A. Ostermeier, "Completely Derandomized Self-Adaptation in Evolution Strategies," *Evolutionary Computation*, Vol. 9, Issue 2, pp. 159–195, 2001.

[30] N. Hansen, "The CMA Evolution Strategy: A Tutorial," arXiv preprint arXiv:1604.00772, 2016.

[31] D. Kingma, M. Welling, "Auto-Encoding Variational Bayes," in *ICLR*, 2014.

[32] J. Ho, A. Jain and P. Abbeel, "Denoising Diffusion Probabilistic Models," in *NeurIPS*, 2020.

[33] D. J. Rezende and S. Mohamed, "Variational Inference with Normalizing Flows," in *ICML*, 2015.

[34] G. Papamakarios, T. Pavlakou and I. Murray, "Masked Autoregressive Flow for Density Estimation," in *NIPS*, 2017.

[35] I. Kobyzev, S. J. D. Prince and M. A. Brubaker, "Normalizing Flows: An Introduction and Review of Current Methods," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 11, pp. 3964-3979, 2021,

[36] T. Needham, "A Visual Explanation of Jensen's Inequality," in *The American Mathematical Monthly*, Vol. 100, No. 8, pp. 768–771, 1993.

[37] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR*, 2015.

[38] W. G. Cochran, "Sampling techniques," John Wiley & Sons, 2007.

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

[40] G. Hinton, O. Vinyals, J. Dean, "Distilling the Knowledge in a Neural Network," arXiv preprint arXiv:1503.02531, 2015.

[41] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[42] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NeurIPS*, 2017.

[43] G. Peyré and M. Cuturi, "Computational Optimal Transport," *Foundations and Trends®in Machine Learning*, Vol. 11, No. 5-6, pp. 355–607, 2019.

[44] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. H. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, T. Vayer, "POT: Python Optimal Transport," *Journal of Machine Learning Research*, Vol. 22, No. 78, pp.1–8, 2021.

[45] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning*, Vol. 22, No. 268, pp. 1–8, 2021.

[46] T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.," in *ICML*, 2018.