Combining Reinforcement Learning and Imitation Learning through Reward Shaping for Continuous Control

by

Yuchen Wu

A thesis submitted in conformity with the requirements
for the degree of Bachelor of Applied Science

Department of Engineering Science
University of Toronto

Combining Reinforcement Learning and Imitation Learning through Reward Shaping for Continuous Control

Yuchen Wu
Bachelor of Applied Science

Department of Engineering Science
University of Toronto
2020

Abstract

The potential benefits of reinforcement learning to real robotics systems are limited by its uninformed exploration that leads to lack of sample efficiency. To address this drawback, we propose a method that combines reinforcement and imitation learning by shaping the reward function with a state-action dependent potential that is learned from demonstrations. The shaping potential specifies high-value areas of the state-action space that are worth exploring first and help accelerate the policy learning. In particular, we use deep generative models to represent the shaping potential and optimize the policy using off-policy actor-critic style RL algorithms, such that our method can apply to continuous state and action spaces. We evaluate our methods on a wide range of simulated manipulation tasks and show that our method significantly improves the sample efficiency of reinforcement learning. More crucially, unlike the majority of existing methods that incorporate demonstrations as hard constraints on policy optimization, our approach only uses the demonstration data to bias exploration and thereby does not suffer from sub-optimal demonstrations.

Acknowledgements

I would like to thank Professor Jonathan Kelly, Professor Florian Shkurti and Trevor Ablett for the guidance and support over the last year.

I would also like to thank Melissa Mozifian who helped me carry out the initial experimental results of this project.

Additionally, I am grateful to my friends for the support and encouragement they have given me throughout my undergraduate studies.

Contents

1 Introduction **1**

2 Related Work **3**

2.1 Addressing the Sample Efficiency Problem in Reinforcement Learning . . . 3

2.1.1 Model-Based RL . . . 3

2.1.2 Model-Free RL . . . 4

2.1.3 Rewards and Exploration Bonuses . . . 5

2.2 Handling Distribution Shift in Imitation Learning . . . 6

2.3 Overcoming Demonstration Sub-Optimality via Combined RL and IL . . . 7

3 Background **9**

3.1 Off-Policy Actor-Critic Algorithms for Continuous Control . . . 9

3.2 State-Action, Potential-Based Reward Shaping . . . 11

3.3 Flow Based Generative Models - Normalizing Flows . . . 12

3.4 Generative Adversarial Networks . . . 13

4 RL from Demonstration via Shaping through Generative Models **15**

4.1 Assumptions and Preliminaries . . . 15

4.2 Shaping Potential Based on Generative Models . . . 16

4.2.1 Shaping Potential Based on Normalizing Flows . . . 16

4.2.2 Shaping Potential Based on Generative Adversarial Networks . . . 17

4.3 Updated Learning Objectives for SAC and TD3 . . . 17

4.4 Other Changes to TD3 and SAC that Facilitate Learning from Demonstrations and Sparse Rewards . . . 18

4.5 Algorithm Summary and Pseudo-Code . . . 20

5 Experiments **22**

5.1 Baselines . . . 23

5.2 Specification and Categorization of Tasks . . . 24

5.3 Generating Demonstrations . . . 25

5.4 Illustrative Example 1: Visualizing and Evaluating Shaping Potentials . . . 26

5.5 Illustrative Example 2: Visualizing and Evaluating Policies Learned from Sub-Optimal Demonstrations 31
5.6 Experiments on Short-Horizon Tasks 32
5.7 Experiments on Long-Horizon Tasks 34
5.8 Discussion 35

6 Conclusion and Future Work **37**

A Hyper-Parameters Used for All Manipulation Tasks **38**

B Visualization of the Manipulation Tasks **40**

Bibliography **42**

List of Tables

5.1 Specification and categorization of tasks 24

A.1 Hyper-parameters used for all tasks 39

List of Figures

5.1 Description and demonstration of the *Reach-2D* task 26
5.2 Direct visualization of GAN/normalizing flow based shaping potentials for *Reach-2D* 27
5.3 Indirect visualization of GAN/normalizing flow based shaping potentials: $\Phi - \sigma^2$ plot 28
5.4 $\Phi - \sigma^2$ plot of normalizing flow based shaping potentials trained with different hyper-parameters 29
5.5 Results of running TD3+Shaping and baseline algorithms on *Reach-2D* with optimal demonstrations 30
5.6 Results of running TD3+Shaping and baseline algorithms on *Gym-PegInsertion-2D* 31
5.7 Results of running TD3+Shaping and baseline algorithms on *Gym-PegInsertion* with sub-optimal demonstrations 32
5.8 Results of running TD3+Shaping and baseline algorithms on *Gym-PickAndPlace* with sub-optimal demonstrations 33
5.9 Results of running TD3+Shaping, TD3+BC and TD3+BC+QFilter on *MW-PressButton* 35

B.1 Visualization of tasks implemented based on OpenAI Gym Robotics API . 40
B.2 Visualization of tasks in MetaWorld 41

Chapter 1

Introduction

Machine learning techniques, typically imitation learning (IL) and reinforcement learning (RL), have been making significant progress in a wide variety of complex real-world robotic control and manipulation tasks [1][2]. However, the drawbacks of IL and RL are both obvious. IL, where the policy is trained to mimic demonstrated behaviours, requires a large amount of training data or on-line corrections [3], which is not desired or even possible in many real-world robotic systems. It also performs poorly when the demonstrations are sub-optimal. RL, on the other hand, suffers from slow convergence due to lack of data efficiency and unnecessary interactions with the environment. In addition, designing a dense, informative reward function can be challenging for many tasks in robotics.

Extensive amount of works have been trying to address the aforementioned problems by combining the two methods, IL and RL, in various ways. For example, *Inverse RL (IRL)* and *Adversarial IL* methods, such as GAIL [4], AIRL [5] and DAC [6] avoid designing the reward function by inferring it from demonstrations. Rajeswaran et al. [7] and Vercerik et al. [8] proposed to add a *Behavioral Cloning (BC)* objective in addition to the RL objective to the policy, biasing the policy towards demonstrated behaviors. However, a common problem with these methods is that they assume the demonstrations to be optimal, and their performance may drop significantly when sub-optimal demonstrations are given. Methods that benefit from both RL and IL but do not suffer from suboptimal demonstrations also exist. For example, Mulling et al. [9] proposed to initialize the policy via BC and the fine-tune it via pure RL afterwards, which showed decent performance in training a real robot arm to play table tennis. Nair et al. [10] introduced *Q-filter* where the policy only learns to imitate good demonstrated behaviors based on the Q-value estimates. Though these methods take sub-optimality of demonstrations into consideration, they are not very effective in practice due to the distribution shift issue [3] and Q-value estimation errors. Brys et al. [11] proposed a novel idea of using state-action based *reward shaping* [12] to incorporate sub-optimal demonstrations into RL without biasing the learned policy. However, the class of shaping potentials they considered was limited to discrete action and low dimensional state space.

In this thesis, we extend the idea in [11] to high dimensional, continuous state and action spaces so that it can be used for robotic control and manipulation tasks. We use a *shaping potential* in the form of deep generative models, e.g. Generative Adversarial Networks (GANs) [13] and normalizing flows (NFs) [14], and incorporate reward shaping into off-policy actor-critic algorithms such as TD3 [15] and SAC [16] that naturally support continuous state and action spaces. We evaluate our work using two popular RL and IL environment suites, MetaWorld [17] and OpenAI Gym [18], which include many control and manipulation tasks with well documented baseline results using the state-of-the-art RL algorithms. We compare our method against several popular RL and IL methods, especially methods that can overcome sub-optimality in demonstrations in theory, e.g. RL with BC pre-training [9] and RL with BC plus Q-filter [10].

In summary, we notice that most existing methods that combine RL and IL cannot gracefully handle sub-optimal demonstrations, and the performance of the policies trained via these methods drop significantly when sub-optimal demonstrations are given. To address this problem, we propose to combine IL and RL using reward shaping through deep generative models. We show that our method is applicable to complex robotic control and manipulation tasks in simulation, and it can perform better in handling sub-optimal demonstrations than existing methods.

Chapter 2

Related Work

2.1 Addressing the Sample Efficiency Problem in Reinforcement Learning

Though reinforcement learning (RL) algorithms have demonstrated significant results on a wide range of simulated robotic control and manipulation tasks, their applicability to real world robotic systems is hampered by their low sample efficiency. Over the past a few decades, significant amount of work has been dedicated to increase the sample efficiency of RL in different ways. In this section, we review some representative research directions that have shown promising result in addressing this problem for robotic tasks, which include learning the environment model (i.e. model-based RL), leveraging off-policy data, designing unbiased reward function and some efficient exploration strategies.

2.1.1 Model-Based RL

One of the major branching points in RL algorithms is based on whether a model of the environment is learned. Model-based RL algorithms have always been an active RL research field and are known in general to outperform model-free RL methods in terms of sample efficiency [19]. In practice, they have also been successfully applied to many simulated as well as real-world robotic systems, such as inverted pendulums [20], legged robots [21], and manipulators [22]. However, most model-based RL algorithms that demonstrate great sample efficiency use relatively low-capacity model classes such as Gaussian process [20][23] or time-varying linear models [24][25] to represent the environment, which introduce additional assumptions and thereby only applicable to simple tasks with low dimensional state space. Although more expressive model classes, e.g. neural networks, have also been adopted in recent model-based RL algorithms [26][27][28], the advantage of these methods in terms of sample efficiency are not as much when used in more complex environments. This is because learning the environment model requires more training data, and the learned model is more prone to over-fit.

In particular, when it comes to image-based environments, model-based methods typically learn a latent space model of the environment with extra loss added to match the latent space to the original observation space, such as SLAC [29] and PlaNet [30]. The problem with these methods is that they achieve better sample efficiency than model-free RL at the cost of high system complexity and multiple learning objectives. In addition to the losses on policy and value optimizations, there are several auxiliary losses such as transition loss, observation loss and reward loss, which require careful balancing during training. Therefore, these methods are known to be sensitive to hyper-parameter settings and hard to transform from one environment to another.

In general, the challenge of model-based RL is the representation and learning of the environment model. In high dimensional spaces, learning a good model of the environment may be more difficult than learning the policy itself, and the agent may exploit biases in the learned model, resulting in sub-optimal behavior in the actual environment. Due to this issue, the difficulty of generalizing model-based RL algorithms to complex environments is much higher than that for model-free RL algorithms. Moreover, for image-based environments, there are model-free RL algorithms, e.g. SAC+AE [31], that show similar sample efficiency to model-based RL methods, but with only minor modifications to the commonly used model-free RL methods. Therefore, in this work, we put our focus on incorporating demonstrations into recent model-free RL algorithms, as discussed in the next section, to avoid building an overly complicated system and such that our method can possibly generalize to image-based environments easily.

2.1.2 Model-Free RL

Unlike model-based RL algorithms, model-free RL methods are usually easy to implement and apply to a wide range of environments with varying complexity. A large amount of model-free RL algorithms have been proposed over the last few years, and some them have shown working very well across multiple domains, such as TD3 [15], SAC [16], DDPG [32], TRPO [33], PPO [34], etc. These algorithms have also been commonly used as baselines in many benchmark RL environments [17][18][35].

In model-free RL, there used to be two main approaches to represent and train agent, policy optimization and Q-learning. Methods using the policy optimization approach, such as TRPO and PPO, optimize the policy directly with respect to the performance objective, and always use data collected by the most recent policy (i.e. on-policy). This class of methods are generally more stable and insensitive to hyperparameters as compared to Q-learning, and they naturally support continuous observation and action spaces, which is the case for most robotic domains. However, the poor sample efficiency of these methods is predetermined by their on-policy learning constraint, and the resulting computational cost quickly becomes intractable as the task complexity increases. In contrary, methods using the Q-learning approach, such as DQN [36] and C51 [37], aim to learn the optimal

action-value function based on the Bellman equation [38]. These methods are more sample efficient as they allow re-use of past experiences. However, pure Q-learning methods tend to be unstable with many failure modes [39] and are only applicable to environments with discrete action spaces.

Recently, the Deterministic Policy Gradient (DPG) algorithm proposed by Silver et al. [40] gives a way to combine policy optimization and Q-learning by using a parametrized deterministic policy to perform the optimization in Q-learning, which leads to an actor-critic style RL algorithm. This formulation makes it possible to apply off-policy learning to environments with continuous action spaces and balance between sample efficiency and stability. Their subsequent work, DDPG [32], which leverages deep neural networks to parametrize the policy and the action-value function, shows results in several robotic control and manipulation tasks with significantly better sample efficiency than on-policy algorithms [41]. However, DDPG tends to be more sensitive to hyper-parameters due to its Q-learning manner. More recently, two variants of DDPG, known as TD3 and SAC, use clipped double Q learning and entropy regularized RL [42] to further address the action-value function approximation and exploration issues of DDPG, respectively. Both algorithms show improved performance in both cumulative return and sample efficiency on many complex tasks in robotics. In particular, SAC+AE [31] simply trains SAC in latent space with an auxiliary reconstruction loss and shows comparable sample efficiency to the state-of-the-art model-based methods [29][30] on several image-based control tasks from DeepMind Control Suite [35]. Moreover, off-policy algorithms can learn from experiences not necessarily generated by the policy, which provides more options to incorporate environmental or human knowledge to the learning process, such as expert demonstrations. Therefore, in developing our method to combine RL and IL, we primarily consider using off-policy actor-critic style methods, i.e. TD3 and SAC, as our base RL algorithms.

2.1.3 Rewards and Exploration Bonuses

Besides the representation and training strategies of the policy, the problem of low sample efficiency in RL is also caused by the sparse and uninformative reward signal used in many robotic tasks, as well as inefficient exploration strategies that cause shallow exploration and unnecessary re-visit of many states.

Designing an informative reward function is challenging in many cases as it requires a fairly amount of task-specific prior knowledge from the experimenter and, if not designed properly, may lead to *reward hacking* as discussed by Ng et al. [43] and Amodei et al. [44], where the agent ends up maximizing the given reward without performing the intended task. A common strategy to circumvent this problem is to use *reward shaping* [12][43], where the optimal policy remains unchanged between the Markov Decision Processes before and after the reward transformation. This policy invariant property provides more flexibility and tolerance in reward function design. Therefore, reward shaping has been used in many

recent RL methods [45][46][47][48]. Our proposed method also builds upon reward shaping, however, our method is different as we consider the case where the observation and action spaces are both continuous and the shaping rewards come from expert demonstrations.

Besides reward shaping, existing works on exploration in RL also introduced the notion of *intrinsic motivation/rewards* which is usually an exploration bonus additional to the environment reward signal. There are many ways to generate this exploration bonus, such as using *visitation count* [49][50][51], which encourages the agent to explore unseen states, and *information gain* [52][53][54], which encourages the agent to perform actions that minimize certain prediction error. The prediction error can be in many forms. For example, VIME [52] uses the error of environment dynamic prediction, and RND [55] uses the error of predicting the output of a randomly initialized neural network. However, although many of these methods show improved sample efficiency, the actual benefit from them are limited as they do not incorporate any prior knowledge of the environment. Also, as we consider the safety issues for many real world robotic tasks, there might be more states which the agent should avoid than the states which the agent should be encouraged to explore.

2.2 Handling Distribution Shift in Imitation Learning

In contrast to RL, imitation learning (IL) does not require a pre-defined reward signal but learns by imitating the demonstrations from experts such as humans [56]. The simplest IL method is *Behavioral Cloning (BC)*, where the the policy is learned directly from expert demonstrations via supervised learning. Although BC has been applied successfully in tasks like autonomous driving [57][58], it is prone to compounding errors that lead to distribution mismatch between states visited by the policy and by the demonstrations [3]. In other words, small errors in the policy cause the agent to deviate from the training distributions, making future errors more likely. In this section, we briefly review the main methods for imitation learning that address this distribution shift problem.

Ross et al. [3] proposed DAGGER that addressed the distribution shift problem by combining BC with dataset aggregation. In DAGGER, the policy is executed to generated a certain number of trajectories, and the demonstrator is then asked for on-line corrections to the off-course states. After that, the corrected data is added to the training dataset so that the training distribution matches the test distribution. There are a few drawbacks of this method. First, it requires access to the demonstrator throughout the training process, which is not desired or possible in many situations. Second, although DAGGER solves the distribution shift problem, it still requires much more demonstrations when compared to other IL methods proposed concurrently or later, which will be reviewed below. Third, there is no direct way to mark a certain state as bad or undesirable, and the demonstrator may be asked to provide demonstrations at states that are not likely to be visited normally.

Another popular IL method is *Inverse Reinforcement Learning (IRL)* whose goal is

to infer the underlying reward function from the demonstrated behavior [59]. Maximum Entropy IRL (MaxEnt IRL) [60], for example, models the demonstrations with a Boltzmann distribution based on a learned cost function $c_\theta(\tau)$ parametrized by θ:

$$p_\theta(\tau) = \frac{1}{Z}\exp(-c_\theta(\tau))$$

where $\tau = \{s_1, a_1, s_2, a_2, ...\}$ is a trajectory and the partition function Z is the integral of $\exp(-c_\theta(\tau))$ over all possible trajectories. The parameters θ are optimized to maximize the likelihood of the demonstrations. However, estimating the partition function Z is computationally difficult for large, continuous domains. IRL algorithms typically alternate between updating the cost function $c_\theta(\tau)$ and running RL to find the optimal policy with respect to the current $c_\theta(\tau)$.

Recently, Ho & Ermon [4] and Finn et al. [61] suggested that GANs can be viewed as a sample-based algorithm for the MaxEnt IRL problem. The method presented in Ho & Ermon [4], known as Generative Adversarial Imitation Learning (GAIL), uses the adversarial structure of GAN and trains a policy to match the state-action distribution in expert demonstrations. This method simultaneously learns the policy and the reward function that is implicit within the discriminator, and has been successfully applied to various control tasks in the OpenAI Gym Mujoco environments [18]. There are several algorithms, such as AIRL [5] and DAC [6], that use similar adversarial structure as GAIL, and we refer this class of methods as *Adversarial Imitation Learning (Adversarial IL)*.

Both IRL and Adversarial IL, though already leverage RL in their learning process, are considered to be imitation learning because they do not use a pre-defined reward function that indicates the agent's performance. This lack of indication causes the performance of a learned policy to be limited by the performance of demonstrations. In real world robotic systems, optimal demonstrations are difficult to obtain due to system noises and human capacities. In the following section, we review some methods that not only learn from demonstrations but also make use of a pre-defined reward function to achieve higher performance than the demonstrations.

2.3 Overcoming Demonstration Sub-Optimality via Combined RL and IL

Designing a dense, informative reward function is difficult for most tasks in robotics, but a sparse reward that only indicates the task objective is normally easy to provide. For example, for object insertion tasks [8], it is sufficient to give a reward of 0 when the object is inserted and -1 otherwise. The lack of information in guiding the agent towards the objective can be compensated by expert demonstrations in several ways, such as initializing the policy using IL, learning guided exploration bonuses from demonstrations, and augmenting the

training data for off-policy methods.

Initializing the policy via IL and then fine-tuning with RL is the simplest way to combine and benefit from both RL and IL, and it has been successfully applied to real world robotic tasks [9]. Moreover, the learned policy will not be biased towards demonstrated actions so that this method does not suffer from sub-optimal demonstrations. However, the effectiveness of this method can be limited because, at early stage of RL, bad experiences may be collected due to the distribution drift problem [3] and these experiences can destroy initialization of the policy. Reward shaping, which our method builds upon, is equivalent to Q-value initialization as discussed by Wiewiora et al [12], because the shaping potential biases the Q-value by exactly the same of its amount. Initializing the Q-value function should be preferred over policy initialization because it does not suffer from bad experiences. Prior work most closely related to ours is [11], which also incorporates demonstrations via reward shaping. However, their method is limited to low dimensional state space and discrete action space, and thereby does not apply to most robotic tasks. Our method uses deep generative models to represent shaping potential so that it can scale up to high-dimensional state and action spaces. In addition, we use off-policy actor-critic methods to handle continuous action spaces.

The second idea of combining RL and IL is to use demonstrations as an augmented dataset for RL, as done in [8][10][62]. However, this idea has to combine with other IL and RL tricks, such as importance sampling [63][64], because otherwise the augmented data will not be helpful due to errors introduced by Q-value extrapolation [65]. Our method also adds demonstrations to the replay buffer. Unlike [8][10], we assume that the demonstrations contain only state-action pairs without corresponding rewards and subsequent states, which is a more realistic assumption for demonstrations generated for real-world robotic tasks.

The third idea is to have a hybrid learning objective from both RL and IL, as used in [7][10][66]. Rajeswaran et al. [7] combined the two objectives by simply updating the policy via a weighted sum of the policy optimization loss and a BC loss, while Zhu et al. [66] did it by combining the pre-defined reward function with a learned reward function from demonstrations via Adversarial IL. The problems with these methods are 1) the relative weighting between the RL and IL objectives is hard to tune, and 2) the learned policy will be biased towards demonstrated behaviors, which might be suboptimal. Nair et al. [10] used the same way of combining the two objectives as [7], but introduced *Q-Filter* to handle sub-optimal demonstrations, where they only kept the IL loss for which the demonstrated action has higher estimated Q-value than the action returned by the policy. However, we show that Q-filter may not only filter bad demonstrations but also filter good demonstrations due to Q-value estimation error, and thereby limit the benefit from the IL objective.

Chapter 3

Background

3.1 Off-Policy Actor-Critic Algorithms for Continuous Control

We consider an infinite-horizon Markov Decision Process (MDP), which is described by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}(s_{t+1}|s_t, a_t)$ is the transition probability, $\mathcal{R}(s_t, a_t, s_{t+1}) \in \mathbb{R}$ is the reward function, and $\gamma \in (0, 1]$ is the discount factor [38]. At each discrete time step t with a given state $s_t \in \mathcal{S}$, the agent selects an action $a \in \mathcal{A}$ with respect to its policy $\pi : \mathcal{S} \to \mathcal{A}$, receiving a reward r and the new state of the environment $s_{t+1} \in \mathcal{S}$. The return at time t, R_t, is defined as the discounted sum of reward starting from time t: $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$.

The goal of ordinary RL is to find a policy π that maximizes the expected return:

$$J(\pi) = E_{s_i \sim p_\pi, a_i \sim \pi}\left[R_0\right] = E_{s_i \sim p_\pi, a_i \sim \pi}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right] \tag{3.1}$$

where p_π is the discounted state visitation distribution of policy π. In actor-critic style methods, the policy is also called the *actor*, and is updated through an action-value function Q, known as the *critic*, which describes the expected return after taking an action at state s_t and following policy π:

$$Q^\pi(s_t, a_t) = E_{r_{i \geq t}, s_{i>t} \sim \rho_\pi, a_{i>t} \sim \pi}[R_t | s_t, a_t] \tag{3.2}$$

We consider that both the actor and the critic are represented by deep neural networks where $\pi = \pi_\phi(s)$, parametrized by ϕ, and $Q = Q_\theta(s, a)$, parametrized by θ. The critic Q_θ can be learned using temporal difference learning [67] based on the Bellman equation [38] that specifies the following relationship between (s_t, a_t) and (s_{t+1}, a_{t+1}):

$$Q^\pi(s_t, a_t) = r + \gamma E_{s_{t+1}, a_{t+1}}[Q^\pi(s_{t+1}, a_{t+1})] \text{ where } a_{t+1} \sim \pi(s_{t+1}) \tag{3.3}$$

In case where the policy π_ϕ is deterministic and the action space is continuous, π_ϕ can be updated based on the *Deterministic Policy Gradient Theorem* [40]:

$$J_\pi(\phi) = \mathbb{E}_{s\sim p_\pi}\left[Q(s,a)|_{a=\pi(s)}\right] \tag{3.4}$$

$$\nabla_\phi J_\pi(\phi) = \mathbb{E}_{s\sim p_\pi}\left[\nabla_a Q^\pi(s,a)|_{a=\pi(s)}\nabla_\phi \pi_\phi(s)\right] \tag{3.5}$$

To address the over-estimation problem in Q-value approximation, Fujimoto et al. proposed in their TD3 algorithm [15] to use a clipped double Q-learning method coupled with target policy smoothing, which produces the following objective for Q_θ:

$$y = r_t + \gamma \min_{i=1,2} Q_{\theta_i'}(s_{t+1}, \pi_{\phi'}(s_{t+1}) + \epsilon^{\text{TD3}}) \text{ with } \epsilon^{\text{TD3}} \sim \text{clip}(\mathcal{N}(0, \sigma^{\text{TD3}}), -c^{\text{TD3}}, c^{\text{TD3}}) \tag{3.6}$$

$Q_{\theta_1'}$ and $Q_{\theta_2'}$ are frozen target networks that make the objective y be fixed over multiple updates. Selecting the smaller Q estimation from two target networks alleviates the over-estimation problem in Q-learning, and the clipped Gaussian noise ϵ^{TD3} added to the target policy π_ϕ' avoids overfitting to narrow peaks in value function estimation. The loss function for Q_θ is then:

$$J_Q(\theta) = E_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{B}}\left[(Q_\theta(s_t,a_t) - y)^2\right] \tag{3.7}$$

where $\mathcal{B}$ is the replay buffer.

An alternative to the above formulation is *Maximum Entropy RL* discussed in Ziebart et al. [42], where the goal is to maximize both the expected return and entropy of the policy:

$$J^{\text{ME}} = E_{s_i\sim p_\pi, a_i\sim\pi}\left[\sum_{t=0}^{T} \gamma^t r(s_t,a_t) + \alpha H(\pi(\cdot|s_t))\right] \tag{3.8}$$

Given this objective, the Bellman target and loss function for Q_θ are as follows:

$$y^{\text{ME}} = r_t + \gamma\, \mathbb{E}_{a_{t+1}\sim\pi_\theta(s_{t+1})}\left[\min_{i=1,2} Q_{\theta_i'}(s_{t+1},a_{t+1}) + \alpha \log\pi(a_{t+1}|s_{t+1})\right] \tag{3.9}$$

$$J_Q^{\text{ME}}(\theta) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{B}}\left[(Q(s_t,a_t) - y)^2\right] \tag{3.10}$$

For continuous action spaces, Haarnoja et al. proposed in their SAC algorithm [16] to use a differentiable stochastic policy designed to be a squashed Gaussian:

$$\pi_\phi = \tanh(\mu_\phi, \sigma_\phi \odot \xi) \text{ where } \xi \sim \mathcal{N}(0, I) \tag{3.11}$$

and the policy is updated again via gradient descent with the reparameterization trick [68]. The policy is reparametrized using a neural network transformation:

$$a_t = f_\phi(\epsilon; s_t) \text{ with } \epsilon \sim \mathcal{N}(0, I) \tag{3.12}$$

and the learning objective for π_ϕ and its approximated gradient are:

$$J_\pi^{\mathrm{ME}}(\phi) = E_{s\sim\mathcal{B},\epsilon\sim\mathcal{N}}\left[\alpha \log \pi_\phi(f_\phi(\epsilon;s)|s) - Q(s, f_\phi(\epsilon;s))\right] \tag{3.13}$$

$$\hat{\nabla}_\phi J_\pi^{\mathrm{ME}}(\phi) = \nabla_\phi \log \pi_\phi(\hat{a}|s) + (\nabla_{\hat{a}} \log \pi_\phi(\hat{a}|s) - \nabla_{\hat{a}} Q(s,\hat{a}))\nabla_\phi f_\phi(\epsilon;s) \tag{3.14}$$

where $\hat{a}$ is evaluated at $f_\phi(\epsilon;s)$.

3.2 State-Action, Potential-Based Reward Shaping

Reward shaping, introduced by Ng et al. [43], refers to transforming a MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, with a sparse reward function to a new MDP $\widetilde{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \widetilde{\mathcal{R}}, \gamma)$ with a relatively denser and more informative reward function. The transformation is done as follows:

$$\widetilde{r}_t(s_t, a_t, s_{t+1}) = r(s_t, a_t, s_{t+1}) + \gamma\Phi(s_{t+1}) - \Phi(s_t) \tag{3.15}$$

The function Φ is a state-based *shaping potential*, and the *shaping reward* f is the discounted difference of potentials at time t and $t+1$:

$$f(s_t, s_{t+1}) = \gamma\Phi(s_{t+1}) - \Phi(s_t) \tag{3.16}$$

Under this transformation, the optimal action-value functions between the original MDP $\mathcal{M}$ and the transformed MDP $\widetilde{\mathcal{M}}$ satisfies the following equation:

$$Q^*(s,a) = \widetilde{Q}^*(s,a) + \Phi(s) \tag{3.17}$$

Since the difference between the original and transformed action-value functions do not depend on the actions at any state s, an optimal policy corresponding to $Q^*(s,a)$ satisfies

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a) = \operatorname*{argmax}_a \widetilde{Q}^*(s,a) = \widetilde{\pi}^*(s) \tag{3.18}$$

which implies that the optimal policy for $\widetilde{\mathcal{M}}$ will also be optimal for $\mathcal{M}$.

Wiewiora et al. [12] extended state based reward shaping to state-action based reward shaping, with the modified shaping reward being:

$$\widetilde{r}(s_t, a_t, s_{t+1}) = r(s_t, a_t, s_{t+1}) + f(s_t, a_t, s_{t+1}) = r(s_t, a_t, s_{t+1}) + \gamma\Phi(s_{t+1}, a_{t+1}) - \Phi(s_t, a_t) \tag{3.19}$$

, which gives rise to the following action-value function:

$$Q^*(s,a) = \widetilde{Q}^*(s,a) + \Phi(s,a) \tag{3.20}$$

In this case, since the shaping potential also depends on actions, the optimal policy corresponding to the transformed MDP $\widetilde{\mathcal{M}}$ is no longer guaranteed to be the optimal one

for the original MDP $\mathcal{M}$. However, based on Equation (3.20), we can still find the optimal policy for $\mathcal{M}$, which is defined to be

$$\pi^*(s) = \underset{a}{\text{argmax}} \left[\widetilde{Q}^*(s,a) + \Phi(s,a) \right] \tag{3.21}$$

3.3 Flow Based Generative Models - Normalizing Flows

Normalizing Flow is a popular class of generative model, which composes a sequence of invertible and differentiable mappings (i.e. bijections) to transform a simple probability distribution into a more complex distribution via the *change-of-variables formula* for probability distributions. Given a random variable $y \in \mathbb{R}^n$ with $z \sim p(z) = \mathcal{N}(0, I_n)$, and an differentiable, invertible mapping $f : \mathbb{R}^n \to \mathbb{R}^n$ with $y = f(z)$, the change of variables formula is defined as:

$$p(y) = p(z) \left| \det \left(\frac{\partial f^{-1}}{\partial z} \right) \right| = p(z) \left| \det \left(\frac{\partial f^{-1}}{\partial z} \right) \right|^{-1} \tag{3.22}$$

A normalizing flow is obtained by applying a series of such mappings $f_k, k \in 1, ..., K$, as first introduced by Rezende and Mohamed [14]:

$$z_K = f_K \circ f_{K-1} \circ ... \circ f_1(z_0), z_0 \sim p_0(z_0) \tag{3.23}$$

$$p_K(z_K) = p_0(z_0) \prod_{k=1}^{K} \left| \det \left(\frac{\partial f^{-1}}{\partial z} \right) \right|^{-1} \tag{3.24}$$

The major difference between each kind of normalizing flow is the way it parametrizes the bijective functions $f_i(z)$. In our case, we consider a particular type of parametrization proposed in Papamakarios et al. [69] named Masked Autoregressive Flow (MAF), which is derived from the following autoregressive model whose conditionals are parametrized as Gaussians:

$$p(z^i | z^{1:i-1}) = \mathcal{N}(z^i | \mu^i, (\exp \alpha^i)^2) \text{ where } \mu^i = f_{\mu^i}(z^{1:i-1}), \alpha^i = f_{\alpha^i}(z^{1:i-1}) \tag{3.25}$$

f_{μ^i} and f_{α^i} are scalar functions represented by a single neural network with masked entries, $f_{\psi^i} = (f_{\mu^i}, f_{\alpha^i})$, where ψ represents the parameters of the neural network. The neural network outputs both mean and log standard deviation of the i^{th} conditional given $z^{1:i-1}$ where $i \leq n$. With this model, the bijective function is defined by the following recursion:

$$z_k^i = \mu^i + z_{k-1}^i \exp \alpha^i \tag{3.26}$$

The primary reason that we consider this bijective function is because its autoregressive structure makes the Jacobian of f^{-1} a triangular matrix so that the absolute determinant

can be easily computed as follows:

$$\left|\det\left(\frac{\partial f_{\psi}^{-1}}{\partial z}\right)\right| = \exp\left(-\sum_{i}\alpha^{i}\right) \tag{3.27}$$

Due to this advantage, the log likelihood of any observed data can be computed efficiently based on Equation (3.24). Moreover, the training criterion of flow-based generative models is simply the negative log-likelihood (NLL) over the training dataset $\mathcal{D}$.

$$\mathcal{L}^{1}_{f,\mathcal{D}}(\psi) = -\sum_{z_K\in\mathcal{D}}\sum_{k=1}^{K}\log\left|\det\left(\frac{\partial f_{\psi_k}^{-1}}{\partial z_{k-1}}\right)\right| \tag{3.28}$$

Because of this criterion, an optimized normalizing flow will have high density on data that is close to the training distribution, and low density on data away from that distribution.

In practice, a regularization term is sometimes added to control the smoothness of the transformed distribution. In our case, we choose to use the norm of the gradient of the learned log density with respect to input z_K:

$$\mathcal{L}^{2}_{f,\mathcal{D}}(\psi) = \sum_{z_K\in\mathcal{D}}||\nabla_{z_K}\log p_K(z_K)||^2 \tag{3.29}$$

The final cost function is then

$$\mathcal{L}_{f,\mathcal{D}}(\psi) = \mathcal{L}^{1}_{f,\mathcal{D}}(\psi) + \eta^{\mathrm{NF}}\ \mathcal{L}^{2}_{f,\mathcal{D}}(\psi) \tag{3.30}$$

where η^{NF} is a hyper-parameter controlling the weight of the regularization term.

3.4 Generative Adversarial Networks

Another popular class of generative model is the family of *Generative Adversarial Networks (GANs)* [13]. A GAN composes two deep neural networks, the generator G_ϕ, parametrized by ϕ, and the discriminator D_ψ, parametrized by ψ. The discriminator D is learned to differentiate samples generated by G and from the true distribution. The generator G is trained to fool the discriminator. Let p_t be the true distribution of a random variable x and p_g be the generated distribution from G, the training of GAN is a minimax optimization process with the goal being to minimize the distance between p_t and p_g.

There are multiple methods of measuring the distance between two distributions, and each method leads to a different minimax objective. In our case, we consider *Wasserstein GAN (WGAN)* proposed by Arjovsky et al. [70], which uses the *Earth Mover's distance* (or *Wasserstein-1 distance*) to compare p_t and p_g. The minimax objective of WGAN is

constructed based on the *Kantorovich-Rubinstein duality*:

$$\mathcal{L}^1(\phi, \psi) = \min_{G_\phi} \max_{D_\psi \in \mathcal{F}} \mathbb{E}_{x \sim p_t}[D_\psi(x)] - \mathbb{E}_{\widetilde{x} \sim p_g}[D_\psi(\widetilde{x})] \tag{3.31}$$

where $\mathcal{F}$ is the set of 1-Lipschitz functions and p_g is the distribution defined by $\widetilde{x} = G(z)$ with $z \sim \mathcal{N}(0, I)$. To enforce the 1-Lipschitz constraint on D, we use the method in Gulrajani et al [71], which imposes a soft version of the constraint with a penalty term on D's gradient norm

$$\max_{D_\psi} \mathbb{E}_{\hat{x} \sim p_{\hat{x}}} \left[(||\nabla_{\widehat{x}} D_\psi(\widehat{x})|| - gp^{\text{GAN}})^2 \right] \tag{3.32}$$

where $\widehat{x} = \epsilon x + (1 - \epsilon)\widetilde{x}$ with $\epsilon \sim U[0, 1], x \sim p_t, \widetilde{x} \sim p_g$, and $gp^{\text{GAN}} \in [0, 1]$ is the target norm of the gradient. Combining this soft constraint with the minimax objective, the resulting learning objective is:

$$\min_{G_\phi} \max_{D_\psi} \mathbb{E}_{x \sim p_t}[D_\psi(x)] - \mathbb{E}_{\widetilde{x} \sim p_g}[D_\psi(\widetilde{x})] - \eta^{\text{GAN}} \left[(||\nabla_{\widehat{x}} D_\psi(\widehat{x})|| - gp^{\text{GAN}})^2 \right] \tag{3.33}$$

where η^{GAN} is a hyper-parameter controlling the weight of the soft constraint.

It is important to note that Equation (3.33) also trains the discriminator to output higher values on samples from the true distribution than on samples from the generated distribution, which inspired us to develop our GAN based shaping potential discussed in 4.2. The main reason that we choose Wasserstein GAN over other type of GANs is because the gradient penalty term in Equation (3.32) can be used to control the smoothness of the shaping potential as it regularizes the gradient of the discriminator output with respect to its input.

Chapter 4

RL from Demonstration via Shaping through Generative Models

In this chapter, we describe how we combine RL and IL via reward shaping through generative models. First, we present the two forms of shaping potentials, one based on normalizing flows and the other based on GANs. Then, we derive the new learning objectives of two popular off-policy actor-critic algorithms: SAC and TD3 using reward shaping. However, the shaping potentials considered in our case are agnostic to the RL algorithm used. We choose these two algorithms because they are the state-of-the-art at the time of writing, and they are representative off-policy actor-critic algorithms: TD3 learns a deterministic policy, while SAC learns a stochastic policy. Finally, we list other changes to SAC and TD3 that we make to facilitate learning from demonstrations and sparse rewards.

4.1 Assumptions and Preliminaries

In this work, we consider systems with unknown dynamics and a sparse reward function that only indicates task completion. We use the infinite-horizon MDP formulation described by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, as outlined in 3.1. Our goal also remains the same: finding a policy π that maximizes the expected return $J(\pi)$ in Equation (3.1) for ordinary RL, or the expected return and entropy of the policy $J^{\text{ME}}(\pi)$ in Equation (3.8) for Maximum Entropy RL.

We assume that the tasks are iterative in nature with a fixed, known goal set $\mathcal{G} \subset \mathcal{S}$. This is common in many manipulation tasks, such as object insertion, placement and assembly. The reward function that specifies task completion is defined to be: $r(s, a) = -\mathbb{1}_{\mathcal{G}^C}(s)$, where $\mathcal{G}^C$ is the complement of $\mathcal{G}$ under $\mathcal{S}$. We define task success as convergence to $\mathcal{G}$ within a pre-defined time limit T measured in terms of simulation steps. T is sometimes

called *episode length* or *task horizon*. Note that given this definition of the reward function, the problem we consider is equivalent to the minimum-time control problem in optimal control because the agent gets higher return by converging to the goal set faster. Though our method is also applicable to other choices of sparse or dense reward functions, we focus on this particular type of reward because it is common for robotic tasks and best demonstrates the effectiveness of our method.

We further assume that a set of optimal or sub-optimal demonstrations $\mathcal{D} = \{(s_i, a_i), i = 1...N\}$ is given beforehand. The demonstrations can be provided, for example, from human teleoperation or a hard-coded policy. Note that our choice of demonstrations are different from some other works on RL from Demonstration, such as [10], which assume the demonstrations are in the form (s, a, r, s'). Having access to the reward signal in demonstrations means that one can learn both the policy and the action-value function using demonstrations. However, we argue that getting access to the reward is only possible when the demonstrator is executing the same task as the one to be learned by the agent, so that it does not allow easy extension to multi-task learning or different state spaces between the demonstrator and the agent. Therefore, we stick with the form (s, a) for our demonstrations to preserve extensibility and do not train the action-value function using data in $\mathcal{D}$.

4.2 Shaping Potential Based on Generative Models

The main idea of state-action based reward shaping is to construct a informative shaping potential that gives higher values for desired state-action pairs, which is assumed to be in the demonstrations. We notice that both the density estimation from a normalizing flow and the output from the discriminator of a GAN trained on the demonstrations $\mathcal{D}$ match this criteria.

4.2.1 Shaping Potential Based on Normalizing Flows

Given a normalizing flow f_{ψ^*} that has been optimized on demonstrations $\mathcal{D}$ with $z_K = (s, a) \in \mathcal{D}$, we define the shaping potential based on the density estimation of f_{ψ^*} on (s, a) as follows:

$$\Phi^{\text{NF}}_{\psi^*, k^{\text{NF}}, b^{\text{NF}}}(s, a) = k^{\text{NF}} \log \left(p_{\psi^*}(s, a) + b^{\text{NF}}\right) \tag{4.1}$$

where $k^{\text{NF}} \in \mathbb{R}^+$ is a hyper-parameter that controls the scale of the shaping potential, and b^{NF} is a small constant that prevents the log density from reaching negative infinity.

To train the normalizing flow on demonstration $\mathcal{D}$, we use mini-batch gradient descent with Adam optimizer [72]. The hyper-parameters used for training the normalizing flow are listed in Appendix A.

4.2.2 Shaping Potential Based on Generative Adversarial Networks

Given a GAN that has been trained on demonstrations $\mathcal{D}$ with $x = (s, a) \in \mathcal{D}$, and its discriminator D_{ψ^*}, we define the shaping potential based on the output of D_{ψ^*} as follows:

$$\Phi^{\text{GAN}}_{\psi^*, k^{\text{GAN}}, b^{\text{GAN}}}(s, a) = k^{\text{GAN}}(D_{\psi^*}(s, a) - b^{\text{GAN}}) \tag{4.2}$$

where $k^{\text{GAN}} \in \mathbb{R}^+$ is again the hyper-parameter that controls the scale of the shaping potential, and b^{GAN} is a constant shift that is set to the mean output of D_{ψ^*} on $\mathcal{D}$.

In terms of training the GAN, we follow the same process proposed in [70] where both G_ϕ and D_ψ are learned with Adam optimizer [72] while G_ϕ is updated after every $n^{\text{GAN}}_{\text{critic}}$ times of D_ψ update. Pseudo-code of the training process is shown in Algorithm 1, and the hyper-parameters for GAN training and GAN based shaping potentials are listed in Appendix A

Algorithm 1 Learn a Shaping Potential Based on WGAN-GP [70]

Require: The gradient penalty coefficient η^{GAN}, the number of discriminator iterations per generator iteration $n^{\text{GAN}}_{\text{critic}}$, the batch size m, Adam hyper-parameters α^{GAN}, β^{GAN}_1, β^{GAN}_2

Require: Initial discriminator parameters ψ_0, initial generator parameters ϕ_0.

while ϕ has not converged **do**
  for $t = 1, ..., n^{\text{GAN}}_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample demonstration data $x = (s, a) \sim \mathcal{D}$, latent variable $z \sim p(z)$, $\epsilon \sim U[0, 1]$.
      $\tilde{x} \leftarrow G_\phi(z)$
      $\hat{x} \leftarrow \epsilon x + (1 - \epsilon)\tilde{x}$
      $L^{(i)} \leftarrow D_\psi(\tilde{x}) - D_\psi(x) + \eta^{\text{GAN}}(||\nabla_{\hat{x}} D_\psi(\hat{x})||_2 - gp^{\text{GAN}})^2$
    $\psi \leftarrow \text{Adam}(\nabla_\psi \frac{1}{m} \sum_{i=1}^m L^{(i)}, \psi, \alpha^{\text{GAN}}, \beta^{\text{GAN}}_1, \beta^{\text{GAN}}_2)$
  Sample a batch of latent variables $\{z^{(i)}{}_{i=1}^m\} \sim p(z)$.
  $\phi \leftarrow \text{Adam}(\nabla_\phi \frac{1}{m} \sum_{i=1}^m -D_\psi(G_\phi(z)), \phi, \alpha^{\text{GAN}}, \beta^{\text{GAN}}_1, \beta^{\text{GAN}}_2)$

4.3 Updated Learning Objectives for SAC and TD3

We follow the state-action based reward shaping method in [12], and incorporate Equation (3.20) into the Bellman equation (Equation (3.3)). The transformed Bellman equation is then

$$\widetilde{Q}^\pi(s_t, a_t) + \Phi(s_t, a_t) = r + \gamma E_{s_{t+1}, a_{t+1}}[\widetilde{Q}^\pi(s_{t+1}, a_{t+1}) + \Phi(s_{t+1}, a_{t+1})], a_{t+1} \sim \pi(s_{t+1}) \tag{4.3}$$

$$\widetilde{Q}^\pi(s_t, a_t) = r - \Phi(s_t, a_t) + \gamma E_{s_{t+1}, a_{t+1}}[\widetilde{Q}^\pi(s_{t+1}, a_{t+1}) + \Phi(s_{t+1}, a_{t+1})] \tag{4.4}$$

For TD3, the transformed target of the action-value function $\widetilde{Q}$ and its corresponding

loss are:

$$\widetilde{y} = r_t - \Phi(s_t, a_t) + \gamma \left(\min_{i=1,2} Q_{\theta_i'}(s_{t+1}, \pi_{\phi'}(s_{t+1}) + \epsilon^{\text{TD3}}) + \Phi(s_{t+1}, \pi_{\phi'}(s_{t+1}) + \epsilon^{\text{TD3}}\right) \quad (4.5)$$

$$J_{\widetilde{Q}}(\theta) = E_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{B}}\left[(\widetilde{Q}_\theta(s_t, a_t) - \widetilde{y})^2\right] \quad (4.6)$$

where $\epsilon^{\text{TD3}} \sim \text{clip}(\mathcal{N}(0, \sigma^{\text{TD3}}), -c^{\text{TD3}}, c^{\text{TD3}})$. The policy loss is:

$$J_\pi(\phi) = \mathbb{E}_{s\sim p_\pi}\left[-\widetilde{Q}(s, a) - \Phi(s, a)|_{a=\pi(s)}\right] \quad (4.7)$$

Similarly, for SAC, the transformed target and loss for $\widetilde{Q}$ are:

$$\widetilde{y}^{\text{ME}} = r_t - \Phi(s_t, a_t) \quad (4.8)$$

$$+\gamma\, \mathbb{E}_{a_{t+1}\sim\pi_\theta(s_{t+1})}\left[\min_{i=1,2} \widetilde{Q}_{\theta_i'}(s_{t+1}, a_{t+1}) + \Phi(s_{t+1}, a_{t+1}) + \alpha \log\pi(a_{t+1}|s_{t+1})\right] \quad (4.9)$$

$$J^{\text{ME}}_{\widetilde{Q}}(\theta) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{B}}\left[(\widetilde{Q}(s_t, a_t) - \widetilde{y})^2\right] \quad (4.10)$$

And the new loss for policy is:

$$J^{\text{ME}}_\pi(\phi) = E_{s\sim\mathcal{B},\epsilon\sim\mathcal{N}}\left[\alpha \log \pi_\phi(f_\phi(\epsilon; s)|s) - \widetilde{Q}(s, f_\phi(\epsilon; s)) - \Phi(s, f_\phi(\epsilon; s))\right] \quad (4.11)$$

4.4 Other Changes to TD3 and SAC that Facilitate Learning from Demonstrations and Sparse Rewards

In 4.2 and 4.3, we present the shaping potentials based on two types of generative models and the updated learning objectives of TD3 and SAC with reward shaping. Since reward shaping does not alter other part of the RL training process, one can use exactly the same learning procedure outlined in the original TD3 and SAC paper [15][16] with the updated learning objectives to incorporate the shaping potentials learned from demonstrations. However, for complex tasks with high dimensional state-action space and/or long task horizon, we suggest to use the following tricks to facilitate learning from demonstrations and sparse rewards. Note that the following modifications 1) do not make any assumptions additional to those discussed in 4.1; 2) do not bias the learned policy towards demonstrated behaviors.

1) Mixing 1-step and n-step return

   Using a mix of 1-step and n-step return when updating the critic function helps propagate the Q-values along the trajectories of n steps. The idea is to minimize the difference between the action-value at (s_t, a_t) and the return of the rollout $(s_{t+i}, a_{t+i}, r_{t+i}, s_{t+i+1})_{i=0}^{n-1}$ with $s_{t+i+1} = P(\cdot|s_{t+i}, a_{t+i})$ following a policy π close to the current policy $\pi_\phi(s)$. The Bellman objective for n-step return $\tilde{y}^n$ can be easily derived from the Bellman equation

and is similar to $\tilde{y}$ in Equation (4.5):

$$\widetilde{y}^n = \sum_{i=0}^{n-1} r_{t+i} - \Phi(s_t, a_t) + \gamma^n \left(\min_{i=1,2} Q_{\theta_i'}(s_{t+n}, \pi_{\phi'}(s_{t+n}) + \epsilon^{\text{TD3}}) + \Phi(s_{t+n}, \pi_{\phi'}(s_{t+n}) + \epsilon^{\text{TD3}})\right) \tag{4.12}$$

2) Using Demonstrations as an Augmented Dataset
In addition to the RL rollout experiences, we can use the state-action pairs from the demonstration set $\mathcal{D}$ to train the policy in TD3 and SAC. Note that, unlike [8], we do not load demonstrations into the RL replay buffer $\mathcal{B}$ but use $\mathcal{D}$ as a separate replay buffer. When updating the policy π, we sample a fixed number of (s, a) pairs from both $\mathcal{B}$ and $\mathcal{D}$, which ensures that we train π on the demonstrations throughout the entire RL training process.

3) Initializing the policy via BC
Before training the policy π using TD3 or SAC with the updated learning objectives, we can first initialize the policy using BC. For deterministic policy, such as the policy learned via TD3, we use the MSE criterion:

$$J_\pi^{\text{BC}}(\phi) = \sum_{(s,a)\in\mathcal{D}} (\pi_\phi(s) - a)^2 \tag{4.13}$$

For stochastic policy, such as the policy learned via SAC, we use the negative log likelihood loss as our training criterion:

$$J_\pi^{\text{BC}}(\phi) = \sum_{(s,a)\in\mathcal{D}} -\log p(\pi_\phi(s) = a) \tag{4.14}$$

4) Ensemble of GAN/Normalizing Flow Based Shaping Potential
Instead of using the shaping potential that comes from a single GAN or normalizing flow, we consider using an ensemble of shaping potentials based on K models of the same type. All models learn from the same demonstrations but are initialized differently. The resulting shaping potential is the average output from all K models: $\phi_{\psi^*,k,b} = \frac{1}{K}\sum_{i=1}^{K} \phi_{\psi_i^*,k,b}$

5) Scheduled Decay on the Scale of the Shaping Potential
For experiments with long-horizon tasks, we consider replacing the scaling parameter $k^{\text{NF/GAN}}$ with a decreasing sequence $k_t^{\text{NF/GAN}}$ such that: $\lim_{t\to\infty} k_t^{\text{NF/GAN}} = 0$. Using this decreasing sequence means that we gradually switch from optimizing the action-value function Q_θ for the transformed MDP $\widetilde{\mathcal{M}}$ to the original MDP $\mathcal{M}$. We found that this resulted in policy with higher performance despite that the learned policy via RL should not be affected by the shaping potential in theory. We compare experimental

results with and without the scheduled decay in 5.7 and provide more detailed discussions there.

4.5 Algorithm Summary and Pseudo-Code

To summarize, our algorithm combines RL and IL through state-action based reward shaping. We proposed two forms of shaping potentials based on two types of deep generative models - GAN and normalizing flow, and the shaping potentials are learned from a modest set of demonstrations. We also derived the new learning objectives for TD3 and SAC incorporating the shaping potentials. Moreover, we made a few other changes to the standard TD3/SAC training flow to help learn from demonstrations and sparse rewards, including a mix of 1-step and n-step return, BC pre-training, sampling from the demonstrations, etc. The pseudo-code of our algorithm is shown in Algorithm 2, and the hyper-parameters used in our experiments are listed in Appendix A.

Algorithm 2 TD3/SAC from Demonstration via Reward Shaping

Require: Optimal/sub-optimal demonstrations $\mathcal{D} = \{(s_i, a_i), i = 1...N\}$

Offline Pre-training

Train K shaping potentials based on normalizing flows or GANs: $\Phi^{\text{GAN/NF}}_{\psi_i^*,k,b}(s,a), i = 1,...,K$, from Eqn. 4.1 or 4.2.

Use the average of the shaping potentials: $\phi^{\text{GAN/NF}}_{\psi^*,k,b} = \frac{1}{K}\sum_{i=1}^{K}\phi_{\psi_i^*,k,b}$

Given MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$, Consider MDP $\widetilde{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \widetilde{r}, \gamma)$ from Eqn. 3.19 with

$\widetilde{r}_t = r(s_t, a_t, s_{t+1}) + \gamma\Phi_{\psi^*,k,b}(s_{t+1}, a_{t+1}) - \Phi_{\psi^*,k,b}(s_t, a_t)$

Off-Policy Actor-Critic Training with Shaping and BC Initialization

Initialize actor and critic networks for $\widetilde{M} : \widetilde{Q}_{\theta_1}, \widetilde{Q}_{\theta_2}, \pi_\phi$

Pre-train π_ϕ on $\mathcal{D}$ using BC with MSE criterion (Eqn. 4.13) for TD3, or NLL criterion (Eqn. 4.14) for SAC.

Initialize target networks $\theta_1' \leftarrow \theta_1, \theta_2' \leftarrow \theta_2, \phi' \leftarrow \phi$

Initialize 1-step replay buffer $\mathcal{B}$ and n-step replay buffer $\mathcal{B}^n$ to empty.

while not converged **do**

  for episode $e = 1...E$ **do**

    for step $t = 1...T$ **do**

      Sample action a from $\pi_\phi(s)$, observe reward r and new state s' from $\mathcal{M}$.

      Store transition tuple (s, a, r, s') in both $\mathcal{B}$ and $\mathcal{B}^n$.

  for batch $b = 1...B$ **do**

    Sample mini-batch $\mathcal{B}_b$ and $\mathcal{B}_b^n$ of (s, a, r, s') from $\mathcal{B}$ and $\mathcal{B}^n$, respectively.

    Sample mini-batch $\mathcal{D}_b$ of (s_d, a_d) from $\mathcal{D}$

    Update critics from Eqn. 4.6 for TD3 or Eqn. 4.10 for SAC using $\mathcal{B}_b$ and $\mathcal{B}_b^n$

    if b mod d **then**

      Update policy from Eqn. 4.7 for TD3 or 4.11 for SAC using $\mathcal{B}_b$, $\mathcal{B}_b^n$ and $\mathcal{D}_b$

  Update target networks:

  $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$

  $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$

  Clear n-step replay buffer $\mathcal{B}^n$

Chapter 5

Experiments

In this section, we present experimental results and discussions of comparing the TD3 version of our method, named TD3+Shaping, against several RL and IL methods on simulated robotic control and manipulation tasks. Our evaluation metrics are sample efficiency, episode return and hyper-parameter sensitivity. In particular, we answer the following questions:

1. What does the learned shaping potential look like? How do we know if it will be helpful or not?

2. Will our method bias the learned policy when sub-optimal demonstrations are given? Does our method get higher return than pure BC methods or RL+IL methods with a hybrid objective?

3. Given either optimal or sub-optimal demonstrations, does our method outperform better than pure RL methods in terms of sample efficiency?

4. Is our method sensitive to any hyper-parameter, which makes it difficult to tune?

This section is organized as follows: In 5.1, we list the baseline RL and IL algorithms to be compared against in our experiments. 5.2 and 5.3 describe the tasks and how the optimal and sub-optimal demonstrations are generated. Then, in 5.4, we consider a toy 2D *reaching* task and show how to visualize and evaluate the learned shaping potentials, which answers the first question above. After that, in 5.5, we visualize the sub-optimal demonstrations of a 2D *peg insertion* task and compare the resulting trajectory generated by the learned policies from our method and RL+IL methods that assume optimal demonstrations. In 5.6 and 5.7, we compare our methods against the baseline algorithms on several manipulation tasks with varying difficulties. Finally, we conclude our findings and answers to the above questions in 5.8.

5.1 Baselines

We use the following RL, IL and RL+IL algorithms as our baselines:

1) Pure RL method: Twin Delayed DDPG (TD3) [15]
   Note that this is also the underlying RL algorithm of our method.

2) Pure IL method: Behavioral Cloning (BC)
   This method uses supervised learning on demonstrations. We will consider learning a deterministic policy using the MSE criterion in Equation (4.13).

3) RL+IL method via initialization: TD3+BC Initialization
   The policy is initialized via BC on demonstrations and fine-tuned using TD3.

4) RL+IL method via a hybrid objective: TD3+BC
   Like TD3+BC Initialization, the policy is first initialized via BC on demonstrations. When training the policy using TD3, we keep the BC objective $J^{\text{BC}}(\pi)$ as a soft regularization term and weigh the TD3 objective in Equation (3.4) using a hyper-parameter λ:
   $$J_\pi^{\text{TD3+BC}}(\phi) = \lambda\, J_\pi^{\text{TD3}}(\phi) + J_\pi^{\text{BC}}(\phi) \tag{5.1}$$
   Since the regularization term biases the policy towards demonstrated actions, we use multiple value of λ and see its impact on sample efficiency and performance of the learned policy. In particular, we choose λ to be 1e-1, 1e-2, 1e-3 and 1e-4.

5) RL+IL method via hybrid objective and filtering: TD3+BC+QFilter
   This method uses the same learning objective as TD3+BC, but regularization is only applied to states where the estimated Q-value of the demonstrated action is higher than the estimated Q-value of the policy output. Q-filter is introduced by Nair et al. in [10].

Note that we only use TD3, BC with MSE criterion and some combinations of the two as our baseline algorithms. We choose these algorithms because: 1) we use the TD3 version of our method in all experiments; 2) both TD3 and BC with MSE criterion are simple to implement and are part of our method; 3) both algorithms work with deterministic policy so that they can be combined directly without change to the representation of the policy. However, as we have discussed in section 4.3, one can easily learn a stochastic policy using the SAC version of our method. Furthermore, there are many other RL and IL baseline algorithms not considered in this work, such as IRL methods [42][59] and adversarial IL methods [4]. Learning stochastic policies using SAC+Shaping and comparing our method to other RL and IL methods are left for future works.

5.2 Specification and Categorization of Tasks

The tasks used in our experiments are listed in Table 5.1. Except the Reach-2D task, all tasks are constructed based on the OpenAI Gym [18] and MetaWorld (MW) [17] API for Robotics environments. We categorize the tasks according to its horizon, which refers to the number of environment steps between each reset and is chosen to be about 3 times greater than the minimum number of steps taken to reach the goal state. We categorize the tasks this way because all of our tasks are state-based, and we notice that task horizon is the determining factor of the performance of our method in terms of both episode return and hyper-parameter sensitivity. However, one should note that the difficulty of a task is not solely dependent on the task horizon (or the minimum number of steps taken to reach the goal). There are many other factors affecting the difficulty of a task, such as the dimension of state and action spaces and whether or not objects & obstacles are involved.

For reward function design, as we have mentioned in 4.1, we always use sparse rewards that only indicate task completion: $r(s, a) = -\mathbb{1}_{\mathcal{G}^C}(s)$, which yields a control policy that minimizes time taken to reach the goal set $\mathcal{G}$. We refer the reader to the original papers of OpenAI Gym [18] and MetaWorld [17] for the precise definition of the goal set $\mathcal{G}$ of each task. Generally speaking, the goal set is the collection of states close to the task's goal state, and the goal state can be easily inferred from the task name. For example, for a *peg insertion* task, $\mathcal{G}$ contains states where the peg is inserted in the hole with a small tolerance. We consider a task as successfully completed if the agent reaches and stays in $\mathcal{G}$ until the end of the episode.

Task Name	$\dim(\mathcal{S})$	$\dim(\mathcal{A})$	Task Horizon
Toy Tasks			
Reach-2D	2	2	20
Gym-PegInsertion-2D	6	4	40
Short-Horizon Tasks			
Gym-PegInsertion	6	4	40
Gym-PickAndPlace	25	4	40
Long-Horizon Tasks			
MW-PressButton	9	4	150
MW-DrawerClose	9	4	150
MW-Hammer	9	4	150

Table 5.1: Specification of tasks in our experiments including state space $\mathcal{S}$, action space $\mathcal{A}$, and task horizon. We divide the tasks into three categories. Toy tasks are used for illustrative purposes. Short-Horizon and Long-Horizon tasks are used for comparing our method with the baseline algorithms in 5.1. See Appendix B for the visualization of the simulated manipulation tasks.

5.3 Generating Demonstrations

All algorithms that involve IL are given the same demonstrations, which can be optimal, near-optimal or sub-optimal. The demonstrations normally consist of multiple trajectories that successfully complete the task. As discussed above, given the task completion reward, the optimal policy should spend minimum time reaching the goal set $\mathcal{G}$. In practice, it is very difficult to have such a policy that provides optimal demonstrations using a hard-coded script or human teleoperation. Therefore, we first generate some sub-optimal demonstrations using a hard-coded policy. Then, we run TD3+BC and use the learned policy to generate better demonstrations iteratively until the performance of the policy stops improving.

For sub-optimal demonstrations, we mainly consider the case where the demonstrated trajectories do not achieve the RL goal of the task, i.e. minimize time taken to reach $\mathcal{G}$, but still successfully complete the task. We generate sub-optimal demonstrations through hard-coded policies for all tasks. In particular, we show in 5.5 what the sub-optimal demonstrations may look like and how it affects the learned policies using our method and other RL+IL methods.

Moreover, it is worth mentioning that the demonstrations can be sub-optimal for other reasons, such as noise in measurement or actuation. One can approximate noise in actuation by adding Gaussian noise to the output of an optimal policy when generating the demonstrative trajectories. Similarly, one can add Gaussian noise to the policy's inputs to simulate measurement noise. We did not focus on sub-optimal demonstrations due to noise in our experiments because we found that they had much less impact on the optimality of the learned policy using any RL+IL method, unless when too much noise was added in which case none of the algorithms converged.

5.4 Illustrative Example 1: Visualizing and Evaluating Shaping Potentials

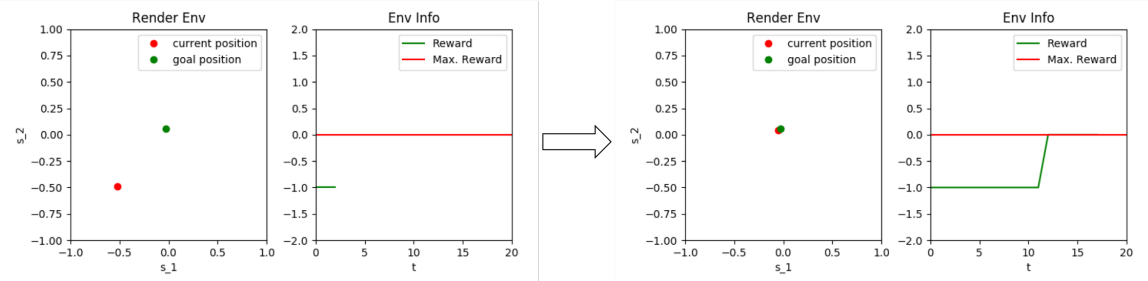

Figure 5.1: The *Reach-2D* task used for visualizing the shaping potentials based on GANs and normalizing flows. The plot on the left shows the task state at $t = 2$ where the point mass (in red) has not reached the goal position (in green) and thereby gets a reward -1. The plot on the right shows the environment state at $t = 13$ where the point mass reaches the goal and gets a reward 0.

To evaluate our GAN and normalizing flow (NF) based shaping potentials, we created a toy *reaching* task in a 2D environment, named *Reach-2D*, as shown in Figure 5.1. The goal of the task is to control a point mass (red) to reach a target position (green), which is fixed at $g = (0, 0)$ in this experiment. The agent observes the current position of the point mass $s = (s_1, s_2)$ and specifies the target velocity $v = (v_1, v_2)$ for the mass. The goal set $\mathcal{G}$ is the collection of states centered at the target position $(0, 0)$ with a tolerance of 0.05: $||s - g|| < 0.05$.

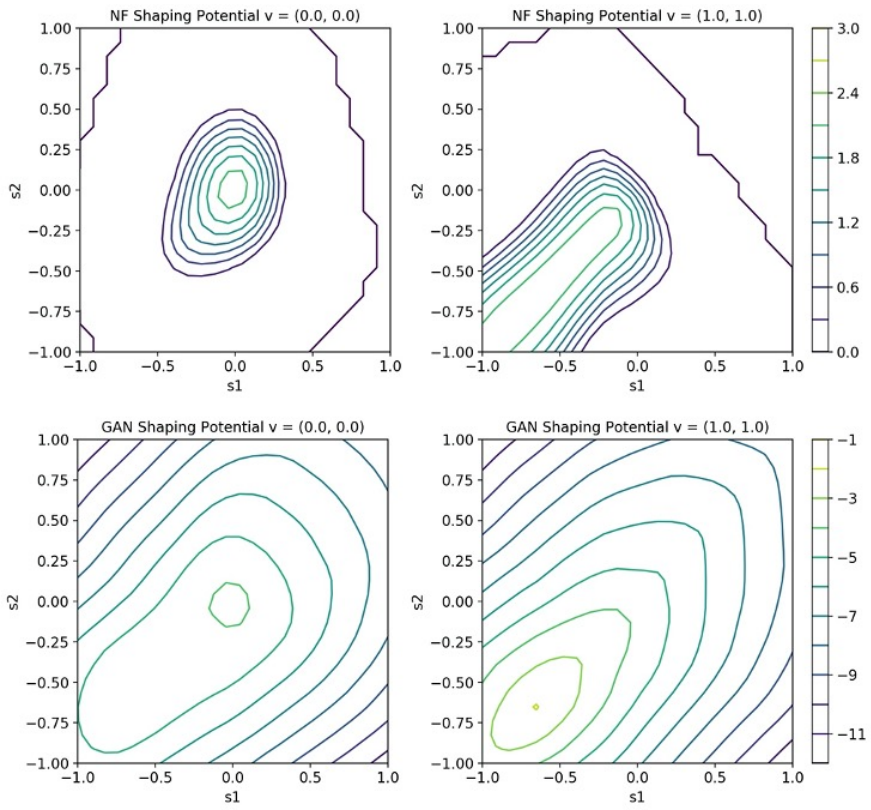

Figure 5.2: The learned shaping potentials based GAN and normalizing flow from optimal demonstrations for Reach-2D. The initial position of the point mass is always $(-0.8, -0.8)$ and the goal position is kept at $(0, 0)$. As expected, the shaping potentials model the demonstration data distribution and thereby encourage the agent to reach and stay at the goal state.

This 2D environment allows us to visualize the learned shaping potentials directly over the entire state space $\mathcal{S}$ given an action $a \in \mathcal{A}$. In Figure 5.2, we plot the learned shaping potentials when the point mass is initialized at $s_0 = (-0.8, -0.8)$ and the demonstrations are optimal. We can see that when setting $v = (1.0, 1.0)$, which is the maximum velocity towards the target position from $(\hat{s}, \hat{s}), \hat{s} < 0$, both NF and GAN based shaping potentials assign higher values to states along the straight line between $s_0 = (-0.8, -0.8)$ and $g = (0, 0)$. When letting $v = (0.0, 0.0)$, both shaping potentials put higher values near $g = (0, 0)$. This result indicates that the potentials encourage the agent to move towards the target position and stop after reaching the target, which is consistent with the demonstrated behavior. We know that at early stage of RL training, the action-value function $\tilde{Q}$, i.e. the critic in TD3/SAC, is normally close to zero everywhere and the potential Φ outweighs $\tilde{Q}$. So, according to Equation (3.21), the policy is trained to maximize the output of the shaping potential and thereby gets biased towards the target $g = (0, 0)$.

For environments with high dimensional state and action spaces, it is not possible to

visualize the shaping potential directly as in Figure 5.2, so we consider an alternative method to evaluate the shaping potential. Given a shaping potential Φ trained on demonstration dataset $\mathcal{D}$, we plot the average output of the shaping potential given the (s, a) pairs in $\mathcal{D}$ perturbed by Gaussian noise ϵ with zero mean and σ^2 variance: $\{(s, a) + \epsilon \mid (s, a) \in \mathcal{D}, \epsilon \sim \mathcal{N}(0, \sigma^2)\}$. Figure 5.3 shows the relationship between ϕ and σ^2 of the shaping potentials that we have visualized directly in Figure 5.2. We notice that Figure 5.2 and Figure 5.3 provide very similar information: the learned shaping potentials output higher values given (s, a) pairs close to those in $\mathcal{D}$. Moreover, Figure 5.3 shows that both shaping potentials are sensitive to small perturbations to (s, a) in $\mathcal{D}$, i.e when σ^2 is small. See *Decreasing Region* of the two plots in Figure 5.3. However, the shaping potentials become flat when σ^2 gets large, in which case the inputs are far from the demonstrated (s, a) pairs. See *Flat Region* of both plots in Figure 5.3.

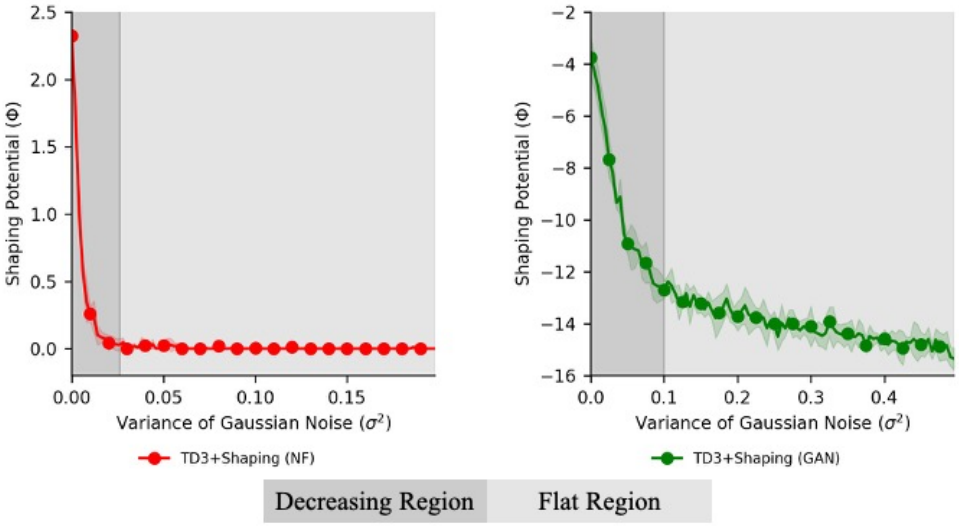

Figure 5.3: Plot of the average output from GAN (right) and normalizing flow (left) based shaping potentials verses (s, a) pairs from in demonstration dataset D perturbed by Gaussian noise $\mathcal{N}(0, \sigma^2)$. The shaping potentials drop quickly inside the *Decreasing Region* and are almost flat outside this region.

In practice, we found that the width of *Decreasing Region* strongly affects the performance of our method. The reason is that, when *Decreasing Region* is too narrow, the shaping potential becomes sparse as it gives different outputs only at states close to the demonstrations; if *Decreasing Region* is too wide, the shaping potential becomes too smooth and the agent may gets distracted by the noise in the action-value function $\widetilde{Q}$. Another important factor that affects the effectiveness of a shaping potential is its scale, which determines the relative weight between the shaping reward and the true reward from the environment. The hyper-parameters controlling the width of *Decreasing Region* and scale of the shaping potentials are the regularizers $\eta^{\text{NF/GAN}}$ in Equation (3.30)/(3.33) and the

scaling factors $k^{\text{NF/GAN}}$ in Equation (4.1)/(4.2). In Figure 5.4, we show an example of tuning the hyper-parameters k^{NF} and η^{NF} of a normalizing flow based shaping potential trained on optimal demonstrations for *Reach-2D*. For simple tasks, such as the toy tasks and short-horizon tasks listed in Table 5.1, we can choose the hyper-parameters $k^{\text{NF/GAN}}$ and $\eta^{\text{NF/GAN}}$ according to the $\Phi - \sigma^2$ plot. The learned shaping potentials are helpful and never bias the policy. However, we will discuss in 5.7 that the performance of policy may become highly sensitive to the scale and shape of the shaping potential for long-horizon tasks, in which case we cannot determine these hyper-parameters solely from the $\Phi - \sigma^2$ plot.

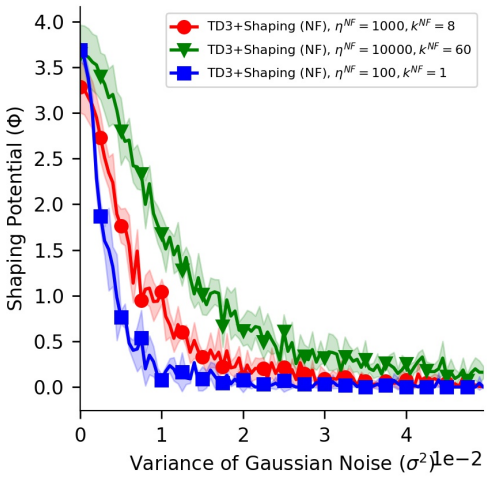

Figure 5.4: $\Phi - \sigma^2$ plot of normalizing flow based shaping potentials trained with 3 different choices of the scaling factor k^{NF} and weight on the regularization term η^{NF}. The empirical mean is computed from 4 seeds and the error bars represent 1σ standard deviation. The learned shaping potentials are very consistent across seeds, and the width of the *Decreasing Region* can be controlled by the two hyper-parameters.

Finally, we present the results of running baseline algorithms and our method on the *Reach-2D* task in Figure 5.5. The demonstrated trajectory is optimal and reaches the goal state in 7 steps. Since the task is simple and optimal demonstrations are given, we can see that an optimal policy can be learned from just BC without any help from RL. Both TD3+BC and TD3+BC+QFilter converged quickly within 20000 steps of exploration. Our methods, TD3+Shaping (GAN/NF), converged slightly slower but still found the optimal policy within 40000 steps. However, TD3+BC Initialization only converged on 3/4 seeds, which is why the green line has low mean but high variance. TD3 was not able to converge on any seed within 80000 simulation steps' exploration.

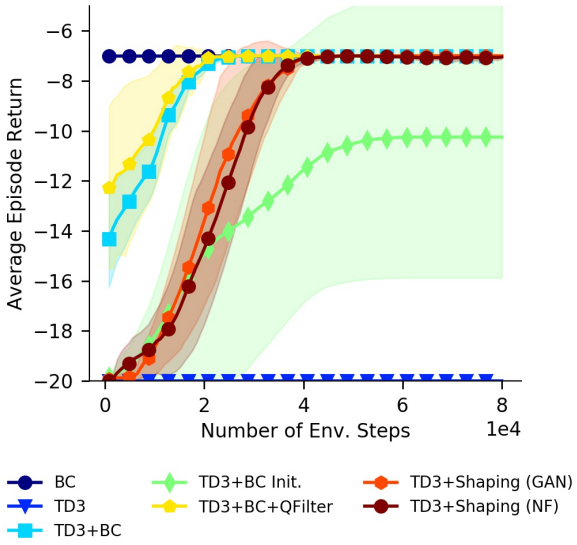

Figure 5.5: Average episode return vs. simulation steps of our method and the baseline algorithms on *Reach-2D* with optimal demonstrations. The empirical mean is computed from 4 seeds and the error bars represent 1σ standard deviation. Both TD3+Shaping, BC and TD3+BC(+QFilter) converged to the optimal policy that spends 7 simulation steps to reach the goal state. Pure TD3 was not able to converge on any seed within 8e4 simulation steps, while TD3+BC Initialization only converged on 3/4 seeds.

5.5 Illustrative Example 2: Visualizing and Evaluating Policies Learned from Sub-Optimal Demonstrations

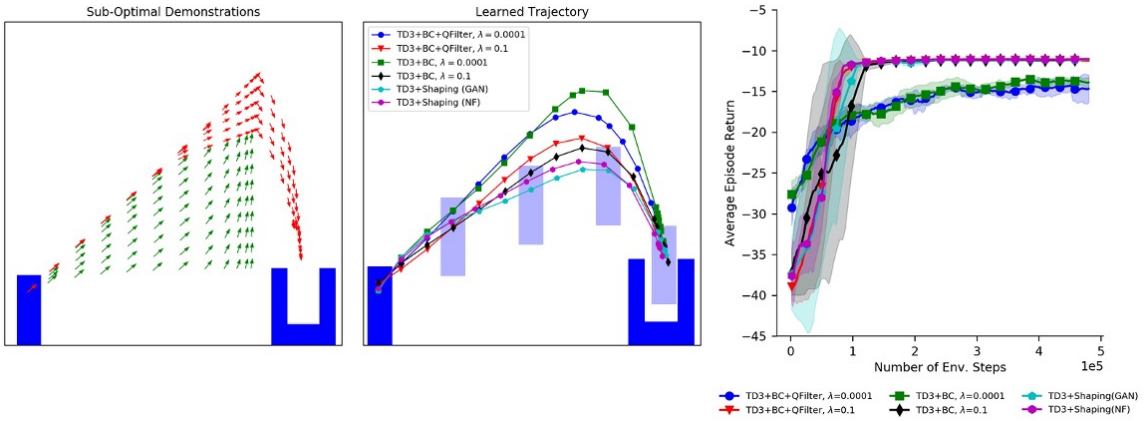

Figure 5.6: Illustration of our method's robustness to sub-optimal demonstrations. The left most figure shows the sub-optimal demonstrations where sub-optimality is introduced by exaggerating the lift of the peg. Crucially, the suboptimal demonstrations are in an area of the state space where the optimal trajectory needs to pass through, so the RL and IL objectives clash. Figure in the middle shows the learned trajectory from TD3+Shaping and TD3+BC(+QFilter) with two choices of λ: 0.1 and 0.0001. The right most figure shows the average episode returns over time. The empirical mean is computed from 4 seeds and the error bars represent 1σ standard deviation.

We use another toy task, named *Gym-PegInsertion-2D*, to illustrate the impact of sub-optimal demonstrations on RL+IL methods. This task is implemented based on the OpenAI Gym API for Robotics environments but we limit the state and action spaces to a 2D plane. As shown in the left figure of Figure 5.6, we provide sub-optimal demonstrations that exaggerate the lift of the peg before inserting it into the hole. These demonstrations are shown as red arrows. In addition, we add demonstrations that pushes the learned policy away from the optimal trajectory, shown as green arrows. These extra sub-optimal demonstrations are given in area of the state space where the optimal trajectory passes through, so the IL objective clashes the RL objective. The learned trajectories are shown in the second figure in Figure 5.6. In particular, we see that the RL+Shaping methods converge to the optimal policy, regardless of any sub-optimality in the demonstrations. On the other hand, the TD3+BC method is sensitive to the relative weight of the TD3 and BC objective, i.e. λ in Equation (5.1). When the role of the TD3 objective gets reduced by setting $\lambda = 0.0001$, the learned policy does not manage to find the optimal solution. Although there exists values of λ that does not bias the learned policy as much (e.g. $\lambda = 0.1$), we cannot know what the best value is without trial and errors.

For complex tasks, one must carefully choose the value of λ to balance between sample efficiency of RL and optimality of the learned policy. However, for RL+Shaping methods, we

are much less concerned about learning sub-optimal policies because the shaping potential does not bias the policy in theory. Furthermore, we can determine the most sensitive parameters via the $\Phi - \sigma^2$ before running RL with the shaping potential, as discussed in 5.4.

5.6 Experiments on Short-Horizon Tasks

We present the results of running baseline algorithms and TD3+Shaping on *Gym-PegInsertion* and *Gym-PickAndPlace* in Figure 5.7 and 5.8, respectively. Both tasks are implemented in Mujoco [73] based on the OpenAI Gym API. We consider these two tasks as having short task horizon because both of them can be successfully completed within 15 simulation steps given an optimal policy. We ran TD3+Shaping with multiple $k^{\text{NF/GAN}}$ values to evaluate its sensitivity to the choice of $k^{\text{NF/GAN}}$.

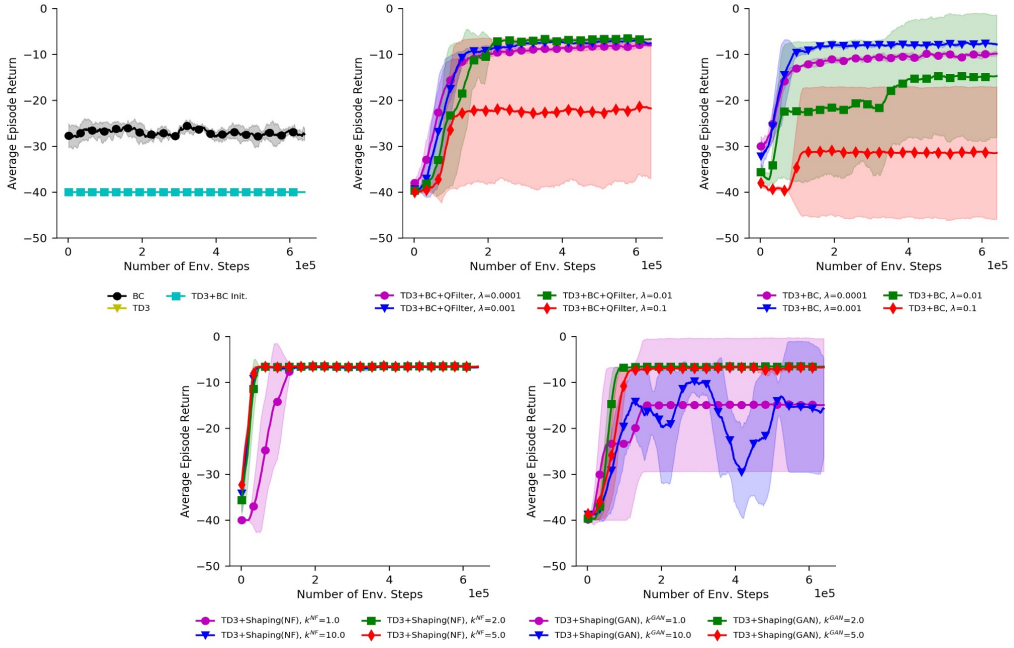

Figure 5.7: Results of running TD3+Shaping and baseline algorithms on *Gym-PegInsertion* with sub-optimal demonstrations. Figures in the top row show the average episode return over time for baseline algorithms, and figures in the bottom row show the corresponding curves of TD3+Shaping methods. The empirical mean is computed from 4 seeds and the error bars represent 1σ standard deviation. With some choices of hyper-parameter, TD3+BC(+QFilter) and TD3+Shaping(GAN) were not able to converge on all seeds, which is why the curves show low mean and high variance.

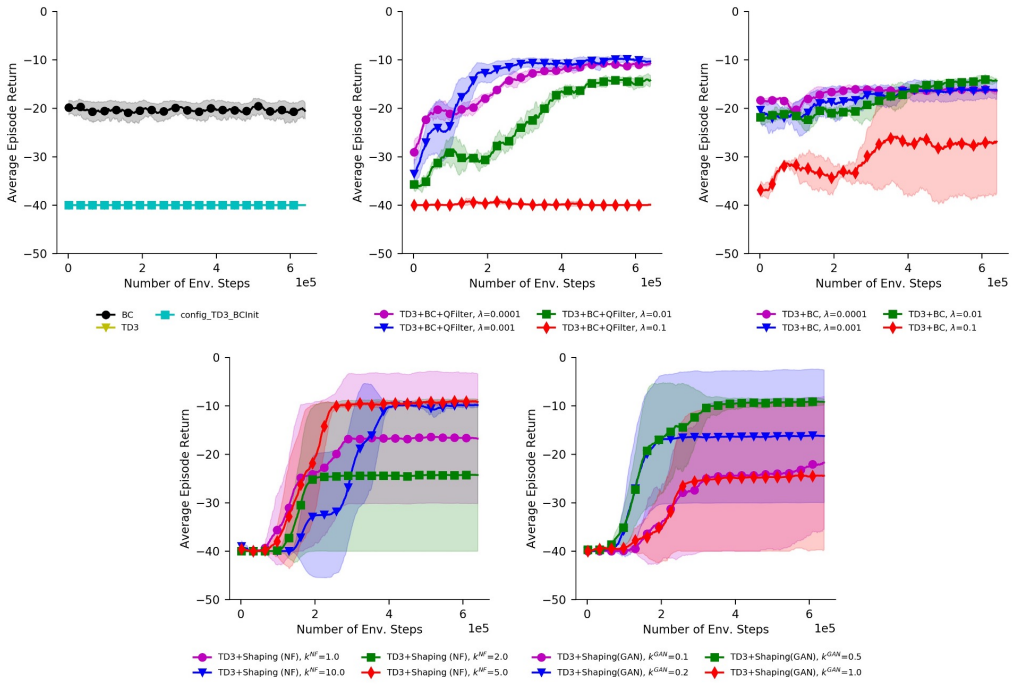

Figure 5.8: Results of running TD3+Shaping and baseline algorithms on *Gym-PickAndPlace* with sub-optimal demonstrations. Figures in the top row show the average episode return over time for baseline algorithms, and figures in the bottom row show the corresponding curves of TD3+Shaping methods. The empirical mean is computed from 4 seeds and the error bars represent 1σ standard deviation. For TD3+Shaping, different choices of $k^{\text{GAN/NF}}$ did not affect the optimality of the learned policy but had an impact on its robustness to seeds. Some learning curves of the TD3+Shaping method have low mean and high variance because the method converged on some seeds but did not on others.

For both tasks, TD3 cannot converge on any seed due to insufficient exploration, while BC is able to complete the two tasks but the learned policies are clearly not optimal as compared to the average episode return achieved by TD3+BC, TD3+BC+QFilter and TD3+Shaping methods.

For TD3+BC, its performance, in terms of sample efficiency and average episode return, varies with difference value of λ. When λ is small (e.g. $\lambda <= 0.001$), TD3+BC converges fast but the final policy is strongly biased towards demonstrated trajectories that are not optimal. In particular, one can see from Figure 5.8 that, when $\lambda < 0.01$, the learned policy via TD3+BC is not much better than the policy trained through just BC. On the other hand, when λ is large (e.g. $\lambda >= 0.01$), TD3+BC becomes less sample efficient. Figure 5.7 shows that when choosing $\lambda = 0.1$, TD3+BC cannot manage to converge on 3/4 seeds, which is why the red line in the third plot of the first row has low mean but high variance.

Adding Q-filter to TD3+BC does not solve the problem of learning sub-optimal policies completely. As shown in Figure 5.8, although TD3+BC+QFilter seems to get higher episode return than TD3+BC for some choices of λ, the performance of the policies still varies.

Moreover, since Q-filter may filter out good demonstrations, TD3+BC+QFilter generally converges lower than TD3+BC, which means it is less sample efficient.

Finally, for TD3+Shaping, we can see that different choice of $k^{\text{NF/GAN}}$ does affect the convergence of the policy. However, as long as the policy converges, it is always optimal unless the scale of the shaping gets too large. For example, all learning curves of TD3+Shaping in Figure 5.8 become flat after a certain amount of exploration. Some curves have low mean but high variance because the algorithm does not converge on all seeds. The fact that all curves are flat at the end indicates that the performance of converged policies are stable and optimal unlike those learned from RL+BC.

5.7 Experiments on Long-Horizon Tasks

In addition to the two short-horizon tasks discussed in 5.6, we also considered a few tasks with longer horizon, which greatly increases the difficulty of finding good policies via RL. These tasks come from MetaWorld [17] without any change to the task objective and setup. For all tasks, it takes more than 40 simulation steps to complete and thereby is almost impossible for any pure RL algorithm to converge under the sparse reward setting. Figure 5.9 shows the results of running TD3+Shaping, TD3+BC and TD3+BC+QFilter on one of these tasks, *MW-PressButton*.

As expected, the performance of the policies learned from TD3+BC given sub-optimal demonstrations is highly dependent on the choice of λ. Even with the Q-filter, the policies still get biased by sub-optimal demonstrations. From figures in the bottom row of Figure 5.9, we can see that using the Q-filter, in fact, does not make much difference in the average episode return of the converged policies.

However, for TD3+Shaping, we notice that it is also not able to learn optimal policies given sub-optimal demonstrations. Moreover, even with optimal demonstrations, the performance of a converged policy seems to be dependent on the hyper-parameters, e.g. $k^{\text{GAN/NF}}$. To confirm that the learned policies are sub-optimal due to the shaping potential, we tried replacing the scaling factor $k^{\text{GAN/NF}}$ with a decreasing sequence $k_t^{\text{GAN/NF}}$ such that $\lim_{t\to\infty} k_t^{\text{GAN/NF}} \to 0$. The results are shown in the right most plots in Figure 5.9, where $k_t^{\text{GAN/NF}}$ decreases linearly from $k^{\text{GAN/NF}}$ to 0 in the gray region. By comparing the third and forth plots in both rows of Figure 5.9, one can see that the policies improve significantly after decreasing the scale of shaping potential, which implies that the shaping potential does affect the optimality of the learned policy for long-horizon tasks.

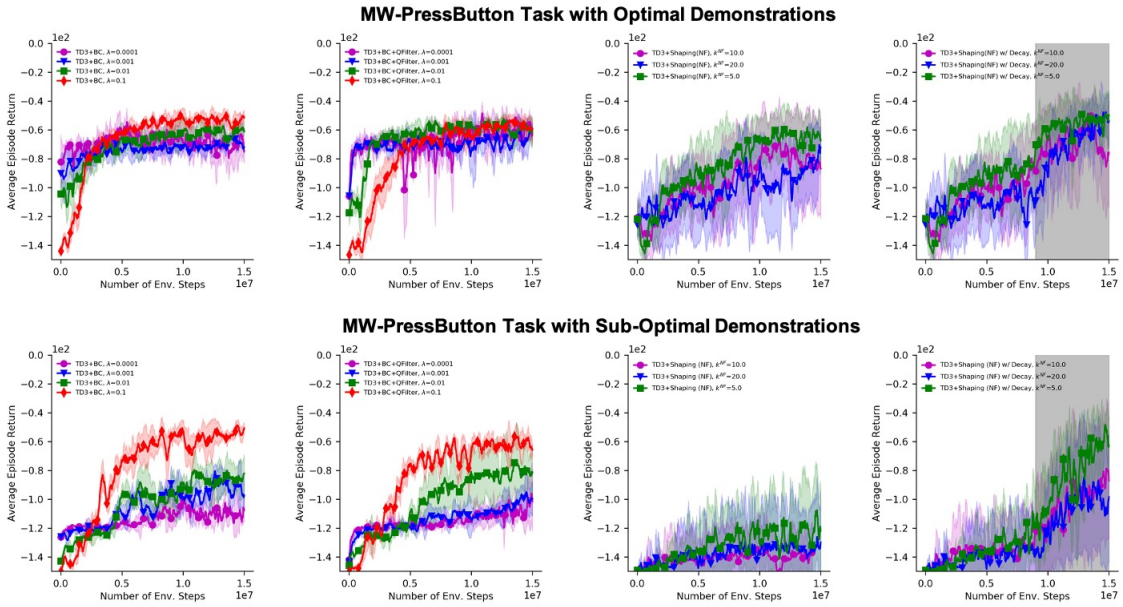

Figure 5.9: Results of running TD3+BC, TD3+BC+QFilter and TD3 with normalizing flow based shaping on *MW-PressButton*. Figures in the top row show the learning curves given optimal demonstrations, and figures in the bottom row show the learning curves given sub-optimal demonstrations. The two figures in the right show the results of applying a decay schedule to the scaling factor k^{NF}. k^{NF} decreases linearly to 0 in the gray region. The empirical mean is computed from 8 seeds and the error bars represent 1σ standard deviation.

5.8 Discussion

We conclude our findings and answers to the questions asked at the beginning of this section:

1. In our method, the shaping potential trained on demonstration dataset $\mathcal{D}$ assigns high values to (s, a) pairs close to those in $\mathcal{D}$. In addition, it is sensitive within a certain region around the states and actions in $\mathcal{D}$, but almost flat outside this region. We found that the performance of the shaping potential depends largely on the size of this region and its overall scale, which are controlled by $k^{\text{GAN/NF}}$ and $\eta^{\text{GAN/NF}}$. In practice, we can tune these two hyper-parameters using the $\Phi - \sigma^2$ plot introduced in 5.4 before training the policy via RL.

2. In theory, policies learned through our method should always be optimal with respective to the RL objective. We showed that this is true for short-horizon tasks in 5.6, where as long as our method converges, the resulting policy is always optimal. However, we also found that this statement does not hold for long-horizon tasks, as discussed in 5.7. In particular, we showed that decreasing the shaping potential after the policy converges could further increase the performance of the policy, which suggests that the shaping

potential biases the policy.

3. Our method is significantly more sample efficient than pure RL methods. It is also more efficient than initializing the policy via IL and then fine-tuning through RL. However, our method is generally less sample efficient than methods with a hybrid RL and IL objective.

4. The performance of our method is very sensitive to the scale and shape of the learned shaping potential, and thereby sensitive to the scaling factor $k^{\text{GAN/NF}}$ and weight on the regularization term $\eta^{\text{GAN/NF}}$. However, this does not make our method difficult to tune because one can adjust these two parameters according to the $\Phi - \sigma^2$ plot before running RL using the shaping potential.

Chapter 6

Conclusion and Future Work

In this thesis, we propose a method that combines reinforcement learning and imitation learning through reward shaping. We use shaping potentials based on deep generative models and actor-critic style RL algorithms such that our method can apply to environments with continuous state and action spaces. The main advantages of our method are 1) it improves the sample efficiency of RL, and 2) the learned policy is unbiased in the presence of sub-optimal demonstrations.

Through several robotic manipulation tasks with varying difficulty, we showed that our method is more efficient than pure RL methods and is able to converge quickly under sparse reward setting. Furthermore, we showed that, for short-horizon tasks with sub-optimal demonstrations, our method is able to learn the optimal policy, and the performance of the converged policy is not sensitive to hyper-parameters. In contrast, policies learned through methods with a hybrid RL and IL objective are not always optimal. However, for long-horizon tasks, our method becomes sensitive to hyper-parameters and cannot manage to find the optimal policy. Therefore, our next step is to address this problem and improve the performance of our method on long-horizon tasks. Besides that, we also seek to further improve the sample efficiency of our method so that its performance is more consistent across different seeds. In particular, we will consider using more sophisticated exploration strategies and prioritized experience replay which favors transitions that get the true reward from the environment.

Appendix A

Hyper-Parameters Used for All Manipulation Tasks

In Table A.1, we describe the details of the important hyper-parameters used in our experiments. The hyper-parameters for TD3 are mostly adopted from the original TD3 implementation in [15]. The hyper-parameters for GAN and normalizing flow based shaping potentials are obtained by trial and errors.

	Toy Tasks	Short-Horizon Tasks	Long-Horizon Tasks
TD3 Hyper-Parameters			
Discount Rate (γ)	0.95		0.99
Number of Training Epochs	100	400	2500
Env. Steps / Epoch ($E \times T$)	200	400	1500
Number of Batch / Epoch (B)	40		
Batch Size	256		
Actor/Critic Layer Sizes	[256, 256, 256]		
Actor/Critic Update Freq. (n_{critic})	2		
Target Policy Noise (σ^{TD3})	0.2		
Target Policy Noise Clip (c^{TD3})	0.5		
Critic Learning Rate	1e-3		
Actor Learning Rate	1e-3		
Polyak Average (τ)	0.05		
n-step Return (n)	Not Used		30
Buffer Size	1000000		
BC Initialization Hyper-Parameters			
Number of Training Epochs	2000		
Batch Size	128		
Hybrid RL + BC Objective Hyper-Parameters			

Batch Size	128
GAN Based Shaping Potential Hyper-Parameters	
Ensemble (K)	2
Number of Training Epochs	10000
Batch Size	128
Latent Dimension	$dim(\mathcal{S}) + dim(\mathcal{A})$
Gradient Penalty (gp^{GAN})	1.0
D_ψ / G_ϕ Update Freq. ($n_{\text{critic}}^{\text{GAN}}$)	5
Adam Learning Rate α^{GAN}	1e-4
Adam β_1^{GAN}	0.5
Adam β_2^{GAN}	0.9
Normalizing Flow Based Shaping Potential Hyper-Parameters	
Number of Ensembles	2
Number of Training Epochs	10000
Batch Size	128
Layer Sizes	[256, 256]
Number of Bijective Functions (K)	4
Learning Rate	1e-4

Table A.1: List of hyper-parameters used for all tasks in our experiments.

Appendix B

Visualization of the Manipulation Tasks

Figure B.1 shows the RGB images of the 3 tasks which we implemented using the OpenAI Gym API for Robotics environments. Figure B.2 shows the RGB images of all manipulations tasks provided in MetaWorld. The image is copied directly from MetaWorld's open-source GitHub repository. In our experiments, we adopted these tasks without change to their objectives and setup.

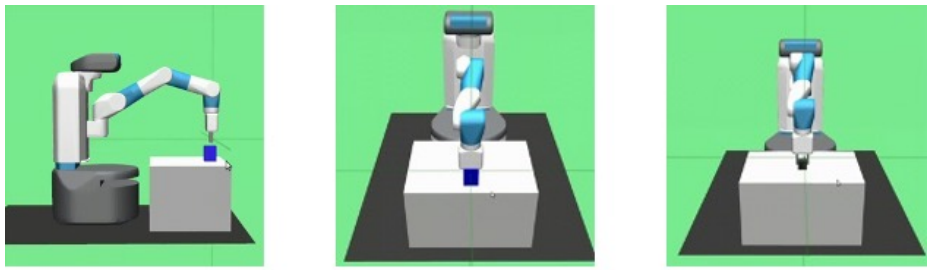

Figure B.1: RGB images of the three manipulation tasks implemented based on OpenAI Gym Robotics API [18]

.

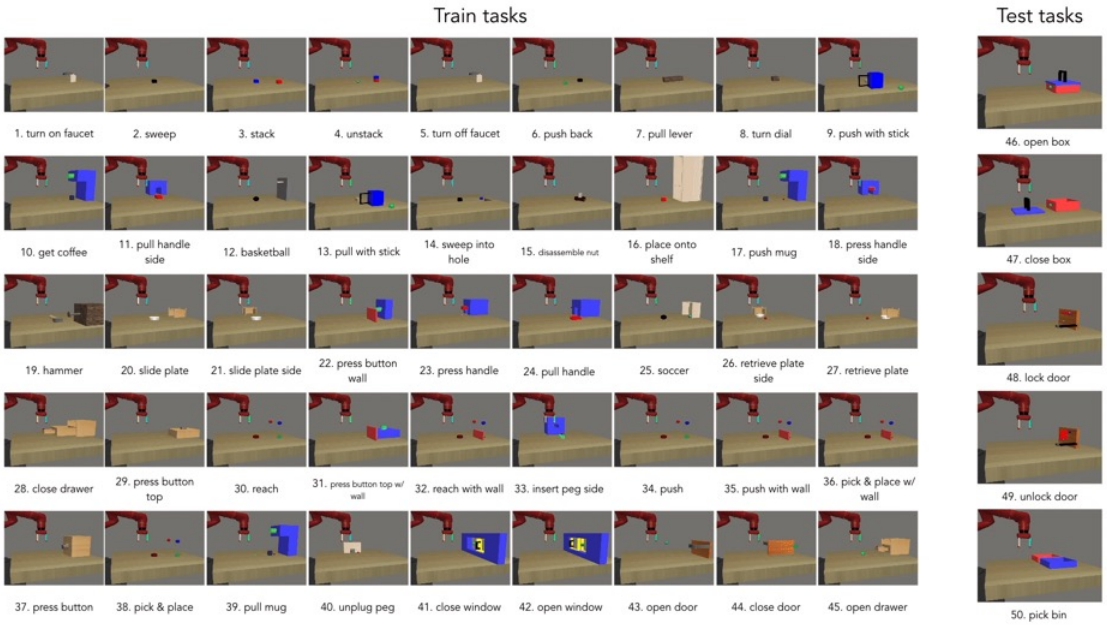

Figure B.2: RGB images of the 50 manipulation tasks from MetaWorld [17].

.

Bibliography

[1] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. "One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning". In: *arXiv:1802.01557 [cs]* (2018).

[2] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *arXiv:1806.10293 [cs, stat]* (2018).

[3] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *arXiv:1011.0686 [cs, stat]* (2011).

[4] Jonathan Ho and Stefano Ermon. "Generative Adversarial Imitation Learning". In: *arXiv:1606.03476 [cs]* (2016).

[5] Justin Fu, Katie Luo, and Sergey Levine. "Learning Robust Rewards with Adversarial Inverse Reinforcement Learning". In: *arXiv:1710.11248 [cs]* (2018).

[6] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. In: *arXiv:1809.02925 [cs, stat]* (2018).

[7] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations". In: *arXiv:1709.10087 [cs]* (2018).

[8] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards". In: *arXiv:1707.08817 [cs]* (2018).

[9] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. "Learning to select and generalize striking movements in robot table tennis". In: *The International Journal of Robotics Research* 32.3 (2013), pp. 263–279. ISSN: 0278-3649, 1741-3176.

[10] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. "Overcoming Exploration in Reinforcement Learning with Demonstrations". In: *arXiv:1709.10089 [cs]* (2018).

[11] Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E. Taylor, and Ann Nowé. "Reinforcement Learning from Demonstration through Shaping". In: *IJCAI*. 2015.

[12] Eric Wiewiora, Garrison Cottrell, and Charles Elkan. "Principled Methods for Advising Reinforcement Learning Agents". In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. 2003, pp. 792–799.

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Networks". In: *arXiv:1406.2661 [cs, stat]* (2014).

[14] Danilo Jimenez Rezende and Shakir Mohamed. "Variational Inference with Normalizing Flows". In: *arXiv:1505.05770 [cs, stat]* (2016).

[15] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *arXiv:1802.09477 [cs, stat]* (2018).

[16] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *arXiv e-prints* (2018), arXiv:1801.01290.

[17] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning". In: *arXiv:1910.10897 [cs, stat]* (2019).

[18] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "OpenAI Gym". In: *arXiv:1606.01540 [cs]* (2016).

[19] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. *A Survey on Policy Search for Robotics*. 2013.

[20] Marc Peter Deisenroth and Carl Edward Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. 2011, pp. 465–472.

[21] David Meger, Juan Camilo Gamboa Higuera, Anqi Xu, Philippe Giguere, and Gregory Dudek. "Learning legged swimming gaits from experience". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 2332–2338.

[22] Marc Deisenroth, Carl Rasmussen, and Dieter Fox. "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning". In: *Robotics: Science and Systems VII*. 2011.

[23] Joschka Boedecker, Jost Tobias Springenberg, Jan Wülfing, and Martin Riedmiller. "Approximate real-time optimal control based on sparse Gaussian process models". In: *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. 2014, pp. 1–8.

[24] R. Lioutikov, A. Paraschos, J. Peters, and G. Neumann. "Sample-based informationl-theoretic stochastic optimal control". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 3896–3902.

[25] Sergey Levine and Pieter Abbeel. "Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. 2014, pp. 1071–1079.

[26] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning". In: *arXiv:1708.02596 [cs]* (2017).

[27] Justin Fu, Sergey Levine, and Pieter Abbeel. "One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors". In: *arXiv:1509.06841 [cs]* (2016).

[28] Igor Mordatch, Nikhil Mishra, Clemens Eppner, and Pieter Abbeel. "Combining model-based policy search with online model learning for control of physical humanoids". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 242–248.

[29] Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. "Stochastic Latent Actor-Critic: Deep Reinforcement Learning with a Latent Variable Model". In: *arXiv:1907.00953 [cs, stat]* (2019).

[30] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. "Learning Latent Dynamics for Planning from Pixels". In: *arXiv:1811.04551 [cs, stat]* (2019).

[31] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. "Improving Sample Efficiency in Model-Free Reinforcement Learning from Images". In: *arXiv:1910.01741 [cs, stat]* (2019).

[32] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning". In: *arXiv:1509.02971 [cs, stat]* (2019).

[33] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. "Trust Region Policy Optimization". In: *arXiv:1502.05477 [cs]* (2017).

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *arXiv:1707.06347 [cs]* (2017).

[35] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. "DeepMind Control Suite". In: *arXiv:1801.00690 [cs]* (2018).

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing Atari with Deep Reinforcement Learning". In: *arXiv:1312.5602 [cs]* (2013).

[37] Marc G. Bellemare, Will Dabney, and Rémi Munos. "A Distributional Perspective on Reinforcement Learning". In: *arXiv:1707.06887 [cs, stat]* (2017).

[38] Richard Bellman. "Dynamic programming and stochastic control processes". In: *Information and Control* 1.3 (1958), pp. 228–239. ISSN: 0019-9958.

[39] J.N. Tsitsiklis and B. Van Roy. "An analysis of temporal-difference learning with function approximation". In: *IEEE Transactions on Automatic Control* 42.5 (1997), pp. 674–690. ISSN: 00189286.

[40] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic Policy Gradient Algorithms". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. 2014, I–387–I–395.

[41] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. "Deep Reinforcement Learning that Matters". In: *arXiv:1709.06560 [cs, stat]* (2019).

[42] Brian D. Ziebart, J. Andrew Bagnell, and Anind K. Dey. "Modeling Interaction via the Principle of Maximum Causal Entropy". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. 2010, pp. 1255–1262.

[43] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping". In: *Proceedings of the Sixteenth International Conference on Machine Learning*. 1999, pp. 278–287.

[44] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. "Concrete Problems in AI Safety". In: *arXiv:1606.06565 [cs]* (2016).

[45] John Asmuth, Michael L. Littman, and Robert Zinkov. "Potential-based shaping in model-based reinforcement learning". In: *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*. 2008, pp. 604–609.

[46] Sam Devlin and Daniel Kudenko. "Theoretical Considerations of Potential-Based Reward Shaping for Multi-Agent Systems". In: *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*. 2011, pp. 225–232.

[47] Kyriakos Efthymiadis, Sam Devlin, and Daniel Kudenko. "Overcoming erroneous domain knowledge in plan-based reward shaping". In: *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. 2013, pp. 1245–1246.

[48] Adam Eck, Leen-Kiat Soh, Sam Devlin, and Daniel Kudenko. "Potential-Based Reward Shaping for POMDPs". In: *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*. 2013, pp. 1123–1124.

[49] Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. "Unifying Count-Based Exploration and Intrinsic Motivation". In: *arXiv:1606.01868 [cs, stat]* (2016).

[50] Georg Ostrovski, Marc G. Bellemare, Aaron van den Oord, and Remi Munos. "Count-Based Exploration with Neural Density Models". In: *arXiv:1703.01310 [cs]* (2017).

[51] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. "#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning". In: *arXiv:1611.04717 [cs]* (2017).

[52] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. "VIME: Variational Information Maximizing Exploration". In: *arXiv:1605.09674 [cs, stat]* (2017).

[53] Shakir Mohamed and Danilo Jimenez Rezende. "Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning". In: *arXiv e-prints* (2015), arXiv:1509.08731.

[54] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. "Curiosity-driven Exploration by Self-supervised Prediction". In: *arXiv e-prints* (2017), arXiv:1705.05363.

[55] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. "Exploration by Random Network Distillation". In: *arXiv:1810.12894 [cs, stat]* (2018).

[56] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. "A survey of robot learning from demonstration". In: *Robotics and Autonomous Systems* 57.5 (2009), pp. 469–483. ISSN: 0921-8890.

[57] Dean A. Pomerleau. "ALVINN: An Autonomous Land Vehicle in a Neural Network". In: *Advances in Neural Information Processing Systems 1*. Ed. by D. S. Touretzky. 1989, pp. 305–313.

[58] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. "End to End Learning for Self-Driving Cars". In: *arXiv e-prints* (2016), arXiv:1604.07316.

[59] Pieter Abbeel and Andrew Y. Ng. "Apprenticeship learning via inverse reinforcement learning". In: *Twenty-first international conference on Machine learning - ICML '04*. 2004, p. 1.

[60] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. "Maximum Entropy Inverse Reinforcement Learning". In: *Proc. AAAI*. 2008, pp. 1433–1438.

[61] Chelsea Finn, Paul F. Christiano, Pieter Abbeel, and Sergey Levine. "A Connection between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models". In: *CoRR* abs/1611.03852 (2016).

[62] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. "Deep Q-learning from Demonstrations". In: *arXiv:1704.03732 [cs]* (2017).

[63] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. "Safe and Efficient Off-Policy Reinforcement Learning". In: *arXiv e-prints* (2016), arXiv:1606.02647.

[64] Nan Jiang and Lihong Li. "Doubly Robust Off-policy Value Evaluation for Reinforcement Learning". In: *arXiv e-prints* (2015), arXiv:1511.03722.

[65] Scott Fujimoto, David Meger, and Doina Precup. "Off-Policy Deep Reinforcement Learning without Exploration". In: *ICML*. 2019.

[66] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, and Nicolas Heess. "Reinforcement and Imitation Learning for Diverse Visuomotor Skills". In: *arXiv:1802.09564 [cs]* (2018).

[67] Richard S. Sutton. “Learning to predict by the methods of temporal differences”. In: *Machine Learning* 3.1 (1988), pp. 9–44. ISSN: 1573-0565.

[68] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *arXiv:1312.6114 [cs, stat]* (2014).

[69] George Papamakarios, Theo Pavlakou, and Iain Murray. “Masked Autoregressive Flow for Density Estimation”. In: *arXiv:1705.07057 [cs, stat]* (2018).

[70] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein GAN”. In: *arXiv:1701.07875 [cs, stat]* (2017).

[71] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. “Improved Training of Wasserstein GANs”. In: *arXiv:1704.00028 [cs, stat]* (2017).

[72] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (2014).

[73] E. Todorov, T. Erez, and Y. Tassa. “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033.