Deep Reinforcement Learning for Continuous Control Environments with Multi-modal State-space and Sparse Rewards

by

Cancheng Zeng

A Thesis Submitted to

The Hong Kong University of Science and Technology

in Partial Fulfillment of the Requirements for

the Degree of Master of Philosophy

in Master of Philosophy in Computer Science and Engineering

Date: _____, Hong Kong

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

Cancheng Zeng

Date: _____

Deep Reinforcement Learning for Continuous Control Environments with Multi-modal State-space and Sparse Rewards

by

Cancheng Zeng

This is to certify that I have examined the above MPhil thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the thesis examination committee have been made.

Professor Dit-Yan Yeung, Thesis Supervisor and Acting Head
Department of Computer Science and Engineering

Department of Computer Science and Engineering, HKUST

Date: _____

Acknowledgments

Foremost, I'd like to express my sincere gratitude to my advisor Prof. Dit-Yan Yeung, for his kindness, generosity, open-mindedness, and his continuous support of my study.

Besides my advisor, I'd like to thank my fellow labmates Shi Xingjian, Li Siyi, Gao Zhihan and Leonard Lausen, for the countless valuable discussions in these two years.

Contents

List of Figures

List of Tables

Deep Reinforcement Learning for Continuous Control Environments with Multi-modal State-space and Sparse Rewards

by Cancheng Zeng

Department of Computer Science and Engineering, HKUST

The Hong Kong University of Science and Technology

Abstract

One of the most important problems in artificial intelligence is learning to solve control problems without human supervision. Recent advances in end-to-end deep reinforcement learning methods have achieved significant progress in this domain, on a number of problems including a subset of Atari games [3], Go game [4], and several simple robot control environments [5]. However, a general solution to realistic robot control problems is still missing. Most realistic robot control problems have multi-modal state space, which usually have both low-dimensional motion sensor input and high-dimensional image sensor input. Apart from that, a smooth and informative reward function is usually unavailable in realistic environments. Most realistic robot control problems only have sparse and discrete reward functions.

This thesis investigates several continuous control problems with the property of multi-modal state space and sparse reward functions. We propose several solutions for improving the performance of flat reinforcement learning methods on a subset of the proposed problems, including Wasserstein trust-region method, exceptional advantage regularization method, and robust concentric Gaussian mixture policy model. A hierarchical reinforcement learning method is also proposed to solve the target tasks without domain-specific knowledge given a pre-defined set of primary tasks.

Chapter 1

Introduction

1.1 Overview

Deep reinforcement learning methods have become popular in the machine learning field in the recent years. Through the combination of deep neural network models and reinforcement learning theory, advances in deep reinforcement learning have achieved human-level control performance in playing Atari games [3], mastered the board game GO [4], learned to play Texas Hold'em pokers [6], to solve simple robot control tasks [5] and to play a simplified version of the game Dota 2 [7].

However, the capability of contemporary deep reinforcement learning methods is still limited when we consider real-world problems, such as playing the StarCraft games, controlling robots to do meaningful tasks and making stock trading decisions. Generally speaking, the contemporary end-to-end deep reinforcement learning methods have been able to solve two classes of problems. The first class of problems consists of single-agent environments with unknown state-transition dynamics but trivial task logics, such as the Atari games. The second class of problems consists of the multi-agent games which could have relatively complex task logics but in a closed system, such as the board game Go.

General real-world robot problems, however, are more challenging. Compared with video game environments like [3], robot problems have continuous action spaces instead of discrete spaces. This has made the policy exploration problem much more difficult. Apart from that, most robot control problems have multi-modal state space consisting of both low-dimensional and high-dimensional inputs. The multi-modality property further introduces complexity to reinforcement learning problems because the local-minimum problem is much more difficult than in supervised learning domains. Finally, the lack

of smooth and informative reward signals in realistic robot problems pose significant challenges to contemporary end-to-end deep reinforcement learning methods.

In this study, we investigate a set of challenging robot control environments and propose flat and hierarchical reinforcement learning methods to solve them.

1.2 Background in Reinforcement Learning (RL)

Reinforcement learning [8] is the problem of learning a policy that maps from situations to actions in order to maximize a numerical reward signal. The learner must discover the optimal policy by examining different possibilities. Reinforcement learning is different from supervised learning that the learner is not provided labeled examples by a knowledgeable external supervisor. Apart from that, reinforcement learning agents are trained on the data that are experiences generated by itself, instead of any existing training data. Therefore, reinforcement learning agents not only need to exploit their experience, but also need to efficiently explore the problem space so that the experience is useful. Reinforcement learning is also different from unsupervised learning, whose objective is finding hidden structures in unlabeled data without any guiding signal.

Most reinforcement learning study is based on the problem formulation of Markov Decision Processes (MDPs). MDP is a framework defining the problem of learning from interaction. The learner is called the *agent*. Everything outside that interacts with the agent is called the *environment*. The agent and environments interact during a finite sequence of discrete time steps, $t = 0, 1, 2, 3, ...$ At each time step t, the agent receives an observation of the environment's *state*, $s \in S$ and selects an *action* $a \in A$. The agent receives a numerical reward r, and also the next state s'. A finite-horizon discounted Markov decision process (MDP) is defined by the tuple $(S, A, P, r, \rho_0, \gamma)$ where the $\rho_0 : S \mapsto \mathbb{R}$ is the initial state distribution, $P(s'|s, a) : S \times A \mapsto \mathbb{R}$ is the distribution describing the state transition dynamics, $r : S \times A \mapsto \mathbb{R}$ is the expected reward function, and $\gamma \in (0, 1]$ is the discount factor.

Let $\pi : S \times A \mapsto [0, 1]$ denote a policy, the following equations are definitions of the

action-value function Q_π, the value function V_π and the advantage function A_π:

$$Q_\pi(s_t, a) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\Big[\sum_{l=0}^{\infty} \gamma^l r(s_t)\Big], \tag{1.1}$$

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots}\Big[\sum_{l=0}^{\infty} \gamma^l r(s_t)\Big], \tag{1.2}$$

$$\text{where } a_t \sim \pi(a_t|s_t), \text{for } t \geq 0 \tag{1.3}$$

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s), \tag{1.4}$$

The action-value function Q estimates the average total discounted reward following the policy π after taking the action a at the current state s_t. The value function V estimates the average total discounted reward following the policy π both at the current state and in the future. The advantage function A is the difference between Q and V, which indicates the advantage of taking the action a at the current state over the policy π.

1.3 Scope of Study

Realistic robot control problems in reinforcement learning involve many sources of complexity. However, we are only concerned with some of the typical problems. We focus on a set of reinforcement learning problems with multi-modality state space and continuous action space. We also specifically study some problems that not only have the previously mentioned properties, but also have sparse reward functions.

1.3.1 Multi-modality State Space

The state spaces of a vast majority of the problems in previous reinforcement learning studies are in single-modality, such as a low-dimensional vector of internal game state or locomotion sensor signal, or a single high dimensional image stream. Most recent works in reinforcement learning for continuous control focus on the RL tasks with low-dimensional state input only [5]. Some study [9] also tried to handle tasks with a single image-input but in limited complexity.

However, it is more common in real-world environments that a robot agent has multiple types of sensors installed simultaneously. In such cases, the state space of the reinforcement learning problem is a combination of input with different modalities. The different modalities in the state space could provide complementary information that cannot be

substituted. For a typical robot locomotion problem, the low-dimensional motion sensor input usually provides accurate information about the agent's physical status. The image input, on the other hand, may provide information on the external environments and object relationships. If the motion sensor input is missing, the robot agent would not perform its motor control well, on the other hand, if the image sensor is missing, the robot agent would not know anything about its task and environment.

Apart from motion and image sensors, real-world robot agents may have other modalities such as wireless network antenna input, human language instructions and laser sensors. This thesis focus on robot control environments whose state observation consists of a low-dimensional motion sensor signal and an image sensor stream, which is among the most typical cases of real-world robot problems.

In the context of reinforcement learning, the previously described multi-modality problem is significantly more difficult than single-modality problems. The reason is that the feature of image observations is much more difficult to learn than the features of motion sensor signals. End-to-end reinforcement learning methods would typically prioritize the low-dimensional input in the initial phase of training, and get stuck until the agent is able to extract useful information from the image input. The local-minimum problem in this case is more challenging than the cases in supervised learning, because the generation of training samples are also controlled by the agent's acting policy.

1.3.2 Sparse Reward Function

The sparse / delayed reward signal problem has been one of the major problems in reinforcement learning domains for the past 20 years. The reinforcement learning problems with sparse rewards refer to the environments where informative reward signals are very rare. For example, the agent in such environments may just receive a constant negative reward at most timesteps but only receives a positive reward when some specific rare event happens. The agent may not be able to learn anything about a good policy if it has never triggered such events that generate informative reward signals. Apart from that, the agent may not be able to improve further after experienced such events if it has already been very conservative in the degree of exploration. The core problem is how the agent can perform efficient exploration in the environment.

Many methods have been proposed to solve the sparse reward problem in environments. However, none of the methods are able to solve the general case. Some of them could only solve special cases of reinforcement learning problems, that are far from practical problems, while the others require a tremendous amount of domain-specific engineering component.

A smooth and informative reward function can generally make the training of reinforcement learning agents easier. Theoretically, any reinforcement learning problems when given a reward function that is informative enough could be solved easily by contemporary methods. However, such a reward function itself is rarely accessible in real-world environments. Reward shaping [10] techniques may be used to improve the reward function, but such methods usually also rely on domain-specific engineering. In our study, we try to solve several realistic sparse environments without the application of reward shaping.

1.3.3 Assumptions on Relationship of Tasks

Apart from the objective to solve sparse multi-modal reinforcement learning problems with end-to-end reinforcement learning problems, we are also particularly interested in a paradigm, where the objective is to solve a *target task* given a set of *source tasks*. The policies that solve the source tasks, namely the *actuator policies*, are available to the agent when it is faced with the target task. We call them actuator policies because these policies perform low-level control skills instead of making the highest level decision in the target task. We assume that the target task can be solved by a sequential execution of a subset of the actuator policies. A *decider agent* needs to be trained, which controls the scheduling of the actuator policies.

This paradigm is related to hierarchical reinforcement learning (HRL), which focuses on learning and utilizing closed-loop partial policy / skills to solve a complex task. However, our case is different from the standard setting of HRL. The standard HRL setting assumes that the agent needs to extract the useful closed-loop partial policy from the original target task. However, we focus on extracting and reusing partial policies from other tasks.

This paradigm is also related to transfer learning in reinforcement learning [11], because the source-task policies are reused by the decider agent to solve another task. However, conventional transfer learning paradigms usually don't pose assumptions on a

hierarchical relationship between the target task and the source tasks.

This is the only assumption on the relationship between the source tasks and the target task. It is not known whether any specific source task is useful. A source task policy might be useful in some cases, or it could be completely useless. The execution time of an actuator policy is also unknown. An actuator policy might be executed for only 1 time-step before the agent switch to another one, or it might also be executed for the whole episode.

Since the set of source-task policies is pre-defined and there is no autonomous method for selecting then, the selection of the source tasks is considered a domain-specific design component. However, it is common that real-world complex problems can be decomposed into primary tasks. For example, classical robot locomotion methods organize the abstract problems (such as trajectory planning) and low-level problems (such as motor control) in hierarchical structures. Apart from that, it is also common that human beings would decompose the original problem into a series of easier problems when the problem it faced is too complex.

If this hierarchical task relationship assumption was not present, the problem would belong to the class of the fully autonomous hierarchical reinforcement learning problem. In this case, the agents need to learn not only a task structure but also all actuator policies from scratch. The solution to such problems is currently considered in-feasible to obtain for general cases [12].

1.4 Research Questions

We will propose several reinforcement learning environments for continuous control in section 3.1 according to the stated scope, and the proposed problems will be of the main focus of the study. The primary research question that we would like to study is:

Can we develop an agent that can achieve sufficient performance (in terms of total episode reward) in the reinforcement learning environments defined in section 3.1 within a feasible training time, with an RL model that doesn't rely on domain-specific techniques (including reward shaping, manual definition of policy hierarchy, manual / heuristic methods to define sub-goals, or manual criteria on the switching of actuator policies), with or without a predefined set of source tasks?

We will first answer the question on whether a hierarchical architecture is necessary:

Research Question 1: *Can a contemporary flat reinforcement learning method solve the proposed environments?*

We will then answer the question on whether an end-to-end hierarchical reinforcement learning can solve this: A model that solves the question needs to address two issues: reusing the policies learned in source tasks and solving the target task. The two issues lead to the following two research questions respectively:

Research Question 2.1: *Can a trained actuator policy achieve an acceptable performance (in terms of the expected total return of the original source task) in a target environment, which might have different initial state distribution?.*

Research Question 2.1: *Can a decider agent be developed to solve the target tasks defined in 3.1 given the trained actuators policies, without other domain-specific designs?.*

Chapter 2

Related Works

2.1 Policy Gradient Methods

Policy gradient methods are a class of reinforcement learning methods that learn a parametrized policy that select actions without consulting any value estimates. A value function may be still used to learn the policy parameter, but not for selecting actions. The vast majority of deep reinforcement learning methods for continuous control belong to the class of policy gradient methods instead of methods that are based on action-value functions, because it is infeasible to compute the optimal action from general action-value functions with a continuous action space. Policy gradient methods optimize a parametrized policy model $\pi_\theta(a|s), a \in A, s \in S$, so that the expected return:

$$J(\theta) = \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t r_t\Big]$$

is maximized. The constant γ is a discount factor and T is the episode length.

Policy gradient methods maximize the expected return iteratively by estimating the gradient $g := \nabla\mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t r_t\Big]$, which has the form [13]:

$$g = \mathbb{E}_{t,s_t,a_t}\Big[\Psi_t \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{q(a_t|s_t)}\Big] \tag{2.1}$$

where $q(.)$ is the distribution that generates the samples. If the samples are generated by the policy π_θ, then policy gradient can be simplified using the REINFORCE trick [14]:

$$g = \mathbb{E}_{t,s_t,a_t}\Big[\Psi_t \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)}\Big] = \mathbb{E}_{t,s_t,a_t}\Big[\Psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)\Big] \tag{2.2}$$

The term Ψ_t may be one of the following [13]:

1. $\sum_{t=0}^{\infty} r_t$: expected total return

2. $\sum_{t'=t}^{\infty} r_t$: expected return following a_t

3. $\sum_{t'=t}^{\infty}[r_t - b(s_t)]$

4. $Q_\pi(s_t, a_t)$

5. $A_\pi(s_t, a_t)$

6. $\hat{A}_t^{(1)} := \delta_t = r_t + V_\pi(s_{t+1}) - V_\pi(s_t)$: 1-step TD residual

7. $\hat{A}_t^{(2)} := \delta_t + \gamma\delta_{t+1} = r_t + \gamma r_{t+1} + \gamma^2 V_\pi(s_{t+2}) - V_\pi(s_t)$: 2-step TD residual

8. $\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_{t+l} = r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) - V(s_t)$: k-step TD residual

9. $GAE(\lambda) = \frac{1}{(1+\lambda+\lambda^2+\ldots)}\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \ldots\right) \approx \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}$, where $\lambda \in [0, 1]$: the generalized advantage estimator proposed by [13]

Many methods that belong to the class of policy gradient methods have been proposed to solve reinforcement learning problems for continuous control. They can be classified into two categories: on-policy methods and off-policy methods. On-policy methods perform gradient updates only with the data sampled by the current policy, while off-policy methods could use the data that is generated by any other distributions. Off-policy methods are generally more sample efficient, but need to handle the problems related to the deviation of sampling distribution from the current distribution. Apart from that, many reinforcement learning techniques are only valid for on-policy data and can not be used in off-policy methods. Therefore, we will mainly focus on on-policy policy gradient methods. The following sections will discuss some state-of-art policy gradient methods in details.

2.1.1 Trust Region Policy Optimization

The method proposed by [15], namely Trust Region Policy Optimization (TRPO) is one of the early works in deep reinforcement learning for continuous control. The method formulates the policy optimization problem as a constraint optimization problem:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & J(\theta) \\ \text{subject to} \quad & \overline{D_{KL}}(\pi_{\theta_{old}} \| \pi_\theta) \leq \delta_{KL} \end{aligned} \tag{2.3}$$

where $\pi_{\theta_{old}}$ is the old policy parameter before performing the parameter update, and $\overline{D_{KL}}$ is the average KL divergence over the samples, δ_{KL} is the hyper-parameter that control the step size of updates.

The method solves this problem through 4 steps. The first step computes g following the Equation 2.1. The second step computes the Riemannian metric tensor of the parameter space of the policy model $A = H_\theta \left(\overline{D_{KL}}(\pi_{\theta_{old}} \| \pi_\theta)\right)$, where $H_\theta(.)$ denotes the Hessian matrix with respect to θ. The third step obtains which is the natural gradient update direction $s = A^{-1}g$ by solving $g = As$ through the conjugate gradient algorithm. The fourth step computes the maximum step-size $\beta = \sqrt{2\delta_{KL}/s^T As}$ and performs a line search to ensure the improvement of the objective function.

The method is able to solve some simple robot control tasks in [16] within a few millions of training samples. However, both the computation of the Hessian matrix and the conjugate gradient algorithm are infeasible for a neural network model with nontrivial size.

2.1.2 Kronecker-factored Trust Region Policy Gradient

The work of [9] proposes to reduce the computation time solving the natural gradient by the Kronecker-factored approximation method. The method is named Actor-critic Kronecker-factored Trust Region method (ACKTR). The Fisher Information Matrix $F = \mathbb{E}[\nabla_\theta L \nabla_\theta L^T]$, where $L = \log \pi(a|s)$ is used here to approximate the Riemannian metric tensor. The matrix is approximated by:

$$F = \mathbb{E}[\nabla_\theta L \nabla_\theta L^T] = \mathbb{E}[aa^T \otimes \nabla_s L \nabla_s L^T] \approx \mathbb{E}[aa^T] \otimes \mathbb{E}[\nabla_s L \nabla_s L^T]$$

where a is the input activation vector corresponding to the neural network weight parameters and s is the corresponding preactivation vector, The natural gradient is then solved by:

$$s = A^{-1}g \approx F^{-1}g \approx \left(\mathbb{E}[aa^T] \otimes \mathbb{E}[\nabla_s L \nabla_s L^T]\right)^{-1} g = \mathbb{E}[aa^T]^{-1} g \mathbb{E}[\nabla_s L \nabla_s L^T]^{-1} \quad (2.4)$$

The method further speeds up the optimization by reusing recently computed statistic matrices and also by performing asynchronous computation. The method manages to reduce the computation time so that it is comparable to current first-order gradient descent algorithms. The method manages to computationally efficiently perform trust

region natural policy gradient optimization without sacrificing the reinforcement learning performance. A remaining problem is that the method is still batch-based and has a high memory occupation.

2.1.3 Proximal Policy Gradient Method

The author of [17] proposes a first-order policy optimization method, namely proximal policy optimization (PPO). The method clips the surrogate objective according to the following equation:

$$L_{CLIP}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right] \tag{2.5}$$

where $r_t(\theta) = \pi_\theta / \pi_{\theta_{old}}$ is the likelihood ratio of the current policy over the sample policy, and $\epsilon \in [0, 1]$ is a hyperparameter.

The algorithm performs minibatch stochastic optimization at each epoch of collected agent's experience, for a number of epochs. Therefore it is actually an off-policy method since the policy has already deviated from the sampling policy once updated for the first minibatch in each epoch.

The advantage of PPO method is that it has both a relatively low time computational complexity and memory complexity. It can achieve a fast convergence rate in terms of the reinforcement learning performance (total episode reward). It also provides some kind of constraint at each epoch so that the policy doesn't deviate too far from the sampling policy. Although the PPO method has a high convergence rate, there is no evidence that shows the method could achieve a better final performance.

2.1.4 Other Methods in Deep Reinforcement Learning for Continuous Control

Several off-policy methods have also been proposed in the problem domain of deep reinforcement learning for continuous control, including the Deep Deterministic Policy Gradient Method (DDPG) [18], Actor-Critic with Experience Replay (ACER) [19], and Trust-PCL [20]. These methods are off-policy algorithms only utilize first-order information about the policy functions, and are not in the main focus of our study because no evidence has shown that any off-policy methods can significantly outperform the above-mentioned

methods in terms of training time, final performance and the stability of performance during training time.

2.2 Hierarchical Reinforcement Learning Methods

The paradigm of hierarchical reinforcement learning focuses on reducing the complexity of problems through temporal abstraction [12]. A hierarchical reinforcement learning agent invokes temporally extended activities, which are low-level closed-loop partial policies instead of primary actions. Hierarchical reinforcement learning agents have two advantages: First, it does not need to make a decision at every time step and thus the time scale is reduced. Second, it could reuse the existing temporally extended activities and thus the learning efficiency might be improved if the temporally extended activities are useful.

The study of hierarchical reinforcement learning has a long history of around 20 years [21]. Many methods have been proposed, and some of the representative works include option framework [21], hierarchies of abstract machines [22], and MAXQ value function decomposition [23].

All the hierarchical reinforcement learning method needs to solve 2 problems. The first problem is to define and learn the low-level partial policies. Second is to learn the selection and scheduling of the low-level partial policies, including the selection of the partial policies and the scheduling of these decision points.

The first problem has been studied from different point of views. The ideal problem formulation is that the agent should directly discover the useful low-level policies. However, the solution for general cases in this formulation infeasible although some methods have been proposed [24], [25] for some special cases.

The most practical formulation is to manually define the low-level partial policies as well as the scheduling criteria of them. However, it heavily relies on domain-specific engineering, and manual defining a set of predefined low-level partial policies with their scheduling criteria may not be possible. Another popular formulation is to propose a set of auxiliary MDPs to define and train the low-level policies. The partial-policies will also be terminated if the current auxiliary MDP has terminated in the target environment. However, this formulation is also limited to some special cases which are mainly goal-directed discrete environments.

The formulation of this thesis is the low-level policies could be trained in a set of given source tasks. But the problem of selecting and scheduling them is left to be solved. This formulation can handle many general cases in the sense that the target problem can be solved as long as it can be decomposed into a series of source tasks. The formulation may not be feasible for some complex problems like the board game GO, but a vast majority of real-world problems could be decomposed by a series of simple primary tasks. For example, the task of navigating a robot to a target location can be decomposed into the tasks of moving forward, turning left/right, going downstairs/upstairs, bypassing an obstacle and opening/closing a door. The task of assembling a hardware product can be decomposed into combining two specific components, plugging in the wires, installing the screws and etc.

The solution of learning to select among low-level policies, has been thoroughly studied by Semi-Markov Decision Process theory since the work of [21]. However, a general solution to the scheduling of the partial policies is still missing.

The following sections, will discuss the related works of hierarchical reinforcement learning methods in details.

2.2.1 Option Framework

The option framework [21] is one of the earliest studies in hierarchical reinforcement learning. The author uses the notion of *option* to define the partial policies. A complete definition of option consists of an input set I, a partial policy π, a termination condition β. The input set I is a set of states where the option is permitted to be executed. The partial policy π defines the policy to be executed during the activation period of the option. The termination condition β is a mapping from the current state and the current action to a scalar value that controls when the option should be terminated.

An option is Markov if its policy is Markov. Semi-Markov options, on the other hand, are options whose policies not only take the current state as input, but also are based on the state history since initiation of the current option. A policy over options μ selects option o in state s with probability $\mu(s, o)$. The concept of multi-step model is proposed as a generalization of single-step models. For any option o, let $E(o, s, t)$ denote the event of the option o initialized in state s at time t. The return of the multi-step model is

defined as:

$$R(s,o) = E\{r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{t+\tau} r_{t+\tau} | E(o,s,t\}$$

where $t+\tau$ is the termination time of o. The state transition model for the option is:

$$P(s'|s,o) = \sum_{\tau=1}^{\infty} p(s',\tau)\gamma^{\tau}$$

for all states $s' \in S$, where $p(s',\tau)$ denotes the probability that the option terminates after τ steps and results in the state s'.

A generalized form of the Bellman optimality equation is then proposed:

$$V_O^*(s) = \max_{o \in O_s} \left[R(s,o) + \sum_{s'} P(s'|s,o) V_O^*(s') \right] \tag{2.6}$$

The option framework proposes a formulation of hierarchical reinforcement learning and connects it to SMDP theory. The authors also propose the method for training the policy over options. However, the question on how the hierarchy of options is developed are not answered.

2.2.2 Modulated Hierarchical Controller

The study of [1] applies hierarchical reinforcement learning in deep reinforcement learning for continuous control problems. They propose a two-level hierarchical agent architecture containing a high-level (HL) policy and a low-level (LL) policy. The high-level policy produces a modulation signal which the low-level agent then takes as part of its input. The architecture is demonstrated in Figure 2.1. The hierarchical agent is characterized by a policy π with parameter θ. The policy is defined by the composition of the high-level controller F_H and low-level controller F_L. The low-level controller is a feed-forward neural network that maps the preprocessed current state $o(s)$ and the control signal from the high-level controller c to the policy output.

$$\pi(a|s) = F_L(o(s), c) \tag{2.7}$$

where the preprocessed state $o(s)$ removes task-specific information from the current state. The high-level controller is a recurrent neural network: $F_H = (f_z, f_c)$ that produces the new control signal at every K time step:

$$z_t = f_z(s_t, z_{t-1}) \tag{2.8}$$

$$c_t = f_c(z_{t_r}) \tag{2.9}$$

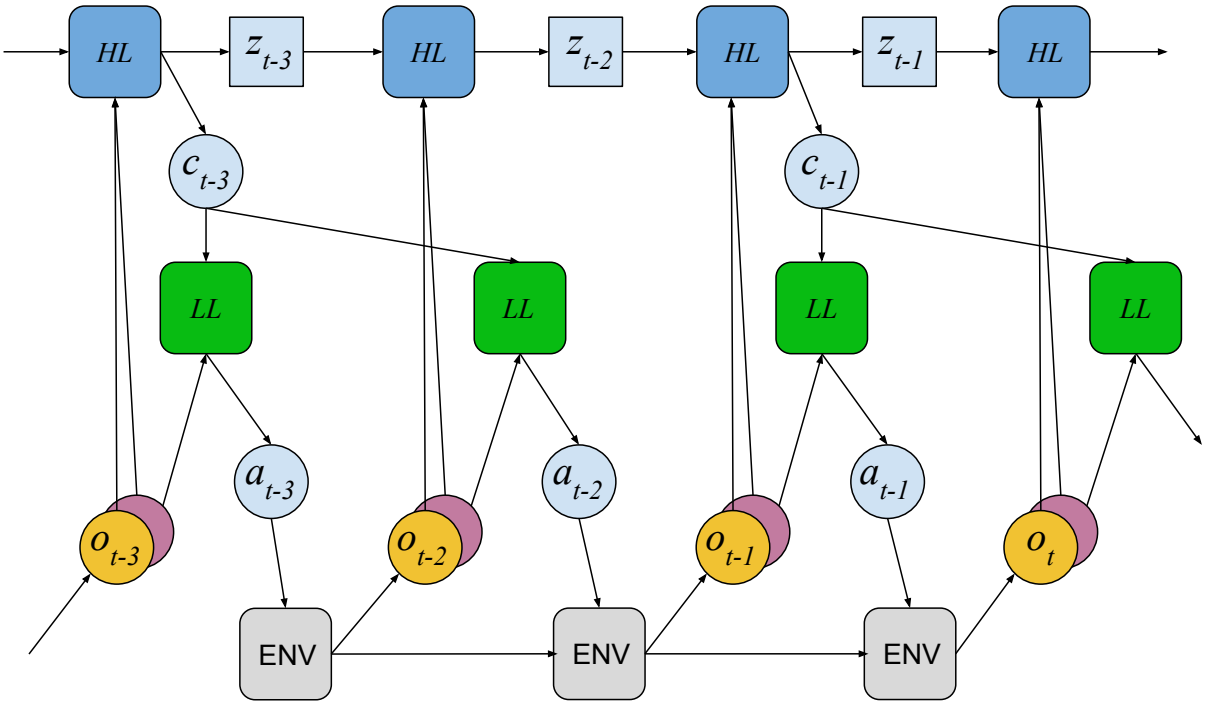


Figure 2.1: The hierarchical structure of the modulated controller [1], which consists of the recurrent high-level controller (HL, dark blue) and feedforward low-level controller (LL, green). The high-level controller has access to all observations (yellow and red). While both controllers observe sensory input at the world-clock frequency, the modulatory control signal from the high-level is updated every K steps (here K = 2)

where t_r is the most recent update output of the control signal. A predefined Gaussian noise component is added to the output c_t during the training of the high-level controller, so that the agent has a better performance in exploration. The training of the hierarchical agent consists of two separate phases: pre-training and transfer. The tasks used for pre-training are relatively simple tasks that facilitate the development of generic locomotion skills. After pre-training, the high-level controller is re-initialized and the weights of low-level controllers are frozen. The high-level controller is then trained in the transfer task, which has a sparse reward function.

This method manages to solve a few problems with sparse reward functions and low-dimensional state inputs, which could not be solved by any flat reinforcement learning methods. The limitation is that the architecture requires heavy problem-specific engineering components. The observation of the low-level agent $o(s)$ is designed manually. the hierarchical relationships between the two agents, including the definition of pre-training environments and the modulation model are also predefined. Apart from that, the low-level agent can only be executed for a predefined number (K) of time-steps before it gets

terminated. This design could only be useful in a limited class of problems.

2.2.3 Hierarchical Reinforcement Learning by Meta-learning

Another work [2], namely meta learning shared hierarchies (MLSH), proposes a meta-learning approach for learning hierarchically structured policies. The MLSH method formulates the problem as learning a finite set of MDPs P_M with the same state-action space, with a universal agent architecture. The agent consists of two sets of parameters (ϕ, θ), where ϕ is the set of task-independent parameters and θ is the set of task-specific parameters. The meta-learning objective is to optimize the expected return during the agent's entire lifetime:

$$\text{maximize}_{\phi} \mathbb{E}_{M \sim P_M}[R] \tag{2.10}$$

The detailed structure of the agent is shown in Figure 2.2. The architecture is also a two-level hierarchical reinforcement learning agent, the components $\phi_1, \phi_2, \ldots$ are the parameters of the low-level agent policies (called sub-policies) and are shared across different tasks. θ is the parameters of the high-level agent of the current task. The policy of the high-level agent, namely master policy, samples an action at fixed periods and selects the sub-policy to be executed. The agent is trained in the following manner: First a task is

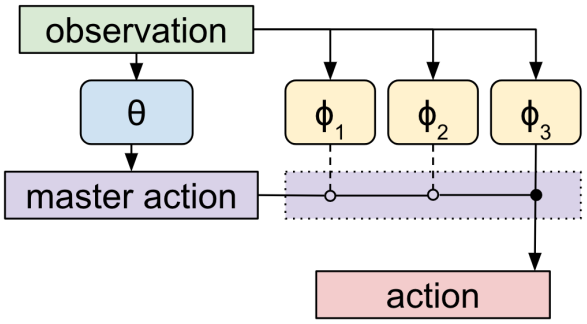


Figure 2.2: The agent setup of the modulated controller [2],θ represents the master policy, which selects a sub-policy to be active. In the diagram, ϕ_3 is the active sub-policy, and actions are taken according to its output.

sampled M is sampled and the parameter θ is re-initialized randomly. Then there is a warmup phase, where only θ is trained to be optimal. Then the agent enters the joint update phase, where both θ and ϕ are updated simultaneously. The method achieves good performance in several control environments. Including a moving-bandit Ant-robot task, an Ant robot maze navigation task and an obstacle course task.

However, the tasks are actually simplified versions of the control tasks defined in [5]. One modification is that the state of the Ant robot is periodically reset to prevent episode termination due to the falling over of the robot. The complexity of the task is thus largely reduced. It is not clear whether the method would perform well on the original version of the Ant environment. The MLSH method also relies on a warm-up period to learn the high-level agent's policy. However, the authors didn't provide a solution to learn the high-level agent's policy θ at the beginning where the parameter ϕ is also randomly initialized. Finally also predefines the temporal relationship between the high-level policy and low-level policy, and the master agent only makes decisions at pre-defined periods.

2.2.4 Goal-directed Learning Methods

Several works [26], [27] have attempted to solve sparse-reward robot control environments through the learning of auxiliary MDPs. The method of [26], namely Scheduled Auxiliary Control (SAC-X), is one of the representative works and is based on several main principles. First, the original MDP is modified, where the agent receives a vector reward signal at each timestep, consisting of the reward for the original MDP and a set of internal auxiliary rewards. Second, each internal auxiliary reward function is assigned a low-level agent policy, namely intention. Third, there is a high-level agent that selects and executes the intentions. Fourth, the learning of intentions is performed off-policy on the same experience and is performed simultaneously for different intentions. The definitions of internal auxiliary rewards are based on predefined goal states. The internal auxiliary reward with goal state g is defined as:

$$r_g(s,a) = \begin{cases} \delta_g(s), & \text{if } d(s,g) \\ 0, & \text{else} \end{cases} \tag{2.11}$$

The parameter θ of the intentions is trained with the aggregation of loss of all the reward functions:

$$\mathcal{L}(\theta) = \mathcal{L}(\theta; M) + \sum_{k=1}^{|A|} (\theta; \mathcal{A}_\mathcal{K}) \tag{2.12}$$

Where M is the MDP with the original reward function and $A = \{A_1, \ldots, A_k\}$ is the set of MDPs with internal auxiliary rewards. The training of the intention parameter is formulated as a multi-task RL problem. The high-level scheduler policy is the trained

using an off-policy policy gradient method with θ being fixed. The method manages to solve the object grasping and stacking problems in a robot-arm environment.

One limitation of SAC-X is that the method relies on predefined auxiliary MDPs. Proposing auxiliary MDPs could be solved by heuristic algorithms in for some special problems, but is infeasible for general problems. Apart from that, the method limits the definition of auxiliary MDPs to be goal-directed tasks, which is impractical for high-dimensional realistic problems.

2.2.5 Other Methods Targeting at Environments with Sparse Reward Function

Apart from the above-mentioned works, several other methods have also been proposed to solve environments with sparse reward recently, including Unicorn [28], FuN [29], HRL with Stochastic Temporal Grammar [30], policy sketch method [31] and strategic attentive writer [32]. These methods are only verified in discrete environments and no evidence has shown that they are applicable to our target problem domains.

Chapter 3

Methodology

3.1 Target Environments

3.1.1 Summary of Environment Design

We propose and develop a set of tasks that are representatives of deep reinforcement learning environments with continuous action space, multi-modal state space and sparse reward functions. The design of the robot physical systems is based on the 3D robot environments [33] in OpenAI Gym [16]. The external environments and reward functions of the target tasks are different from the original environments in OpenAI Gym. There are three classes of 3D robot agents in the environments OpenAI Gym: Ant, Humanoid and Swimmer. The Ant agent is a quadruped robot with 8-dimensional action space, the Humanoid agent is a humanoid robot with 16-dimensional action space, and the Swimmer agent is a worm-like robot with 2-dimensional action space. The three agents are shown in Figure 3.1, Figure 3.2 and Figure 3.3.

We mainly focus on different tasks based on the Ant agent, because the agent has the capability of solving a rich variety of high-level tasks, while the learning of agent's basic control skills is not too challenging. Compared to the Ant agent, the Swimmer agent is too simple and its capability of solving hierarchical tasks is limited. The Humanoid agent, on the other hand, has a much more unstable physical system compared to Ant, and poses too much difficulty on its basic locomotion skills.

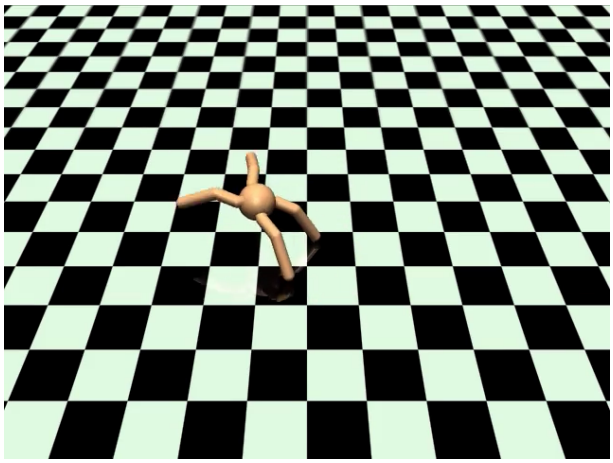


Figure 3.1: The Ant agent

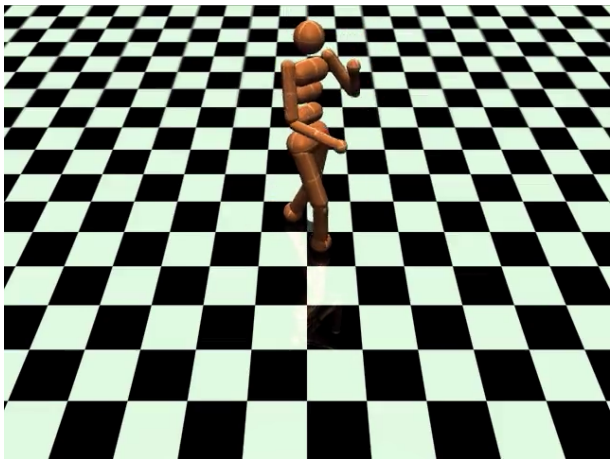


Figure 3.2: The Humanoid agent

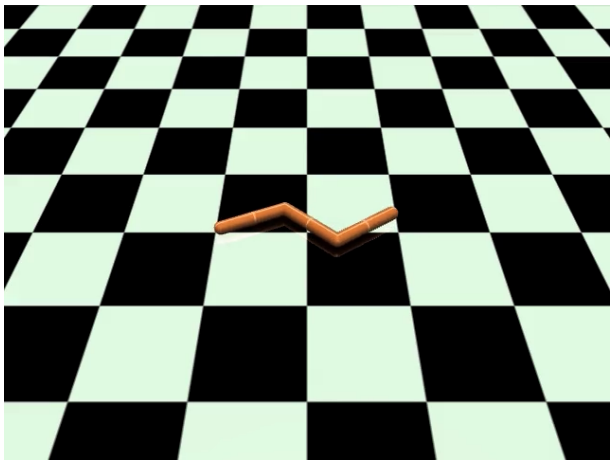


Figure 3.3: The Swimmer agent

3.1.2 Detailed Environment Specifications

We provide a detailed description of the experiment environments in this section. All of the environments are based on the Ant task [16].

In the original Ant task, the agent receives a 111-dimensional motion sensor input and produces an 8-dimensional action output. The agent's state input consists of a 13-dimensional vector that contains the robot's pose information, a 14-dimensional vector that represents its velocity information, and an 84-dimensional vector that contains contact force information. However, information about the agent's absolute position in the world map is not available.

For the proposed environments, the agent receives not only the 111-dimensional motion sensor input, but also a $64 \times 64 \times 1$ dimensional grayscale image observation. A sample image observation is shown in Figure 3.1.2. The image is not in a high resolution but is sufficient for the agent to observe the necessary information for the proposed tasks.

Figure 3.4: A sample image observation of the target environments

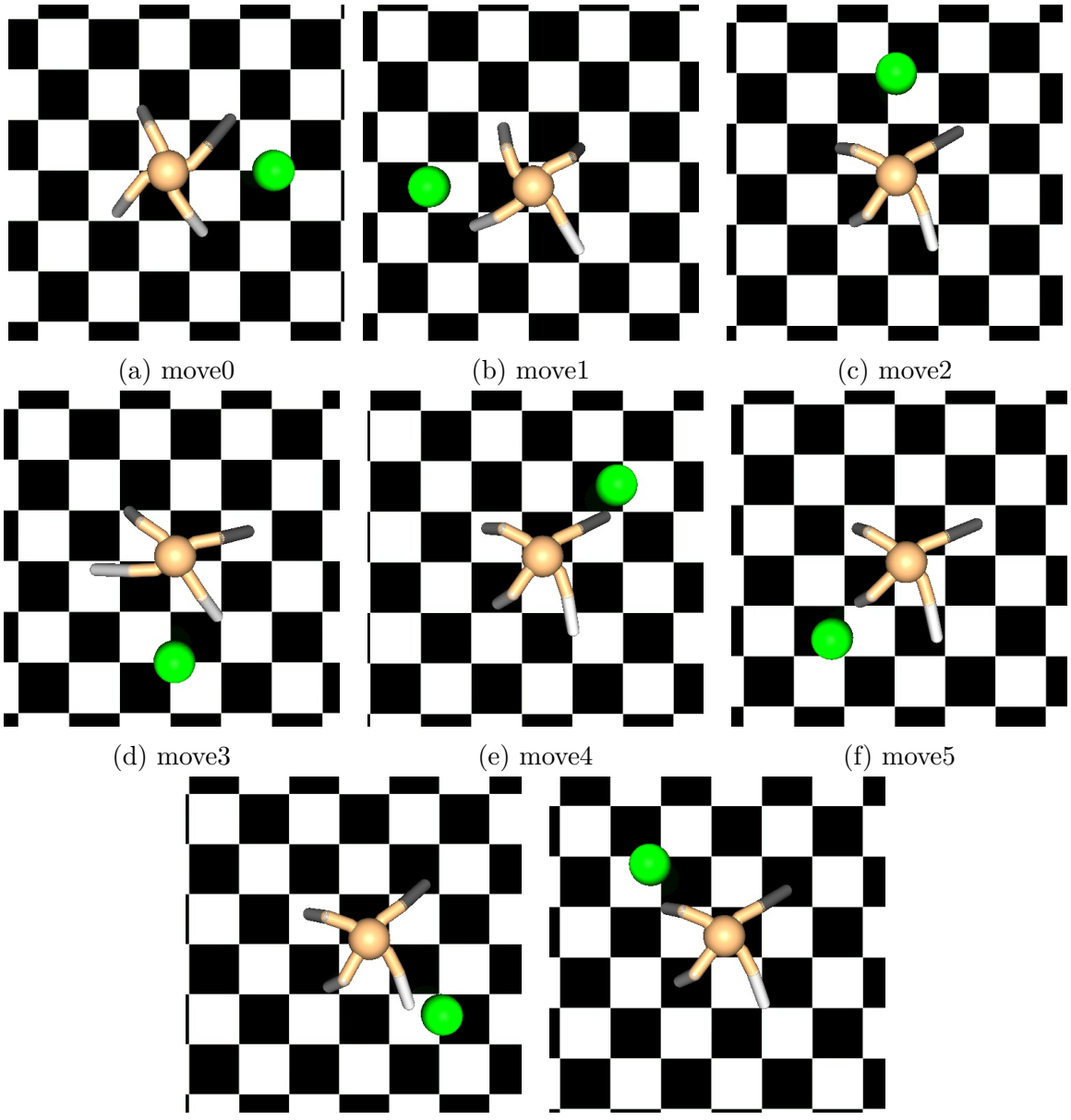
A basic task, namely move0, is similar to the Ant environment in OpenAI Gym [16] except that the state space has an extra image observation. The agent is required to move toward a specific direction $g_0 = (1, 0)$, and the reward at each time-step is given by:

$$r = v_g + 1 - c_p - c_c \tag{3.1}$$

where $v_g = v \cdot g_0$ is the forward reward, which rewards the agent for moving toward the target direction. A sphere object presents in the environments at a constant distance from the robot agent to represent the target direction. c_p is the control cost, which is the power that the agent is consuming, and c_c is the contact cost, which penalizes the agent for collisions. The episode terminates when the agent enters the unrecoverable state of being upside-down, or if the episode length reaches 1000 time-steps.

Apart from move0, a set of similar tasks with different target directions are denoted as move1, move2, move3, ..., move7. These tasks are the same as move0 except that the

goal direction is different. The image observation is actually redundant for all these low-level tasks, because the agent only needs to move toward one specific direction in their corresponding environments. Snapshots of these source tasks are shown in Figure 3.5.



(a) move0 (b) move1 (c) move2

(d) move3 (e) move4 (f) move5

(g) move6 (h) move7

Figure 3.5: The source tasks

Apart from these simple task, we also propose several tasks that are more complex. We propose several multi-modality environments such as moveg2, movecont, dynamicg8. These tasks require the agent to learn not only from the state representation but also the image representation. We also propose sparse multi-modality environments, such as reachcont. The agent receives sparse reward signals in these environments compared with the previous environments. The target direction or location is represented by a sphere

object and can be seen in the image observation. The set of all the proposed environments are described in details in table 3.1.2.

3.2 Wasserstein Actor Critic Kronecker-factored Trust Region Policy Optimization Method

As is reported in [34], the result of the current state-of-art deep reinforcement learning methods for continuous control, including TRPO, ACKTR, PPO and DDPG are difficult to be reproduced, due to they're heavily influenced by a variety of factors including random seed, neural network architecture, activation functions of neural network layers and even software implementation. This fact has reduced the reliability of these methods, and introduces difficulty in comparing their performance.

Particularly, the reproducibility problem with trust region methods including TRPO and ACKTR could be due to the training gets stuck in local minimum states. These trust region methods have a naturally decreasing learning rate due to the use of KL-divergence measure.

We propose a new policy optimization method, namely Wasserstein Actor Critic Kronecker-factored Trust Region Policy Optimization (W-KTR), in order to achieve a better performance on the proposed tasks, in terms of the agent's final performance, total training time and reproducibility.

The proposed W-KTR method focus on the problem scope of deep reinforcement learning for continuous control, specifically robot control problems. The physical meaning of action of these environments usually stands for mechanics-related physical quantity, such as motor torque and target motor phase. However, the traditional KL-divergence based trust region algorithms may not be suitable for these problems. Because the KL-divergence only focuses on the intersection of policy distributions and may not be a good measure of the deviation of policy. A small perturbation in the mean value of a policy with Gaussian distribution from 1.0 to 1.01 will lead to a large KL-divergence when the variance has a small value such as 0.003. However, the perturbation may not lead to a significant difference in reality.

Therefore, we propose to use the Wasserstein metric, to measure the deviation of the policy updates. The Wasserstein metric as a trust region criteria has also been studied in

Task name	Goal	Reward	Description
move0	velocity: $g_0 = (1, 0)$	$v_g + 1 - c_p - c_c$	move toward a target direction
move1	velocity: $g_1 = (-1, 0)$	$v_g + 1 - c_p - c_c$	move toward a target direction
move2	velocity: $g_2 = (0, 1)$	$v_g + 1 - c_p - c_c$	move toward a target direction
move3	velocity: $g_3 = (0, -1)$	$v_g + 1 - c_p - c_c$	move toward a target direction
move4	velocity: $g_4 = (\sqrt{2}/2, \sqrt{2}/2)$	$v_g + 1 - c_p - c_c$	move toward a target direction
move5	velocity: $g_5 = (-\sqrt{2}/2, -\sqrt{2}/2)$	$v_g + 1 - c_p - c_c$	move toward a target direction
move6	velocity: $g_6 = (\sqrt{2}/2, -\sqrt{2}/2)$	$v_g + 1 - c_p - c_c$	move toward a target direction
move7	velocity: $g_7 = (-\sqrt{2}/2, \sqrt{2}/2)$	$v_g + 1 - c_p - c_c$	move toward a target direction
moveg2	velocity samples from: $\{g_0, g_1\}$	$v_g + 1 - c_p - c_c$	each episode has a random sampled target direction
movecont	velocity samples from the unit circle	$v_g + 1 - c_p - c_c$	each episode has a random sampled target direction
dynamicg8	velocity samples from: $\{g_0, g_1, \ldots, g_7\}$	v_g	the target direction is re-sampled with probability 0.005 at each time-step
reachcont	position samples from the unit circle	$I(\|x - g\|_2^2 < 0.5) - 0.01$	the episode is terminated when reaching a target position
reachcontreg	position samples from the unit circle	$5I(\|x - g\|_2^2 < 0.5) - 0.01$	a new target is sampled and the episode continuous once the target position is reached
constdirreachreg	target direction samples from: $\{g_0, g_1, \ldots, g_7\}$	$5I(x_g > 1) - 0.01$	a reward is given once the agent has moved toward the target direction for a constant distance, then a new target is sampled

Table 3.1: Summary of Ant-based environments

previous literatures [35]. We reformulate the trust region policy optimization problem as follows:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & J(\theta) \\ \text{subject to} \quad & \overline{W_2}(\pi_{\theta_{old}}, \pi_\theta) \leq \delta_W \end{aligned} \tag{3.2}$$

where $\overline{W_2}(\pi_{\theta_{old}}, \pi_\theta)$ is the average Wasserstein-2 metric between the old policy and the current policy.

The Wasserstein-2 metric is defined by the following equation [36]:

$$W_2(P, Q) = \inf_{\Gamma \in \mathbb{P}(X \sim P, Y \sim Q)} \left[\mathbb{E}_{X,Y \sim \Gamma} \|X - Y\|_2^2 \right]^{1/2} \tag{3.3}$$

where $\mathbb{P}(X \sim P, Y \sim Q)$ is the set of all joint distribution of X, Y with marginals P, Q respectively.

Specifically, when P and Q are Gaussian distributions with mean m_1, m_2 and covariance matrix Σ_1 and Σ_2, the squared value of Wasserstein-2 distance $W_2^2(.)$ is defined as the following equation [37] :

$$W_2^2(P, Q) = \|m_1 - m_2\|_2^2 + \text{Tr}\left(\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}\right) \tag{3.4}$$

The problem is solved by performing gradient updates iteratively following the natural gradient with the Riemannian metric as W_2 metric:

$$s = H_\theta \left(\overline{W_2^2}(\pi_{old}, \pi) \right)^{-1} g := A^{-1} g \tag{3.5}$$

If the non-diagonal entries in covariance matrices are all zero, the metric $W_2^2(.)$ is simply a second order term over the means and the square root of the covariance matrices:

$$W_2^2(P, Q) = \|m_1 - m_2\|_2^2 + \|\Sigma_1^{1/2} - \Sigma_2^{1/2}\|_2^2 \tag{3.6}$$

The metric can actually be represented as a squared error between two parameter vectors:

$$W_2^2(P, Q) = \|\text{vec}[m_1, diag(\Sigma_1^{1/2})] - \text{vec}[m_2, diag(\Sigma_2^{1/2})]\|_2^2 \tag{3.7}$$

The solution of this kind of expressions can then be computed using the Kronecker-factor approximation technique [9].

The policy is then updated following the gradient direction s with ADAM stochastic optimization algorithm [38]. The step size is adjusted in an adaptive manner according to the resulting W_2 distance at each training step.

3.3 Efficient Exploration Through Exceptional Advantage Regularization

One of the challenges with multi-modality robot environments is to simultaneously learn features from different sources of state inputs.

The state spaces of the studied problems have two modalities: the locomotion sensor signal and the image observation. The complexity in learning the features from the two inputs is significantly different. Because the learning of motion sensor signal features are trivial while the learning of image features takes much more time. The agent is would get stuck at a local minimum after it has learned the features of the locomotion state vectors but still haven't learned anything from the image input.

As far as we know, the distribution type of the continuous stochastic policy in reinforcement learning is usually chosen to be a normal distribution with diagonal covariance matrices in the previous studies of on-policy policy gradient methods. We have observed a phenomenon in the training process of policy gradient methods that, the variance of policy is extremely unlikely to be increased, even when the agent has just escaped from a local minimum. This phenomenon has lead to in-efficiency in exploration. Therefore, a regulation method is necessary to encourage the agent to perform exploration without hurting the reinforcement learning performance.

A conventional regularization method to encourage exploration is entropy regularization:

$$g' = g + \beta_{ent} \nabla_\theta \mathbb{E}[H(\pi_\theta(a|s))] \tag{3.8}$$

where β_{ent} is the weight controlling the penalty on low entropies. The entropy of a normal distribution $\mathcal{N}(\mu, \Sigma)$ is defined as:

$$H(\pi) = \frac{1}{2} \ln \det(2\pi e \Sigma) \tag{3.9}$$

If the policy distribution is a normal distribution with diagonal covariance matrix, the entropy regularization basically introduces a constant bias to the gradient of the logarithm of variance parameters. As a result, the entropy regularization method is usually to tune

in order to efficiently encourage exploration without significant degradation on learning performance.

We propose a novel method that can efficiently encourage exploration. We add a term, namely Exceptional Advantage Regularization loss (EAR), to the original gradient of the variance parameters:

$$g'_{\Sigma} = g_{\Sigma} + \beta_{ex}\mathbb{E}_{\hat{A}^{GAE}(s)>0}\left[\hat{A}^{GAE}(s)\max\left(0, \nabla_{\Sigma}\log\pi(a|s)\right)\right] \tag{3.10}$$

where β_{ex} is the weight controlling the bias on exploration.

The EAR loss adds a bias for the positive gradients of the variance parameters of the samples with positive advantage values. By introducing the EAR loss, more importance is added to the samples low likelihood which produce exceptionally positive advantage values. As a result, the positive experiences that are rarely encountered would be paid more attention. The problem remaining is to identify the samples with low likelihood. We simply take the positive gradients of the variance parameters with respect to the log-likelihood here to filter out the samples whose likelihoods are too high.

3.4 Efficient Exploration Through Robust Concentric Gaussian Mixture Policy

Apart from the EAR method for exploration, we propose to use an alternative probability distribution type, namely Robust Concentric Gaussian Mixture, instead of a diagonal Gaussian policy used in previous works.

The proposed policy distribution, namely Robust Concentric Gaussian Mixture (RCGM) Policy, is a mixture of two Gaussian distributions:

$$\pi(a|s) = (1-\alpha_{ex})\mathcal{N}(\mu, \Sigma) + \alpha_{ex}\mathcal{N}(\mu, q_{ex}\Sigma) \tag{3.11}$$

where the constant α_{ex} is the weight of second distribution. The weight α_{ex} can take a value in the range of $(0, 1)$, such as 0.05, and the constant $q_{ex} \in (1,)$, for example can be 5, controls the variance of the second deviation. The larger q_{ex} is, the higher the likelihood will be in the tails of the distribution.

The KL-divergence of two RCGM policy does not have a closed form expression even for this special case. However, an empirical estimation of the KL-divergence can still be

computed based on the training samples. Apart from that, the Wasserstein-2 distance between two RCGM policy can be given by:

$$W_2^2(\pi_0(a|\mu_0,\Sigma_0),\pi_1(a|\mu_1,\Sigma_1) = \\ (1-\alpha_{ex})W_2^2\Big(\mathcal{N}(\mu_0,\Sigma_0),\mathcal{N}(\mu_1,\Sigma_1)\Big) + \alpha_{ex}W_2^2\Big(\mathcal{N}(\mu_0,q_{ex}\Sigma_0),\mathcal{N}(\mu_1,q_{ex}\Sigma_1)\Big) \tag{3.12}$$

Compared to the commonly used Diagonal Gaussian distribution, the RCGM distribution could be much more robust since it has longer tails. Therefore, it is less likely that the agent gets stuck at sub-optimal policies.

3.5 Hierarchical Reinforcement Learning method

3.5.1 Hierarchical Reinforcement Learning Architecture

We propose a two-level hierarchical model to solve the target problems. The hierarchical model consists of a top-level decider agent and a set of bottom-level actuator agents. The actuator agents' policies is a mapping from the state to the primary action space, and are trained in the source-task environments. The actuator policies are fixed during the training of the target task.

The decider agent takes an action at every time-step. It may either decide which actuator-policy should be executed, or continue using the current actuator policy that is decided earlier. Therefore, assume there are n_a sub-policies, the action space of the decider agent is actually an $(n_a + 1)$-discrete action space. The overall decision-making process of the decider agent is shown in Algorithm 1.

The observation space of the decider agent consists of 2 parts, the original multi-modal state and the meta state. The meta state contains information about the current sub-policy being executed and the number of time-steps since the last decision has been made.

The decider agent is parameterized by two policy networks, θ_s and θ_d. The network θ_s, namely switcher policy, outputs a scalar value that indicates whether the agent should simply continue using the current acting actuator policy, or terminate the current actuator policy and make the decision on the selection of actuator policy again. The network θ_d, namely the decider policy, outputs an n_a-discrete action space. The output of the network represents the decision on the selection of the acting actuator policy.

Algorithm 1 The decider agent mechanism

function DECIDERACT(self,s_t)
  $a_{decider} \sim \pi_{decider}(s_t)$
  if $a_{decider} \neq 0$ **then**
    $self.currentActuator \leftarrow self.allActuators[a_{decider} - 1]$
  $a_{actuator} \leftarrow self.currentActuator.act(s_t)$
  return $a_{actuator}$

3.5.2 Generalized advantage estimation for the decider agent

We propose a generalized advantage estimation method for the decider agent of the proposed hierarchical reinforcement learning model.

Assume that a decider agent makes decisions at time $t_1, t_2, \ldots$, then the execution length of the corresponding actuator policies are $l_i = t_{i+1} - t_i, i = 1, 2, \ldots$.

The definition of reward of the decider action at t_i is given by:

$$\bar{r}_{t_i} := \sum_{l=0}^{t_{i+1}-t_i-1} \gamma^l r_{t_i+l} \tag{3.13}$$

Define the TD residual $\delta_{t_i}^V$ for $i = 0, 1, \ldots$ by:

$$\delta_{t_i}^V := \bar{r}_{t_i} + \gamma^{t_{i+1}-t_i} V(s_{t_{i+1}}) - V(s_{t_i}) \tag{3.14}$$

Then the k-step advantage estimation is given by:

$$\hat{A}_{t_i}^{(1)} := \delta_{t_i}^V = \bar{r}_{t_i} + \gamma^{t_{i+1}-t_i} V(s_{t_{i+1}}) - V(s_{t_i}) \tag{3.15}$$

$$\hat{A}_{t_i}^{(2)} := \delta_{t_i}^V + \gamma^{t_{i+1}-t_i}\delta_{t_{i+1}}^V = \bar{r}_{t_i} + \gamma^{t_{i+1}-t_i}\bar{r}_{t_{i+1}} + \gamma^{t_{i+2}-t_i} V(t_{i+2}) - V(s_{t_i}) \tag{3.16}$$

$$\hat{A}_t^{(3)} := \delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3}) - V(s_t) \tag{3.17}$$

$$\begin{aligned} \hat{A}_{t_i}^{(k)} &:= \sum_{d=0}^{k-1} \gamma^{t_{i+d}-t_i}\delta_{t_{i+d}}^V \\ &= \bar{r}_{t_i} + \gamma^{t_{i+1}-t_i}\bar{r}_{t_{i+1}} + \cdots + \gamma^{t_{i+k-1}-t_i}\bar{r}_{t_{i+k-1}} + \gamma^{t_{i+k}-t_i} V(s_{t_{i+k}}) - V(s_t) \end{aligned} \tag{3.18}$$

We can define the unnormalized generalized advantage estimator as a exponentially-weighted sum of these k-step advantage estimators [13]:

$$\begin{aligned}
\hat{A}_{t_i}^{GAE_{unnorm}(\lambda)} &:= \hat{A}_{t_i}^{(1)} + \lambda^{t_{i+1}-t_i}\hat{A}_{t_i}^{(2)} + \lambda^{t_{i+2}-t_i}\hat{A}_{t_i}^{(3)} + \cdots + \lambda^{t_{i+k-1}-t_i}\hat{A}_{t_i}^{(k)} \\
&= \delta_{t_i}^V + \lambda^{t_{i+1}-t_i}(\delta_{t_i}^V + \gamma^{t_{i+1}-t_i}\delta_{t_{i+1}}^V) \qquad (3.19) \\
&\quad + \lambda^{t_{i+2}-t_i}(\delta_t^V + \gamma^{t_{i+1}-t_i}\delta_{t_{i+1}}^V + \gamma^{t_{i+2}-t_i}\delta_{t_{i+2}}^V) + \dots \\
&\quad + \lambda^{t_{i+k-1}-t_i}\sum_{d=0}^{k-1}\gamma^{t_{i+d}-t_i}\delta_{t_{i+d}}^V \qquad (3.20) \\
&= (\delta_{t_i}^V \sum_{b=0}^{k-1}\lambda^{t_{i+k-1}-t_i} + \gamma^{t_{i+1}-t_i}\delta_{t_{i+1}}^V \sum_{b=1}^{k}\lambda^{t_{i+b}-t_i} \\
&\quad + \gamma^{t_{i+2}-t_i}\delta_{t_{i+2}}^V \sum_{b=2}^{k}\lambda^{t_{i+b}-t_i} + \dots \qquad (3.21) \\
&\quad + \gamma^{t_{i+k-1}-t_i}\delta_{t_{i+k-1}}^V \lambda^{t_{i+k-1}-t_i}) \\
&= \sum_{d=0}^{k-1}\delta_{t_i}^V\gamma^{t_{i+d}-t_i}\sum_{b=0}^{k-1}\lambda^{t_{i+b}-t_i} \qquad (3.22)
\end{aligned}$$

The normalized generalized advantage estimator is then given by:

$$\hat{A}_{t_i}^{GAE(\lambda)} = \frac{\hat{A}_{t_i}^{GAE_{unorm}(\lambda)}}{\sum_{b=0}^{k-1}\lambda^{t_{i+b}-t_i}} \qquad (3.23)$$

The unormalized GAE estimator is usually used in practice instead of the normalized one, with a postprocessing step of batch normalization to adjust the scale and bias of the advantages.

This technique is an adaption of the generalized advantage estimation method [13] in the context of hierarchical reinforcement learning, and originates from the idea of eligibility trace [39]. The goal of the proposed GAE estimator for hierarchical reinforcement learning is to provide an effective variance reduction scheme for the corresponding optimization procedures.

3.5.3 Training of the Switcher Policy

We train the switcher agent using a flat reinforcement learning method with respect to the return of the target task. However, an extra loss term is added so that the switcher network will be encouraged to produce termination signals less frequently.

$$J'(\theta_s) = J'(\theta_s) + \mathbb{E}[\pi_s(s)] \qquad (3.24)$$

Where π_s is the switcher policy distribution. The second term is the basically expected frequency that the switcher network produces the positive output.

3.6 Training of the Actuator Policies

This section discusses the related techniques in training the actuator policies in the proposed hierarchical reinforcement learning formulation.

3.6.1 Effective Training of Actuator Policies

Directly applying the actuator policies trained in the source tasks to the target task is not likely to work. Because the initial state distribution could have a significant difference. For example, consider the target task as dynamicg4 and the set of source tasks as move0, move1, move2, move3. When the goal direction in dynamicg4 changes from $(-1, 0)$ to $(1, 0)$ the correct decider policy should be switching from move1 to move0. However, the actuator policy of move0 could fail to perform well, because the initial state is sampled from the states generated by move1 policy. At such an initial state, the agent might be moving toward the direction $(-1, 0)$ in a high velocity and was not encountered during the training of move0.

A straight-forward method to solve this problem is to perform joint training of actuator policy in the target task with a uniform random decision policy and fixed-length switcher policy. However, this method increases the training time linearly with respect to the number of source tasks.

There is a study [40] that proposes a technique to improve the robustness of policies. The method tries to artificially add variety to the environments, so that the trained policy will be able to deal with a larger variety of conditions. However, the method doesn't work in our case. Because there is no simple way to increase the variety of the initial states in our problems without sampling degenerate states.

3.6.2 Domain Randomization by Cross-sampling Initial States

We propose a novel method for training robust actuator agents, namely domain randomization by cross-sampling initial states. We train all the source tasks simultaneously. During the training, each actuator agent keeps an experience buffer that stores its state history. At the beginning of each episode, the actuator environment randomly samples a state from the combination of its initial state distribution and the experience buffer of other tasks. This will guarantee that each actuator agents will encounter states generated

by other agents. As a result, the proposed method is more efficient than the direct joint training method.

3.6.3 Synchronous Scheduling of Actuator Learning

One of the problems related to the joint training of multiple tasks is 'strategy collapse' [7]. This phenomenon refers to the status where the agents converge to sub-optimal policies due to their policy collectively form a sub-optimal equilibrium. We propose a technique, namely synchronous scheduling of actuator learning , to solve this problem, which actively controls the training schedule of actuator agents. The training of an actuator agent is paused if it largely surpasses the lowest performance, until all the other agents have reached the agent's performance level. This method requires a universal definition of performance level for all the source tasks, which is challenging when the source tasks are irrelevant and the episode rewards are in different scales. The method is suitable for our experiment settings, where the source tasks here are basically the same type of tasks with symmetrical specifications. Another solution could be increasing the time span of the experience buffer, so that an agent will be less influenced by the current status of other agents.

Chapter 4

Experiments

4.1 Experiments on the Basic Task *move0*

The methods for solving the basic task move0 are discussed in this section.

First, we would like to verify the ability to reproducing consistent learning performance discussed in [34]. We choose the ACKTR [9] method and test it on the move0 task. As is discussed previously, the move0 task is actually a single-modal environment since the image observation is redundant. Separate neural network parameters are used for the actor network and critic network. The neural networks have the same architecture that consists of two fully connected layers with 64 hidden units after the motion sensor input. The neural network outputs the mean parameter of the policy distribution. The standard deviation of the policy distribution is parameterized by an independent parameter vector. The batch-size is set to 8000, the KL-divergence set to 0.0001 and 20 parallel agents are used to generate experience. The resulting learning performance is shown in Figure 4.1. It appears that the performance of one experiment gets stuck at a score below 3000 while another continues to improve after reaching the score of 4000.

We'd also like to compare the learning performance of the ACKTR method across different learning rates. The experiment result showing the performance of the original ACKTR algorithm with different learning rates is shown in Figure 4.2. The result shows that the agents are likely to get stuck at the score of around 3000.

We'd also like to verify if the proposed W-KTR method is less prone to local minimums on the task move0. The experiment results comparing the performance of W-KTR agents with different learning rates is shown in Figure 4.3. The result shows that none of the W-KTR agents gets stuck at the local minimum around 3000, and their final performance

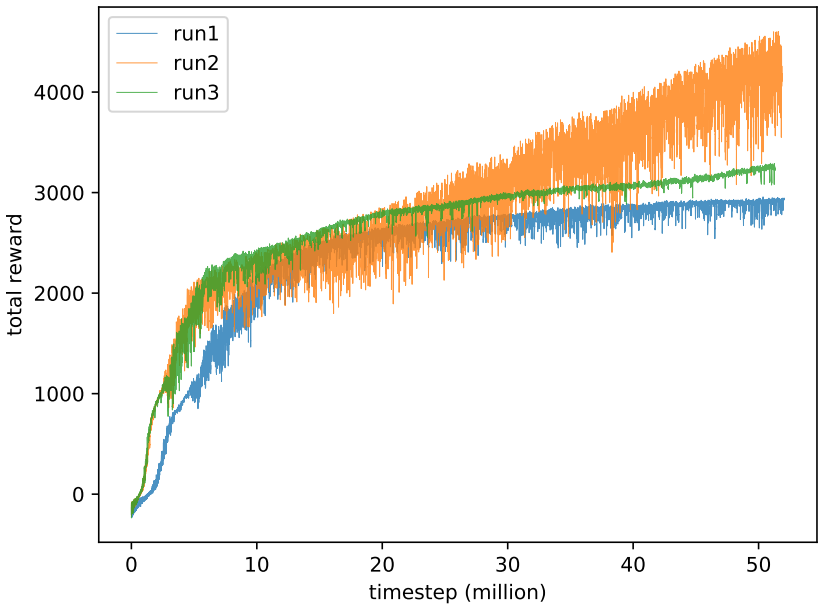


Figure 4.1: Inconsistent performance produced by ACKTR agents with the same parameters but different runs. The vertical axis is the total return averaged over the recent 20 episodes and the horizontal axis is the number of million time-steps

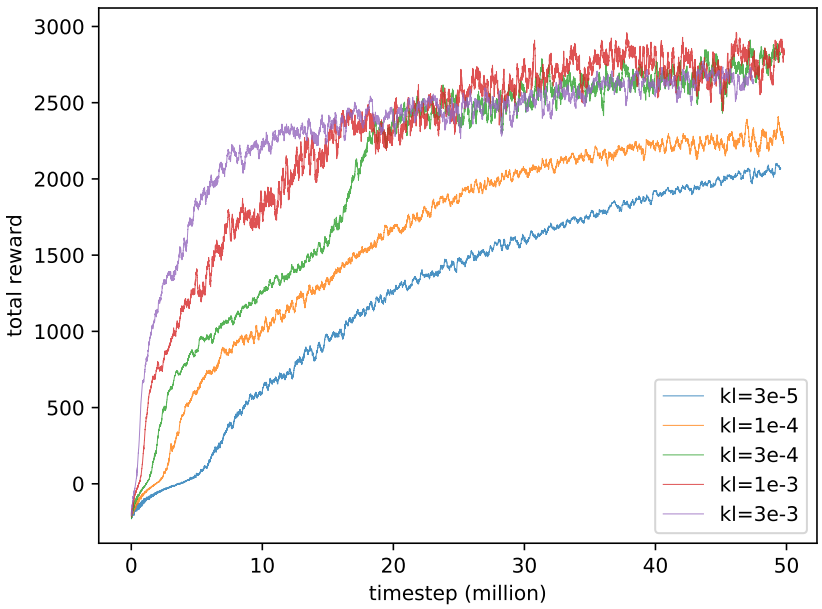


Figure 4.2: Performance of ACKTR agents with different KL-divergence constraints. All the agents are trained with batch-size 4000 and 20 parallel agents. The vertical axis is the total return averaged over the recent 200 episodes and the horizontal axis is the number of million time-steps

can reach a total return of around 6000. However, the W-KTR agent appears to have a much slower rate of improvement on the performance before 50 million time-steps.

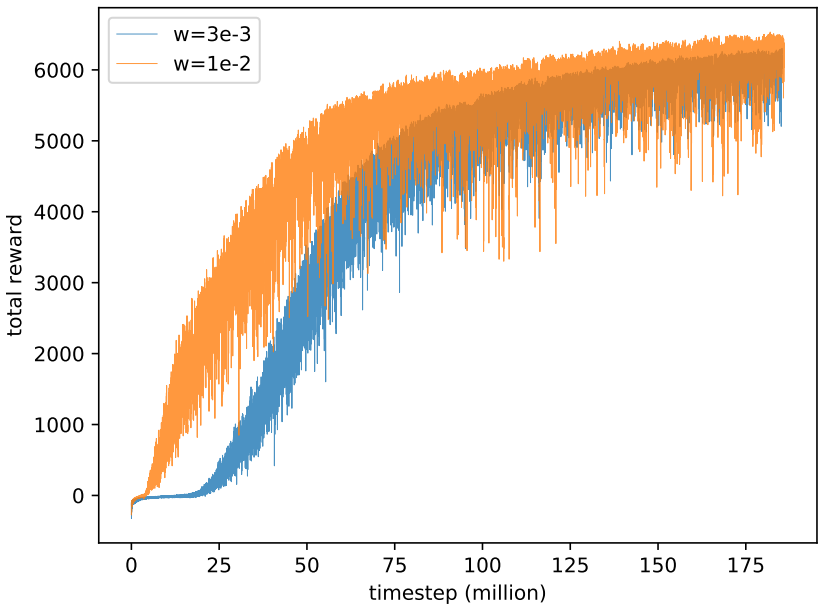


Figure 4.3: Performance of W-KTR agents with different W2-metric constraints. All the agents are trained with batch-size 4000 and 20 parallel agents. The vertical axis is the total return averaged over the recent 20 episodes and the horizontal axis is the number of million time-steps

We also test if applying a decaying Wasserstein constraint at the early phase of training will improve the learning performance. The experiment of a W-KTR agent with a decaying W-2 constraint from 0.02 to 0.00003 in the first 15 million time-steps is shown in Figure 4.4. The agent can achieve a more efficient learning performance in the first 50 million time-steps. However, the drawback of this method is that more hyper-parameters need to be tuned.

The experiment results show that the proposed W-KTR algorithm is able to achieve the same level of final performance as the contemporary state-of-art methods. The major drawback is that the algorithm does not perform well in training time. The decaying learning rate feature of KL-divergence based trust region algorithms is able to achieve a proper performance improvement rate in the early phase of training.

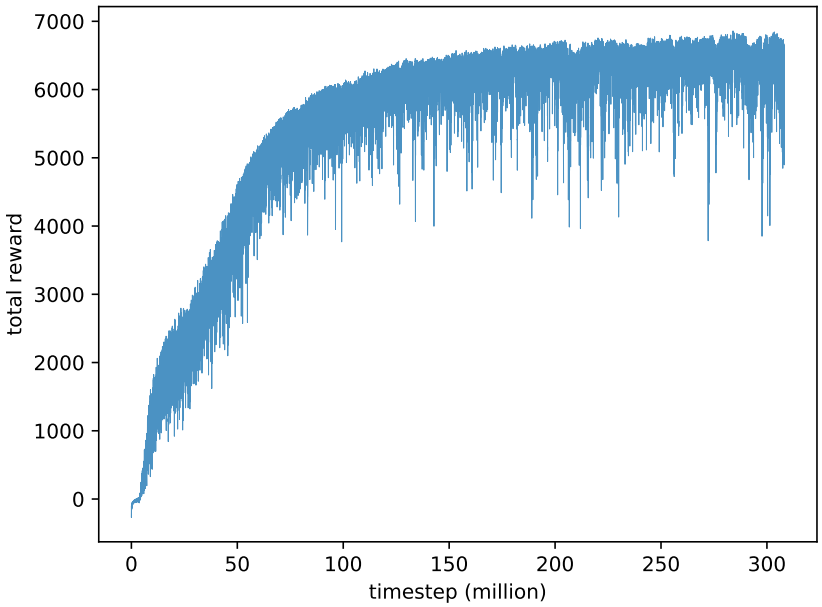


Figure 4.4: Performance of a W-KTR agent with a decaying W2-metric from 0.02 to 0.00003 in the first 15 million time-steps. The agent is trained with batch-size 4000 and 20 parallel agents. The vertical axis is the total return averaged over the recent 20 episodes and the horizontal axis is the number of million time-steps

4.2 Experiment on Flat Reinforcement Learning Solutions to Multi-modality Tasks

4.2.1 Conventional Flat reinforcement Learning Methods

Contemporary end-to-end flat reinforcement learning methods may not be good at solving tasks with multi-modal state space.

Take the task movecont as an example. In this task, a goal direction as sampled uniformly at random from the continuous range of angles: $[0, 2\pi)$, at the beginning of each episode. The agent can only obtain the information about the target direction from the image, where there is a sphere object at the corresponding position. It is easy for the agent to learn to balance itself and stay at a stable pose based on the current motion sensor readings, in order to reduce the control cost and contact cost, as well as preventing the game over condition. However, the agent needs to extract the location of the sphere object in the image and move toward the object to solve the task. Learning the image features takes much more data than learning from the motion sensor readings, and the agent may fail to learn anything meaningful from the image while focusing on the motion sensor features. Even if the agent finally manages to extract image features, the

policy might have already at a low-entropy distribution, and would not be able to make the necessary explorations for the optimal solution. Therefore, an efficient exploration technique is needed for this kind of tasks with multi-modal state space.

We would first like to examine the effectiveness of the conventional entropy regularization method for encouraging exploration of reinforcement learning agents with continuous action space. We use a multi-modal ACKTR agent. The agent uses separate networks for actor and critic function approximation with the same architecture. The image input gets feed into 3 convolutional layers with filter size 3 by 3 and stride 2, with 16, 8, 4 filters respectively. The output features of the convolutional network are passed to a fully-connected layer of 16 units. The output 16-unit fully connected layer is then concatenated with the motion sensor inputs. Finally, the concatenated feature layer is feed into 2 fully connected layers with 64 units and output the mean parameter of the policy.

The experiment result on the entropy regularization technique is shown in Figure 4.5. The agents are trained using the ACKTR algorithm minibatch size 2560 and KL-divergence constraint 0.0003. The samples are produced by 32 parallel agents.The result shows that the agent easily fails to learn the reinforcement learning task when the weight of entropy term is a little bit too large.

The average Standard Deviation of the policy distribution during training is visualized in Figure 4.6. The figure shows that the entropy regularization term might too much bias on the agent to perform exploration instead of improving its performance.

4.2.2 Experiment on Exceptional Advantage Regularization

The effectiveness of exceptional advantage regularization method on multi-modality tasks is discussed in this section. First we demonstrate the general patterns of the distribution on the advantage values on the simple multi-modality task moveg2. The task samples a target direction from a set of 2 opposite directions at the beginning of each episode, and the agent is rewarded for moving toward the direction.

The experiment result on the average total reward of a normal ACKTR agent without any regularization is shown in Figure 4.7. The average reward per time-step during training is shown in Figure 4.8, and the average standard deviation is shown in Figure 4.9.

The distribution of advantage values at the first epoch (epoch 0) is shown in Figure 4.10a. It can be seen that the advantage values are distributed uniformly because the

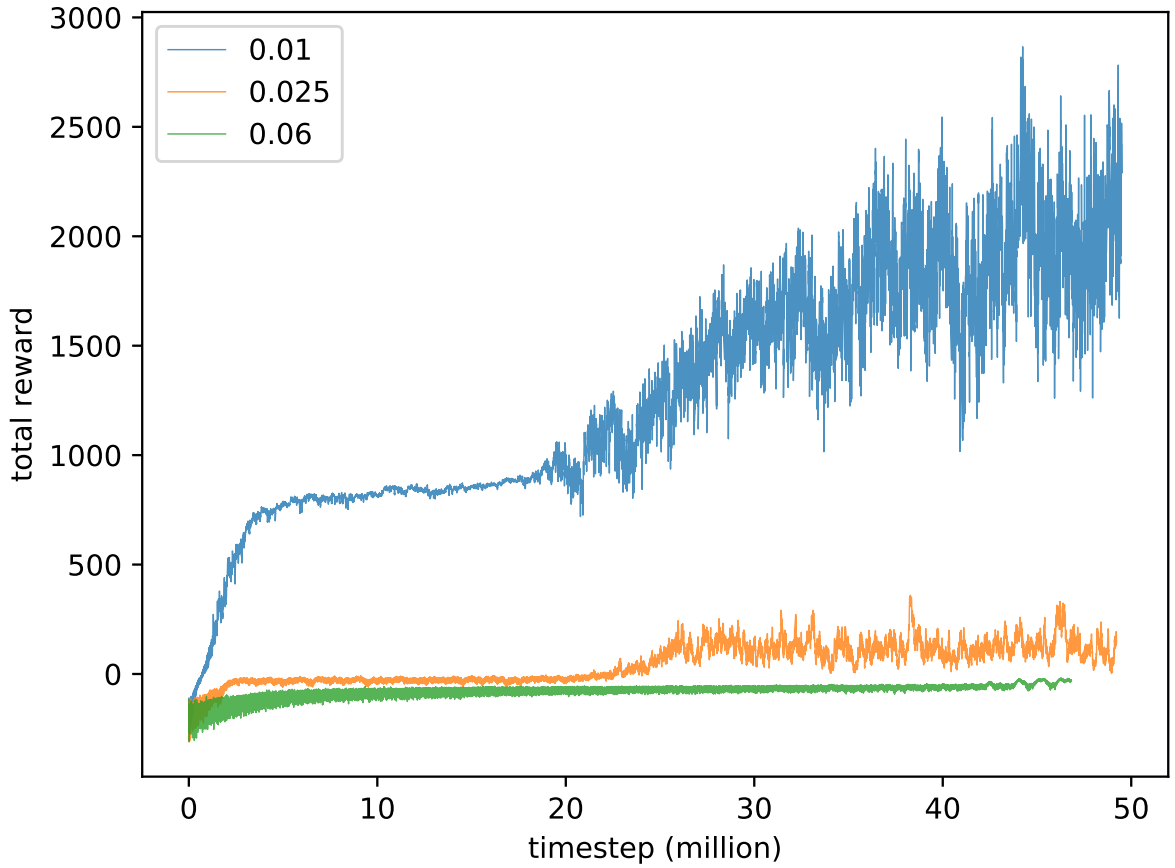


Figure 4.5: Performance of agents with different weight on entropy regularization term, the horizontal axis is the number of million timesteps and the vertical axis is the total episode reward averaged over the last 32 episodes

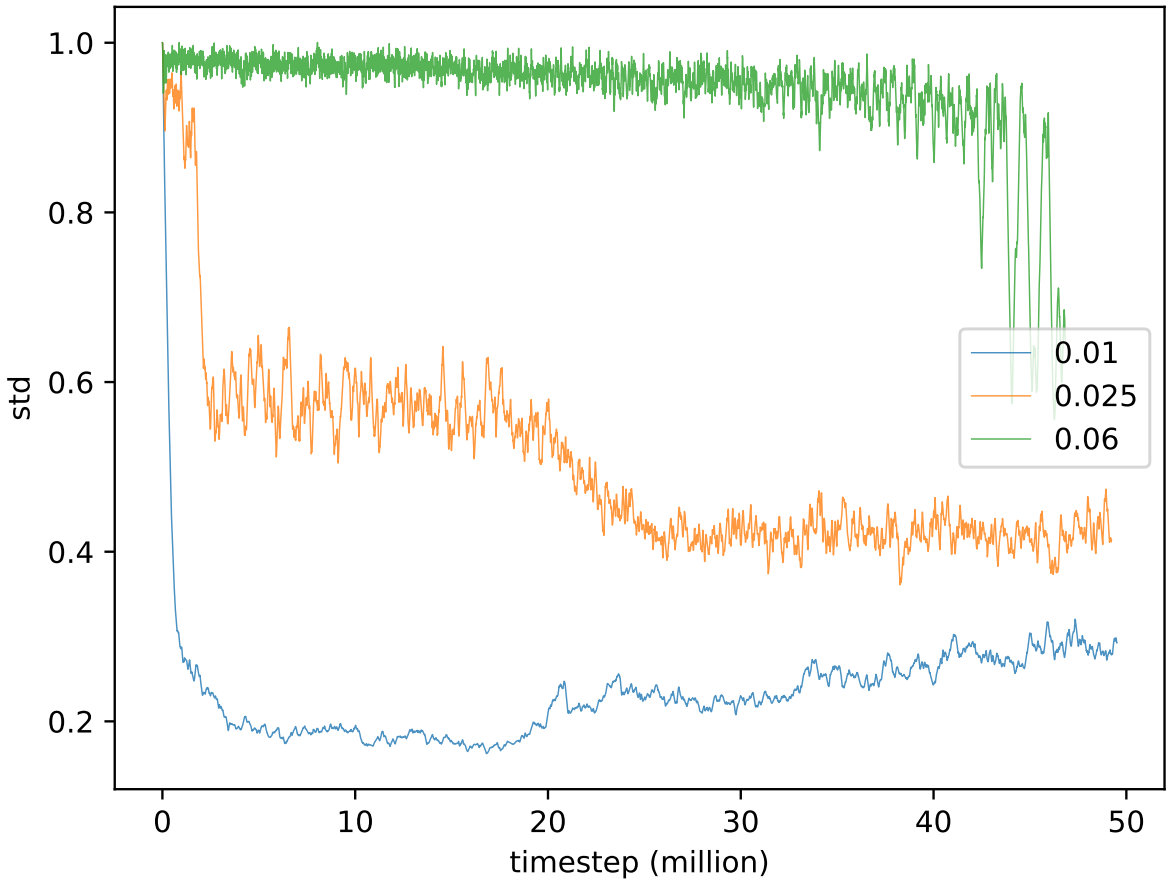


Figure 4.6: Average standard deviation of the policy of agents in Figure 4.5, the horizontal axis is the number of million timesteps.

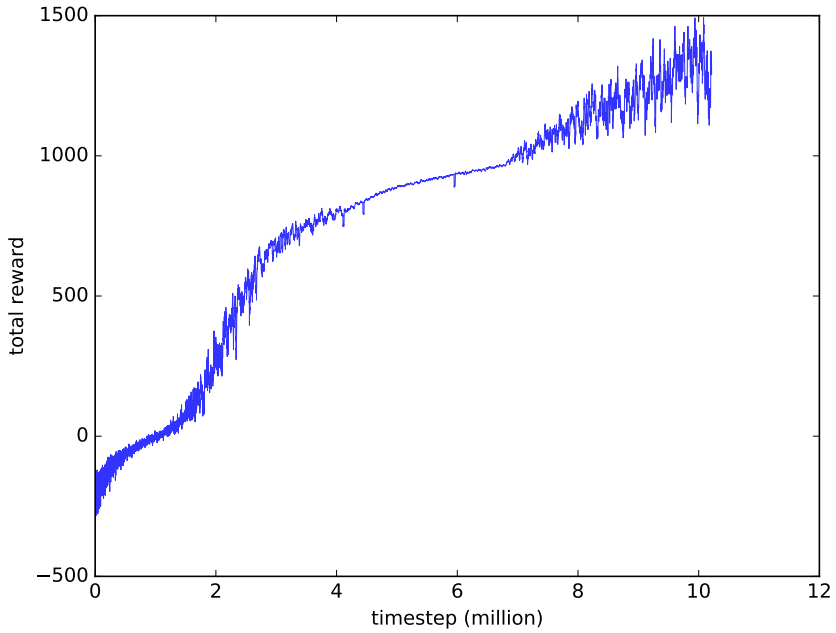


Figure 4.7: Performance of ACKTR agent on the moveg2 task, the horizontal axis is the number of million timesteps and the vertical axis is the total episode reward averaged over the last 20 episodes

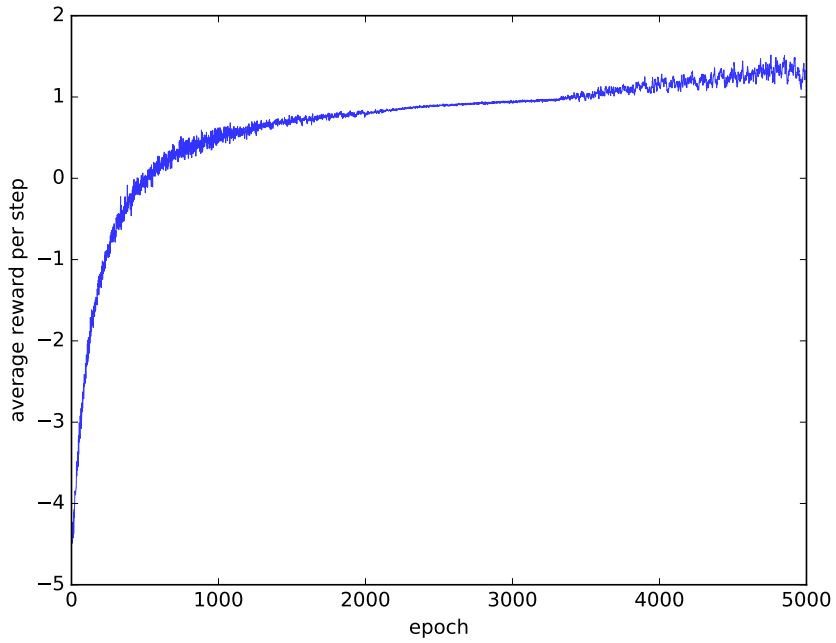


Figure 4.8: The average reward per time-step of ACKTR agent on the moveg2 task, the horizontal axis is the number of training batches

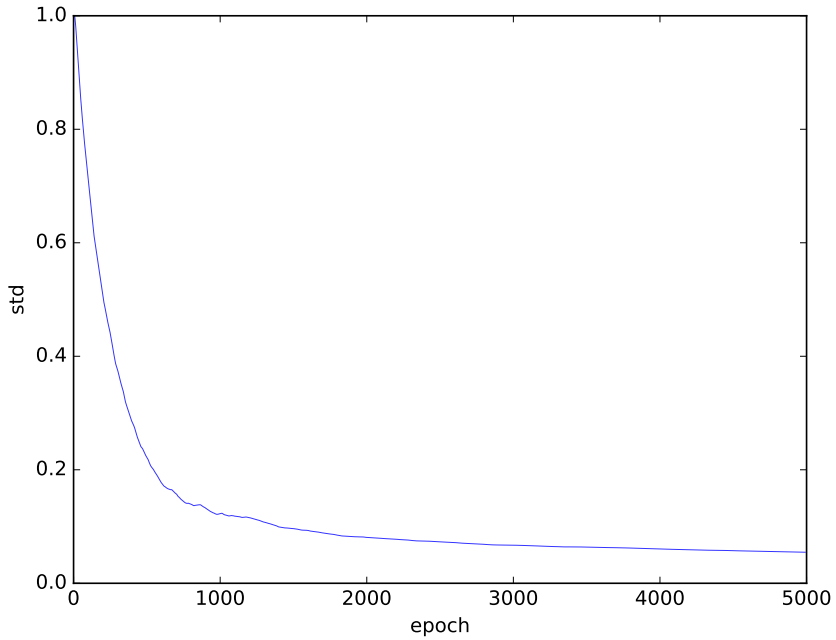


Figure 4.9: The average standard deviation of ACKTR agent's policy on the moveg2 task, the horizontal axis is the number of training batches

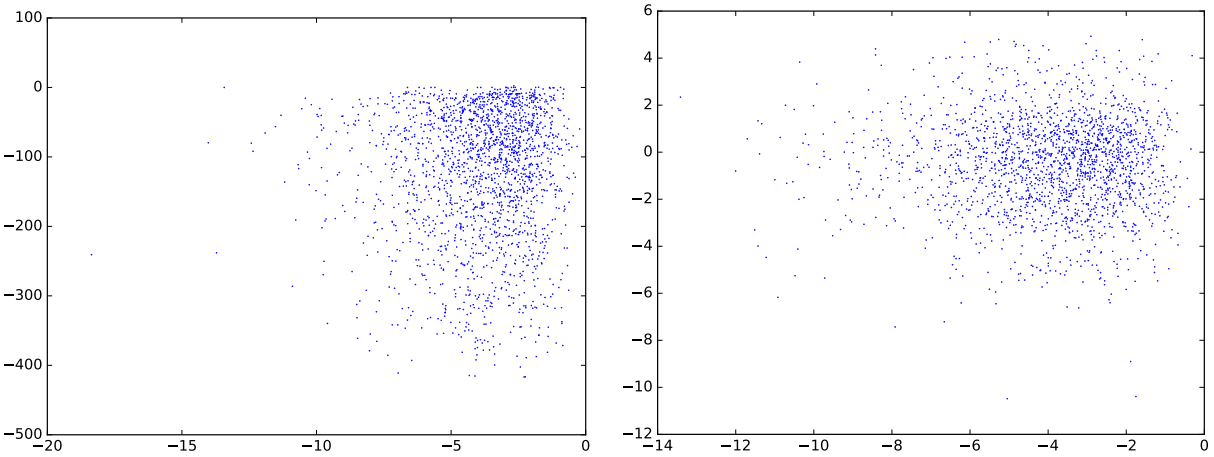
critic model is just initialized. Figure 4.10b shows the distribution at epoch 3000. It can be seen that after the critic model has been trained for some time, the marginal distribution advantage value tend to follow a normal distribution with zero mean. Figure 4.10c shows the distribution of advantage values at epoch 4900. The advantage values are likely to have a positive bias at some samples with low log-likelihood when the agent in this case, where the agent just managed to escape from a local minimum.

A large number of samples with low likelihood but positive advantage value indicates that it might be beneficial to the agent if the degree of exploration is increased. However, figure 4.9 on the change of average standard deviation shows that the agent actually keeps decreasing the standard deviations of its policy even at this case. Therefore, the application of an exceptional advantage regularization term, aims to solve this problem.

The performance of exceptional advantage regularization (EAR) technique is tested on the 'movecont' task. We keep the same neural network configuration as in section 4.2.1, with the modification that the EAR technique is used. The experiment result on the performance of an ACKTR agent with different weights exceptional advantage regularization is shown in Figure 4.11. The average standard deviation of the policies is also shown in Figure 4.12. All the agents are trained with batch-size 2560 and KL-divergence constraint 0.0003. The weight-0 agent is the same as the original ACKTR agent without any exploration regularization, and it could only achieve a total reward of around 2000 at the end of training. The agent with exceptional advantage regularization weight 0.04 can improve rapidly in the early phase before 40 million timestep, but gets stuck at around 3500. This shows that the exceptional advantage regularization method might cause an adverse effect on the final performance. The agent with exceptional advantage regularization weight 0.01 achieves the best final performance. Its average standard deviation shows that the agent manages to increase its policy entropy and re-explore the environment after it has escaped from a local minimum.

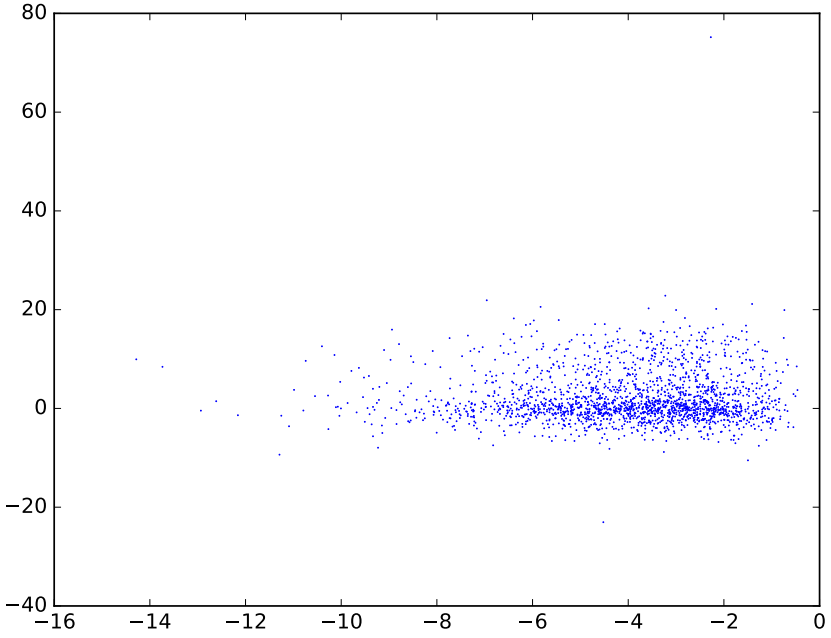
4.2.3 Experiment on Robust Concentric Gaussian Mixture Policy Model

The effectiveness of robust concentric Gaussian mixture policy agent in exploration of task movecont is tested in this section. We use the same configuration as in section 4.2.1 except that the distribution type is changed to the robust concentric Gaussian mixture



(a) Epoch 0

(b) Epoch 3000



(c) Epoch 4900

Figure 4.10: The distribution of advantage values of the ACKTR agent at epoch 0, 3000 and 4900 respectively. The horizontal axis is the log-likelihood value

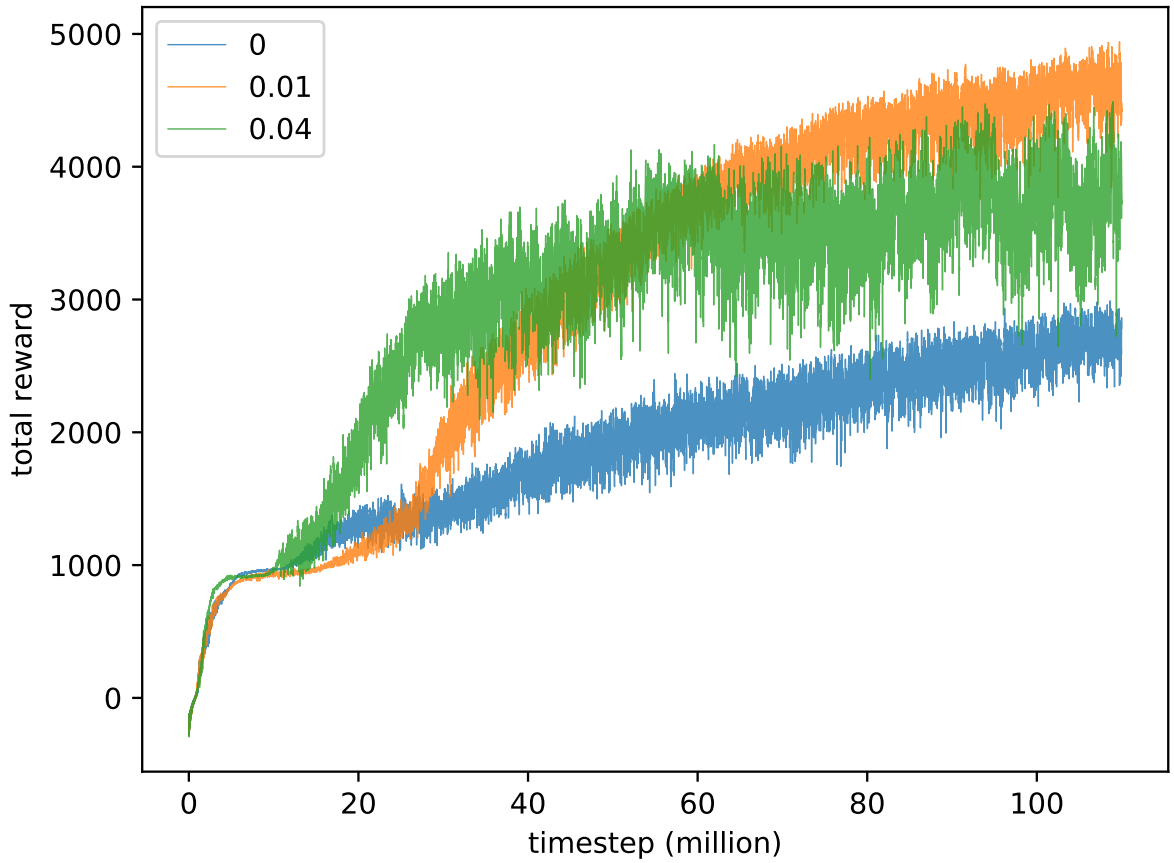


Figure 4.11: Performance of agents with different exceptional advantage regularization weights, the horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 32 episodes

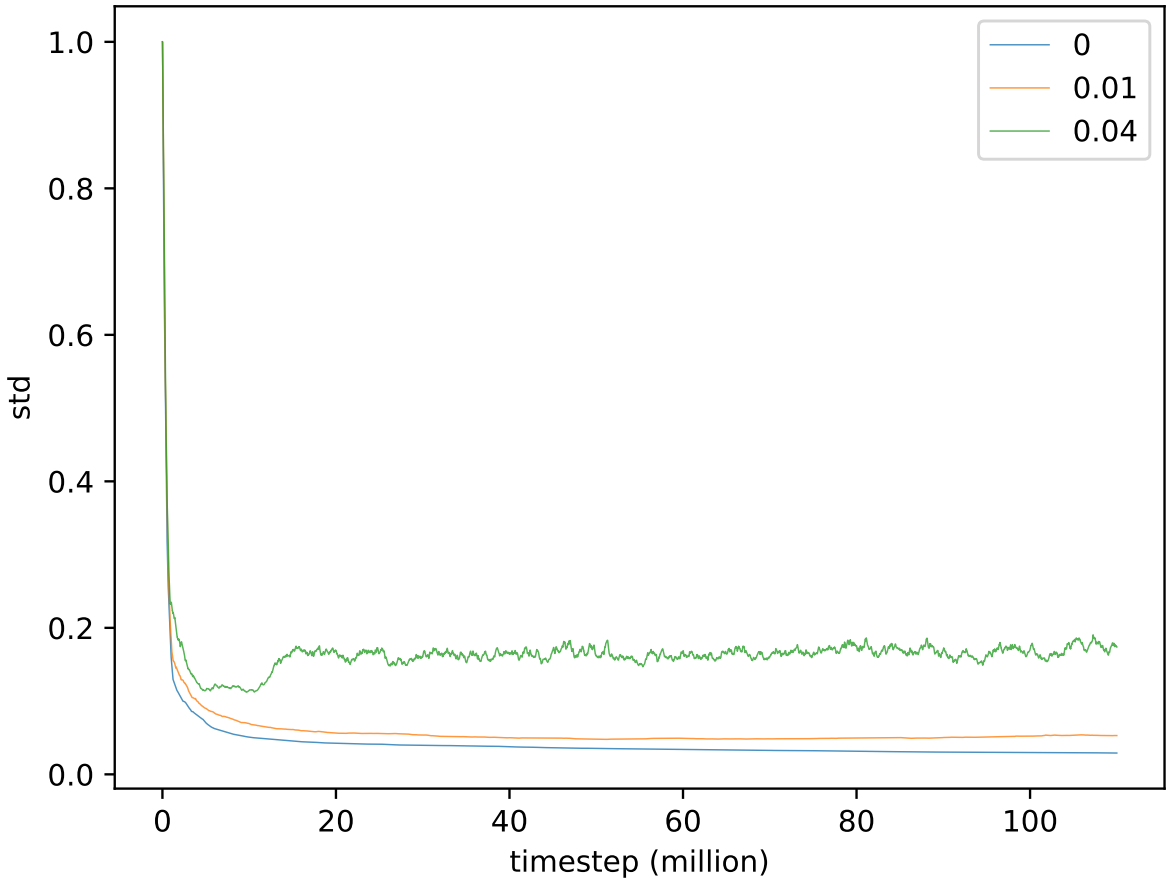


Figure 4.12: The average standard deviation of agents with different exceptional advantage regularization weights, the horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 32 episodes

policy. We use the empirical estimation of the KL-divergence since there is no closed-form solution on it.

The experiment result on the performance of an ACKTR Gaussian mixture policy agent is shown in Figure 4.13. The Gaussian mixture policy agents have slower rates on performance improvement compared to pure Gaussian policy agents. However, the agent is still able to achieve a good final performance, with a total reward of around 4000.

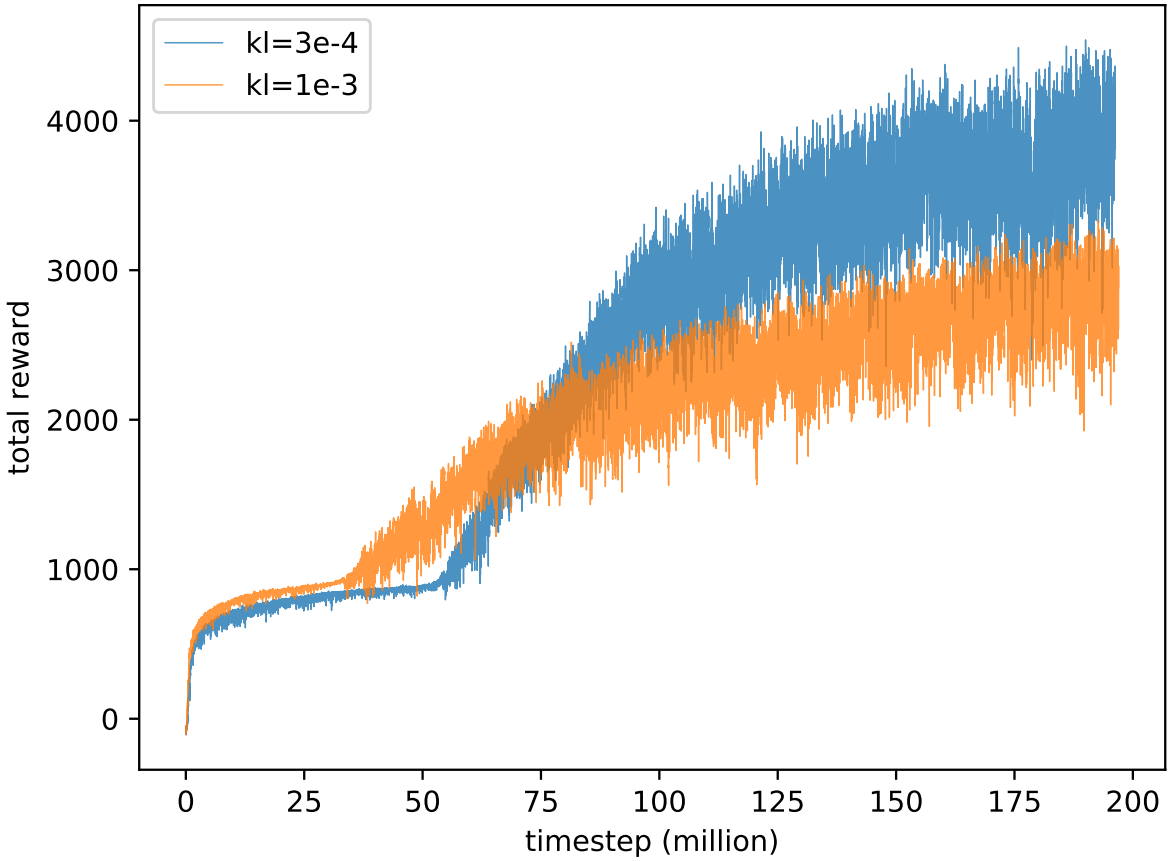


Figure 4.13: Performance of ACKTR agents with different KL-divergence constraints, the horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 32 episodes.

4.3 Experiments on Hierarchical Reinforcement Learning Methods for Multi-modality and Sparse environments

This section discusses the experiment results on the proposed hierarchical reinforcement learning model.

Experiments are done on a set of target tasks: dynamicg8 and reachcont and reachcontreg. The target task dynamicg8 is a typical problem that has multi-modality state-space and smooth reward signal, and the target task reachcont is a typical problem that has both multi-modality state-space and sparse reward signal. The set of source tasks is {move0, move1, ..., move7} for all the experiments.

The problem of hierarchical reinforcement learning consists of two parts: the training of actuator agents and the decider agent. The two problems will be discussed in separate sections.

4.3.1 Performance of flat reinforcement learning solutions

We would firstly like to examine the performance of contemporary reinforcement learning methods on the two typical tasks: dynamicg8 and reachcont. We use the same parameter settings as in section 4.2.2. We also test whether exceptional advantage regularization is helpful in these two tasks.

The result on the task dynamicg8 is shown in Figure 4.14, and the result on the task reachcont is shown in Figure 4.15. It can be seen that agents fail to solve the tasks. When the task logic becomes more complex and reward function become more sparse, the difficulty of learning useful image features increases, and the task becomes infeasible for contemporary flat reinforcement learning algorithms.

4.3.2 Training the Actuator Agents with Domain Randomization by Cross-sampling Initial States

As is discussed previously, the learning of the actuator agent of a source task from the source task set is not always the same problem as training a flat reinforcement learning agent for that single task. The initial states generated by actuator agents for other source

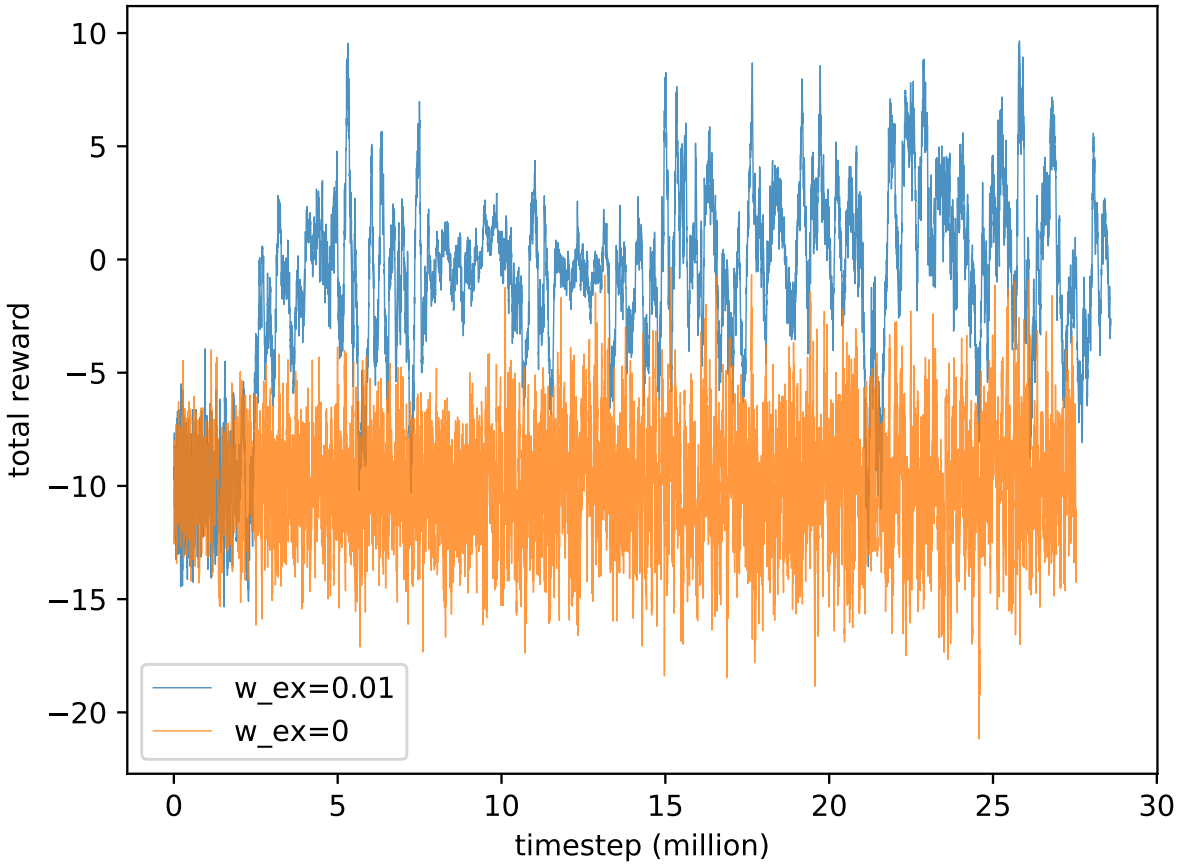


Figure 4.14: Performance of flat reinforcement learning method on the task dynamicg8, with different weights on exceptional advantage regularization. The horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 200 episodes

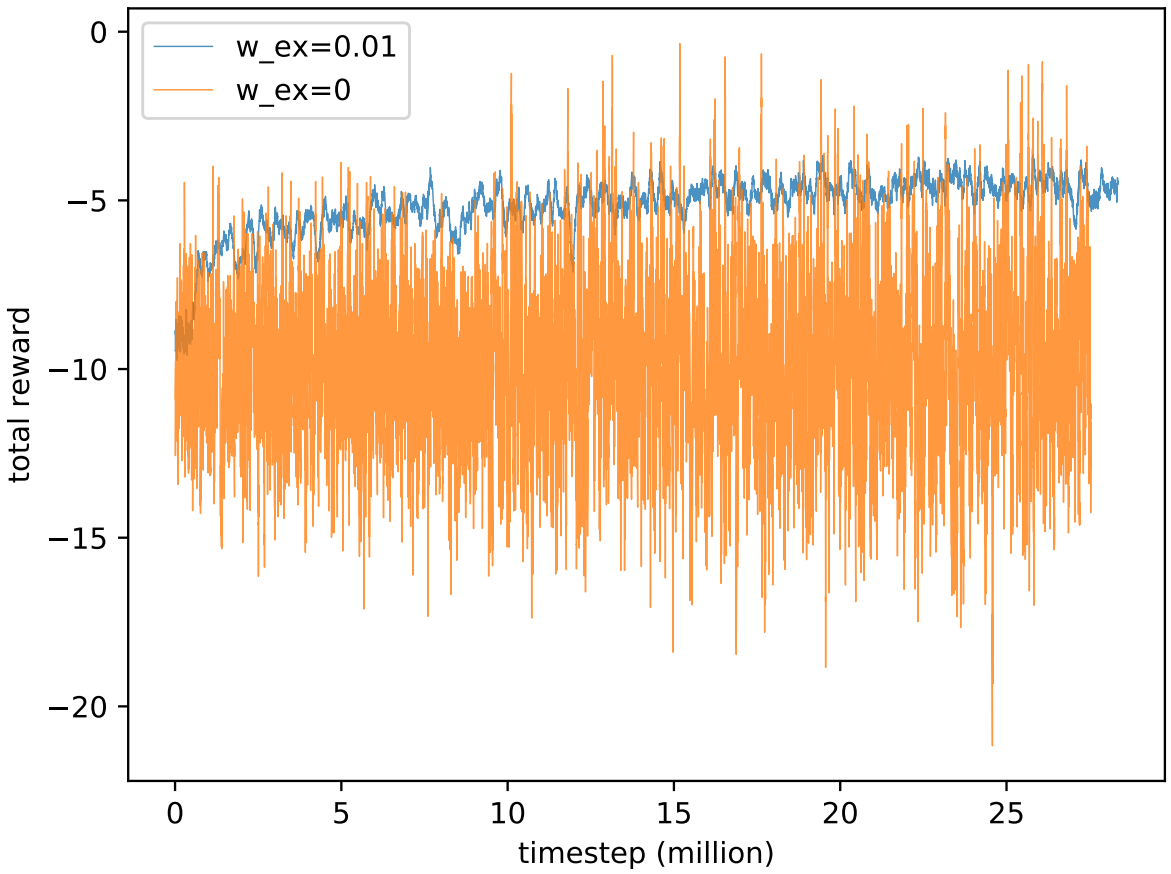


Figure 4.15: Performance of flat reinforcement learning method on the task reachcont, with different weights on exceptional advantage regularization. The horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 200 episodes

tasks must be handled, but is usually not encountered in the original source task.

We proposed that the domain randomization by cross-sampling initial states method can handle the problem of novel initial states. The performance of this method is tested in this section. We use the same configuration as in section 4.1 for each actuator policy.

The experiment result on the performance of all the actuator agents is plotted in Figure 4.16. It can be seen that the actuator agents get stuck at sub-optimal performance levels.

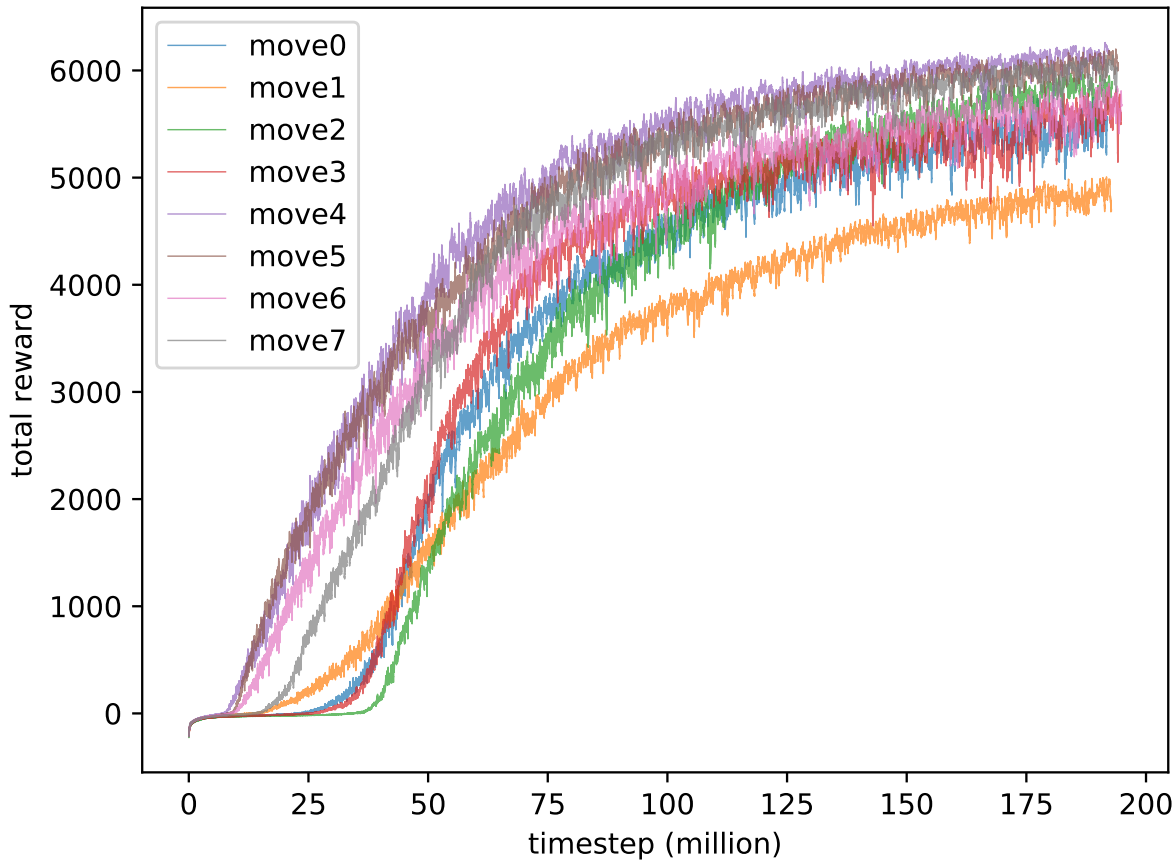


Figure 4.16: Performance of actuator agents with domain randomization by cross-sampling initial states, the horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 200 episodes

Therefore, the proposed synchronous scheduling of actuator learning method aims to prevent this problem. The experiment result of the technique is shown in Figure 4.17. In this experiment, the policy training of the actuator agents who outperforms the global lowest-performance by 1000 is paused until all other actuator agents outperform this agent. The result shows that all the actuator agents are able to reach a final performance of around 6000, although the performance of different agents diverge initially. This has

verified that the proposed method can successfully prevent any actuator agents getting stuck at sub-optimal performance levels.

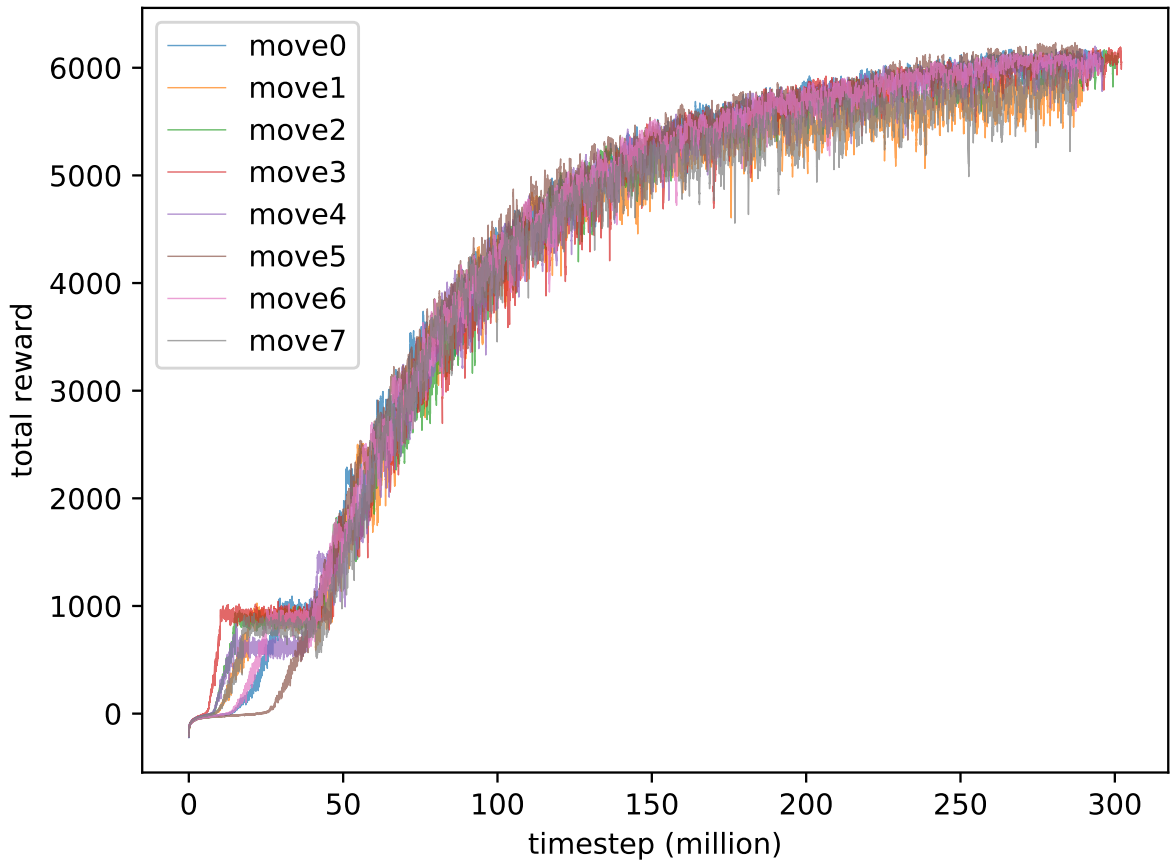


Figure 4.17: Performance of actuator agents using synchronous scheduling of actuator learning , the horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 200 episodes

Therefore the learning of source tasks is successfully solved by the proposed technique of domain randomization by cross-sampling initial states with synchronous scheduling of actuator learning method.

4.3.3 Training the Decider Agent

The training of the decider agent consists of two phases: the training of the decision policy and the switcher policy. We train them in two separate phases. In the first phase, the switcher policy is initialized so that it outputs a termination signal whenever the current decision policy has been executed for more than l_c time steps, which is set to 10 in our experiments. The decision policy is trained with the switcher policy being fixed. After the performance of the decision policy converges, the switcher policy is then trained with

the decision policy fixed.

Phase 1: Decision policy training

The configuration of neural network parameters is the same as the ACKTR agent in section 4.2.1 except that the type of distribution is categorical distribution instead of Gaussian distribution. The experiment result on the performance of the decision policy training on dynamicg8 is shown in Figure 4.18. The result shows that the agent is able to achieve a performance of around 2900. The performance is far from the theoretically optimal score of 6000, but the result shows that the agent could at least learn a meaningful policy.

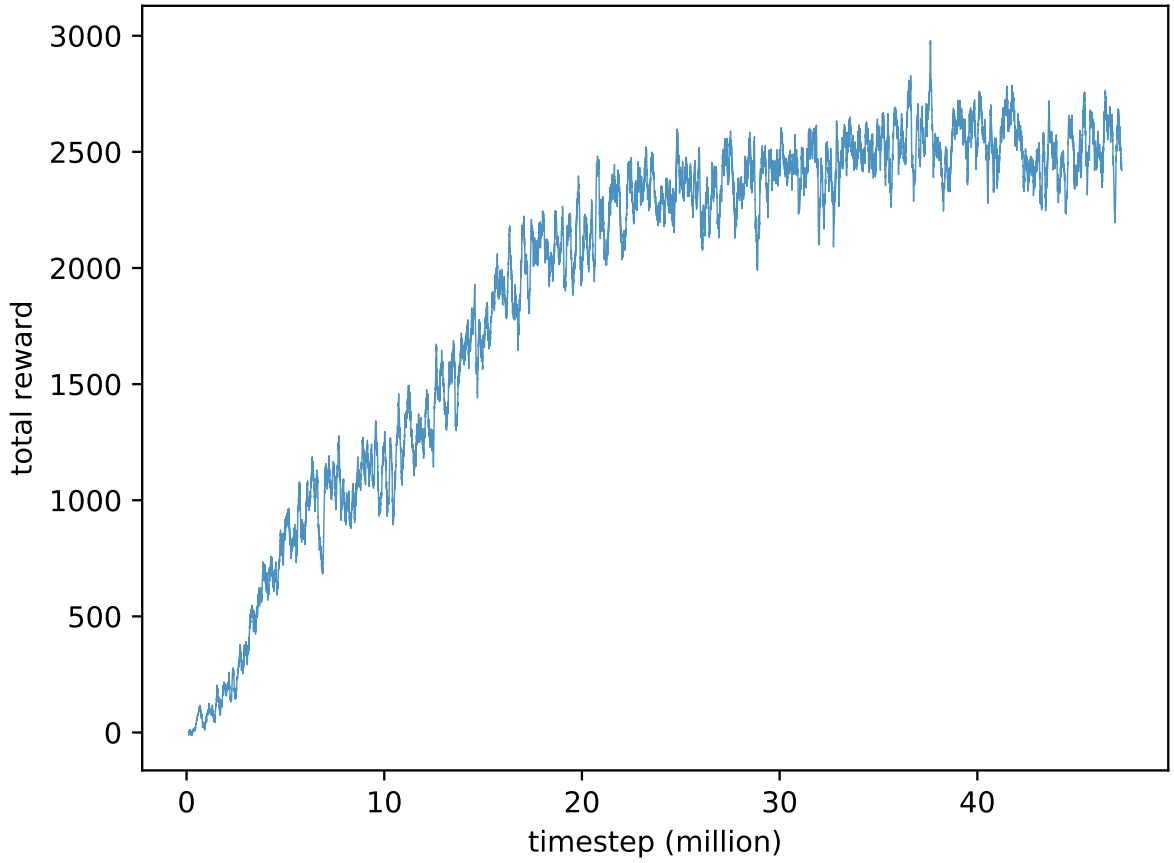


Figure 4.18: Decision policy training performance of the task dynamicg8, the horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 200 episodes

The performance of the training of decision policy on task reachcont is shown in Figure 4.19. The agent achieves an average reward of around 0.75, and is considered having solved the task.

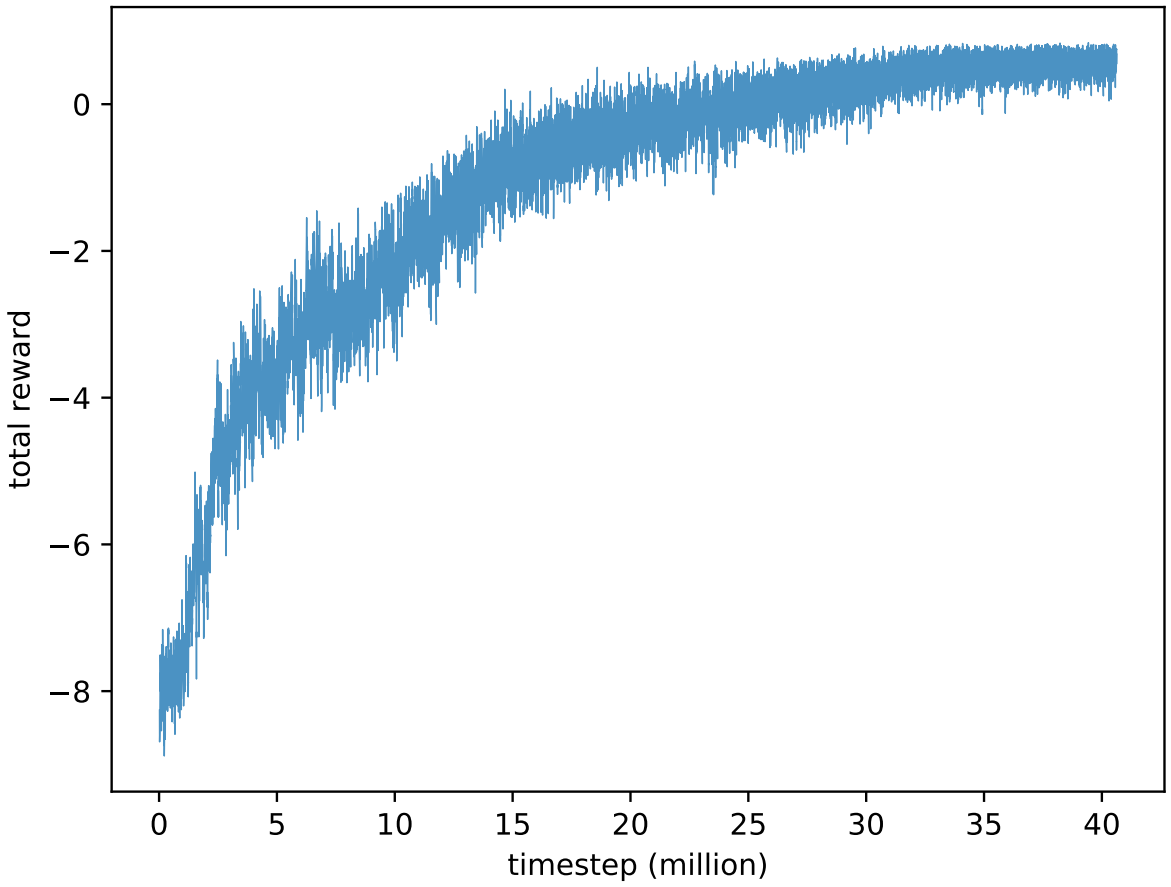


Figure 4.19: Decision policy training performance of the task reachcont, the horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 200 episodes

The performance of the training of decision policy on task reachcontreg is shown in Figure 4.20. The task appears to be more difficult than reachcont because the episode length is much longer. The hierarchical reinforcement learning agent still manages to solve the task at the end.

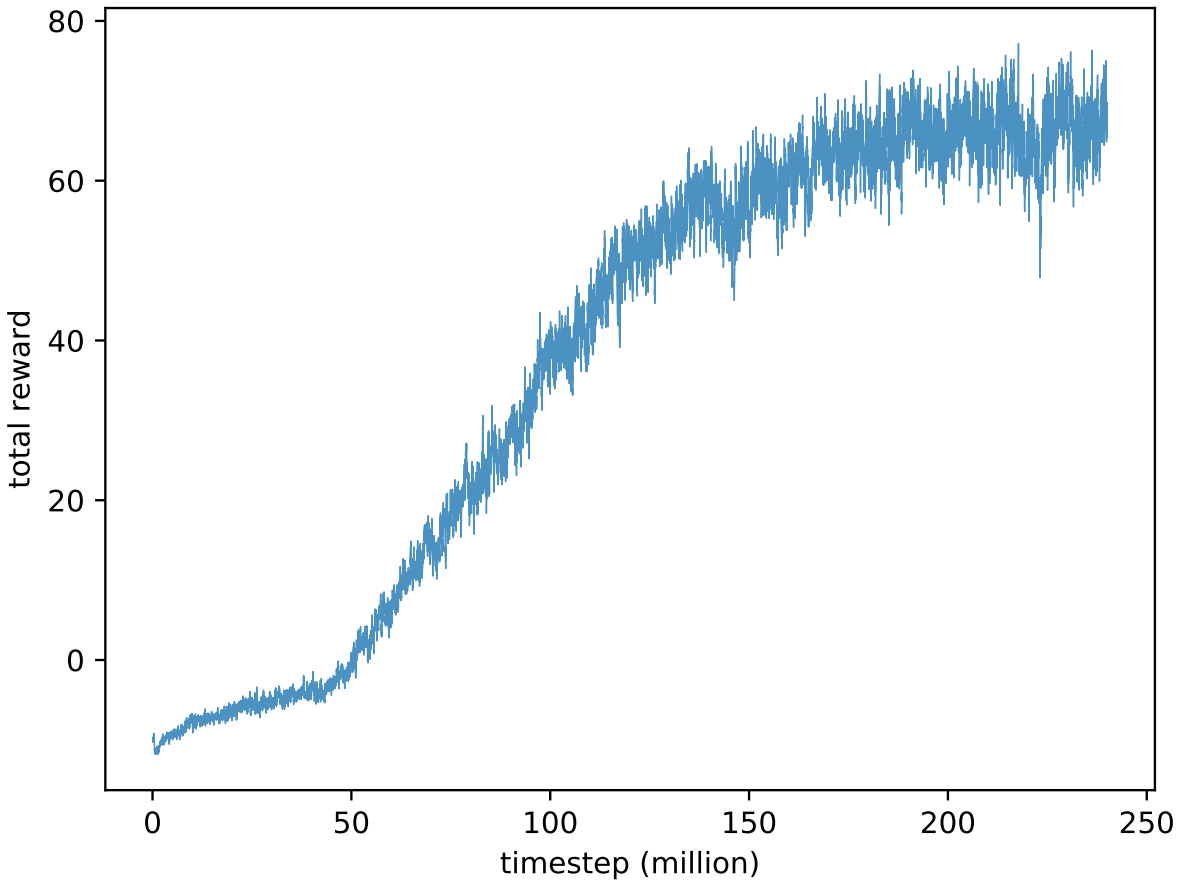


Figure 4.20: Decision policy training performance of the task reachcontreg, the horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 200 episodes

The result of the training of the decision policy shows that the hierarchical reinforcement learning agent is able to achieve a reasonable level of performance. The result is as expected because previous works in SMDP have been studied intensively.

Phase 2: Switcher policy training

At the beginning of this phase, the switcher policy is re-initialized randomly so that it outputs a termination signal with a probability of around 0.5. This reinitialized switcher policy is different from the switcher policy in phase 1 which outputs a termination signal

deterministically after the current actuator policy has been executed for a pre-defined number of steps. The switcher policy is then trained from scratch.

The experiment result on the performance during the training of the switcher policy on the task dynamicg8 is shown in Figure 4.21. The best performance during this phase is nearly 3000, which is slightly better than the best performance of 2500 in phase 1. However, the performance suddenly drops to a very low value at around 30 million training steps.

The average execution length of the decision policies is plotted in Figure 4.22. The agent manages to find a better switcher policy since the average execution length becomes around 10 when the performance reaches the highest level at 30 million training steps. It means that the decision frequency of the decision policy is reduced. It can also be seen that the average execution length increases suddenly to the maximum value 500 after that.

The training of switcher policy beneficial in the sense that the agent can maintain a good performance and also reduce the decision frequency of the decision policy. A video recording of the agent can be viewed at the website: [`https://youtu.be/ytW7rTpgRAs`].

The performance of the reachcont switcher policy is shown in Figure 4.23. The final performance has been improved from 0.75 in the decision policy training to around 0.81. The training of switcher policy appears to beneficial to the performance of the target task.

The average execution length of the decision policies is plotted in Figure 4.24. The final average value of the execution length appears to be reduced to 1.6, compared to 5 in the decision policy training phase.

Therefore, the switcher policy is able to improve the reinforcement learning performance at the expense of increasing the decision frequency of the decision policy.

The performance of the reachcontreg switcher policy is shown in Figure 4.25 and the average execution length of actuator policies is shown in Figure 4.26. A video recording of the agent can be viewed at the website: [`https://youtu.be/Y0uHEpJ7Cck`]. The best performance, which as 95, surpasses the final performance of 70 in phase 1. The average execution length is 1.0 when the performance reaches the highest level. This means that temporal abstraction is infeasible for this task. However, the training also follows a pattern that the performance degrades in the later phase of training when the average execution

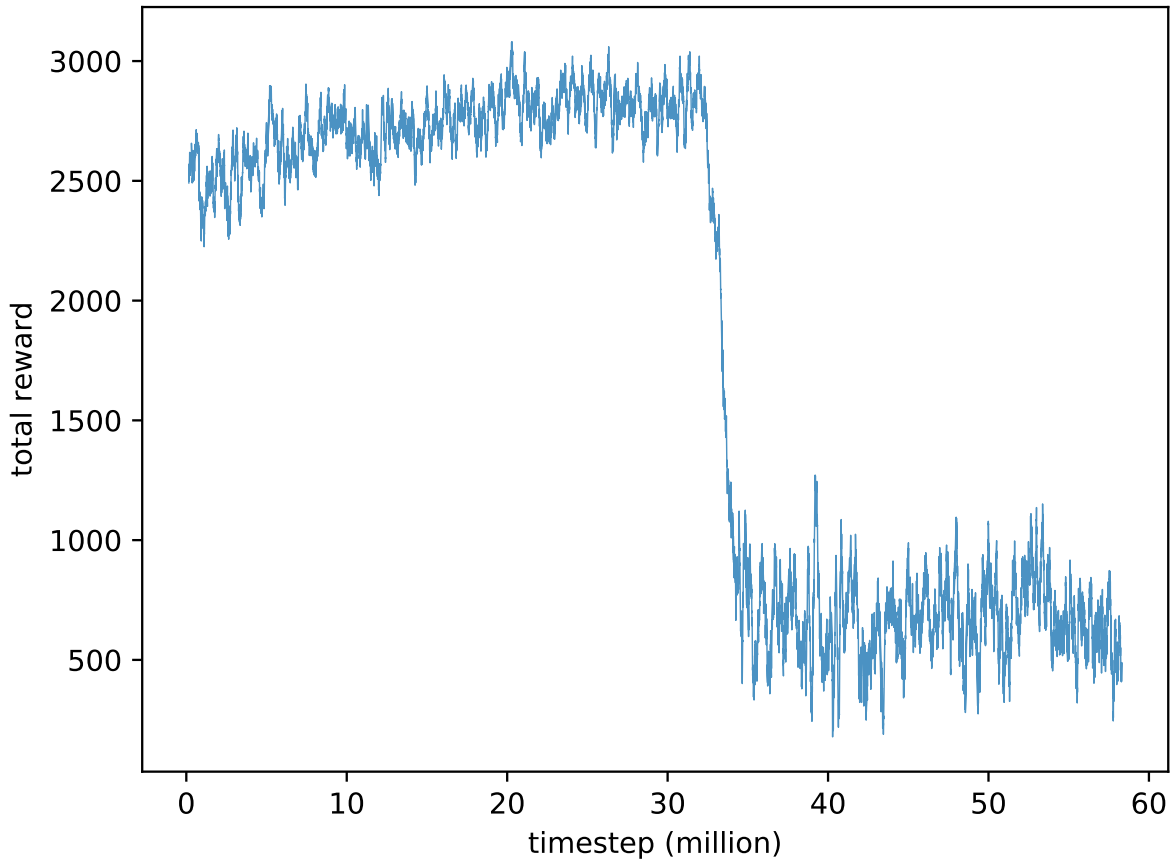


Figure 4.21: Switcher policy training performance of the task dynamicg8, the horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 200 episodes

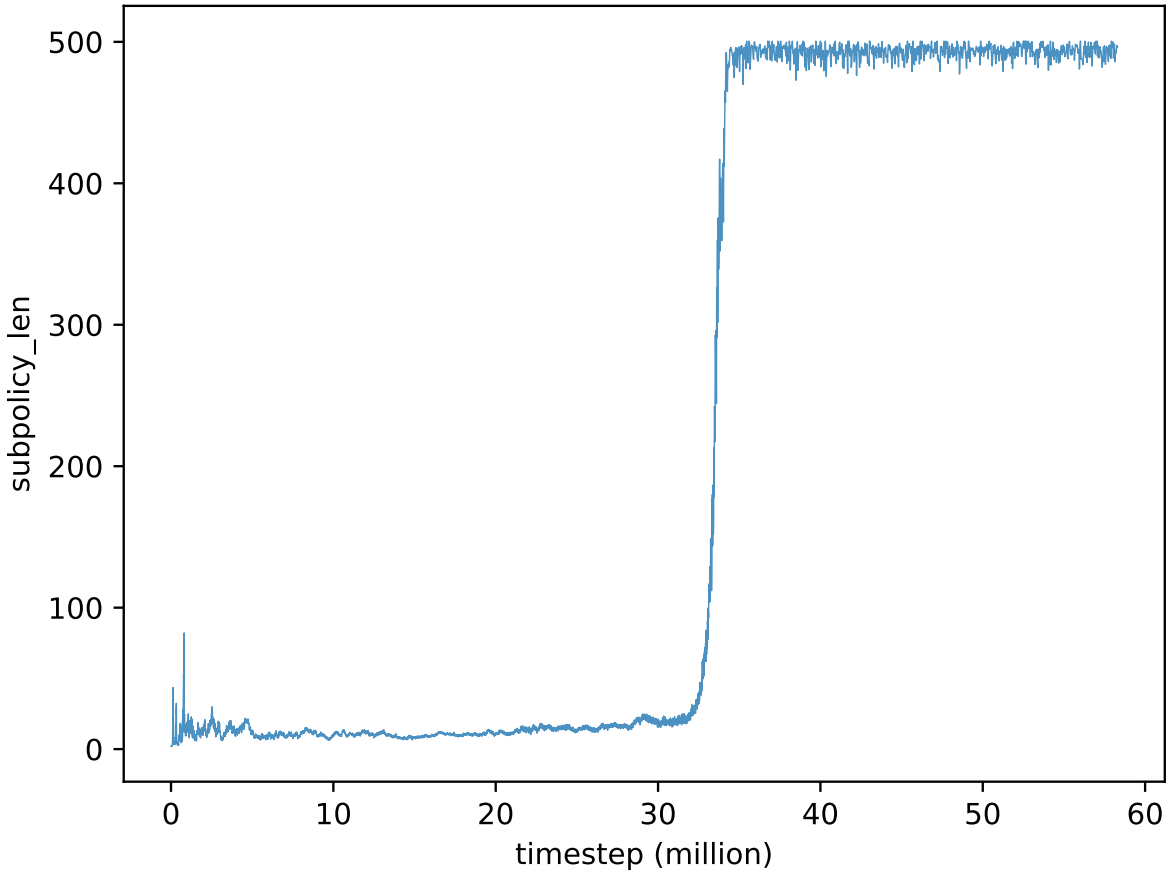


Figure 4.22: Average execution length of the actions of decision policies of the task dynamicg8, the horizontal axis is the number of million time-steps and the vertical axis is the decision policy's average execution length of the last batch.

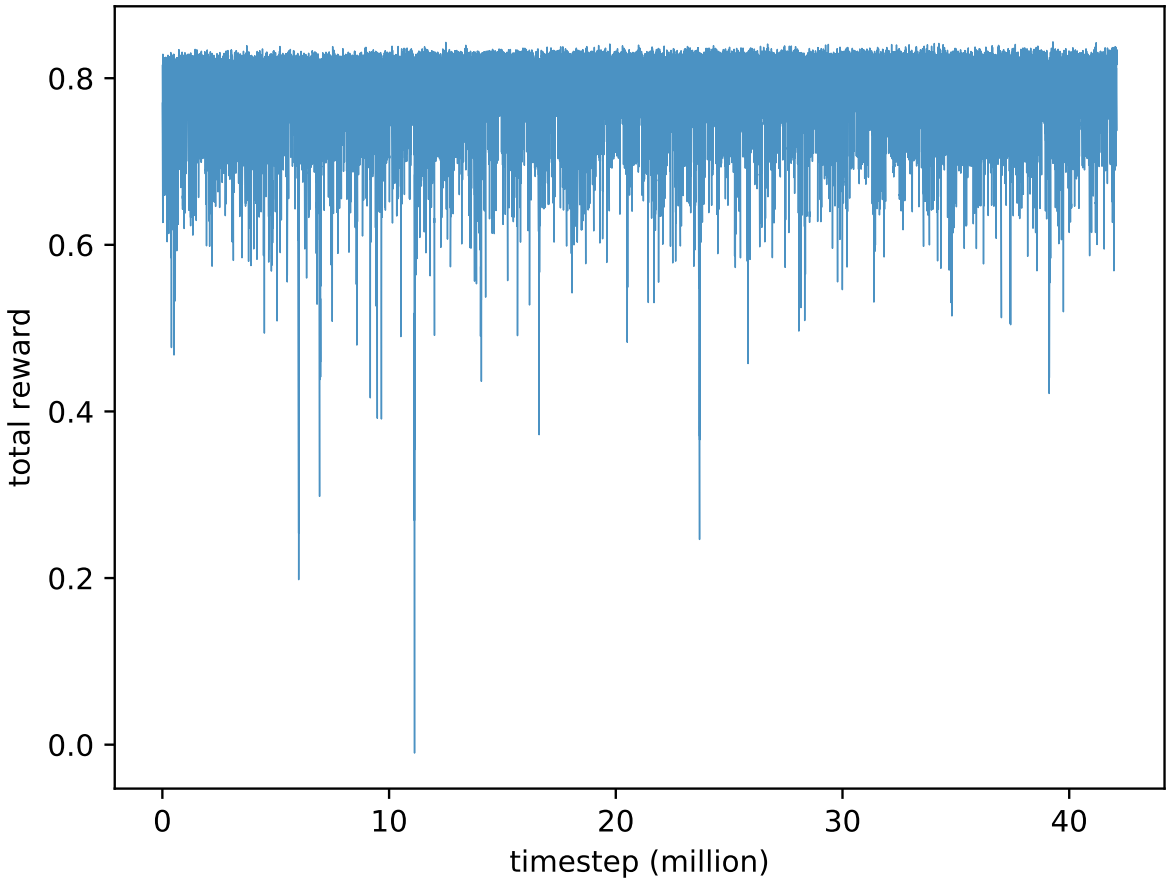


Figure 4.23: Switcher policy training performance of the task reachcont, the horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 32 episodes

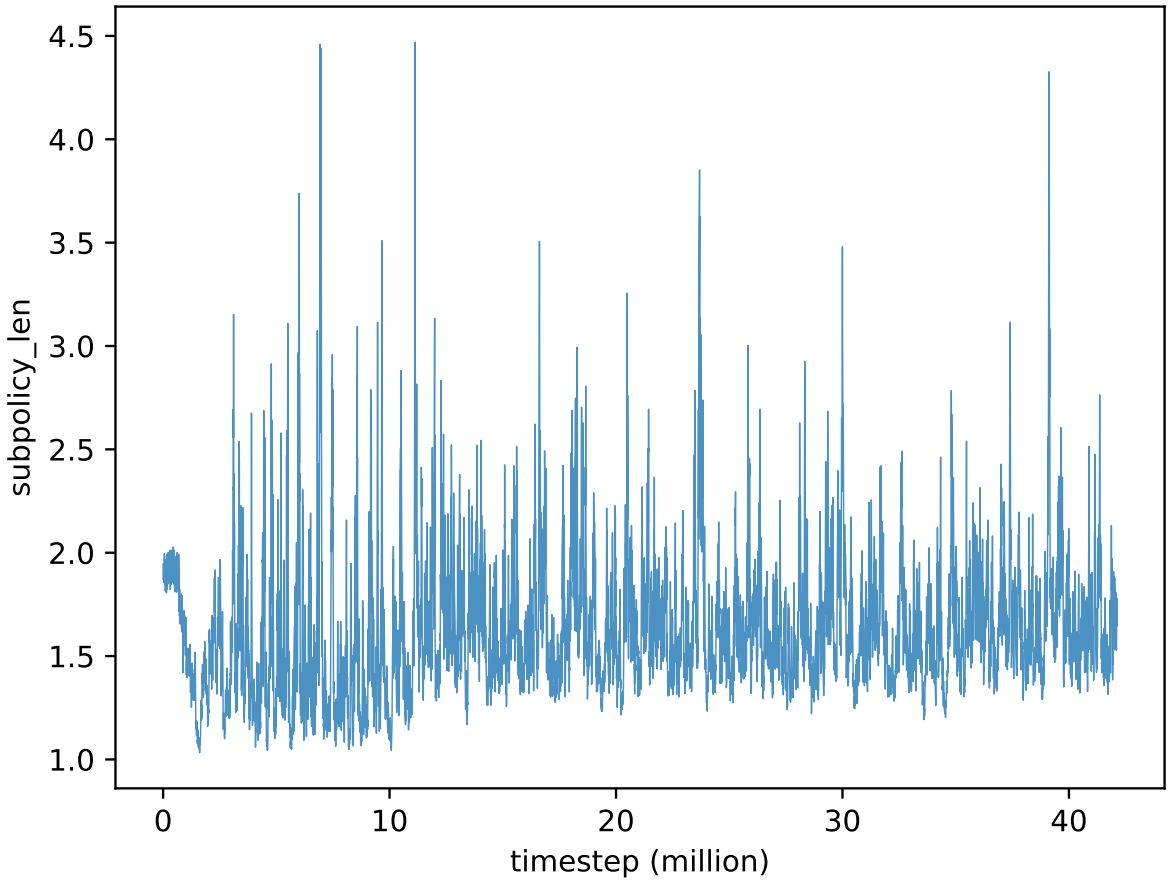


Figure 4.24: Average execution length of the actions of decision policies of the task reach-cont, the horizontal axis is the number of million time-steps and the vertical axis is the decision policy's average execution length of the last batch.

length begins increase.

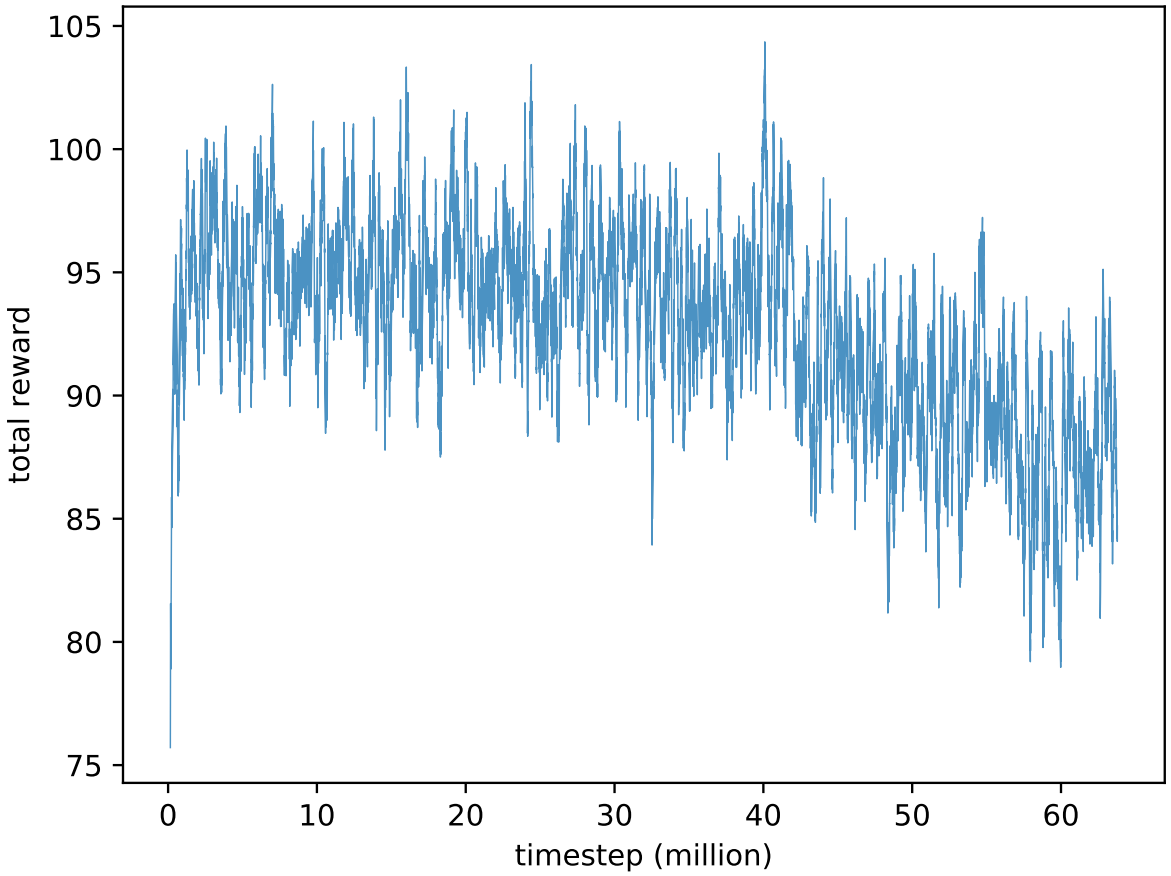


Figure 4.25: Switcher policy training performance of the task reachcontreg, the horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 200 episodes

One reason that the performance degrades in the later phase of switcher policy training could be due to the penalty term that penalize the termination of actuator policy. The term could have a stronger impact at the late phase of training, where the scale of the advantage term in the reinforcement learning loss decreases.

The above experiment results show that a switcher policy that controls the scheduling intervals of the decision policy can actually be trained in an end-to-end manner. However, a better penalty term on frequent actuator policy termination could be designed, to prevent the degradation of performance in the late phase of training.

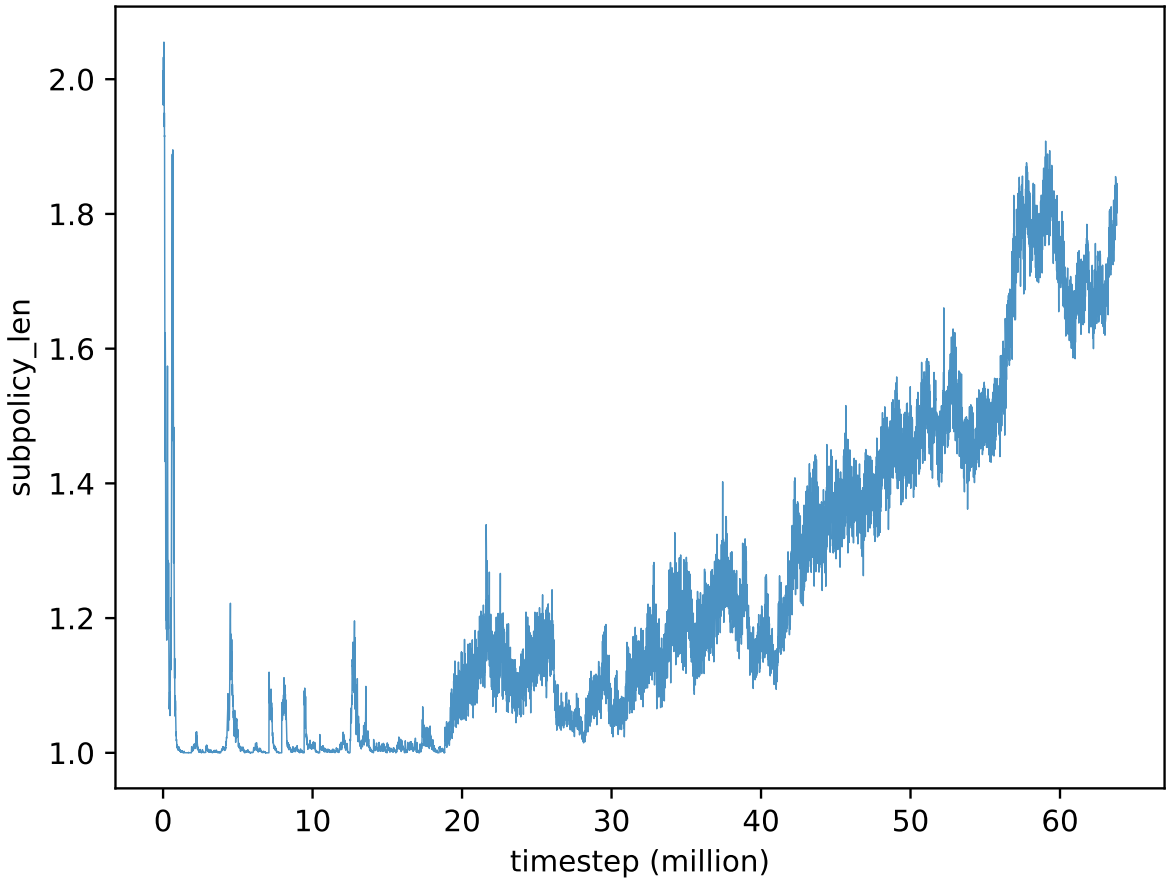


Figure 4.26: Average execution length of the actions of decision policies of the task reach-contreg, the horizontal axis is the number of million time-steps and the vertical axis is the decision policy's average execution length of the last batch.

4.4 Conclusion on Experiment Results

We have proposed a set of reinforcement learning tasks that have the property of multi-modal state space. Some of them also have sparse reward functions. Control experiments have been done and the results show that the contemporary reinforcement learning algorithms cannot solve the tasks. We have proposed several techniques that aim to improve the performance of flat reinforcement learning methods on the proposed task. Firstly we propose the Wasserstein actor critic Kronecker-factored trust region policy optimization (W-KTR) method aiming to improve the learning performance. However, the experiment results on the W-KTR method show that the method does not significantly outperform other contemporary state-of-art methods. We have also proposed two techniques aiming to achieve better exploration performance. The first technique, namely exceptional advantage estimation, has proven to improve the final performance of flat reinforcement learning performance on the task movecont. The second technique, namely robust concentric Gaussian mixture model, also manages to solve the movecont task but is less efficient in term of the number of training samples.

Although the above-mentioned methods are able to improve the performance of flat reinforcement learning methods on these tasks, a long training time and a large of training data are still required. Apart from that, tasks with both multi-modal state space and sparse reward functions could hardly be solved by flat reinforcement learning methods. The performance of the proposed method should better be verified in a bigger variety of tasks in future works.

We have also proposed an autonomous hierarchical reinforcement learning framework. The proposed domain randomization by cross-sampling initial states method has been proven to successfully learn the actuator policies of the source tasks designed in this study. The proposed hierarchical agent was able to achieve a reasonable performance in the task dynamicg8 and successfully solve the task reachcont. Both the decision policy and the switcher policy are trained in an end-to-end manner.

Chapter 5

Discussion

This thesis studies reinforcement learning solutions to robot control tasks with multi-modal state space and sparse reward functions.

Firstly, we have proposed a set of experiment environments. We have shown that contemporary deep reinforcement learning algorithms for continuous control cannot solve the proposed problems which have multi-modal state space.

Secondly, we have proposed several techniques aiming to improve the performance of the contemporary flat reinforcement learning methods, including Wasserstein actor critic Kronecker-factored trust region policy optimization (W-KTR) method, exceptional advantage estimation, and robust concentric Gaussian mixture model. Some of them, especially the exceptional advantage estimation method, lead to improvement on the final performance of the agent. However, an efficient flat reinforcement learning solution to multi-modal problems in terms of the number of training steps is still missing. It is worth exploring in the future whether there is any technique that can solve the multi-modal problems without getting stuck at local minimums for such a long time.

Thirdly, we have proposed an autonomous hierarchical reinforcement learning method for the tasks with multi-modal state space and sparse reward functions given a predefined set of source tasks. The experiments show positive results, indicating that the proposed method can solve the tasks in an end-to-end manner.

The experiments in this thesis only show preliminary results on a limited set of tasks. The capability of the method on general robot problems also needs to be verified in a bigger variety of problems in future works. The drawback of this hierarchical reinforcement learning framework is that it requires a properly pre-defined set of source tasks. Designing the source-task set is easy for the proposed problems, but we are not sure whether the job is feasible for real-world robot control tasks in general. Apart from that, we have not

thoroughly investigated how the scheduling of decision policy training and switcher policy training could be designed. We have a pre-defined switcher policy at first, and schedule the training of decision policy and switcher policy into two separate phases in our experiments. Future works can investigate whether it is possible to train the decision policy and the switcher policy jointly. Apart from that, how to properly balance the switcher policy for better reinforcement learning performance and lower decision frequency of the decider policy is worth studying.

In conclusion, realistic continuous control tasks with multi-modal state space and sparse reward functions still pose a significant challenge to flat and hierarchical reinforcement learning methods. Our study proposes several novel solutions, but an efficient solution that achieves optimal performance without human intervention is still missing.

References

[1] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver, "Learning and transfer of modulated locomotor controllers," *arXiv preprint arXiv:1610.05182*, 2016, https://arxiv.org/pdf/1610.05182.

[2] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," *arXiv preprint arXiv:1710.09767*, 2017.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[6] J. Heinrich and D. Silver, "Deep reinforcement learning from self-play in imperfect-information games," *arXiv preprint arXiv:1603.01121*, 2016.

[7] S. Zhang, H. Pondé, G. Brockman, R. Józefowicz, J. Pachocki, D. Farhi, J. Raiman, C. Dennison, M. Petrov, P. Dębiak, F. Wolski, J. Tang, S. Sidor, and B. Chan, "Openai five," Jun 2018. [Online]. Available: https://blog.openai.com/openai-five/

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[9] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Advances in Neural Information Processing Systems*, 2017, pp. 5285–5294.

[10] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 99, 1999, pp. 278–287.

[11] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[12] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.

[13] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[14] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[15] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR, abs/1502.05477*, 2015, http://www.jmlr.org/proceedings/papers/v37/schulman15.pdf.

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

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

[19] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[20] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Trust-pcl: An off-policy trust region method for continuous control," *arXiv preprint arXiv:1707.01891*, 2017.

[21] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1, pp. 181–211, 1999.

[22] R. Parr and S. J. Russell, "Reinforcement learning with hierarchies of machines," in *Advances in neural information processing systems*, 1998, pp. 1043–1049.

[23] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *J. Artif. Intell. Res.(JAIR)*, vol. 13, pp. 227–303, 2000.

[24] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," 2001.

[25] B. Hengst, "Discovering hierarchy in reinforcement learning with hexq," in *ICML*, vol. 2, 2002, pp. 243–250.

[26] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Van de Wiele, V. Mnih, N. Heess, and J. T. Springenberg, "Learning by playing-solving sparse reward tasks from scratch," *arXiv preprint arXiv:1802.10567*, 2018, https://arxiv.org/pdf/1802.10567.

[27] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5055–5065, http://papers.nips.cc/paper/7090-hindsight-experience-replay.pdf.

[28] D. J. Mankowitz, A. Žídek, A. Barreto, D. Horgan, M. Hessel, J. Quan, J. Oh, H. van Hasselt, D. Silver, and T. Schaul, "Unicorn: Continual learning with a universal, off-policy agent," *arXiv preprint arXiv:1802.08294*, 2018.

[29] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," *arXiv preprint arXiv:1703.01161*, 2017.

[30] T. Shu, C. Xiong, and R. Socher, "Hierarchical and interpretable skill acquisition in multi-task reinforcement learning," *arXiv preprint arXiv:1712.07294*, 2017.

[31] J. Andreas, D. Klein, and S. Levine, "Modular multitask reinforcement learning with policy sketches," *arXiv preprint arXiv:1611.01796*, 2016.

[32] A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, J. Agapiou *et al.*, "Strategic attentive writer for learning macro-actions," in *Advances in neural information processing systems*, 2016, pp. 3486–3494.

[33] O. Klimov and J. Schulman, "Roboschool," Jun 2018. [Online]. Available: https://blog.openai.com/roboschool/

[34] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," *arXiv preprint arXiv:1709.06560*, 2017.

[35] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf, "Wasserstein auto-encoders," *arXiv preprint arXiv:1711.01558*, 2017.

[36] C. Villani, *Topics in optimal transportation*. American Mathematical Soc., 2003, no. 58.

[37] D. Chafai, "Wasserstein distance between two gaussians." [Online]. Available: http://djalil.chafai.net/blog/2010/04/30/wasserstein-distance-between-two-gaussians/

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[39] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[40] J. Tobin, W. Zaremba, and P. Abbeel, "Domain randomization and generative models for robotic grasping," *arXiv preprint arXiv:1710.06425*, 2017.