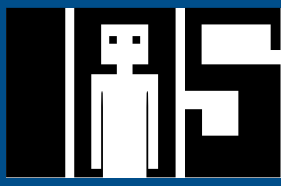
# Approximated Policy Search in Black-Box Optimization

**Ungefähre Richtliniensuche in der Black-Box-Optimierung**
Master thesis by Zhenhui Zhou
Date of submission: March 1, 2021

1. Review: Prof. Dr. Jan Peters
2. Review: Prof. Dr. Heinz Koeppl
3. Review: M.Sc. Puze Liu
4. Review: M.Sc. Julen Urain De Jesus
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT



## Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Zhenhui Zhou, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 1. März 2021

Zhenhui, Zhou

Z. Zhou

# Abstract

In machine learning, it is a fundamental robotic task to learn a multi-modal and intractable distribution, which could be the representation of the rewards following the specific policy, with a well constructed and tractable distribution model. We demonstrate a method based on the policy search using normalizing flow to address this problem. The normalizing flow is able to compress the distribution from the observed space into the latent space and recover it based on the latent space since the direct operations in the observed space are usually not available. The benefits of normalizing flow are the power of expressive and the ability of both efficiently sampling as well as accurately evaluating over the drawn samples. And the strategies of model-based policy search algorithms for updating the constructed model are well suited to this optimization problem when an additional entropy term is applied in the objective. Our method also employs a surrogate function for providing the gradient information. For this objective, we consider both the I-projection and M-projection to avoid premature collapsing. We evaluate our method to learn the gaussian mixture model as well as the sparse reward distribution and apply it for the robotic grasping tasks. The method is able to provide a significantly favorable approximation to the complex multi-modal target distribution and to give a compact and elegant policy model with high expressive power.

# Zusammenfassung

Im maschinellen Lernen ist es eine grundlegende Roboteraufgabe, eine multimodale und unlösbare Verteilung zu lernen, die die Belohnungen nach der spezifischen Richtlinie repräsentieren könnte, mit einem gut konstruierten und handhabbar Verteilungsmodell. Wir zeigen eine Methode, die auf der Richtliniensuche basiert und den Normalisierungsfluss verwendet, um dieses Problem zu beheben. Der Normalisierungsfluss kann die Verteilung vom beobachteten Raum in den latenten Raum transformieren, und auch vom latenten Raum in den beobachteten Raum wieder herstellen, weil die direkten Operationen im beobachteten Raum normalerweise nicht verfügbar sind. Die Vorteile der Normalisierungsfluss sind die Ausdruckskraft und die die Fähigkeit, sowohl effizient Proben zu entnehmen als auch die gezogenen Proben genau auszuwerten. Die Strategien modellbasierter Richtliniensuchalgorithmen zur Aktualisierung des erstellten Modells eignen sich gut für dieses Optimierungsproblem, wenn ein zusätzlicher Entropieterm in der Zielsetzung angewendet wird. Unsere Methode verwendet auch eine Ersatzfunktion zur Bereitstellung der Gradienteninformationen. Für diese Zielsetzung betrachten wir sowohl die I-Projektion als auch die M-Projektion, um ein vorzeitiges Zusammenfallen zu vermeiden. Wir evaluieren unsere Methode, um das Gaußsche Mischungsmodell sowie die spärliche Belohnungsverteilung zu lernen und sie für die Roboter-Greifaufgaben anzuwenden. Das Verfahren ist in der Lage, eine signifikant günstige Annäherung an die komplexe multimodale Zielverteilung zu liefern, und ein kompaktes und elegantes politisches Modell mit hoher Ausdruckskraft zu geben.

# Contents

# Figures

## List of Figures

# 1. Introduction

The classic modeling of the robot and the environment to solve a robotic task is not sufficiently accurate and limited on adaptation to a new change, while the methods in machine learning field are mainly focusing on driving the robot by extracting the relevant information from data. The robotic tasks can be automatically achieved with help of the power of expressiveness and of the flexibility of the methods from machine learning techniques, such as reinforcement learning, when these tasks are formulated properly.

One of the core robotic tasks in reinforcement learning is to learn an unknown and complex target probability distribution, which represents the reward density distribution following a set of policy strategies, by a tractable and plausible policy model probability distribution. And this target probability distribution can be multi-modal, which means it consists of multiple modes with full covariance matrices. After learning the policy model should describe the whole landscape of the reward density distribution. It is also necessary for the policy model after training to generate some new reliable samples and predict rewards on considering samples within specific policies. In short, it requires a policy model with high expressive power and the ability of both efficiently sampling from itself and accurately evaluating over the drawn samples. Many previous works make efforts to represent multi-modal reward density distributions [48, 22, 2, 50, 60], but some have extensive constructions and make it hard to generalize to new situations. As an alternative possibility, we leverage normalizing flow to meet the requirements. Normalizing flow transforms an initial simple probability distribution through a series of mappings to produce any richer and more multi-modal probability distributions. As many recent studies show [53, 35], the expressive power of this transformation is one of the greatest benefits of using normalizing flow. And this transformation makes it possible to use a prior probability distribution to represent an unknown target probability distribution so that the reliable sampling and exact evaluating can be done under the prior distribution and passed through normalizing flow. This problem can be characterized as an optimization problem with randomness and can be solved by stochastic optimization method [3]. As a subfield

of reinforcement learning, the policy search is aiming to search for an optimal policy model distribution using the strategy of updating parameters [10]. And the policy search algorithm can directly be applied to such an optimization problem if it can incorporate a well-defined additional entropy objective [2, 1]. Thus, we demonstrate a method to address this problem with a well-constructed policy model following the strategy of policy search. Note that the gradient based strategy can be more efficient for this approximation and lead to more robust and better optima [27, 25]. So we need to make sure that the gradient information is maintained in general situations, including sparse and discontinuous cases, when we are updating the policy model with the gradient method. It can be achieved by using a continuous surrogate to fit the data from discontinuous observed value pairs. When it cooperates with the policy search strategy, we can focus more on the policy search strategy with a more general constructed policy model with high expressive power. This ability makes the whole model more compact and elegant to train.

And this policy model can also be constructed under specific conditions from the observations. In other words, it can be constructed and treated as a posterior probability distribution given the evidence. When we find such a well-constructed and trained policy model, we can have all the information of the whole target distribution by the approximation of the policy model. And this method can also be sufficiently accurate and flexible to adapt to a new change in the environment.

The construction of this thesis is showing in the following paragraphs.

The Chapter 1 gives a brief introduction to our work. In the Chapter 2, we introduce an overview of related works to ours.

In the Chapter 3, we give the definition and properties of normalizing flow. Then, we explain the expressive power of it. Moreover, we show the basic construction of normalizing flow that we used in our policy model. And we discuss how to construct it.

In the Chapter 4, we transition into the policy search. First, we review the policy search and give the formulation of the problem using the insights from policy search. And also we transfer the maximization problem following the strategy of the policy model into an equivalent minimization problem. Second, we discuss the objective function and show details on how we compute a suitable objective function. Third, we propose the method to maintain the gradient information. Finally, we give a summary and the implementation of the algorithm.

In the Chapter 5, we give an overview of robotic grasping task, because it is a typical scenario of multi-modal in robotic tasks. We are applying our method in this realistic situation to evaluate whether it is qualified. In the beginning, a simplified problem formulation of the grasping task is introduced. And we start from the simulation environment. Thus, we build a simulation environment for the grasping task based on the PyBullet model.

In the Chapter 6, we evaluate our method in a number of experiments with different construction or training settings. What's more, we present the results and the analysis.

Chapter 7 gives the conclusion of this thesis and the outlook of the future work.

# 2. Related Work

To solve the problem of approximating an intractable distribution by a tractable distribution, a number of methods have been developed in recent years and gained a lot of success. In this chapter, we are going to briefly take a look at some methods in the related study fields and their relevant ideas for solving this kind of problem.

**Variational inference** is widely used to approximate probability densities by using the optimization strategy [28, 70, 3]. The main idea is to use a probability density from a set of candidates, which is simple enough for efficient optimization, to approximate the complex and unknown distribution. Regier et al. [61] develop a method for variational inference based on stochastic second-order trust-region optimization and the reparameterization trick. This stochastic optimization based method converges to a stationary point provably. Dhaka et al. [11] present a stochastic optimization based variational inference by viewing this optimization algorithm as producing a Markov chain. Miller et al. [48] demonstrate a practical variational inference method with an increasingly expressive approximation, which is able to learn rich, complex, and high-dimensional Gaussian mixture model. Guo et al. [22] also consider mixing more flexible approximating base distributions for the multi-modal target distribution, by successively adding and optimizing the components from the base family. Titsias et al. [67] focus on developing an unbiased variational inference. Arenz et al. [2] contribute an efficient and gradient-free method for variational inference to learn a multi-modal intractable target distribution using the insights from the policy search, which shows the policy search algorithm can be directly applied to variational inference if the objective is properly defined. On the contrary, the problem in the context of policy search can also be solved with the help of the variational inference.

Recently **variational autoencoder** has been wildly and successfully applied in the deep generative models, especially for handwritten digits, faces and images. The variational autoencoder can be viewed as two coupled but independently parameterized parts: the encoder or recognition model, and the decoder or generative model [31]. The encoder

provides regularised encoding distribution in the latent space, which has good properties allowing the decoder to generate some meaningful data. The variational autoencoder framework has been extended in many fields, e.g. to dynamical models by Johnson et al. [27], to models with multiple levels of stochastic latent variables by Kingma et al. [33]. Kingma et al. [32] introduce a method to use stochastic gradient descent to optimize the variational inference objective, which is relied on the estimator of the posterior from the variational autoencoder. Chung et al. [9], Gregor et al. [21] and Eslami et al. [15] develop variational autoencoder with more complex prior in the latent space for more complicated representations. The idea of variational autoencoder can also help to solve the problem, which is intractable from the observed space, with the help of the posterior from the latent space.

**Normalizing flow** provides a set of bijective transformations to map a simple and base distribution onto the arbitrary complex distribution, which shows a high power of expressive. And it can also be viewed as a generative model that produces the tractable distribution with both efficient sampling and exact density evaluation on samples. Ho et al. [24] applied normalizing flow in the image generation, Kumar et al. [39] employ it in the video generation and Esling et al. [16], Kim et al. [30], Prenger et al. [56] introduce normalizing flow based method in the audio generation. Moreover, the normalizing flow has also been applied in the policy search in reinforcement learning. Mazoure et al. [47] introduce a method combining soft actor-critic updates with a sequence of normalizing flows. Ward et al. [73] offer a normalizing flow based policy search within the actor-critic framework to learn more expressive classes. Touati et al. [68] leverage normalizing flow with deep Q-networks and deep deterministic policy gradient to achieve randomized value functions in high dimension for tracking a rich approximated posterior distribution over the parameters of the value function. Urain et al. [69] extend the normalizing flow to learn stable stochastic nonlinear dynamical systems. Thus, the normalizing flow provides wide possibilities for variational autoencoder to construct the expressive transformation from the latent space to the observed space with the abilities of both efficient sampling and exact evaluation over samples.

**Policy search** is an approach for improving the policy models by observing the information in the previous trials, whose update strategy can be widely used for optimization problems [10]. And this method usually requires a lot of data from trials, which causes inefficiency of learning. In the field of policy search, the objective function or its analytic gradient is usually not known, which is treated as *black box*. In such a complex context, some methods are proposed to solve this kind of optimization problem, e.g. black-box optimization, Bayesian optimization, directed exploration in the search space, and actor-critic

methods [64]. Recently, Mania et al. [46] propose the simplest **black-box optimization**, which refers to a gradient-free random search method. Genetic algorithms [20] has been successfully developed and applied as a kind of black-box optimization, and Such et al. [66] apply the genetic algorithm to the deep neural network based on the modern computational resources. Hansen et al. [23] present a popular method called covariance matrix adaptation evolution strategy. **Bayesian optimization** is well suited to optimize the maximization problem with an expensive to evaluate objective function [18]. It constructs a surrogate statistical model for the objective function, which is typically a Gaussian process [59], to provide a Bayesian posterior distribution given potential candidate new samples. And then it defines an acquisition function to decide where to sample for the next time. For each time the posterior distribution from the built surrogate is updated based on the new observations. Brochu et al. [7] and Lizotte et al. [43] have applied it in reinforcement learning. Snoek et al. [65] present a method associating the Bayesian optimization of hyperparameters with machine learning algorithms to show the great success for training deep neural networks. Klein et al. [34] propose an entropy search based Bayesian optimization method to collect globally good information about hyperparameters by extending additional input variables, such as the dataset size and the evaluation time, to accelerate the optimization speed. Eriksson et al. [14] adopt a local strategy for Bayesian optimization in high dimension, e.g. reinforcement learning problems with sparse reward, by using a collection of independent local surrogate models and apply multi-armed bandit strategy at each updating step to allocate samples from these local models, so to decide which local optimization runs to continue. Kandasamy et al. [29], Gardner et al. [19] and Wang et al. [71] are using an additive structure in the objective function and then exploiting it through Bayesian optimization. Wang et al. [72] and Nayebi et al. [52] are considering mapping the high dimension space into some low dimension subspace to reduce the number of the observations. **Novelty search** is given by Lehman et al. [40], and **quality-diversity** is introduced by Pugh et al. [57], both of which are directed exploration methods and particularly useful for sparse rewards. Peters et al. [55] apply **actor-critic methods** in the high dimension cases within the context of policy search. What's more, Abdolmaleki et al. [1] apply a surrogate-based stochastic search method, which allows a closed-form solution of information geometric trust regions. And the surrogate based stochastic search algorithms are able to give much smooth objective function [1]. Chatzilygeroudis et al. [8] apply prior knowledge on the policy structure to speed up the learning phase of policy search with fewer data.

And also the exploration-exploitation trade-off is the essential point in the policy search that needs to be addressed. **Multi-armed bandit** has been successfully used for the exploration-exploitation tradeoff in reinforcement learning, while a number of algorithms

are applied for solving this problem, e.g. $\epsilon$-greedy, Boltzmann exploration, upper confidence bound, and so on [37]. Maes et al. [44] propose a search algorithm as a multi-armed bandit problem over the search space. Papini et al. [54] treat the policy search problem as a multi-armed bandit and combine multiple importance sampling to estimate the expected return for exploiting the problem structure. Inside the field, Komiyama et al. [36] introduce a multiple-play multi-armed bandit and Fouché et al. [17] extend multiple-play with Thompson sampling.

**Grasping task** is a fundamental robotic task, which has gained increasingly more attention to be addressed by the data-driven based methods in machine learning. Lenz et al. [41] and Mahler et al. [45] works on the grasping prediction to estimate the probability of success based on the visual observation when a specific grasping setting is given. Yan et al. [75] propose a method to train a neural density model for approximating the posterior distribution of successful grasping based on visual observations. Morrison et al. [50] and Redmon et al. [60] focus more on the multi-modal problem in grasping tasks.

# 3. Normalizing Flow

Normalizing flow has attracted increasingly more attention in machine learning, because of its power of expressive and the ability of both sampling and evaluating. Normalizing flow is a general transformation of a simple and tractable distribution into a complex and intractable distribution by a sequence of diffeomorphism mappings [53]. This "diffeomorphism" has the meaning of invertible and differentiable. "Normalizing" means that a normalized distribution will be given after applying the diffeomorphism transformations. "Flow" means that a more complex distribution can be created based on the normalized distribution by applying the diffeomorphism transformations in the opposite direction.

In the following sections, we show the basic definition and properties of normalizing flow. Then, we transition into how normalizing flow provides the power of expressive. Finally, we discuss the autoregressive flow in detail, which is employed in our method.

## 3.1. Definition and Property

Normalizing flow is able to provide a flexible mapping between the observation space and the latent space over a continuous random variable.

Suppose $\boldsymbol{x}$ is a random variable in the observation space with a complex and intractable distribution $p(\boldsymbol{x})$, and $\boldsymbol{u}$ is a random variable in the latent space with a simple and tractable distribution $p(\boldsymbol{u})$. So the basic idea of normalizing flow is to express the random variable $\boldsymbol{x}$ as a transformation $\boldsymbol{T}$ of the random variable $\boldsymbol{u}$:

$$\boldsymbol{x} = \boldsymbol{T}(\boldsymbol{u})\,. \tag{3.1}$$

Since the transformation $\boldsymbol{T}$ is defined to be diffeomorphism, which also requires that $\boldsymbol{x}$ and $\boldsymbol{u}$ have the same dimension $D$ [49, 53], so we obtain the density of $\boldsymbol{x}$

$$p(\boldsymbol{x}) = p(\boldsymbol{u})|\det J_{\boldsymbol{T}}(\boldsymbol{u})|^{-1} \tag{3.2}$$

by a change of the variable [63, 4, 53], where $J_{\boldsymbol{T}}(\boldsymbol{u})$ is the Jacobian matrix $(D \times D)$ of the partial derivatives of the transformation $\boldsymbol{T}$ with respect to the variable $\boldsymbol{u}$

$$J_{\boldsymbol{T}}(\boldsymbol{u}) = \begin{bmatrix} \frac{\partial T_1}{\partial u_1} & \dots & \frac{\partial T_1}{\partial u_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial T_D}{\partial u_1} & \dots & \frac{\partial T_D}{\partial u_D} \end{bmatrix} . \tag{3.3}$$

Simultaneously, the transformation in the opposite direction can be given as

$$\boldsymbol{u} = \boldsymbol{T}^{-1}(\boldsymbol{x}) ,$$

and the probability density

$$p(\boldsymbol{u}) = p(\boldsymbol{x})|\det J_{\boldsymbol{T}^{-1}}(\boldsymbol{x})|^{-1} ,$$

where obviously we have $J_{\boldsymbol{T}^{-1}}(\boldsymbol{x})$ as the Jacobian matrix $(D \times D)$ of the partial derivatives of the inverse transformation $\boldsymbol{T}^{-1}$ with respect to the variable $\boldsymbol{x}$, similar to Equation (3.1) - (3.3).

One of the most important properties of diffeomorphism transformation is composability. That means the composition of a set of diffeomorphism transformations themselves is also diffeomorphism. Let $\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, \dots \boldsymbol{\tau}_N$ be a set of $N$ diffeomorphism transformations, and the composition of them is defined as

$$\boldsymbol{T} = \boldsymbol{\tau}_1 \circ \boldsymbol{\tau}_2 \circ \cdots \circ \boldsymbol{\tau}_N .$$

Then the inverse of the transformation $\boldsymbol{T}$ can be derived as

$$\boldsymbol{T}^{-1} = (\boldsymbol{\tau}_1 \circ \boldsymbol{\tau}_2 \circ \cdots \circ \boldsymbol{\tau}_N)^{-1} = {\boldsymbol{\tau}_N}^{-1} \circ {\boldsymbol{\tau}_{N-1}}^{-1} \circ \cdots \circ {\boldsymbol{\tau}_1}^{-1} .$$

And the determinant of the Jacobian matrix is given by

$$\det J_{\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, \dots \boldsymbol{\tau}_N}(\boldsymbol{u}) = \det J_{\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, \dots \boldsymbol{\tau}_{N-1}}(\boldsymbol{u}) \cdot \det J_{\boldsymbol{\tau}_N}(\boldsymbol{u}) .$$

In consequence, the complex transformation from a distribution $p(\boldsymbol{u})$ into another $p(\boldsymbol{x})$ can be built by composing a series of simpler diffeomorphism mappings, without losing

the diffeomorphism property, without losing the ability to calculate the density of the distributions by the determinant of the Jacobian matrix.

Normalizing flow can provide the operations of both sampling and evaluating the density. We can take the simple and tractable distribution $p(\boldsymbol{u})$ as a known prior distribution, so the samples from the prior distribution with their probability densities can be transferred from the latent space to the observation space using forward "flow" transformation. When we have some observed samples from the intractable distribution $p(\boldsymbol{x})$, it is also possible to evaluate their probability densities by reverse "normalizing" transformation. At first, we let the samples $\boldsymbol{x}$ with initial densities $p_0(\boldsymbol{x}) = \mathbf{1}$ be transformed from the observation space into the latent space by the function $\boldsymbol{T}^{-1}$. As a result of that, we will obtain the normalized samples $\boldsymbol{u} = \boldsymbol{T}^{-1}(\boldsymbol{x})$, and the determinant $\det J_{\boldsymbol{T}^{-1}}(\boldsymbol{x})$. Then, we evaluate the probability densities $p(\boldsymbol{u})$ on the normalized samples $\boldsymbol{u}$ with the prior distribution. Finally, the probability densities $p(\boldsymbol{x})$ can be calculated by the multiplication of the determinant $\det J_{\boldsymbol{T}^{-1}}(\boldsymbol{x})$, and this evaluated densities $p(\boldsymbol{u})$.

In practice, the transformation $\boldsymbol{T}$ can be chosen flexibly by a chain of multiple simple mappings, which can be implemented typically by neural networks.

## 3.2. Power of Expressive

Normalizing flow is sufficiently expressive to represent any interesting distributions by the simple base distribution, which can be seen as one of the greatest benefits of normalizing flow based models. We recall the proof of this universal representation under reasonable conditions on $p(\boldsymbol{x})$, and we recommend readers to read these papers [53, 26, 5] for more formal details.

Suppose $\boldsymbol{x} \in \mathbb{R}^D$ and $\boldsymbol{x}' \in \mathbb{R}^D$, assume $x_i$ is the $i$-th element and $\boldsymbol{x}_{<i}$ represents the first $(i-1)$-th elements of the random variable vector $\boldsymbol{x}$. Then, following [26, 5] let a transformation $\boldsymbol{\tau} : \mathbb{R}^D \to \mathbb{R}^D$ be *triangular*, which means the derivative of $\boldsymbol{\tau}$ is a triangular matrix if for all $i \in \{1, \dots D\}$ the following formula as reasonable conditions

$$x_i' = \tau_i(x_i, \boldsymbol{x}_{<i}) \tag{3.4}$$

is satisfied. What's more, the transformation $\boldsymbol{\tau}$ is *increasing*, when the $\tau_i$ is increasing with respect to the $x_i$ for all $i \in \{1, \dots D\}$ while $\boldsymbol{x}_{<i}$ fixed. And increasing triangular

transformations are universal, i.e. any density distribution can be transformed into any target density distribution by increasing triangular transformations. In consequence, we give the theorem directly: for any two distributions $p(\boldsymbol{x})$ and $p(\boldsymbol{x}')$ over the random variables $\boldsymbol{x}$ and $\boldsymbol{x}'$, an increasing triangular mapping between the random variables

$$\boldsymbol{x}' = \boldsymbol{\tau}(\boldsymbol{x})$$

exists, so that the probability density functions between them

$$p(\boldsymbol{x}') = p(\boldsymbol{x})|\det J_{\boldsymbol{\tau}}(\boldsymbol{x})|^{-1}$$

always holds. And since the Jacobian matrix of this $\boldsymbol{\tau}$ is given as a low triangular matrix, its determinant can be obtained as the product of its diagonal elements

$$\det J_{\boldsymbol{\tau}}(\boldsymbol{x}) = \prod_{i=1}^{D} \frac{\partial \tau_i}{\partial x_i} \,. \tag{3.5}$$

And the $\boldsymbol{\tau}$ is diffeomorphism, because of the inverse function theorem [62]. Therefore, the normalizing flow shows its expressive power and is able to approximate any interesting target distribution by a base simple distribution.

## 3.3. Autoregressive Flow

Autoregressive flow is a universal approximation to represent arbitrarily complex intractable distributions, whose basic idea is to directly implement the construction following the Section 3.2. As it showed in the Equation (3.4), the $i$-th element of transformed random variable $\boldsymbol{x}'$ only depends on the first $i$-th elements of the previous random variable before transformation $\boldsymbol{x}$. Hence, it can be rewritten as

$$x'_i = \tau_i(x_i; \boldsymbol{h}_i) \;\; where \;\; \boldsymbol{h}_i = c_i(\boldsymbol{x}_{<i}) \,,$$

which means that the $i$-th transformation function $\tau_i$ is parameterized by the vector $\boldsymbol{h}_i$ given by the $i$-th conditioner $c_i$. And this conditioner $c_i$ can only accept the elements of random variable $\boldsymbol{x}$ with dimension indices less than $i$ as input. As a result of that, the random variable $\boldsymbol{x}$ is able to determine the behavior of the transformations acting on itself. So it keeps all the necessary information from $\boldsymbol{x}$ after performing the transformation.

It is obvious that the inverse transformation can be obtained as

$$x_i = {\tau_i}^{-1}(x_i'; \boldsymbol{h}_i) \;\; where \;\; \boldsymbol{h}_i = c_i(\boldsymbol{x}_{<i}) \,.$$

However, the vector $\boldsymbol{x}_{<i}$ needs to be computed before the $i$-the element $x_i$, so that the conditioner has an available input to compute the parameter vector $\boldsymbol{h}_i$.

Since the determinant of the lower triangular Jacobian matrix of transformation $\boldsymbol{\tau}$ with respect to the random variable $\boldsymbol{x}$ is equal to the product of its diagonal elements, the log-absolute-determinant can be derived from Equation (3.5) as

$$\log \left|\det J_{\boldsymbol{\tau}}(\boldsymbol{x})\right| = \log \left| \prod_{i=1}^{D} \frac{\partial \tau_i}{\partial x_i} \right| = \sum_{i=1}^{D} \log \frac{\partial \tau_i}{\partial x_i} \,, \tag{3.6}$$

which can be easily and efficiently computed while using log-probability. The log-absolute-determinant in the reversed direction has no difference to Equation (3.6), but just to change the variable $\boldsymbol{\tau}$ with $\boldsymbol{\tau}^{-1}$.

### 3.3.1. Coupling Layer

Coupling layer based construction of normalizing flow is a class of autoregressive flow. The candidate vector waiting to be transformed is divided into two parts, and the first part is used to parameterize the transformation of the second part.

Let $\boldsymbol{x} \in \mathbb{R}^D$ be a random variable. Suppose $d$ denotes the index to divide these two parts, where $d < D$ is satisfied, so that we obtain $\boldsymbol{x} = [\boldsymbol{x}_{\leq d}, \boldsymbol{x}_{>d}]^T$. Assume $\boldsymbol{\tau}$ is the complete transformation, $\boldsymbol{c}$ is the complete conditioner to the transformation, where the first part coupling transform with first part coupling conditioner are constants, i.e. not a function of $\boldsymbol{x}$, the second part coupling transform with second part coupling conditioner are given as $\boldsymbol{\omega}$ with $\boldsymbol{m}$. Then the transformation based on coupling layer is given by

$$\boldsymbol{x}' = \boldsymbol{\tau}(\boldsymbol{x}; \boldsymbol{c}(\boldsymbol{x})) = \begin{cases} \boldsymbol{x}'_{\leq d} = \boldsymbol{x}_{\leq d} \\ \boldsymbol{x}'_{>d} = \boldsymbol{\omega}(\boldsymbol{x}_{>d}; \boldsymbol{m}(\boldsymbol{x}_{\leq d})) \,. \end{cases}$$

In contrast, the inverse transformation can be easily obtained by

$$\boldsymbol{x} = \boldsymbol{\tau}^{-1}(\boldsymbol{x}'; \boldsymbol{c}(\boldsymbol{x})) = \begin{cases} \boldsymbol{x}_{\leq d} = \boldsymbol{x}'_{\leq d} \\ \boldsymbol{x}_{>d} = \boldsymbol{\omega}^{-1}(\boldsymbol{x}'_{>d}; \boldsymbol{m}(\boldsymbol{x}_{\leq d})) \,. \end{cases}$$

As for the evaluation of probability density functions, the Equation (3.6) can be applied. And we here only focus on the forward transformation. The Jacobian matrix of coupling layer based complete transformation can be given as a block lower triangular matrix

$$J_{\boldsymbol{\tau}}(\boldsymbol{x}) = \frac{\partial \boldsymbol{\tau}(\boldsymbol{x}; \boldsymbol{c}(\boldsymbol{x}))}{\partial \boldsymbol{x}^T} = \begin{bmatrix} \boldsymbol{I}_d & \boldsymbol{0} \\ \frac{\partial \boldsymbol{\omega}(\partial \boldsymbol{x}_{>d}; \boldsymbol{m}(\boldsymbol{x}_{\leq d}))}{\partial (\boldsymbol{x}_{\leq d})^T} & \frac{\partial \boldsymbol{\omega}(\boldsymbol{x}_{>d}; \boldsymbol{m}(\boldsymbol{x}_{\leq d}))}{\partial (\boldsymbol{x}_{>d})^T} \end{bmatrix}, \tag{3.7}$$

where $\boldsymbol{I}_d$ is a $(d \times d)$ identity matrix, whose determinant is $1$. Thus, the Jacobian matrix on the lower left in the Equation (3.7) does not appear while computing determinant. Then, we focus on the Jacobian matrix on the lower right in the Equation (3.7). Following [13, 51], the low right block in the Equation (3.7) can be given as

$$\frac{\partial \boldsymbol{\omega}(\boldsymbol{x}_{>d}; \boldsymbol{m}(\boldsymbol{x}_{\leq d}))}{\partial (\boldsymbol{x}_{>d})^T} = \begin{bmatrix} q_{d+1}(x_{>d}^{d+1}) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & q_D(x_{>d}^D) \end{bmatrix}, \tag{3.8}$$

where we use

$$q_i(x_{>d}^i) = \frac{\partial \omega_i(x_{>d}^i; m_i(\boldsymbol{x}_{\leq d}))}{\partial x_{>d}^i} \tag{3.9}$$

for all $i = d+1, \dots D$. Thus, the log-absolute-determinant can be obtained by

$$\log |\det J_{\boldsymbol{\tau}}(\boldsymbol{x})| = \log \left| \prod_{i=d+1}^{D} q_i(x_{>d}^i) \right| = \sum_{i=d+1}^{D} \log q_i(x_{>d}^i) \,. \tag{3.10}$$

The transformation $\boldsymbol{\tau}$ can be implemented by an arbitrary neural network.

### 3.3.2. Neural Spline

The coupling layer based transformation has its limitations in the higher dimension cases, while the simple affine transform is used as coupling transform. In the high dimension cases, it requires more layers to provide sufficient expressive power, which decreases the efficiency and increases the costs. Spline based transformation is a possible solution to address this limitation, especially in the higher dimension cases [51]. The spline is a piecewise-polynomial or a piecewise-rational function, which passes a set of specified points and keeps monotonic inside all intervals between these points. There are a

few options of spline based transformation to use: piecewise-linear, piecewise-quadratic, piecewise-cubic, and piecewise-rational-quadratic splines. We begin with the simplest option and then transition into other more complex options.

Recall that we partition the $D$-dimensional random variable $\boldsymbol{x}$ into two parts as $\boldsymbol{x} = [\boldsymbol{x}_{\leq d}, \boldsymbol{x}_{>d}]^T$ for $d < D$. And the second part coupling transform is given as $\boldsymbol{\omega}$ with the second part coupling conditioner $\boldsymbol{m}$.

For piecewise-linear spline, we divide the part $\boldsymbol{x}_{>d}$ into $k$ bins with equal width $w$. Let the $((D-d) \times k)$ matrix $\hat{\boldsymbol{Q}}$ to denote the condition $\boldsymbol{m}(\boldsymbol{x}_{\leq d})$, whose each row defines the unnormalized probability mass function in the corresponding dimension in the part $\boldsymbol{x}_{>d}$. Then, we normalize each row of the matrix $\hat{\boldsymbol{Q}}$ by the softmax function, and obtain the normalized matrix $\boldsymbol{Q}$. Thus, the probability density function in the $i$-th dimension is derived as

$$q_i(x^i_{>d}) = \frac{Q_{ib}}{w} \tag{3.11}$$

with the bin $b = \lfloor k x^i_{>d} \rfloor$ that contains the scalar value $x^i_{>d}$ for all $i = d+1, \dots D$. And the $i$-th dimension coupling transform can be obtained by the integral of the probability density function in the $i$-th dimension:

$$\omega_i(x^i_{>d}; m_i(\boldsymbol{x}_{\leq d})) = \int_0^{x^i_{>d}} q_i(t)\, dt = \alpha Q_{ib} + \sum_{k=1}^{b-1} Q_{ik}\,, \tag{3.12}$$

where $\alpha = k x^i_{>d} - \lfloor k x^i_{>d} \rfloor$ denotes the relative position of $x^i_{>d}$ in bin $b$. It is obvious that the Equation (3.12) is the integral form of Equation (3.9). Finally, we obtain the log-absolute-determinant as

$$\log |\det J_{\boldsymbol{\tau}}(\boldsymbol{x})| = \log \left| \prod_{i=d+1}^{D} q_i(x^i_{>d}) \right| = \log \left| \prod_{i=d+1}^{D} \frac{Q_{ib}}{w} \right| = \sum_{i=d+1}^{D} \log \frac{Q_{ib}}{w}\,,$$

by the combination of the Equation (3.10) and (3.11).

Piecewise-quadratic spline admits a probability density function for piecewise-linear spline, which uses $k+1$ vertices. Let the $\hat{\boldsymbol{V}}$ be the unnormalized $((D-d) \times (k+1))$ matrix that stores the vertical coordinates of these vertices, and the $\hat{\boldsymbol{W}}$ be the unnormalized $((D-d) \times k)$ matrix that stores the horizontal differences between the neighboring vertices. These two matrices are given by the second part coupling conditioner $\boldsymbol{m}(\boldsymbol{x}_{\leq d})$. Again

we normalize each row of the matrix $\hat{\boldsymbol{W}}$ by the softmax function to obtain $\boldsymbol{W}$. As for normalizing the matrix $\hat{\boldsymbol{V}}$, we use

$$V_{ij} = \frac{\exp\left(\hat{V}_{ij}\right)}{\sum_{k=1}^{K} \frac{\exp\left(\hat{V}_{ik}\right)+\exp\left(\hat{V}_{i\,k+1}\right)}{2} W_{ik}} ,$$

where the denominator makes sure that the $i$-th row of the matrix $\boldsymbol{V}$ represents a valid probability density function [51]. Then, the probability density function can be defined as

$$q_i(x^i_{>d}) = lerp(V_{ib}, V_{i\,b+1}, \alpha) ,$$

where $\alpha = \left(x^i_{>d} - \sum_{k=1}^{b-1} W_{ik}\right) / W_{ib}$ indicates the relative position of the $x^i_{>d}$ in bin $b$. Thus, the $i$-th dimension coupling transform is given as

$$\omega_i(x^i_{>d}; m_i(\boldsymbol{x}_{\leq d})) = \frac{\alpha^2}{2} \left(V_{i\,b+1} - V_{ib}\right) W_{ib} + \alpha V_{ib} W_{ib} + \sum_{k=1}^{b-1} \frac{V_{ik} + V_{i\,k+1}}{2} W_{ik} .$$

And the log-absolute-determinant can be obtained by applying the Equation (3.10), but with interpolating the entries of matrix $\boldsymbol{V}$. The piecewise-quadratic spline based transformation provides better accuracy, efficiency and robustness[51].

# 4. Policy Search

Policy search is employed to directly optimize the parameters of a given policy model. It is well suited in the high dimension cases since it can be gracefully scaled with dimensionality and show the guarantees of convergence [55, 10, 42]. And policy search can be classified into model-free and model-based. They are both based on either policy gradients, expectation maximization based updates, or information theoretic insights [10]. Model-free policy search is used to generate the "samples", or the trajectories, by the interactions with the real robot or robot physical simulation environment, while model-based policy search uses a learned internal simulator of dynamics of the robot from the observed data, i.e., the policy model provides the prediction about how the robot and its environment would behave following the current learned policy model, to help to generate the "samples". Thus, model-based policy search requires fewer interactions with the robot and promises to generate unforeseen situations efficiently from the policy model learned from observed data.

We employ the idea of model-based policy search in this paper, which can be solved by maximization of the reward following the policy. In the following sections, we introduce the model-based policy search and give a strategy to transfer this maximization problem to be an equivalent minimization problem. Then, we discuss the objective functions for the minimization problem and how these can be used in our method. At last, we show how to compute the gradient for updating the parameters of the policy model during the training phase. And we give a summary of policy search using normalizing flow at the end of this chapter.

## 4.1. Model-Based Policy Search

The model-based policy search requires a policy model with the power of expressive so that the policy model is able to learn arbitrary complex interesting target distribution based on the gradient descent update strategy. Since the normalizing flow can provide two main benefits: a) high expressive power to represent any distribution, b) the ability to generate samples and evaluate over samples, we use the policy model based on normalizing flow in our policy search task.

When we use the insights from policy search, this optimization problem can be solve by maximizing the reward following the policy. Assuming that $\boldsymbol{x} \in \mathbb{R}^D$ is a random variable in $D$ dimension space, $q(\boldsymbol{x}) = \tilde{q}(\boldsymbol{x})/Z$ is a normalized target distribution of an unnormalized target distribution $\tilde{q}(\boldsymbol{x})$ with normalization constant $Z = \int_{\boldsymbol{x}} \tilde{q}(\boldsymbol{x})\, d\boldsymbol{x}$. And then we define the reward as $r(\boldsymbol{x}) = \log \tilde{q}(\boldsymbol{x})$. As for the objective function for this policy search problem, we define it as

$$\boldsymbol{\mathcal{R}} = \int_{\boldsymbol{x}} p(\boldsymbol{x}) r(\boldsymbol{x})\, d\boldsymbol{x} - \int_{\boldsymbol{x}} p(\boldsymbol{x}) \log p(\boldsymbol{x})\, d\boldsymbol{x}\,,$$

where $\mathcal{H}(\boldsymbol{x}) = -\int_{\boldsymbol{x}} p(\boldsymbol{x}) \log p(\boldsymbol{x})\, d\boldsymbol{x}$ is an additive entropy, and $p(\boldsymbol{x})$ represents policy model. The objective function $\boldsymbol{\mathcal{R}}$ denotes the reward following the policy and entropy of the policy self. Therefore, the reward maximization problem is described as

$$p^*(\boldsymbol{x}) = \arg\max_{p(\boldsymbol{x})} \boldsymbol{\mathcal{R}}\,.$$

If we go one step further, so the objective function $\boldsymbol{\mathcal{R}}$ can be rewritten as

$$\boldsymbol{\mathcal{R}} = -\int_{\boldsymbol{x}} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{\exp(r(\boldsymbol{x}))}\, d\boldsymbol{x}\,.$$

Actually, the rewritten form of the objective function $\boldsymbol{\mathcal{R}}$ is nothing else but the negative value of the Kullback-Leibler Divergence. Thus, the maximization of the objective function $\boldsymbol{\mathcal{R}}$ is equivalent to the minimization of the Kullback-Leibler Divergence. The Kullback-Leibler Divergence is usually used to measure the distance between two distributions. As a result of that, if we properly define the objective function in the policy search, we can solve this optimization problem by another equivalent way to find an optimal policy model distribution, which is closest or most similar to the target distribution. And we use $\boldsymbol{\mathcal{L}}$ to denote this kind of objective function used to measure how similar between the policy model distributions and target distribution, which can be Kullback Leibler Divergence,

Jensen Shannon Divergence, and so on. Finally, it can be described as to search the optimal policy model $p^*(\boldsymbol{x})$ so that

$$p^*(\boldsymbol{x}) = \arg\min_{p(\boldsymbol{x})} \boldsymbol{\mathcal{L}}(\tilde{q}(\boldsymbol{x}), p(\boldsymbol{x}))\,. \tag{4.1}$$

And assuming that distribution $p(\boldsymbol{x};\boldsymbol{\theta})$ is with respect to the parameter vector $\boldsymbol{\theta}$ so that searching for an optimal policy model distribution $p^*(\boldsymbol{x})$ is equivalent to searching for an optimal parameter vector $\boldsymbol{\theta}^*$. Therefore the Equation (4.1) can be given as

$$p(\boldsymbol{x};\boldsymbol{\theta}^*) = \arg\min_{p(\boldsymbol{x};\boldsymbol{\theta})} \boldsymbol{\mathcal{L}}(\tilde{q}(\boldsymbol{x}), p(\boldsymbol{x};\boldsymbol{\theta}))\,. \tag{4.2}$$

The optimal distribution $p(\boldsymbol{x};\boldsymbol{\theta}^*)$ can be derived using a gradient-based algorithm, following the gradient descent with respect to parameter vector $\boldsymbol{\theta}$. And the parameter vector $\boldsymbol{\theta}$ updating strategy is given by

$$\boldsymbol{\theta} \leftarrow \lambda \nabla_{\boldsymbol{\theta}} \boldsymbol{\mathcal{L}}(\tilde{q}(\boldsymbol{x}), p(\boldsymbol{x};\boldsymbol{\theta}))\,, \tag{4.3}$$

where $\lambda$ is the step size, until it reaches the optimum of parameter vector $\boldsymbol{\theta}^*$. What can not be ignored is that the reparameterization trick is required for computing the gradient related to random variables in the distributions. More details about reparameterization trick are presented in Appendix A. At last, we take intractable distribution $p(\boldsymbol{x};\boldsymbol{\theta}^*)$ to approximate target distribution $\tilde{q}(\boldsymbol{x})$.

## 4.2. I-projection and M-projection

The similarity between two distributions can be measured by Kullback–Leibler (KL) Divergence [38], which can also be called relative entropy. Given two distributions $p_1(x)$ and $p_2(x)$, it is defined as

$$D_{KL}\left[p_1(x) \parallel p_2(x)\right] = \int_x p_1(x) \log\left(\frac{p_1(x)}{p_2(x)}\right) dx\,.$$

The KL Divergence is always non-negative, with zero if and only if $p_1 = p_2$ almost everywhere. It is natural to use KL Divergence as the objective function in the application of policy search problems.

Since KL Divergence is asymmetric, it provides two forms to use in the optimization problem. One is called information-projection or I-projection, the other is called moment-projection or M-projection. In the following sections, we particularly focus on these two divergences, and we use the notations already defined in the Section 4.1.

### 4.2.1. I-projection

I-projection is sometimes also called reverse KL Divergence, which is defined between the target distribution $\tilde{q}(\boldsymbol{x})$ and policy model $p(\boldsymbol{x})$ as

$$\begin{aligned} D_{KL}\left[p(\boldsymbol{x}) \parallel \tilde{q}(\boldsymbol{x})\right] &= \int_{\boldsymbol{x}\sim p(\boldsymbol{x})} p(\boldsymbol{x}) \log\left(\frac{p(\boldsymbol{x})}{\tilde{q}(\boldsymbol{x})}\right) d\boldsymbol{x} \\ &= \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left[\log p(\boldsymbol{x}) - \log \tilde{q}(\boldsymbol{x})\right] , \end{aligned}$$

where the expectation over samples from the policy model can be estimated by Monte Carlo as

$$\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left[\log p(\boldsymbol{x}) - \log \tilde{q}(\boldsymbol{x})\right] \approx \frac{1}{N}\sum_{n=1}^{N} \log p(\boldsymbol{x}) - \log \tilde{q}(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim p(\boldsymbol{x})} .$$

Then, the gradient of reverse KL Divergence with respect to parameters $\boldsymbol{\theta}$ is estimated by

$$\nabla_{\boldsymbol{\theta}} D_{KL}\left[p(\boldsymbol{x}) \parallel \tilde{q}(\boldsymbol{x})\right] \approx \frac{1}{N}\sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}) - \nabla_{\boldsymbol{\theta}} \log \tilde{q}(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim p(\boldsymbol{x})} .$$

Following the Equation (4.2), the optimization problem to fit the policy model $p(\boldsymbol{x})$ with respect to parameters $\boldsymbol{\theta}$ to the target distribution $\tilde{q}(\boldsymbol{x})$ can be solved by

$$p(\boldsymbol{x};\boldsymbol{\theta}^*) = \underset{p(\boldsymbol{x};\boldsymbol{\theta})}{\arg\min}\, D_{KL}\left[p(\boldsymbol{x};\boldsymbol{\theta}) \parallel \tilde{q}(\boldsymbol{x})\right]$$

with gradient-based methods in practice, when the I-projection is used as an objective function. Thus, the parameters $\boldsymbol{\theta}$ is updated based on the Equation (4.3).

However, using I-projection typically underestimates the target distribution and locks onto one of the modes of the target distribution. In other words, the reverse KL Divergence ignores the target distribution $\tilde{q}(\boldsymbol{x}) > 0$ for some $\boldsymbol{x}$, where $p(\boldsymbol{x}) = 0$, without any penalty. Hence, the I-projection tries to avoid spreading the approximation, which is known as *zero forcing*. To address this problem, we add M-projection into the objective function to contribute the extra information about the target distribution, to avoid the policy model locking onto only one mode of the target distribution.

### 4.2.2. M-projection

M-projection is sometimes also known as forward KL Divergence. However, the situation for applying the M-projection is different to the above assumptions in Section 4.1. In contrast, the target distribution $\tilde{q}(\boldsymbol{x})$ needs to have the ability to generate samples, or the samples are given, that can be used to approximate the sampling from target distribution $\tilde{q}(\boldsymbol{x})$. Hence, we employ a buffer to store the samples from the policy model that are evaluated to have the best rewards under the target distribution $\tilde{q}(\boldsymbol{x})$. Assume the buffer is already well designed to approximate the sampling from target distribution, and we demonstrate how to build the buffer in Section 4.4 in more detail.

And the M-projection is defined between the target distribution $\tilde{q}(\boldsymbol{x})$ and policy model $p(\boldsymbol{x})$ as

$$\begin{aligned} D_{KL}\left[\tilde{q}(\boldsymbol{x}) \parallel p(\boldsymbol{x})\right] &= \int_{\boldsymbol{x}\sim\tilde{q}(\boldsymbol{x})} \tilde{q}(\boldsymbol{x}) \log\left(\frac{\tilde{q}(\boldsymbol{x})}{p(\boldsymbol{x})}\right) d\boldsymbol{x} \\ &= \mathbb{E}_{\boldsymbol{x}\sim\tilde{q}(\boldsymbol{x})}\left[\log \tilde{q}(\boldsymbol{x}) - \log p(\boldsymbol{x})\right] , \end{aligned}$$

where $\boldsymbol{x} \sim \tilde{q}(\boldsymbol{x})$ represents the estimation of sampling from the target distribution. Again the expectation over the these samples can be estimated by Monte Carlo as

$$\mathbb{E}_{\boldsymbol{x}\sim\tilde{q}(\boldsymbol{x})}\left[\log \tilde{q}(\boldsymbol{x}) - \log p(\boldsymbol{x})\right] \approx \frac{1}{M}\sum_{n=1}^{M} \log \tilde{q}(\boldsymbol{x}) - \log p(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim\tilde{q}(\boldsymbol{x})} .$$

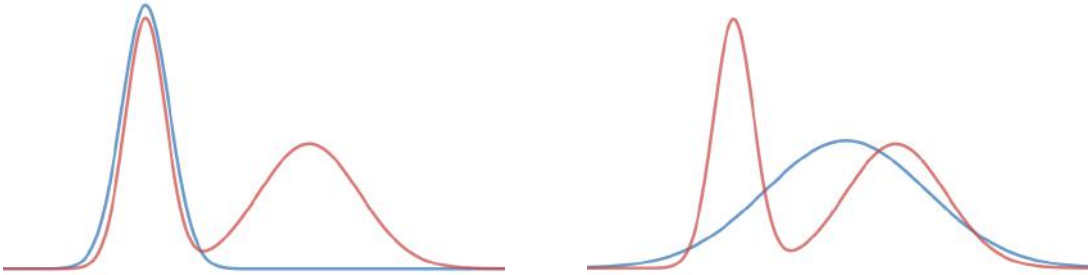
Since the target distribution $\tilde{q}(\boldsymbol{x})$ and its sampling $\boldsymbol{x} \sim \tilde{q}(\boldsymbol{x})$ are independent on the parameters $\boldsymbol{\theta}$, the Monte Carlo estimation for the gradient of forward KL Divergence with respect to the parameter $\boldsymbol{\theta}$ is given by

$$\nabla_{\boldsymbol{\theta}} D_{KL}\left[\tilde{q}(\boldsymbol{x}) \parallel p(\boldsymbol{x})\right] \approx \frac{1}{M}\sum_{n=1}^{M} -\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim\tilde{q}(\boldsymbol{x})} .$$

M-projection has a property named *zero avoiding*, which means the forward KL Divergence only notices all the $\boldsymbol{x}$ as long as $\tilde{q}(\boldsymbol{x}) > 0$, but is not affected by those $\boldsymbol{x}$ with $\tilde{q}(\boldsymbol{x}) = 0$ at all, even if on which the $p(\boldsymbol{x})$ is really different from the $\tilde{q}(\boldsymbol{x})$. Hence, M-projection overestimates the target distribution.

### 4.2.3. Combination of I-projection and M-projection

Neither I-projection alone, either M-projection alone is satisfied with the requirements in practice, because one is always underestimating, while the other is always overesti-



(a) Information projection (b) Moment projection

Figure 4.1.: Information projection and moment projection of a target Gaussian mixture model (red) with two components onto the set of Gaussians (blue). Information projection underestimates the target distribution and then locks onto one mode of the target distribution, while the moment projection overestimating the target distribution and then becoming too spread.

mating. The Figure 4.1 gives a more intuitive display of the behaviors of I-projection and M-projection.

Thus, we leverage both I-projection and M-projection in the objective function, using a weight to control the contribution of these two parts. As a result of that, the optimization problem is described as

$$p(\boldsymbol{x};\boldsymbol{\theta}^*) = \underset{p(\boldsymbol{x};\boldsymbol{\theta})}{\arg\min}\, D_{KL}\left[p(\boldsymbol{x};\boldsymbol{\theta}) \parallel \tilde{q}(\boldsymbol{x})\right] + \beta D_{KL}\left[\tilde{q}(\boldsymbol{x}) \parallel p(\boldsymbol{x};\boldsymbol{\theta})\right] ,$$

where $\beta$ denotes the weight of M-projection, with the objective estimated by Monte Carlo as follows:

$$\begin{aligned} \boldsymbol{\mathcal{L}}(\tilde{q}(\boldsymbol{x}), p(\boldsymbol{x};\boldsymbol{\theta})) &= D_{KL}\left[p(\boldsymbol{x};\boldsymbol{\theta}) \parallel \tilde{q}(\boldsymbol{x})\right] + \beta D_{KL}\left[\tilde{q}(\boldsymbol{x}) \parallel p(\boldsymbol{x};\boldsymbol{\theta})\right] \\ &\approx \frac{1}{N}\sum_{n=1}^{N} \log p(\boldsymbol{x}) - \log \tilde{q}(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim p(\boldsymbol{x})} + \\ &\quad \beta \frac{1}{M}\sum_{n=1}^{M} \log \tilde{q}(\boldsymbol{x}) - \log p(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim \tilde{q}(\boldsymbol{x})} . \end{aligned} \tag{4.4}$$

Then, the gradient of $\boldsymbol{\mathcal{L}}$ with respect to $\boldsymbol{\theta}$ is given as

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\boldsymbol{\mathcal{L}}(\tilde{q}(\boldsymbol{x}), p(\boldsymbol{x};\boldsymbol{\theta})) &= \nabla_{\boldsymbol{\theta}} D_{KL}\left[p(\boldsymbol{x}) \parallel \tilde{q}(\boldsymbol{x})\right] + \beta\nabla_{\boldsymbol{\theta}} D_{KL}\left[\tilde{q}(\boldsymbol{x}) \parallel p(\boldsymbol{x})\right] \\ &\approx \frac{1}{N}\sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}) - \nabla_{\boldsymbol{\theta}} \log \tilde{q}(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim p(\boldsymbol{x})} + \\ &\quad \beta\frac{1}{M}\sum_{n=1}^{M} -\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim \tilde{q}(\boldsymbol{x})} \ .\end{aligned} \tag{4.5}$$

Note that the gradient should be computed over the samples generated from the distribution, we use reparameterization trick to achieve this gradient computing. And when we use normalizing flow based policy model, the reparameterization trick is done in the latent space, and then this gradient computing is passed by the normalizing flow to the observed space, where the policy model is defined. Finally, the Equation (4.3) is applied to optimize the parameters $\boldsymbol{\theta}$, when the gradient is estimated.

## 4.3. Gradient Maintaining for Policy Search

As mentioned in the Section 4.1, the target distribution only provides the evaluation of the probability density values on given observed samples. But a set of discrete observed samples can not give any information about the gradient, on which the policy search method relies to update the parameters. Especially, the gradient is not always existed in the cases of sparse rewards. Thus, a continuous function $f(\boldsymbol{x})$ is employed to approximate the target distribution by fitting these observed samples so that it can provide the gradient almost everywhere. Sometimes the gradient could become too steep in the local region that the gradient-based algorithm is not able to keep stable anymore. Therefore, we try to maintain the gradient by smoothing the approximate function $f(\boldsymbol{x})$.

Given a set of observed samples with the rewards evaluated under target distribution, i.e. the $\log$-value of probability densities $(\boldsymbol{x}_1, r_1), (\boldsymbol{x}_2, r_2), \ldots (\boldsymbol{x}_k, r_k)$, it is intuitive to smooth them by scaling the rewards $r_1, r_2, \ldots r_k$ with a weight smaller than 1. And we choose the weight as the reciprocal of the deviation of these reward values $1/\sigma_k$, if the deviation $\sigma_k > 1$, so give scaled reward values as

$$\hat{r}_i = \begin{cases} r_i/\sigma_k & \sigma_k > 1 \\ r_i & \sigma_k \leq 1 \end{cases} \quad for\ i = 1, 2, \ldots k\,,$$

Then, we leverage the $f(\boldsymbol{x})$ to fit the normalized observed samples $(\boldsymbol{x}_1, \hat{r}_1), (\boldsymbol{x}_2, \hat{r}_2), \ldots (\boldsymbol{x}_k, \hat{r}_k)$, denoted as follows:

$$\log \tilde{q}(\boldsymbol{x}) \approx f(\boldsymbol{x}) \mid_{(\boldsymbol{x}_1,\hat{r}_1),(\boldsymbol{x}_2,\hat{r}_2),\ldots(\boldsymbol{x}_k,\hat{r}_k)} \quad for\ all\ \boldsymbol{x} \in \mathbb{R}^D ,$$

where the $f(\boldsymbol{x})$ can be implemented with neural network in practice. Finally, we use the approximated rewards to compute the objective function and its gradient. The Monte Carlo estimation of I-projection in the objective function from the Equation (4.4) and (4.5) is given as

$$D_{KL}\left[p(\boldsymbol{x};\boldsymbol{\theta}) \parallel \tilde{q}(\boldsymbol{x})\right] \approx \frac{1}{N}\sum_{n=1}^{N} \log p(\boldsymbol{x}) - \log \tilde{q}(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim p(\boldsymbol{x})} ,$$

so we apply the approximation of the target distribution $f(\boldsymbol{x})$ to obtain

$$\frac{1}{N}\sum_{n=1}^{N} \log p(\boldsymbol{x}) - \log \tilde{q}(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim p(\boldsymbol{x})} \approx \frac{1}{N}\sum_{n=1}^{N} \log p(\boldsymbol{x}) - f(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim p(\boldsymbol{x})} .$$

Since the rewards are approximated by the $f(\boldsymbol{x})$ to show the gradient, we must train the $f(\boldsymbol{x})$ before training the policy model to approach the target distribution and make sure the $f(\boldsymbol{x})$ is providing the right gradient direction towards to target distribution. As a result of that, we train the policy model with two steps: first, to train the approximation $f(\boldsymbol{x})$ of the target distribution using a set of observed samples for a number of training episodes; second, to train the policy model to follow the gradient provided by the $f(\boldsymbol{x})$ to approach the target distribution for one training episode. The training episodes trade-off between the first step and the second step needs to be set properly so that the approximation is well qualified but not overfitting.

## 4.4. Application of M-projection with Buffer

The policy search problem must take care of the exploration-exploitation trade-off so that the algorithm explores new territories for the potential global optimums, but also exploits what it has learned so far. Intuitively, it needs to explore more at the beginning and then exploit more, when enough new information has already been explored. Since the target distribution does not provide sampling but only evaluating, we develop a buffer to store a set of samples to approximate the results of sampling from the target distribution, in order to leverage the M-projection to avoid premature convergence of the policy model to

a local optimum. The buffer is actually initialized and updated with generated samples from the exploration phase. It takes the samples with high rewards in and drops out the samples with few worth to exploit. However, the buffer would be too greedy to keep as much available information as possible, when it always only takes the samples with high rewards in. In the end, the samples in the buffer are gathering in a very small space, which may be one of the modes of the target distribution. Thus, we kernelize the buffer with a Gaussian kernel to filter the "duplicate" samples, where "duplicate" means the two samples are so close to being treated just like one. Then, these samples in the buffer help to guide the policy model way to the global optimum.

Let $\phi(\varrho)$ denote the standard Gaussian distribution $\mathcal{N}\ (0,1)$, and work as a Gaussian kernel for each sample. We use $r(\boldsymbol{x},\rho)$ to denote the local region, which is a hypersphere with the center $\boldsymbol{x}$ and the radius $\rho$, and the $\rho$ is given by

$$\phi(\rho) = \frac{1}{\sqrt{2\pi}} e^{-\frac{\rho^2}{2}} = p_c,$$

where the constant $p_c$ is the specified probability density value under $\phi(\varrho)$. Then, we take the samples that are in the same region $r(\boldsymbol{x},\rho)$ as the "duplicate" samples, and we only keep one sample with the greatest reward value in this region. What's more, if the samples generated from the last time and this time are "duplicate", we also keep the sample with the greatest reward value among them. As a result of that, the buffer is able to keep the samples distributed over the whole search space with the worth to exploit. Finally, the samples in the buffer are used to approximate the sampling from the target distribution.

Nevertheless, this approximation may not be so well as expected, because some samples that are not under the target distribution may also be included. Therefore, we let all the samples in the buffer weighted for computing M-projection. From the Equation (4.4) and (4.5), the Monte Carlo estimation of M-projection in the objective function is given as follows:

$$D_{KL}\left[\tilde{q}(\boldsymbol{x}) \parallel p(\boldsymbol{x};\boldsymbol{\theta})\right] \approx \frac{1}{M}\sum_{n=1}^{M} \log \tilde{q}(\boldsymbol{x}) - \log p(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim\tilde{q}(\boldsymbol{x})},$$

where only the second term contributes the gradient to the objective function. Thus, we focus on the second term of M-projection. We use $B(\boldsymbol{x})$ to approximate $\boldsymbol{x} \sim \tilde{q}(\boldsymbol{x})$, and the approximation is given by

$$\frac{1}{M}\sum_{n=1}^{M} - \log p(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim\tilde{q}(\boldsymbol{x})} \approx \frac{1}{M}\sum_{n=1}^{M} - \log p(\boldsymbol{x}) \mid_{\boldsymbol{x}\in B(\boldsymbol{x})} .$$

Then, we use the weight $\gamma(\boldsymbol{x}) = \tilde{q}(\boldsymbol{x})/p(\boldsymbol{x})$ for all the samples in the buffer. For the cases that the target distribution $\tilde{q}(\boldsymbol{x})$ provides a high reward, but the current learned policy model $p(\boldsymbol{x})$ provides a low reward, so the M-projection over these samples $\boldsymbol{x}$ is emphasized with high weights. And it can not be ignored that the target distribution $p(\boldsymbol{x})$ is usually unnormalized so that the above result is not able to directly be weighted. We address this problem by self-normalized importance sampling, which gives the result

$$\begin{aligned}\frac{1}{M}\sum_{n=1}^{M} -\log p(\boldsymbol{x}) \mid_{\boldsymbol{x}\sim\tilde{q}(\boldsymbol{x})} &\approx \frac{1}{M}\sum_{n=1}^{M} -\log p(\boldsymbol{x}) \mid_{\boldsymbol{x}\in B(\boldsymbol{x})} \\ &= \frac{\frac{1}{M}\sum_{n=1}^{M} -\log p(\boldsymbol{x})\gamma(\boldsymbol{x}) \mid_{\boldsymbol{x}\in B(\boldsymbol{x})}}{\frac{1}{M}\sum_{n=1}^{M} \gamma(\boldsymbol{x}) \mid_{\boldsymbol{x}\in B(\boldsymbol{x})}} .\end{aligned}$$

Therefore, the buffer provides the information of exploration by approximating the sampling from target distribution and weighting the approximation results to compute M-projection.

## 4.5. Implementation of Policy Search Using Normalizing Flow

We have theoretically discussed the idea of policy search using normalizing flow, so in this section, we are giving at first the summary of this idea, and then the implementation details. Recall that we define the random variable $\boldsymbol{x} \in \mathbb{R}^D$, the policy model as $p(\boldsymbol{x};\boldsymbol{\theta})$, the unnormalized target distribution as $\tilde{q}(\boldsymbol{x})$ with the reward as $r(\boldsymbol{x}) = \log \tilde{q}(\boldsymbol{x})$, the normalized target distribution $q(\boldsymbol{x}) = \tilde{q}(\boldsymbol{x})/Z$ with normalization constant $Z = \int_{\boldsymbol{x}} \tilde{q}(\boldsymbol{x})\, d\boldsymbol{x}$, the objective function as $\mathcal{L}$. And $\boldsymbol{x} \sim p(\boldsymbol{x};\boldsymbol{\theta})$ represents sampling from the policy model.

We summary the method in the Algorithm 1. Note that we scale the $(sample, reward)$ pairs for smoothing this approximator $f(\boldsymbol{x})$ to avoid the too steep gradient, which could cause the collapsing of the policy model during the training, in the local region. And we are using the reward defined by the unnormalized target distribution, for which we are not able to calculate the normalizing constant $Z$. But this constant $Z$ is not dependent on the parameters $\boldsymbol{\theta}$, so it can be ignored while we computing the gradient w.r.t the parameter $\boldsymbol{\theta}$. What's more, the gradient is related to the random variables of the distribution. In order to achieve this gradient computing, the reparameterization trick is used. And the gradient can be passed by normalizing flow, so to update the parameters $\boldsymbol{\theta}$ of the policy model.

When computing the M-projection in the objective function, the best samples, that have been explored in the previous training epochs, are kept in the buffer to approximate the sampling from the target distribution. And also each of them is weighted with a weight related to the current learned policy model using importance sampling.

For the training set, we use pre-training and entropy decreasing to avoid collapsing of the policy model onto local optimum. Pre-training means that we initialize the policy model at first to have the ability to generate samples in a wide range. And entropy decreasing is used to emphasize the entropy term in the objective function with decreasing weight. We use entropy decreasing from the beginning for a given epoch number to let the policy model can explore areas in the search space. During the entropy decreasing, the M-projection is not able to contribute to the objective function. After that, the M-projection is weighted with the $\beta$ in the objective function.

**Input**: the normalizing flow based policy model $p(\boldsymbol{x};\boldsymbol{\theta})$, the reward function $r(\boldsymbol{x}) = \log \tilde{q}(\boldsymbol{x})$ with the unnormalized target distribution as $\tilde{q}(\boldsymbol{x})$

Initializing $p(\boldsymbol{x};\boldsymbol{\theta})$;
Initializing training settings: set the entropy decreasing, set the inner training iteration, set the weight of M-projection, and other necessary parameters;
**while** *not converged* **do**
  **for** *inner training iterations not reached* **do**
    Sampling from the policy model:
    $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots \boldsymbol{x}_N) \sim p(\boldsymbol{x};\boldsymbol{\theta})$;

    Getting reward on the samples:
    $r_1, r_2, \dots r_N$;

    Smoothing by scaling the reward to generate $(sample, reward)$ pairs:
    $(\boldsymbol{x}_1, \hat{r}_1), (\boldsymbol{x}_2, \hat{r}_2), \dots (\boldsymbol{x}_N, \hat{r}_N)$;

    Fitting the $(sample, reward)$ pairs using a non-linear function:
    $r(\boldsymbol{x}) \approx f(\boldsymbol{x}) \mid_{(\boldsymbol{x}_1,\hat{r}_1),(\boldsymbol{x}_2,\hat{r}_2),\dots(\boldsymbol{x}_N,\hat{r}_N)}$;
  **end**
  Sampling from the policy model:
  $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots \boldsymbol{x}_K) \sim p(\boldsymbol{x};\boldsymbol{\theta})$;

  Updating the buffer with help of a Gaussian kernel for approximating sampling from target distribution:
  $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots \boldsymbol{x}_J) \rightarrow B(\boldsymbol{x})$;

  Computing I-Projection and M-projection as an objective function $\mathcal{L}$ over the samples $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots \boldsymbol{x}_K)$ and $B(\boldsymbol{x})$:
  $\mathcal{L} = D_{KL}\left[p(\boldsymbol{x};\boldsymbol{\theta}) \parallel \exp(r(\boldsymbol{x}))\right] + \beta D_{KL}\left[\exp(r(\boldsymbol{x})) \parallel p(\boldsymbol{x};\boldsymbol{\theta})\right]$;

  Computing the gradient of the objective function $\mathcal{L}$ w.r.t $\boldsymbol{\theta}$ using reparameterization trick:
  $\nabla_{\boldsymbol{\theta}}\mathcal{L} = \nabla_{\boldsymbol{\theta}} D_{KL}\left[p(\boldsymbol{x};\boldsymbol{\theta}) \parallel \exp(r(\boldsymbol{x}))\right] + \beta D_{KL}\left[\exp(r(\boldsymbol{x})) \parallel p(\boldsymbol{x};\boldsymbol{\theta})\right]$;

  Updating the parameter $\boldsymbol{\theta}$:
  $\boldsymbol{\theta} \leftarrow \lambda \nabla_{\boldsymbol{\theta}} D_{KL}\left[p(\boldsymbol{x};\boldsymbol{\theta}) \parallel \exp(r(\boldsymbol{x}))\right] + \beta D_{KL}\left[\exp(r(\boldsymbol{x})) \parallel p(\boldsymbol{x};\boldsymbol{\theta})\right]$;
**end**

**Algorithm 1:** Policy search using normalizing flow.

# 5. Robot Grasping Task Overview

It is a fundamental problem to manipulate a robot arm to grasp an object in machine learning. In order to solve this task, a wide range of methods are developed [58], from analytic methods [74] to data-based learning approaches [6]. And the data-based learning approaches have gained a lot of popularity and success in recent years. The data-based learning approaches are usually aiming to find an optimal grasping setting for the grasper, e.g. the position, the posture, the force-closure, and so on. To achieve this goal, most previous approaches use deep learning to consider the problem as training a neural network for predicting the successful probability of grasping actions, when RGB observations and depth measurements are given [75, 45]. So these methods require an additional component for generating the evaluated and ranked samples. It is less natural but more complex to construct the methods, and also more difficult to adapt to the changing of the tasks. In this work, we are applying the policy search with normalizing flow to directly learn the distribution of the grasping setting of the grasper over the whole space, which provides the abilities of both sampling and evaluating. Thus, the evaluation-ranking process can be eliminated in the tasks for grasping, so that the model becomes more compact and gives the power of adaptive.

In this chapter, we mainly introduce the grasping task based on the simulation. At first, we focus on the formulation of the grasping task and give the problem statement. Currently, the method does not rely on the RGB observations or the depth measurements, but takes all the information of objects as prior knowledge. The reason is that we are more interested in the performance of our method to give a complete picture of the distribution of grasping settings over the whole space. Then, it is illustrated how to properly choose and set the environment with robot grasper, how to simplify the simulation in the environment. Finally, the implementation of the interaction is discussed, which is between the policy model and the environment, and used for training the policy model from trials.

## 5.1. Problem Statement of Grasping Task

We focus on the grasping task for a given simple 3D object model and the parallel-jaw gripper with two fingers. The gripper has 7 degrees of freedom, i.e. the grasping position $\boldsymbol{\mu}_g \in \mathbb{R}^3$, the posture of the gripper $\boldsymbol{\nu}_g \in \mathbb{R}^3$ and the close position of the two fingers $c_g \in \mathbb{R}$. So we are searching for a good grasping setting for the gripper in this joint 7 dimension grasping action space. And a high reward value is given for the successful grasping, while a low reward value used for the failed grasping. Thus, the reward $r$ is defined to be binary over the grasping action space. As a result of that, the grasping actions of the gripper with different given back rewards are considered as a distribution over this space. If we defined the reward $r$ in the $\log$-range, this interesting distribution can be denoted as

$$q(\boldsymbol{\mu}_g, \boldsymbol{\nu}_g, c_g) = e^{r(\boldsymbol{\mu}_g, \boldsymbol{\nu}_g, c_g)}\,, \tag{5.1}$$

which can be used to describe the probability of successful grasping. If we joint the position $\boldsymbol{\mu}_g$, the posture $\boldsymbol{\nu}_g$ and $c_g$ together, we obtain

$$q(\boldsymbol{x}) = e^{r(\boldsymbol{x})}. \tag{5.2}$$

This interesting distribution is not known, but we are going to approximate it with a normalizing flow based policy model $p(\boldsymbol{x})$. As it showed in the Section 4.1, the maximization problem of the reward in the policy search is equivalent to the minimization problem of the "distance" between the interesting target distribution and policy model. Then, we are able to apply the policy search using normalizing flow, which is demonstrated in the Chapter 4 and the Chapter 5, to train the policy model to approximate the target distribution that could be multi-modal.

## 5.2. Building Simulation Environment

We are building the simulation environment for the grasping task at first to evaluate our method. PyBullet is a popular and easy to use Python module in machine learning, which is focusing on sim-to-real transfer. And PyBullet provides abundant robot models and a reliable physical engine for simulation. The interaction with PyBullet is also easy to implement.

Since we focus on grasping the object, we only build the robot hand with two fingers

as the robot model. We ignore the process to drive the robot hand to approach some goal positions with specific postures for grasping, and assume this process is ideal and precise. But we only care about the final position and posture of the robot hand, which could lead to a successful grasping. In order to drive the robot hand, we mount the robot hand on a virtual and invisible robot arm, because of the limitation of PyBullet simulation environment. And this virtual and invisible robot arm can only place the robot hand to the expected position with the expected posture, but has no effect on the grasping task self. The virtual and invisible robot arm consists of 6 links, 3 of which are used for the translation respectively along the three axises, and the rest 3 of which are used for rotation respectively along the three axises. Then, such a robot hand with two fingers is able to be placed through these 6 virtual and invisible links at any position with any orientation precisely, simply, and without dynamics. In this way, we can avoid some noise from irrelevant processes to the grasping and can speed up the simulation.

And the two fingers are always set to be symmetric by using the constraint between them, which means they are always driven to close for the same distance. So the close distance of them is only another one degree of freedom. For now, we set the force on the fingers as a const to ensure it could not make the grasping fail, so the close distance is the only thing that matters of the successful grasping for a specific object, besides the position and posture of the robot hand.

As for the object to grasp, we start from the simple lego model at current. The lego model represents a kind of simple and regular objects for grasping tasks. The object is always placed at a fixed position inside a tray before every grasping attempt. Then, the robot hand is doing the grasping task, when the position, the posture, and the close distance are given. The whole environment is showed in the Figure 5.1. For every grasping attempt, the object is placed at the fixed position for preparation. Next, the robot hand is placed at the to evaluate position with a specific posture. Then, the fingers close for a given distance. Finally, the robot hand tries to lift the object to a height for a while. When the object keeps contacting with the fingers for the whole grasping process, then this grasping is successful, otherwise, this grasping is failed.

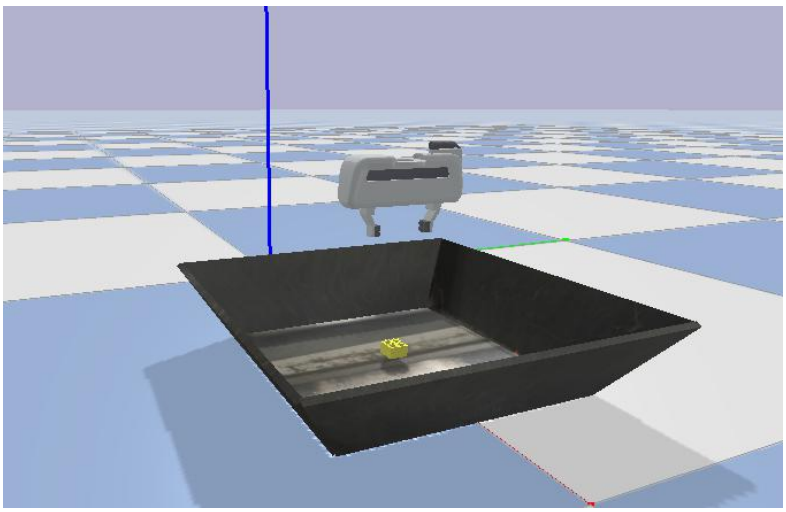


Figure 5.1.: The simulation environment based on the PyBullet for the grasping task evaluated in this paper. It shows the two fingers robot hand with a visual and invisible robot arm, and the simple lego object waiting to grasp inside the tray.

## 5.3. Implementation of Reward Function

The successful grasping gives the high reward back, while the failed grasping gives the low reward back. Actually, it describes the binary reward for the grasping task. However, this kind of binary reward could make it difficult to learn. It is possibly more unstable for the used approximator to estimate the gradient of the target distribution, which could lead to the worse training of the policy model. Note that graspings with low rewards can also differ a lot among themselves, but the binary reward is not able to distinguish them. That means grasping very close to the success can not provide more information than grasping very far to the success, while using the binary reward. So it only relies on being lucky enough to generate the grasping setting which can lead to successful grasping from the whole search space.

It is wise to add an additional part to the reward. This additional part comes from the distance between the robot hand and the object. Only when the robot hand has a proper distance to the object, the grasping can be successful. Therefore, we define the additional part of the reward $r_l$ as the $\log$-value of the density value of a given gaussian

distribution as follows

$$\exp r_l(\|\boldsymbol{\mu}_g - \boldsymbol{\mu}_t\|) \sim \mathcal{N}(l, v)\,, \tag{5.3}$$

where the $\boldsymbol{\mu}_g$ denotes the grasping position of the robot hand, the $\boldsymbol{\mu}_t$ represents the position of the object, and $\mathcal{N}(l, v)$ denotes the Gaussian distribution with the mean value $l$ and the variance $v$ over the distance between robot hand and the object while applying the grasping action. When we apply the Equation (5.3), the Equation (5.1) can be updated as

$$q(\boldsymbol{\mu}_g, \boldsymbol{\nu}_g, c_g) = e^{r(\boldsymbol{\mu}_g, \boldsymbol{\nu}_g, c_g) + r_l(\|\boldsymbol{\mu}_g - \boldsymbol{\mu}_t\|)}\,.$$

Finally, if we joint the variables together, we expand the Equation (5.2) as

$$q(\boldsymbol{x}) = e^{r(\boldsymbol{x}) + r_l(l_{\boldsymbol{x}})}\,,$$

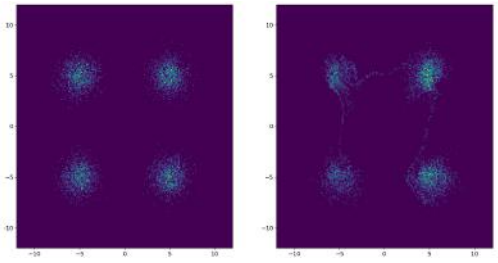
where the $l_{\boldsymbol{x}}$ represents the distance between the to grasp object and the grasping position, when the grasping setting $\boldsymbol{x}$ is given.

# 6. Experiments and Results

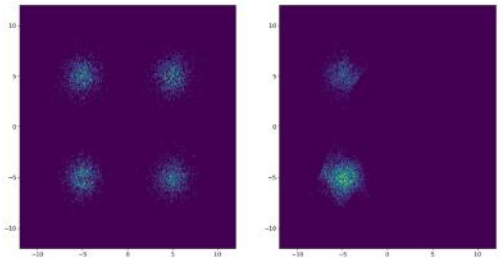
In this chapter, we evaluate our method in several experiments. The normalizing flow-based policy model is applied in several experiments to demonstrate its expressive representation ability to approach Gaussian mixture model with different structure settings. Meanwhile, the effects of the training parameters have also been considered and evaluated. And we use Gaussian mixture model with low covariance matrix to simulate sparse reward cases for evaluating our method. Moreover, we apply it to finish the expected position approaching task for the planar robot in both lower and higher dimension. Finally, we evaluate this method in robotic grasping tasks in the simulation environment based on PyBullet. In all the experiments the policy model would be trained for $10,000$ epochs, and $2,000$ samples at each epoch would be used.

## 6.1. Approximation to Gaussian Mixture Model

The experiment in this section shows the expressive power of normalizing flow based policy model to approximate the Gaussian mixture model. We evaluate the policy model constructed by both coupling layer and neural spline based flow. Note that the Gaussian mixture model is not sparse in this experiment, so it is not wise to do further smoothing of the target distribution. Otherwise, it may cause unstable of the training. Thus, in this experiment we still use approximator $f(\boldsymbol{x})$ to approximate the scaled $(sample, reward)$. But after that, we denormalize the approximated value. The objective function used in this section is computed only by the I-projection, i.e. the reverse KL Divergence. For each construction, we totally train the policy model for $10,000$ epochs and evaluate it with different training settings.



(a) The target distribution (left) and the trained policy model (right).



(b) The target distribution (left) and the trained policy model (right).

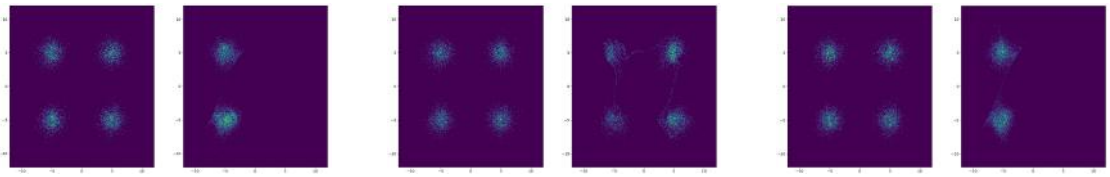
Figure 6.1.: Training policy model with different entropy decreasing speeds. In the subfigure (a) it shows the result using slow entropy decreasing, i.e. the weight of the entropy term in the objective function decreases from $0.9$ to $0.5$ in the first $8,000$ training epochs. In the subfigure (b) it shows the result using fast entropy decreasing, i.e. the weight of the entropy term in the objective function decreases from $0.9$ to $0.5$ in the first $4,000$ training epochs.

### 6.1.1. Using Coupling Layer

At first, we evaluate the coupling layer based policy model. We construct the policy model by $10$ coupling layers with hidden features as $64$. And between any two coupling layers, we add additionally a random permutation layer and a linear layer. For training the coupling layer, we pre-train the policy model to map from a standard Gaussian distribution to a uniform distribution before training.

#### Evaluating with Different Entropy Decreasing Speeds

We compare the results with different entropy decreasing speeds for first. For the slow entropy decreasing, we let the weight of the entropy term in the objective function decrease from $0.9$ to $0.5$ in $8,000$ epochs so that the policy model can do more exploration. And for the fast entropy decreasing, we let the weight of the entropy term in the objective function decrease from $0.9$ to $0.5$ in $4,000$ epochs. The results are shown in the Figure 6.1. In this experiment, if we decrease the entropy slowly enough, so the policy model can converge to all the modes of the target distribution. However, if we decrease the entropy too fast,



(a) The target distribution (left) and the trained policy model (right).

(b) The target distribution (left) and the trained policy model (right).

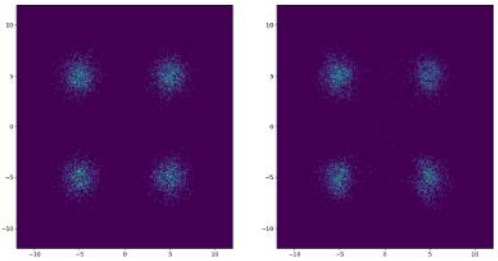
(c) The target distribution (left) and the trained policy model (right).

Figure 6.2.: Training policy model with different inner training iterations. The subfigure (a) shows the result for training the approximator in the inner training loop for $3$ steps. The subfigure (b) shows the result for training the approximator in the inner training loop for $6$ steps. The subfigure (c) shows the result for training the approximator in the inner training loop for $12$ steps.

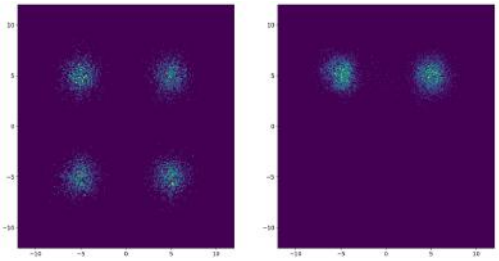
the policy model collapses onto only some modes of the target distribution. Therefore, in the training set we need to make sure that the entropy decreasing is slow enough.

#### Evaluating with Different Inner Iterations

In this experiment, we also test the policy model with different inner iterations. We compare the results, when we set the inner training iterations as $3$, $6$, and $12$. As it shows in the Figure 6.2, when we train the approximator for $6$ steps in the inner training loop, the policy model can converge onto all the modes of the target distribution. However, when we train the approximator for more steps or fewer steps, i.e. for $12$ steps or $3$ steps, the policy model is not able to converge onto all the modes anymore. When the approximator is trained for fewer steps, that means the approximation to the target distribution over the samples generated from the currently trained policy model is not enough, so it can not show the gradient descent direction towards to global optimum. If the approximator is trained for more steps, that means the approximator is overfitting over the samples generated from the current policy model, and the policy model is strongly stuck by the current state. Therefore, the inner training iteration number should be set properly to avoid the collapse of the policy model. Since we are using the approximator to give the gradient towards to the global optimum, it is not qualified if it has been trained too little to fit the $(sample, reward)$ or has been trained too much.



(a) The target distribution (left) and the trained policy model (right).



(b) The target distribution (left) and the trained policy model (right).

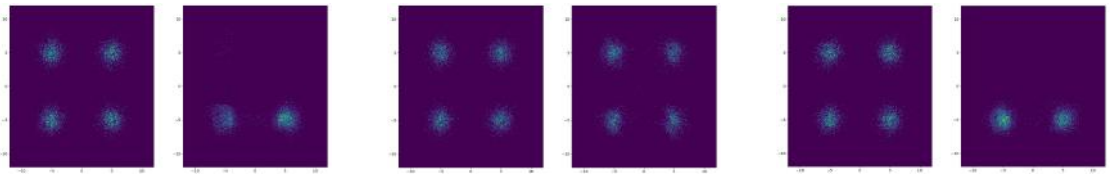
Figure 6.3.: Training policy model with different entropy decreasing speeds. In the subfigure (a) it shows the result using slow entropy decreasing, i.e. the weight of the entropy term in the objective function decreases from $0.9$ to $0.5$ in the first $4,000$ training epochs. In the subfigure (b) it shows the result using fast entropy decreasing, i.e. the weight of the entropy term in the objective function decreases from $0.9$ to $0.5$ in the first $2,000$ training epochs.

### 6.1.2. Using Neural Spline Based Flow

In this experiment, we construct the policy model with the neural spline based flow. In order to achieve more nonlinearity, we employ the piecewise-quadratic neural spline to construct the policy model. And we use two layers of neural spline flow to construct normalizing flow, i.e. the depth is set to be $2$. For every element we quantize the bound range into $8$ bins, so to use the quadratic spline function on these intervals. And in this experiment, the bound for neural spline based flow has been set to be $[-10, 10]$ at each dimension.

#### Evaluating with Different Entropy Decreasing Speeds

At first, we test it with different entropy decreasing speeds. We compare the entropy decreasing from $0.9$ to $0.5$ in $4,000$ epochs and $2,000$ epochs. The results are shown in the Figure 6.3. When we set a slow entropy decreasing, the policy model can converge onto all the modes of the target distribution. But if we set a fast entropy decreasing, the policy model collapses. And compared to the policy model based on the coupling layer, there are no connection lines among modes anymore using neural spline based policy



(a) The target distribution (left) and the trained policy model (right).

(b) The target distribution (left) and the trained policy model (right).

(c) The target distribution (left) and the trained policy model (right).

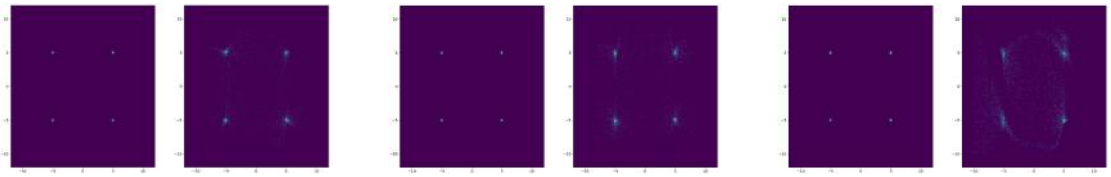
Figure 6.4.: Training policy model with different inner training iterations. The subfigure (a) shows the result for training the approximator in the inner training loop for $6$ steps. The subfigure (b) shows the result for training the approximator in the inner training loop for $12$ steps. The subfigure (c) shows the result for training the approximator in the inner training loop for $24$ steps.

model. The reason is that the neural spline based flow provides more nonlinearity and has been bounded.

#### Evaluating with Different Inner Iterations

In this experiments we also test the policy model with different inner iterations. At first, we train the approximator in the inner training loop for $12$ steps so that the approximator can give the gradient information. The result is presented in the Figure 6.4. And again the if the approximator is trained too much or too little, the policy model can also not converge to all the modes of the target distribution. Since the approximator trained too little is not qualified, and the approximator trained too much is overfitting. And the policy model is constructed with different layers, the inner iterations should be set properly for different constructions.

As the above results show, the policy model based on the neural spline flow has a better performance than based on the coupling layer in our policy search task context. Therefore, we use the policy model based on neural spline flow for the approximation to a more complex target distribution.



(a) The target distribution (left) and the trained policy model (right).

(b) The target distribution (left) and the trained policy model (right).

(c) The target distribution (left) and the trained policy model (right).

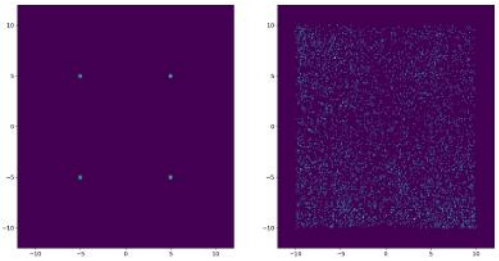
Figure 6.5.: Training policy model with different entropy decreasing speeds. The subfigure (a) shows the results using entropy decreasing from $1.0$ to $0.5$ in first $1,000$ training epochs. The subfigure (b) shows the results using entropy decreasing from $1.0$ to $0.5$ in first $4,000$ training epochs. The subfigure (c) shows the results using entropy decreasing from $1.0$ to $0.5$ in first $8,000$ training epochs.

## 6.2. Approximation to Sparse Reward Model

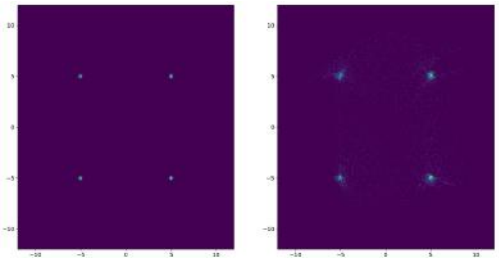
In this experiment, we apply our method in sparse cases. The target distribution is chosen to be Gaussian mixture model but given with low covariance. The policy model is constructed with the neural spline based flow with depth as $2$. The bound for neural spline based flow is set to be $[-10, 10]$ at each dimension. And the bound range is quantized into $8$ bins, at each of which the quadratic spline function is used to give the mapped value element-wise. As for the objective function, we use both M-projection and I-projection. The M-projection always reminds the policy model to exploit the areas possibly with high rewards under the target distribution. Moreover, the entropy decreasing is also used during the training. But at the phase of the entropy decreasing, the M-projection part of the objective is ignored. As soon as the entropy decreasing is finished, the M-projection is able to contribute into the objective with a given weight.

### 6.2.1. Evaluating with Entropy Decreasing Speeds and Inner Iterations

In this experiment, we test the training of policy model with different entropy decreasing speeds. Since the M-projection is applied into the objective function, the information, which is provided by the samples in the buffer for guiding the policy model towards the global optimum, is always available so that the entropy decreasing speed does not have to



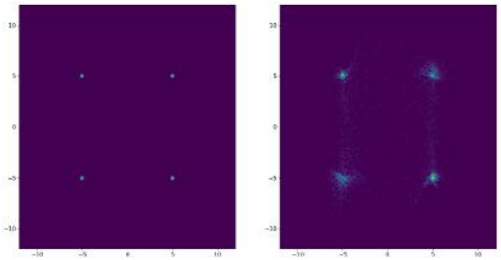
(a) The target distribution (left) and the trained policy model (right).



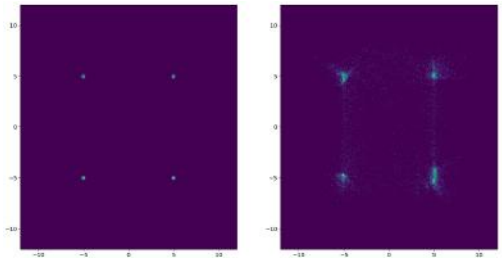
(b) The target distribution (left) and the trained policy model (right).

Figure 6.6.: The states of the policy model during the training. The subfigure (a) shows the policy state when the entropy decreasing is finished, here, it means after $1,000$ training epochs. So the policy model explore the areas as widely as possible. Only when the entropy decreasing phase is finished, the M-projection is able to contribute to the objective function with a weight $\beta = 1.0$. The subfigure (b) shows the state of policy model after $10,000$ training epochs. The trained policy model converges onto all the modes of this sparse target distribution.

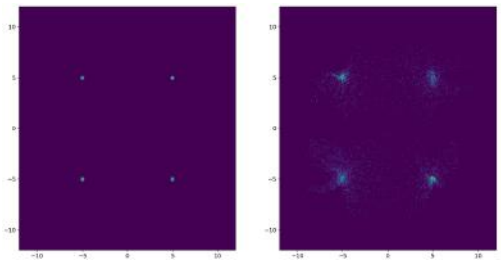
be extremely slow like the cases only using I-projection. However, the M-projection used in our method is approximated over the samples in the buffer. The buffer is always collecting the samples worth exploring from the previous samplings following the policy model. That means the entropy decreasing is only expected to be slow enough to let the buffer collect enough samples for computing M-projection part of the objective. As it shows in the Figure 6.5, the policy model can converge onto the target distribution in all the three settings. However, the policy model with slower entropy decreasing speed is trained not as well as with faster entropy decreasing speed, and covers too much area outside the target distribution. The reason is that the policy model with the slower entropy decreasing speed is not trained enough, when the totally training epoch is fixed. Since the M-projection only contributes into objective function after the entropy decreasing phase, the slower the entropy decreasing speed, the fewer epochs the policy model is trained. Assume that the total training epoch is large enough, the policy model could learn the target distribution as well, but it would unnecessarily increase the time cost. So the policy model should be able to not only explore as much area as possible but also decrease the time cost, before the M-projection is added into the objective function. Figure 6.6 explains the situation. After the entropy decreasing phase for first $1,000$ epochs, the policy model is almost like a uniform distribution to explore all the searching space inside the bounds. Then, we stop



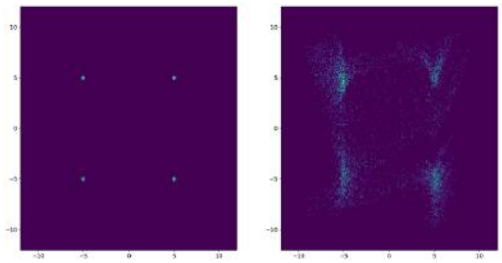
(a) The target distribution (left) and the trained policy model (right).



(b) The target distribution (left) and the trained policy model (right).



(c) The target distribution (left) and the trained policy model (right).



(d) The target distribution (left) and the trained policy model (right).

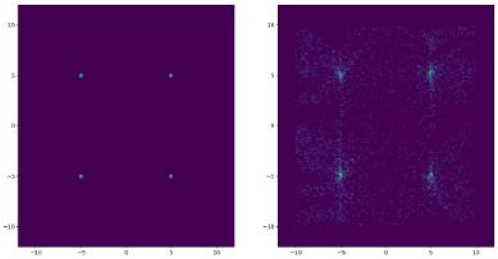
Figure 6.7.: Training policy model with different inner iterations in sparse cases. The subfigure (a) shows the result with inner iterations as $3$. The subfigure (b) gives the result with inner iterations as $6$. The result with $12$ inner iterations is shown in the subfigure (c). And the result with $30$ inner iterations is presented in the subfigure (d).

the entropy decreasing phase and let the M-projection contribute into the objective. It should be noted that in the high dimension search space, we would choose a little slower entropy decreasing speed to let the policy model do enough exploration.

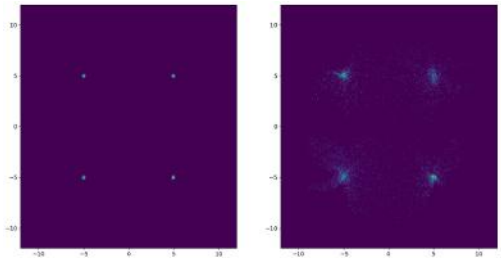
As for the inner iterations at each training epoch, we also test some different settings to compare the results. The Figure 6.7 presents the results with different inner iterations. When the inner training step is set to be $30$, the policy model acts not as well as the others for covering too much area outside the target distribution. And when the inner training step $3$ or $6$, then there are some not expected banded area between two different modes. For the training step $12$, the policy model is trained with more clearly separated modes and less covering of area outside the target distribution. From the results we observe that, if the number of the inner iteration is too large, there could be more resistance for the policy model to step over the not optimal state at current. And the policy model is not able to follow the gradient descent direction if the inner training of the approximator is not enough. Thus, it also requires a proper setting of inner training iterations for cooperating well with the outer training for updating the policy model itself, so that the policy model can give a more accurate approximation after training.

### 6.2.2. Evaluating with Weights of M-projection

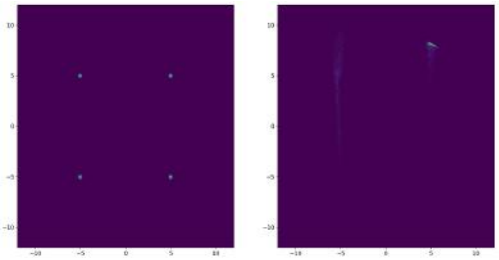
Since we leverage the M-projection in the total objective function for the application in sparse cases, it is crucial to determine the weight of M-projection in the objective function. And it is intuitive that the objective depends too much on the M-projection if the weight is extremely high and vice versa. So we test different possibilities of the weight to show the effects. As the results presented in the Figure 6.8, if the weight $\beta$ is too lower, e.g. $\beta = 0.5$, the policy model approximates the target distribution more widely. But when the weight $\beta$ is set too higher, e.g. $\beta = 10$ or $\beta = 20$, the objective emphasizes more the M-projection part, so that the trained policy model can still not accurately approximate the target distribution. For the cases of sparse reward, we choose the $\beta = 1$ for the training. Actually, the application of both I-projection and M-projection leads to the balance of zero forcing and zero avoiding, where the balance depends on the weight of the M-projection.



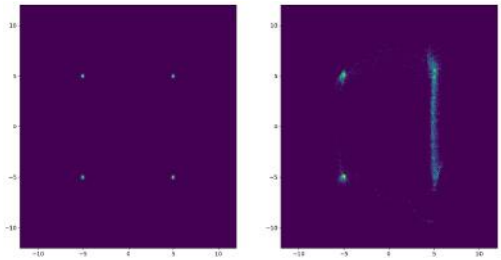
(a) The target distribution (left) and the trained policy model (right).



(b) The target distribution (left) and the trained policy model (right).



(c) The target distribution (left) and the trained policy model (right).



(d) The target distribution (left) and the trained policy model (right).

Figure 6.8.: Training policy model with different weight $\beta$ of the M-projection in the objective in sparse cases. The subfigure (a) shows the result with the weight $\beta$ as $0.5$. The subfigure (b) presents the result with the weight $\beta$ as $1$. The weight $\beta$ equals to $10$, and the result is given in the subfigure (c). The weight $\beta$ is set to be $20$ for training, whose result is presented in the subfigure (d).

## 6.3. Application in Approaching Task for Planar Robot Arm

From the above results, we know how the different training settings affect the result. In this section, we are going to apply our method with the proper training setting in the approaching task for the planar robot arm, which gives a closer approximation of sparse reward to reality. And also we apply our method in higher dimension cases to evaluate its performance. The approaching task for the planar robot arm is better for visualization of the results.

For the planar robot arm, we set the total length to be fixed, and the length of each link to be the quotient of the total length and the number of the links. The dimension number is nothing else but the number of all the links. And the link is joint with the previous link with only one degree of freedom. The relative angle between the link to the previous one is defined as the parameter in this degree of freedom. And the search space is defined as the space consisting of the relative angle of each planar robot link. Then, the position of the top of the planar robot arm can be calculated given these relative angles of all the robot links. Finally, we are aiming to let the top of the planar robot to approach one expected position in the planar as precisely as possible. This expected position is set to be $(2, 0)$. And the expected distribution is defined as a distribution with this expected position as the mean, and the low covariance $0.1$ for each coordinate in the planar. Thus, the target distribution to be approximated in the search space is multi-modal and sparse. There always exists different relative angles of each link to place the top of the planar robot arm to the expected position, and only when this placing with small error would return high reward, i.e. the $\log$ value of the probability density following the expected distribution.

#### For $2$ Dimensions

To begin, we test it in $2$ dimensions, i.e. the planar robot arm has only two links. The training settings of the policy model are set as follows: a) the entropy decreasing is designed from $1.0$ to $0.5$ in the first $2,000$ epochs; b) the inner training iterations is set to be $12$; c) the weight $\beta$ of M-projection in the objective is set to be $1.0$.

The samples, which can be seen as the postures of the planar robot arm following the parameter settings, from both the target distribution and the learned policy model are presented in the Figure 6.9. All postures are distinguished by different colors from the

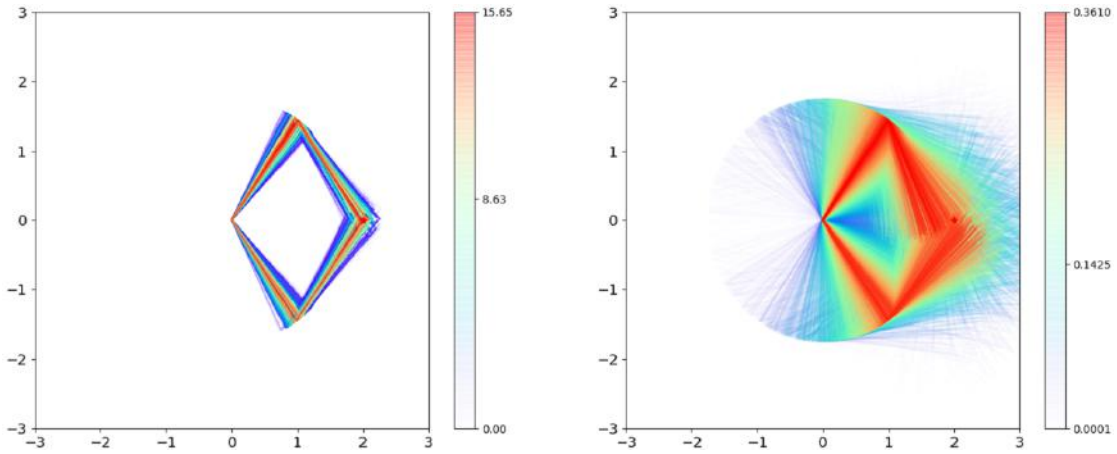



Figure 6.9.: Application of the method in the approaching task for the planar robot arm. The expected position is at $(2, 0)$. In order to visualize the distribution, we use a color bar beside the subfigures to map the rewards onto the postures of the planar robot arm. The postures of the planar robot arm with high rewards are colored close to the warm color, while those with low rewards are colored close to the cool color. And the numbers beside the color bar show the max, min, and mean values of the exponential of the rewards $\exp r(\boldsymbol{x})$ of all the samples visualized in the figures. The left figure shows the target distribution for the planar robot arm to approach the expected position. The right figure shows the policy model trained to approximate the target distribution.

color map. The higher the reward is, the closer the posture is colored to be red with high transparency; the lower the reward is, the closer the posture is colored to be blue with low transparency. On the left side, it shows the samples (postures of the planar robot arm) from the target distribution in the search space. On the right side, the samples (postures of the planar robot arm) from the learned policy model are given. And the numbers beside the color map show the max value, min value, and the mean value of the $\exp r(\boldsymbol{x})$. The policy model is able to approximate the sparse target distribution well. The learned policy model can give a sight of the whole shape of the target reward distribution, and it can also converge onto all the modes of the target distribution. However, the policy model is much smoother than the sparse target distribution. The reason is that we use a much smoother approximator with gradient information to approximate the target distribution without gradient information. Nevertheless, the policy model is still able to generate reliable samples that belong to the target reward distribution with a high possibility.

### For $7$ Dimensions

Next, we go to the $7$ dimensions cases to evaluate our method. And because it is in a higher dimension, we set a smaller kernel to filter the explored samples into the buffer for computing the M-projection. Note that it is hard to clearly present the whole distribution over the postures of the planar robot by coloring, because overlapping of too many abundant postures of the planar robot in $7$ dimensions causes the loss of the details. Then, we add a prior distribution of the planar robot. The added prior distribution forces the policy model to keep focusing the defined region by itself in the search space and then to search more in this region. Thus, it could provide a more clear visualization of the target distribution over all the posture of the planar robot by coloring.

We set the planar robot to have 7 links in the approaching task on the planar. The general training settings of the policy model are set as follows: a) the entropy decreasing is designed from $1.0$ to $0.5$ in the first $2,000$ epochs; b) the inner training iterations is set to be $12$; c) the weight $\beta$ of M-projection in the objective is set to be $1.0$.

The result is shown in the Figure 6.10. Again all the postures are distinguished by different colors in the visualization of both target distribution and learned policy model. The higher the reward is, the closer the coloring of the posture is to the warm colors with high transparency. And the lower the reward is, the closer the coloring of the posture is to the cool colors with low transparency. The figure on the left side represents the target distribution which we would like to approach. The policy model shown on the right is

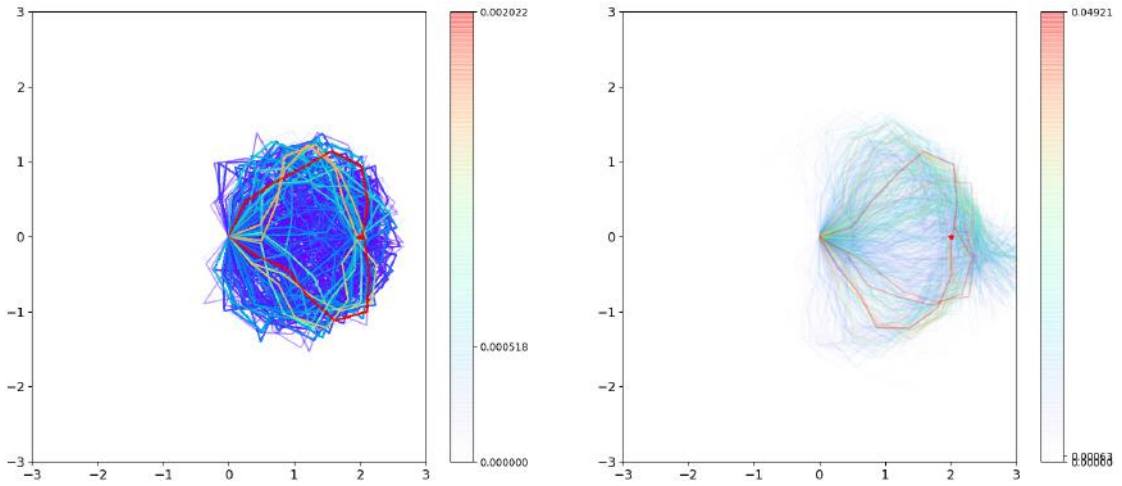



Figure 6.10.: Application of the method in the approaching task for the planar robot arm. The expected position is at $(2, 0)$. In order to visualize the distribution, we still use a color bar beside the subfigures to map the rewards onto the postures of the planar robot arm. The postures of the planar robot arm with high rewards are colored close to the warm color, while those with low rewards are colored close to the cool color. And the numbers of beside the color bar show the max, min, and mean values of the exponential of the rewards $\exp r(\boldsymbol{x})$ of all the samples visualized in the figures. The left figure shows the target distribution for the planar robot arm to approach the expected position. In the higher dimension case, the target distribution is much more complex. The right figure shows the policy model trained to approximate the target distribution.

able to approximate the target distribution well, while considering the prior distribution of the planar robot self for better visualization. Although the policy model gives a much smoother approximation for the sparse target distribution, it can reliably obtain high rewards on original samples with high rewards within the target distribution range. Thus, the learned policy model is plausible for the approximation to the target distribution so to provide the ability to both generating new samples and evaluating on them. And the policy model can catch the multi-modality of the target distribution well. Nevertheless, it can not be ignored that the learned policy model is biased in higher dimensions, which means that the sample with the highest reward following the new learned policy model to approximate may not have the highest reward following the sight of the target reward distribution. Following the policy model, the predicted reward on the generated sample is also available. If it requires a more accurate reward, the reward function to evaluate the reward following the target distribution can still be used.

## 6.4. Application in Robot Grasping Task in Simulation

In order to evaluate the performance of our method in more realistic applications, we apply it in the robotic grasping task. To begin, we build a robotics simulation environment, which is presented in the Chapter 5 in detail, for the evaluation. The built grasper has seven degrees of freedom. And the object to grasp is for now always placed at the position $(0.6, 0, 0)$ with the initial orientation $(0, 0, 0)$. At first, we build the search space with only $3$ dimensions for simplifying the complex task in higher dimensions. Then, we apply it in the complete search space with $7$ dimensions.

#### For $3$ Dimensions

We test it by using only three degrees of freedom: the grasping position coordinates $\mu_x$, $\mu_y$, and the yaw angle $\nu_y$ inside the posture vector of the gripper. The rest degrees of freedom are not variable and fixed on rational values. And the general training settings in this case are set as follows: a) the entropy decreasing is designed from $1.0$ to $0.5$ in the first $2,000$ epochs; b) the inner training iterations is set to be $12$; c) the weight $\beta$ of M-projection in the objective is set to be $1.0$.

After training the policy model is used to generate samples and the approximated reward

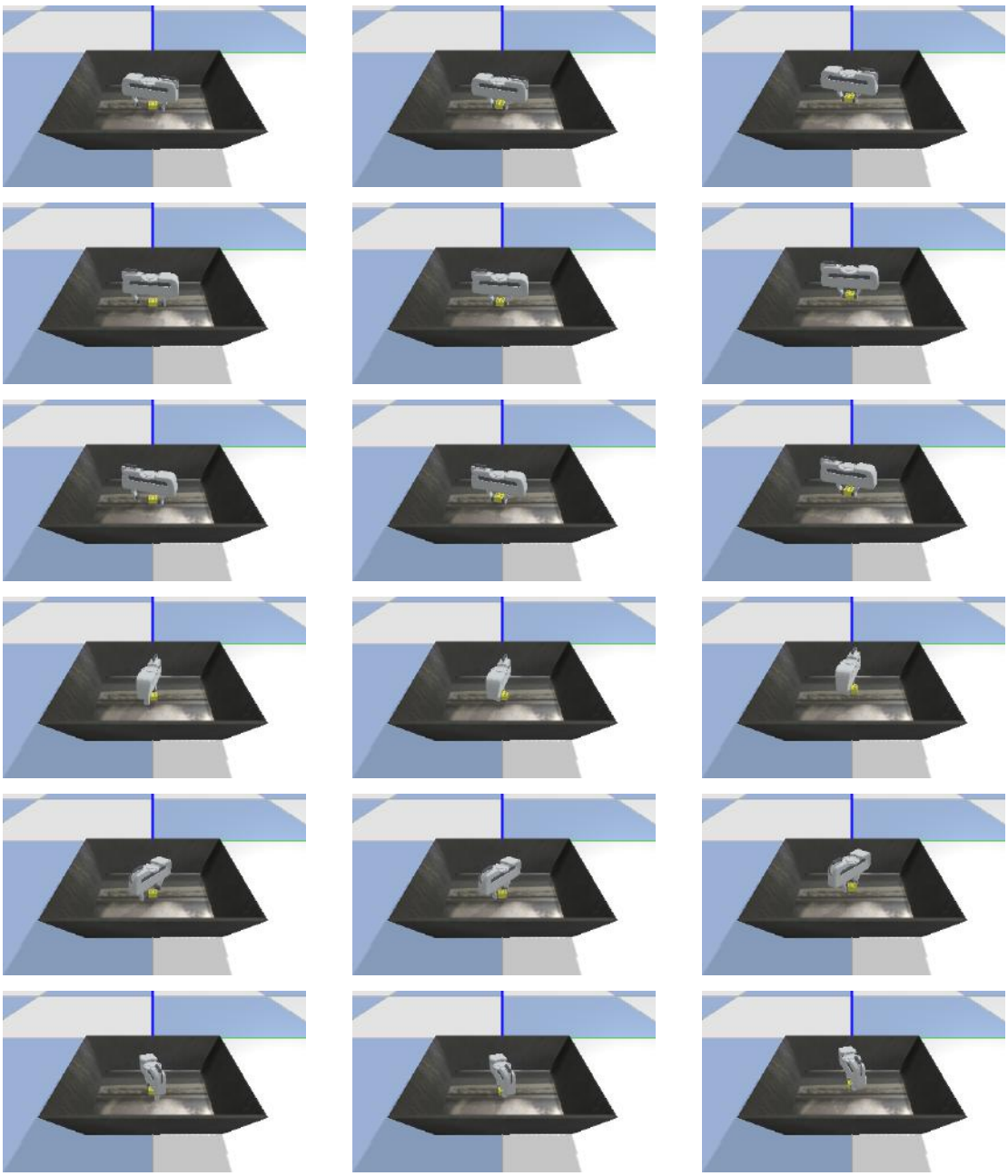
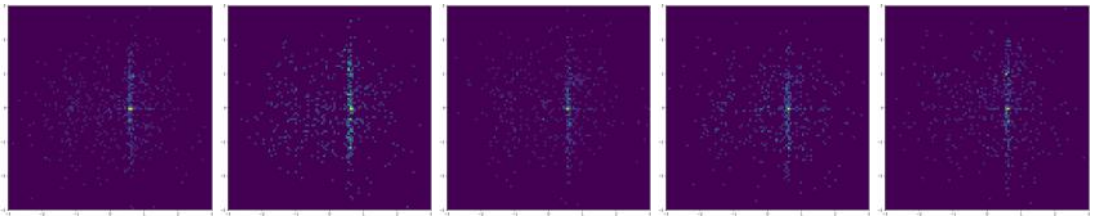
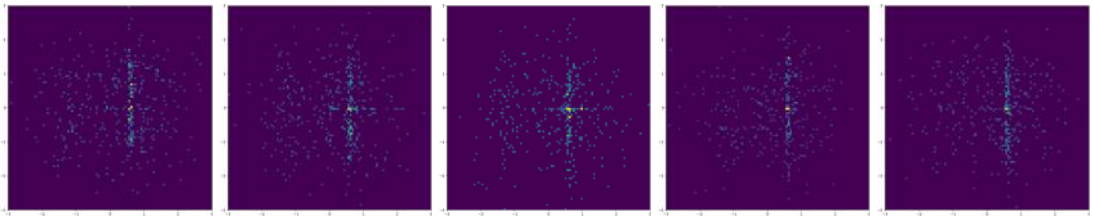


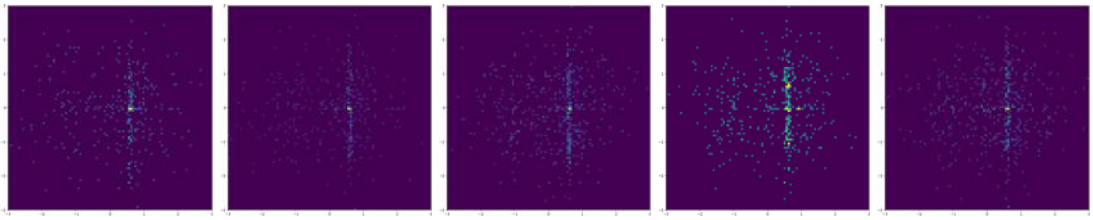
Figure 6.11.: The grasping attempts of the generated samples from the top $25$ approximated rewards following the trained policy model. Subfigures in each row show one complete successful grasping attempt.



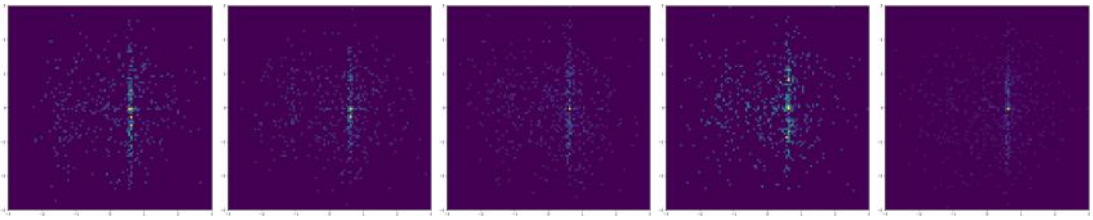
(a) The distribution of the position coordinates $\mu_x$ and $\mu_y$ in $2$ dimensions histogram for sliced layers $[-\pi, -0.9\pi)$, $[-0.9\pi, -0.8\pi)$, $[-0.8\pi, -0.7\pi)$, $[-0.7\pi, -0.6\pi)$ and $[-0.6\pi, -0.5\pi)$.



(b) The distribution of the position coordinates $\mu_x$ and $\mu_y$ in $2$ dimensions histogram for sliced layers $[-0.5\pi, -0.4\pi)$, $[-0.4\pi, -0.3\pi)$, $[-0.3\pi, -0.2\pi)$, $[-0.2\pi, -0.1\pi)$ and $[-0.1\pi, 0)$.



(c) The distribution of the position coordinates $\mu_x$ and $\mu_y$ in $2$ dimensions histogram for sliced layers $[0, 0.1\pi)$, $[0.1\pi, 0.2\pi)$, $[0.2\pi, 0.3\pi)$, $[0.3\pi, 0.4\pi)$ and $[0.4\pi, 0.5\pi)$.



(d) The distribution of the position coordinates $\mu_x$ and $\mu_y$ in $2$ dimensions histogram for sliced layers $[0.5\pi, 0.6\pi)$, $[0.6\pi, 0.7\pi)$, $[0.7\pi, 0.8\pi)$, $[0.8\pi, 0.9\pi)$ and $[0.9\pi, \pi)$.

Figure 6.12.: The distribution of position coordinates along the $x$ and $y$ axis, denoted as $\mu_x$ and $\mu_y$ for different sliced layers of the yaw angle along the $z$ axis, denoted as $\nu_y$. The $2$ dimensions histogram of $\mu_x$ and $\mu_y$ provides high density around the $(0.6, 0)$, which is also the first 2 coordinates of the placed target object.

values on them. We sort the samples by their reward values from high to low. And then we choose the top $25$ samples with the highest approximated reward following the policy model. We present six typical samples from them at each row respectively in the Figure 6.11. We repeat ten times for each considered sample in the evaluation. All the ten trials of each considered sample are successful grasping attempts. Thus, we are able to give a conclusion that the learned policy model can draw reliable samples and give the trusted approximated reward values on all the drawn samples. The results of all the top $25$ samples are shown in the Appendix B.

In order to give a better visualization of the picture of the policy model in such a simplified $3$ dimensions case, we quantify the range of yaw angle $\nu_y$ from $[-\pi, \pi]$ into $20$ layers. And then we present the distribution of position coordinates $\mu_x$ and $\mu_y$ in $2$ dimensions histogram for each sliced layer. The results are given in the Figure 6.12. It shows that the learned policy model is able to approximate the target reward distribution, while giving high density of samples around the target object placed position. From these sliced layers, we can find that in the layers of $[-\pi, -0.9\pi]$, $[-0.8\pi, -0.7\pi]$, $[-0.1\pi, 0]$, $[0, 0.1\pi]$, $[0.7\pi, 0.8\pi, ]$, and $[0.9\pi, \pi]$ the shown sliced distributions of position coordinates $\mu_x$ and $\mu_y$ are a little more narrow and more concentrated than others. So these distributions in the sliced layers could imply those possible modes along the third dimension yaw angle $\nu_y$. Therefore, our method can catch the multi-modality of the target distribution and meet the requirements for providing a plausible approximation.

#### For $7$ Dimensions

At last, we evaluate our method in the $7$ dimensions case of the robotic grasping task. The general training settings are set as follows: a) the entropy decreasing is designed from $1.0$ to $0.5$ in the first $2,000$ epochs; b) the inner training iterations is set to be $12$; c) the weight $\beta$ of M-projection in the objective is set to be $1.0$.

The result is shown in the Figure 6.13. Since it is difficult to draw the whole distribution of the reward, we only present the top $6$ generated samples with the highest approximated reward following the policy model. The grasping attempts shown in the second row and the third row are successful and from different modes, which describes the multi-modality while using policy model to approximate the target distribution. Moreover, the learned policy model is able to generate plausible samples with their predicted rewards. But this policy model approximates the target distribution with a bias. These samples, which have high predictions from the sights of the policy model, do not obtain the same

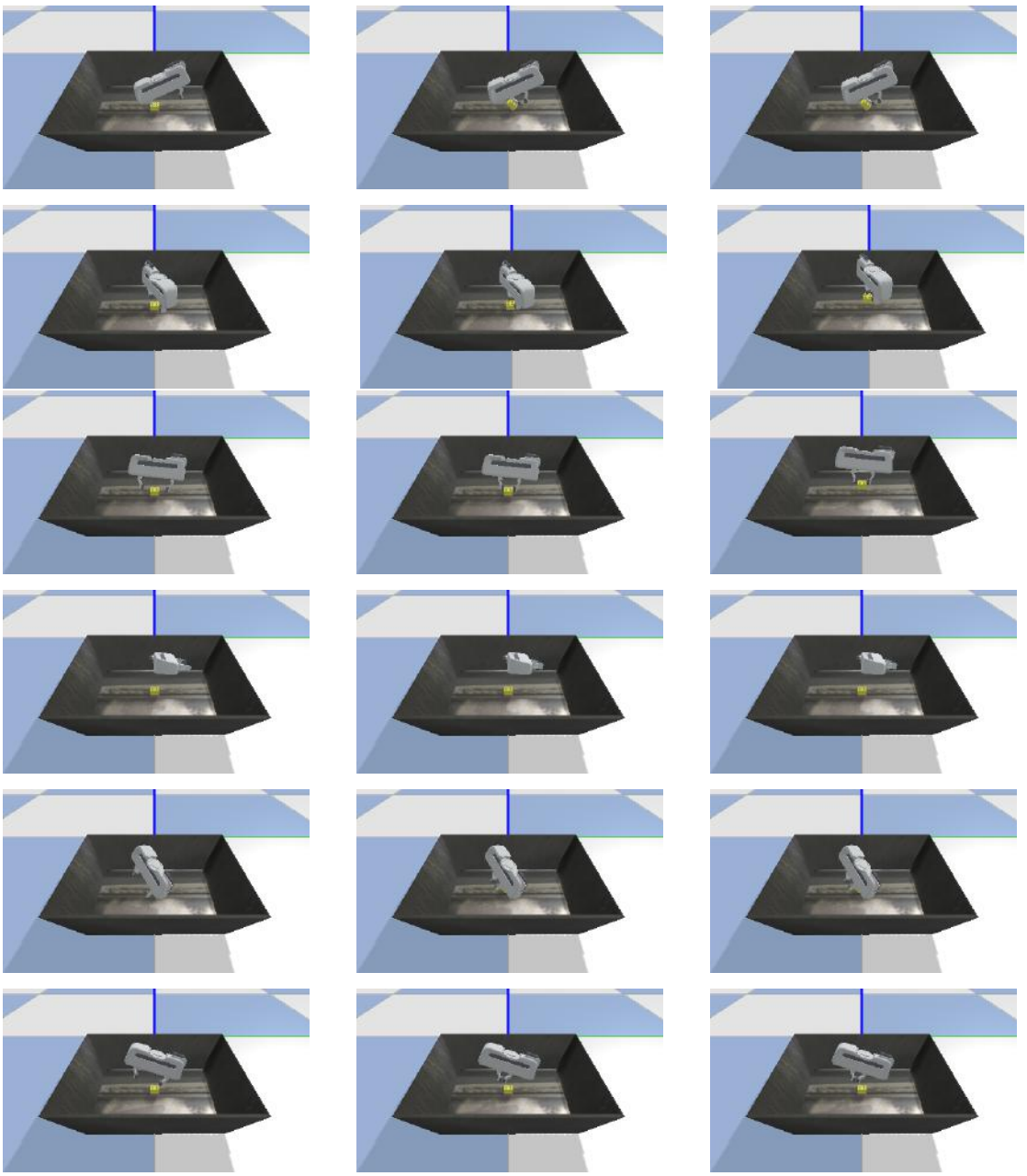


Figure 6.13.: The grasping attempts generated with top $6$ approximated rewards following the trained policy model. Subfigures in each row show one complete successful grasping attempt. The grasping attempts presented in the second row and third row are successful.

high rewards following the reward function. In this case, the generated sample with the highest approximated reward, as shown in the first row in the Figure 6.13, is failed to grasp the object and can not get a high reward. Alternatively, we can also get a better result when we evaluate those samples, which are drawn by the generative policy model, directly using the reward function. Note that the model-based policy search needs a lot of samples to guarantee good performance of the trained model, so it is essential to draw these samples in a wide range to explore as many modes as possible during the training. And in higher dimension search space, it should be trained for a longer time and with a better-suited setting to get a better policy model for approximation. However, it could cause a much higher cost of training than in the lower dimension case, which becomes a not ignorable limitation while this method applied.

# 7. Conclusion and Outlook

In this paper, we demonstrate a method of policy search using normalizing flow to solve the problem which is aiming to approximate an intractable target distribution with a tractable distribution. Considering the insights from the policy search, this target distribution can be used to present the reward distribution. Then, this task can be viewed as to approximate the true reward distribution over the parameter space of the policy, by using a properly built policy model to show the complete picture of the true reward distribution.

This normalizing flow based method not only provides the high power of expressive to approximate arbitrary complex target distribution, but also gives the probability of both sampling and evaluating on the samples at the same time. The policy model constructed with the normalizing flow, which is used to approximate the unknown target distribution, can "normalize" any complex reward distribution from observed space onto a simple and tractable defined prior distribution in the latent space. On the contrary, the prior distribution in the latent space can "flow" to the observed space through the transformation. In order to obtain a policy model built in such a way, we apply the policy search to update all the parameters of this policy model following the gradient. And we transfer the problem of maximizing the reward from the insights of policy search into an equivalent problem of minimizing the KL Divergence between the target distribution and policy model by adding an additional entropy objective. Moreover, the complete objective for the minimization problem consists of both I-projection and M-projection, which avoids the policy model premature collapsing to suboptimal. Since the updating strategy is based on gradient descent, we introduce a surrogate function to approximate the reward by fitting the discrete observed sample-reward pairs for maintaining the gradient and employ the reparameterization trick for stochastic optimization.

For evaluating we apply our method with the complex target distributions such as Gaussian mixture model reward with a full covariance matrix, the sparse reward distribution or binary reward distribution. And we also evaluate the algorithm in more realistic cases,

e.g. the approaching task of the planar robot and a simple grasping task of a typical grasper with two fingers in the simulation environment based on PyBullet physics engine. The demonstrated method is capable of learning high-quality approximations to complex and multi-modal target distributions with high power of expressive. And it is also able to provide the ability to generate new reliable samples that are expected to be inside in the real target distribution and to give the whole picture of the distribution.

However, our method has limitations. The method is biased, which means that the sample with the highest reward from the policy model may not have the highest reward from the sight of target reward distribution. This algorithm would also overestimate the target distribution, so to include too many unexpected samples. The reason could be that the policy model is sometimes so confident to predict high rewards on new samples without penalty. And for the learning phase, the policy model relies on a lot of samples, which causes inefficient training. Moreover, the algorithm is currently still limited to adapt to new contexts of new tasks. In the future, we are going to eliminate the bias of the learned policy model so to make the approximation more precise. We will reduce the data used to train the policy model for speeding up the learning phase by adding possible prior knowledge. We are still expecting to make our method to be more adaptive to more general cases by adding some observed evidence for a general construction of the policy model. And we are looking forward to applying our algorithm in a more sophisticated grasping task, e.g. adding more types of objects to grasp and building a more complex environment to place the objects. It would be quite necessary for us to adapt our method to the real robot in future work.

# Acknowledgments

My deep gratitude goes first to Prof. Dr. Jan Peters and Prof. Dr. Heinz Koeppl, who gave me a lot of valuable suggestions and helped me during the research. My appreciation also extends to Intelligent Autonomous Systems Group for all the advice and supports for the research, especially to my tutors Puze Liu and Julen Urain De Jesus, who guided me through the whole study and shared me the great experience as well as the idea in this research field.

I want to express my appreciation to my best friend Shujia Yan, who gave me the greatest supports and encouragement during the whole time. And I would like to thank my friend Zhichao Pang for his great suggestions. Also, I would thank Jiayi Zhang for her patience and encouragement. And finally, I appreciate my family for the encouraging and supporting, whose value to me is growing in my whole life.

# Bibliography


[1] Abbas Abdolmaleki et al. “Model-based relative entropy stochastic search”. In: *Advances in Neural Information Processing Systems* 28 (2015), pp. 3537–3545.

[2] Oleg Arenz, Gerhard Neumann, and Mingjun Zhong. “Efficient gradient-free variational inference using policy search”. In: *International conference on machine learning*. PMLR. 2018, pp. 234–243.

[3] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. “Variational inference: A review for statisticians”. In: *Journal of the American statistical Association* 112.518 (2017), pp. 859–877.

[4] Vladimir I Bogachev. *Measure theory*. Vol. 1. Springer Science & Business Media, 2007.

[5] Vladimir Igorevich Bogachev, Aleksandr Viktorovich Kolesnikov, and Kirill Vladimirovich Medvedev. “Triangular transformations of measures”. In: *Sbornik: Mathematics* 196.3 (2005), p. 309.

[6] Jeannette Bohg et al. “Data-driven grasp synthesis—a survey”. In: *IEEE Transactions on Robotics* 30.2 (2013), pp. 289–309.

[7] Eric Brochu, Vlad M Cora, and Nando De Freitas. “A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning”. In: *arXiv preprint arXiv:1012.2599* (2010).

[8] Konstantinos Chatzilygeroudis et al. “A survey on policy search algorithms for learning robot controllers in a handful of trials”. In: *IEEE Transactions on Robotics* 36.2 (2019), pp. 328–347.

[9] Junyoung Chung et al. “A recurrent latent variable model for sequential data”. In: *arXiv preprint arXiv:1506.02216* (2015).

[10] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. *A survey on policy search for robotics*. now publishers, 2013.

[11] Akash Kumar Dhaka et al. “Robust, accurate stochastic optimization for variational inference”. In: *arXiv preprint arXiv:2009.00666* (2020).

[12] Laurent Dinh, David Krueger, and Yoshua Bengio. “Nice: Non-linear independent components estimation”. In: *arXiv preprint arXiv:1410.8516* (2014).

[13] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density estimation using real nvp”. In: *arXiv preprint arXiv:1605.08803* (2016).

[14] David Eriksson et al. “Scalable global optimization via local bayesian optimization”. In: *arXiv preprint arXiv:1910.01739* (2019).

[15] SM Eslami et al. “Attend, infer, repeat: Fast scene understanding with generative models”. In: *arXiv preprint arXiv:1603.08575* (2016).

[16] Philippe Esling et al. “Universal audio synthesizer control with normalizing flows”. In: *arXiv preprint arXiv:1907.00971* (2019).

[17] Edouard Fouché, Junpei Komiyama, and Klemens Böhm. “Scaling multi-armed bandit algorithms”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 1449–1459.

[18] Peter I Frazier. “A tutorial on Bayesian optimization”. In: *arXiv preprint arXiv:1807.02811* (2018).

[19] Jacob Gardner et al. “Discovering and exploiting additive structure for Bayesian optimization”. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1311–1319.

[20] David E Golberg. “Genetic algorithms in search, optimization, and machine learning”. In: *Addion wesley* 1989.102 (1989), p. 36.

[21] Karol Gregor et al. “Draw: A recurrent neural network for image generation”. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1462–1471.

[22] Fangjian Guo et al. “Boosting variational inference”. In: *arXiv preprint arXiv:1611.05559* (2016).

[23] Nikolaus Hansen and Andreas Ostermeier. “Completely derandomized self-adaptation in evolution strategies”. In: *Evolutionary computation* 9.2 (2001), pp. 159–195.

[24] Jonathan Ho et al. “Flow++: Improving flow-based generative models with variational dequantization and architecture design”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2722–2730.

[25] Matthew D Hoffman et al. “Stochastic variational inference.” In: *Journal of Machine Learning Research* 14.5 (2013).

[26] Priyank Jaini, Kira A Selby, and Yaoliang Yu. “Sum-of-squares polynomial flow”. In: *arXiv preprint arXiv:1905.02325* (2019).

[27] Matthew J Johnson et al. “Composing graphical models with neural networks for structured representations and fast inference”. In: *arXiv preprint arXiv:1603.06277* (2016).

[28] Michael I Jordan et al. “An introduction to variational methods for graphical models”. In: *Machine learning* 37.2 (1999), pp. 183–233.

[29] Kirthevasan Kandasamy, Jeff Schneider, and Barnabás Póczos. “High dimensional Bayesian optimisation and bandits via additive models”. In: *International conference on machine learning*. PMLR. 2015, pp. 295–304.

[30] Sungwon Kim et al. “FloWaveNet: A generative flow for raw audio”. In: *arXiv preprint arXiv:1811.02155* (2018).

[31] Diederik P Kingma and Max Welling. “An introduction to variational autoencoders”. In: *arXiv preprint arXiv:1906.02691* (2019).

[32] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[33] Diederik P Kingma et al. “Improving variational inference with inverse autoregressive flow”. In: *arXiv preprint arXiv:1606.04934* (2016).

[34] Aaron Klein et al. “Fast bayesian optimization of machine learning hyperparameters on large datasets”. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 528–536.

[35] Ivan Kobyzev, Simon Prince, and Marcus A Brubaker. “Normalizing flows: Introduction and ideas”. In: *arXiv preprint arXiv:1908.09257* (2019).

[36] Junpei Komiyama, Junya Honda, and Hiroshi Nakagawa. “Optimal regret analysis of thompson sampling in stochastic multi-armed bandit problem with multiple plays”. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1152–1161.

[37] Volodymyr Kuleshov and Doina Precup. “Algorithms for multi-armed bandit problems”. In: *arXiv preprint arXiv:1402.6028* (2014).

[38] Solomon Kullback and Richard A Leibler. “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[39] Manoj Kumar et al. “Videoflow: A flow-based generative model for video”. In: *arXiv preprint arXiv:1903.01434* 2.5 (2019).

[40] Joel Lehman and Kenneth O Stanley. “Abandoning objectives: Evolution through the search for novelty alone”. In: *Evolutionary computation* 19.2 (2011), pp. 189–223.

[41] Ian Lenz, Honglak Lee, and Ashutosh Saxena. “Deep learning for detecting robotic grasps”. In: *The International Journal of Robotics Research* 34.4-5 (2015), pp. 705–724.

[42] Sergey Levine and Vladlen Koltun. “Guided policy search”. In: *International Conference on Machine Learning*. 2013, pp. 1–9.

[43] Daniel J Lizotte et al. “Automatic Gait Optimization with Gaussian Process Regression.” In: *IJCAI*. Vol. 7. 2007, pp. 944–949.

[44] Francis Maes et al. “Policy search in a space of simple closed-form formulas: Towards interpretability of reinforcement learning”. In: *International Conference on Discovery Science*. Springer. 2012, pp. 37–51.

[45] Jeffrey Mahler et al. “Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics”. In: *arXiv preprint arXiv:1703.09312* (2017).

[46] Horia Mania, Aurelia Guy, and Benjamin Recht. “Simple random search provides a competitive approach to reinforcement learning”. In: *arXiv preprint arXiv:1803.07055* (2018).

[47] Bogdan Mazoure et al. “Leveraging exploration in off-policy algorithms via normalizing flows”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 430–444.

[48] Andrew C Miller, Nicholas J Foti, and Ryan P Adams. “Variational boosting: Iteratively refining posterior approximations”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2420–2429.

[49] John Milnor and David W Weaver. *Topology from the differentiable viewpoint*. Princeton university press, 1997.

[50] Douglas Morrison, Peter Corke, and Jürgen Leitner. “Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach”. In: *arXiv preprint arXiv:1804.05172* (2018).

[51] Thomas Müller et al. “Neural importance sampling”. In: *ACM Transactions on Graphics (TOG)* 38.5 (2019), pp. 1–19.

[52] Amin Nayebi, Alexander Munteanu, and Matthias Poloczek. “A framework for Bayesian optimization in embedded subspaces”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4752–4761.

[53] George Papamakarios et al. “Normalizing flows for probabilistic modeling and inference”. In: *arXiv preprint arXiv:1912.02762* (2019).

[54] Matteo Papini et al. “Optimistic policy optimization via multiple importance sampling”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4989–4999.

[55] Jan Peters and Stefan Schaal. “Reinforcement learning of motor skills with policy gradients”. In: *Neural networks* 21.4 (2008), pp. 682–697.

[56] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. “Waveglow: A flow-based generative network for speech synthesis”. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 3617–3621.

[57] Justin K Pugh et al. “Confronting the challenge of quality diversity”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 2015, pp. 967–974.

[58] Deirdre Quillen et al. “Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6284–6291.

[59] Carl Edward Rasmussen. “Gaussian processes in machine learning”. In: *Summer school on machine learning*. Springer. 2003, pp. 63–71.

[60] Joseph Redmon and Anelia Angelova. “Real-time grasp detection using convolutional neural networks”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 1316–1322.

[61] Jeffrey Regier, Michael I Jordan, and Jon McAuliffe. “Fast black-box variational inference through stochastic trust-region optimization”. In: *arXiv preprint arXiv:1706.02375* (2017).

[62] Walter Rudin et al. *Principles of mathematical analysis*. Vol. 3. McGraw-hill New York, 1964.

[63] Walter Rudin. *Real and complex analysis*. Tata McGraw-hill education, 2006.

[64] Olivier Sigaud and Freek Stulp. “Policy search in continuous action domains: an overview”. In: *Neural Networks* 113 (2019), pp. 28–40.

[65] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. “Practical bayesian optimization of machine learning algorithms”. In: *arXiv preprint arXiv:1206.2944* (2012).

[66] Felipe Petroski Such et al. “Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning”. In: *arXiv preprint arXiv:1712.06567* (2017).

[67] Michalis K Titsias and Francisco Ruiz. “Unbiased implicit variational inference”. In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR. 2019, pp. 167–176.

[68] Ahmed Touati et al. “Randomized value functions via multiplicative normalizing flows”. In: *Uncertainty in Artificial Intelligence*. PMLR. 2020, pp. 422–432.

[69] Julen Urain et al. “ImitationFlow: Learning Deep Stable Stochastic Dynamic Systems by Normalizing Flows”. In: *arXiv preprint arXiv:2010.13129* (2020).

[70] Martin J Wainwright and Michael Irwin Jordan. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008.

[71] Zi Wang et al. “Batched large-scale bayesian optimization in high-dimensional spaces”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2018, pp. 745–754.

[72] Ziyu Wang et al. “Bayesian optimization in a billion dimensions via random embeddings”. In: *Journal of Artificial Intelligence Research* 55 (2016), pp. 361–387.

[73] Patrick Nadeem Ward, Ariella Smofsky, and Avishek Joey Bose. “Improving exploration in soft-actor-critic with normalizing flows policies”. In: *arXiv preprint arXiv:1906.02771* (2019).

[74] Jonathan Weisz and Peter K Allen. “Pose error robust grasping from contact wrench space metrics”. In: *2012 IEEE international conference on robotics and automation*. IEEE. 2012, pp. 557–562.

[75] Mengyuan Yan et al. “Learning probabilistic multi-modal actor models for vision-based robotic grasping”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 4804–4810.

# A. Reparameterization Trick

In the field of variational inference and variational autoencoder, it is essential to apply the reparameterization trick so that the backpropagation of gradient can flow through a random node to update the gradient-based estimator.

Assume that we have the expectation

$$\mathbb{E}_{p(\boldsymbol{x};\boldsymbol{\theta})}\left[f_{\boldsymbol{\theta}}(\boldsymbol{x})\right] ,$$

where the $p(\boldsymbol{x};\boldsymbol{\theta})$ is the considered distribution w.r.t. the parameter $\boldsymbol{\theta}$ and $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is a differentiable function. Then, we are going to compute the gradient of this expectation w.r.t. the parameter $\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{p(\boldsymbol{x};\boldsymbol{\theta})}\left[f_{\boldsymbol{\theta}}(\boldsymbol{x})\right] = \int_{\boldsymbol{x}} f_{\boldsymbol{\theta}}(\boldsymbol{x})\nabla_{\boldsymbol{\theta}}p(\boldsymbol{x};\boldsymbol{\theta})\, d\boldsymbol{x} + \mathbb{E}_{p(\boldsymbol{x};\boldsymbol{\theta})}\left[\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x})\right] .$$

Actually, the gradient is not guaranteed to be obtained, when the $\nabla_{\boldsymbol{\theta}}p(\boldsymbol{x};\boldsymbol{\theta})$ is not able to be analytically solved. So we define a new prior distribution

$$\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$$

so that the $\boldsymbol{x}$ can be obtained by

$$\boldsymbol{x} = g_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}, \boldsymbol{x}) .$$

Finally, the gradient can be computed by

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}}\mathbb{E}_{p(\boldsymbol{x};\boldsymbol{\theta})}\left[f_{\boldsymbol{\theta}}(\boldsymbol{x})\right] &= \nabla_{\boldsymbol{\theta}}\mathbb{E}_{p(\boldsymbol{\epsilon})}\left[f(g_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}, \boldsymbol{x}))\right] \\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\theta}} f(g_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}, \boldsymbol{x}))\right] \\
&\approx \frac{1}{L}\sum_{l=1}^{L} \nabla_{\boldsymbol{\theta}} f(g_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}_l, \boldsymbol{x}_l)) ,
\end{aligned}$$

where the $\boldsymbol{\epsilon}_l$ and $\boldsymbol{x}_l$ represent the $l$-th element of the vector $\boldsymbol{\epsilon}$ and $\boldsymbol{x}$ respectively.

# B. Robot Grasping Task Result

The results of the generated samples from the 1st to 25th high approximated rewards following the trained policy model are shown in the following figures, while the grasping task is solved in the simplified 3 dimensions case.

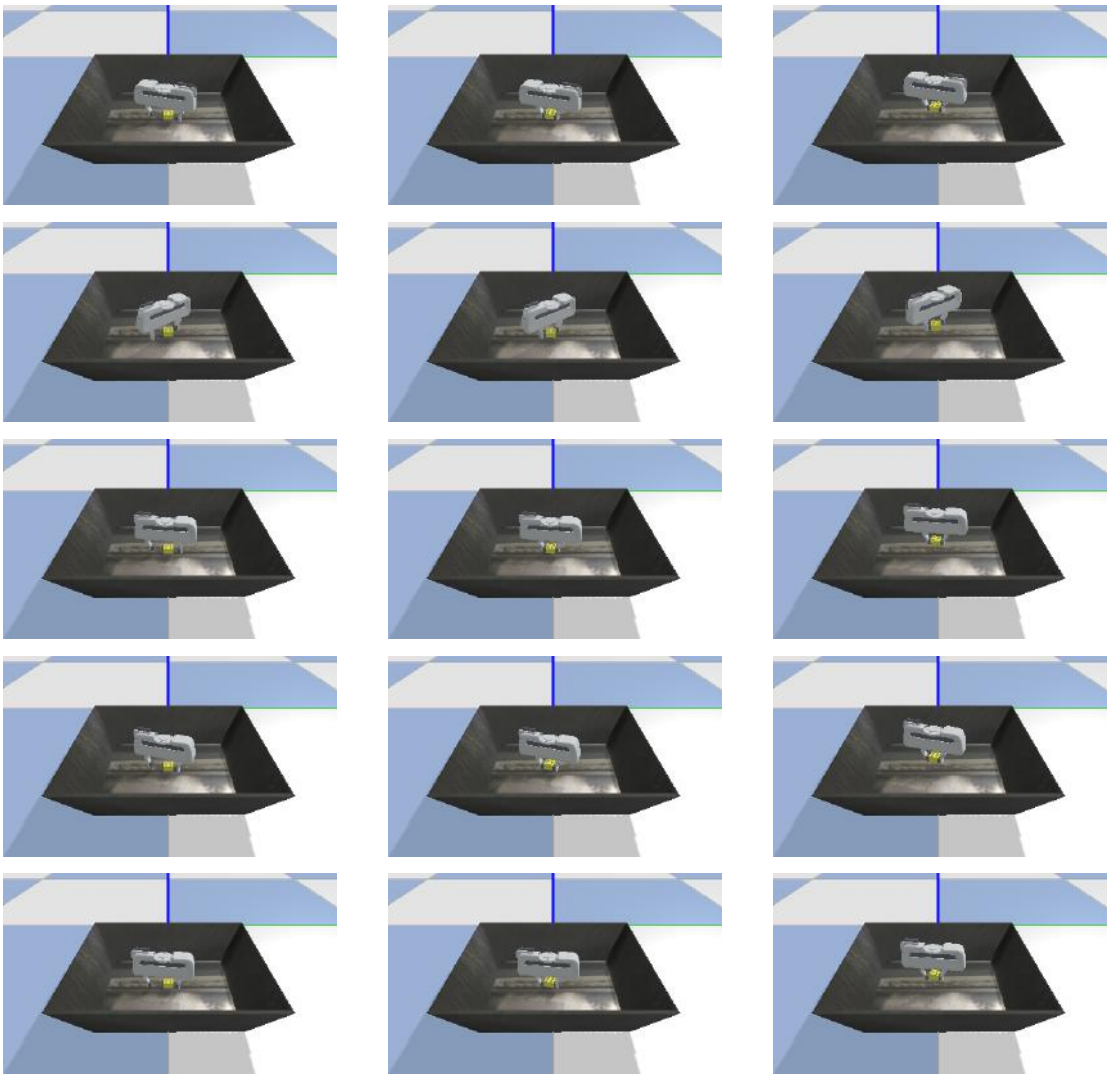


Figure B.1.: The grasping attempts of the generated samples from the 1st to 5th high approximated rewards following the trained policy model. Sub-figures in each row show one complete successful grasping attempt.

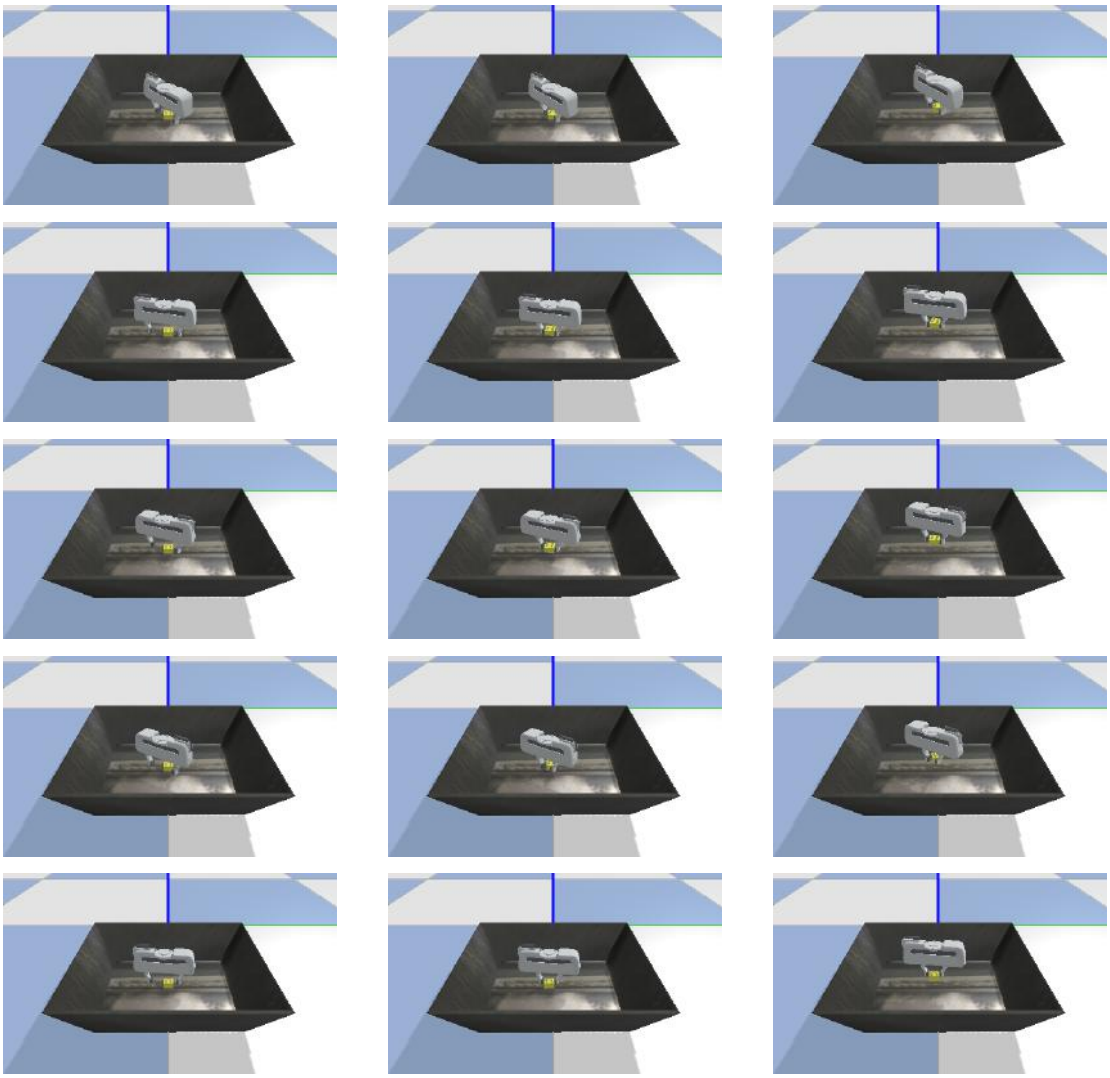


Figure B.2.: The grasping attempts of the generated samples from the 6th to 10th high approximated rewards following the trained policy model. Subfigures in each row show one complete successful grasping attempt.

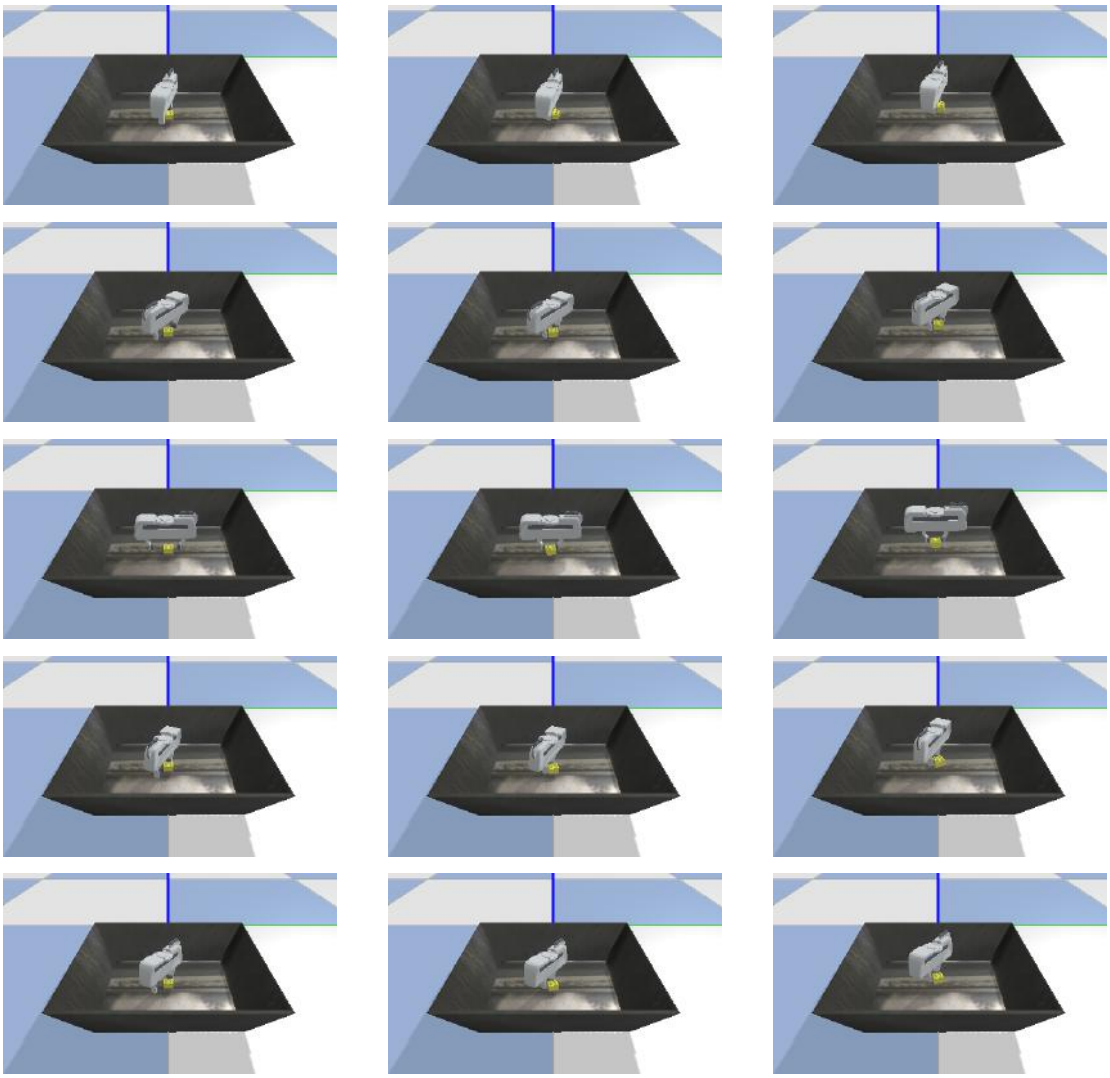


Figure B.3.: The grasping attempts of the generated samples from the 11th to 15th high approximated rewards following the trained policy model. Sub-figures in each row show one complete successful grasping attempt.

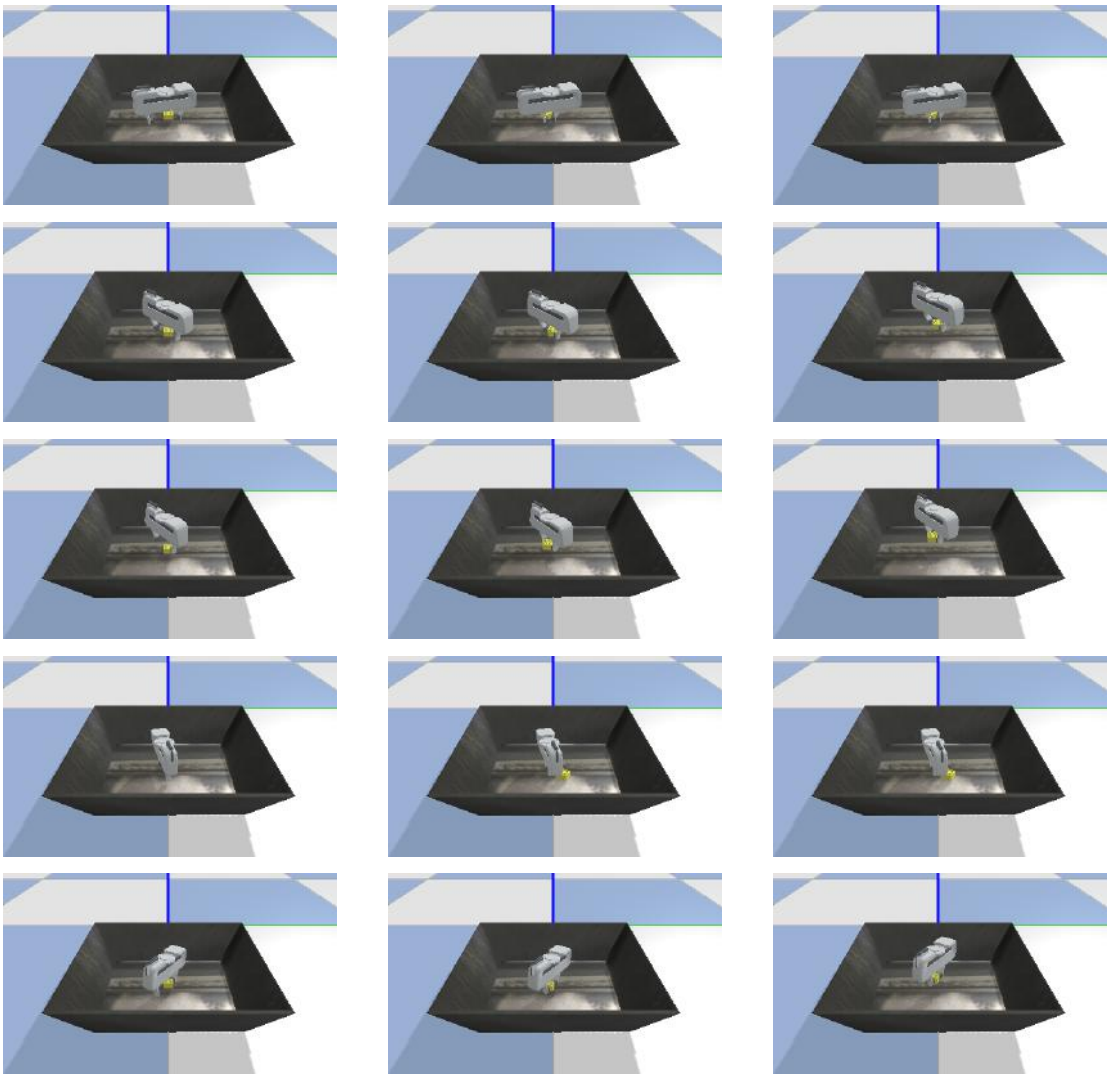


Figure B.4.: The grasping attempts of the generated samples from the 16th to 20th high approximated rewards following the trained policy model. Subfigures in each row show one complete successful grasping attempt.

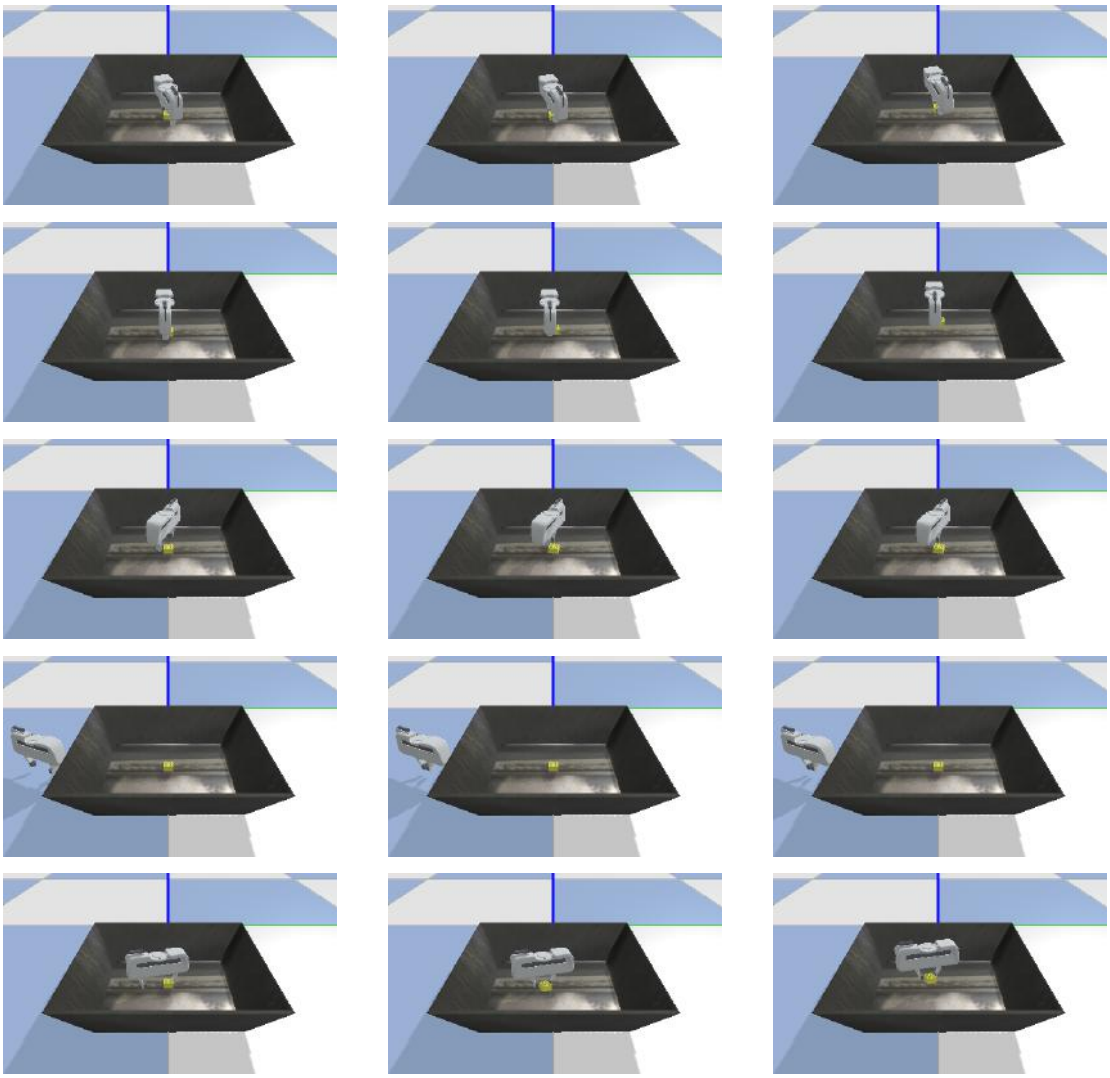


Figure B.5.: The grasping attempts of the generated samples from the 21st to 25th high approximated rewards following the trained policy model. Subfigures in each row show one complete successful grasping attempt.