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

by

Zeng Cancheng

A Thesis Submitted to

The Hong Kong University of Science and Technology

in Partial Fulfillment of the Requirements for

the Degree of Doctor of Philosophy

in Master of Philosophy in Computer Engineering and Science

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.

____

Zeng Cancheng

Date: ______________

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

by

Zeng Cancheng

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-mindness, and his continuous support on 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 Zeng Cancheng

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 of realistic robot control problems have multi-modal state space, which usually includes low-dimensional motion sensor signal and high-dimensional image sensor signal. Apart from that, a smooth and informative reward function is usually unavailable in realistic environments, which only has sparse and discrete reward signals.

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 concentric mixture Gaussian policy model. An 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 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 map 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 generate experience that efficiently explore the problem space. 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, following are definitions of the action value function Q_π, the value function V_π and the advantage function A_π:

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

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

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s), \text{where } a_t \sim \pi(a_t|s_t), \text{for } t \geq 0$$

1.3 Scope of study

Realistic robot control problems in reinforcement learning involves many sources of complexity. However, we are only concerned with some of the typical problems. This study focus on a set of reinforcement learning problems with multi-modality state space and continuous action space. We also specifically study some problems with not only the previously mentioned properties, but also with 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 case, 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 sensor and image sensor, real-world robot agents may have other modalities such as wireless network antenna input, human language instructions and laser sensor. 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 complex than single-modality problems. The reason is that the feature of image observations is much more difficult to learn than the 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 not be able to learn anything about a good policy if it has not triggered event, or the agent may not be able to improve further if it has already stopped exploring the environment after experienced the informative events. 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 relies 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 assume that the target task can be solved by a sequential execution of a subset of the actuator policies. An *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 in 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 to 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 of such problem 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: *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.

To solve the first issue, we need to answer the question of "*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?*".

To address the second issue, we will answer the question of "*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 mothod is a class of reinforcement learning methods that solve the problem by optimizing 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 esitimating the gradient $g := \nabla \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t r_t\Big]$, which has the form [13]:

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

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

Several methods for updating the policy parameters given the policy gradient have been proposed. The following section will discuss some state-of-art works.

2.1.1 Trust Region Policy Optimization

The method proposed by [14], 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:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & J(\theta) \\ \text{subject to} \quad & \overline{D_{KL}}(\pi_{\theta_{old}} \| \pi_\theta) \leq \delta_{KL} \end{aligned} \tag{2.2}$$

where $\pi_{\theta_{old}}$ is the old policy parameter before performing 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 $\beta = \sqrt{2\delta_{KL}/s^T A s}$ 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 [15] 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

The work of [9] proposes to reduce the computation time solving the natural gradient by 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:

$$s = A^{-1} g \approx F^{-1} g \approx \left(\mathbb{E}[aa^T] \otimes \mathbb{E}[\nabla_s L \nabla_s L^T]\right)^{-1} g = \mathbb{E}[aa^T]^{-1} g \mathbb{E}[\nabla_s L \nabla_s L^T]^{-1} \tag{2.3}$$

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 decent 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 [16] proposes a first-order policy optimization method, namely proximal policy gradient with clipping (PPO). The method clips the surrogate objective according to the following equation:

$$L_{CLIP}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t\right)\right] \tag{2.4}$$

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. However, the PPO method may focus too much on the convergence rate, and there is not evidence that shows the method could achieve a better final performance.

2.1.4 Other Methods in Deep Reinforcement Learning for Continuous Control

Other methods related to deep reinforcement learning for continuous control include the Deep Deterministic Policy Gradient Method (DDPG) [17], Actor-Critic with Experience Replay (ACER) [18], and Trust-PCL [19]. These methods are first order off-policy algorithm, and are not the focus of our study because no evidence has shown that any off-policy methods to can significantly outperform the above mentioned methods in terms of training time, final performance and stability.

2.2 Hierarchical Reinforcement Learning Methods

The paradigm of hierarchical reinforcement learning focuses on solving challenging reinforcement learning problems through temporal abstraction [12]. A hierarchical reinforcement learning agent's decisions invoke temporally extended activities instead of taking primary actions, so that it does not need to make decision at every time step.

The history of hierarchical reinforcement learning study can be dated back to late 1990s. Many methods have been proposed in the past 20 years, including option framework [20], hierarchies of abstract machines [21], and MAXQ value function decomposition [22].

Any hierarchical reinforcement learning method needs to solve 2 problems. First is to define and learn the temporally extended activities, which are low-level closed-loop partial policies. Second is to learn the decision scheduling of the low-level partial policies, including the selection of the partial policies and the scheduling of these decision points.

Many paradigms have been proposed to solve the first problem. The ideal formulation is for the agent to directly discover low-level policies that could be reused. However, this formulation has make the problem infeasible although some methods have been proposed [23], [24] in some special cases.

The most widely applied formulation is to manually define the low-level policies, or their corresponding MDPs. The problem is that this formulation basically relies on intensive domain-specific design, and manually defining them may not be feasible for real-world challenging environments.

Another popular formulation is to propose a set of auxiliary tasks to define and train the low-level policies. However, the method of proposing auxiliary tasks is only feasible in some special cases and its development needs domain-specific engineering.

Finally, this thesis assumes the low-level policies are trained in a set of given source tasks that are primary problems. This formulation is general in the sense that the target problem can be solved as long as it can be decomposed into executing a series of some source task policies.

The solution to the first part of the second problem, which is learning to select among low-level policies, has been thoroughly studied since the work of [20]. However, the second part, which is to schedule the decision points, have not been solved for general environments. The most popular choice is to schedule decision points at fixed predefined periods, which is inflexible.

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

2.2.1 Option framework

The work of [20] is among the earliest studies of hierarchical reinforcement learning, which uses the notion of *option* to define the partial policies / subroutines. An option consists of a policy π, a termination condition β and an input set I that indicates whether the partial policy is available at the current state. Once an option is executed, then actions are chosen according to π until $\beta(s)$ outputs a termination signal.

An option is Markov if its policy is Markov. Semi-Markov options, on the other hand, are options whose policies are based on the entire history since the option was initiated. Semi-Markov options include options that terminates after a specific number of time steps, which could be particularly useful when considering policies over options.

A policy over options μ selects option o in state s with probability $\mu(s, o$.

A 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. Then the reward 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:

$$V_O^*(s) = \max_{o \in O_s} \left[R(s, o) + \sum_{s'} P(s'|s, o) V_O^*(s') \right] \tag{2.5}$$

In conclusion, this work 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 are developed are not answered.

2.2.2 Modulated hierarchical controller

The work of [1] is among the first studies that applies hierarchical reinforcement learning on robot 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 outputs 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 curre nt state $o(s)$ and the control signal from the high-level controller c to the policy output.

$$\pi(a|s) = F_L(o(s), c) \tag{2.6}$$

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:

$$z_t = f_z(s_t, z_{t-1}) \tag{2.7}$$

$$c_t = f_c(z_{t_r}) \tag{2.8}$$

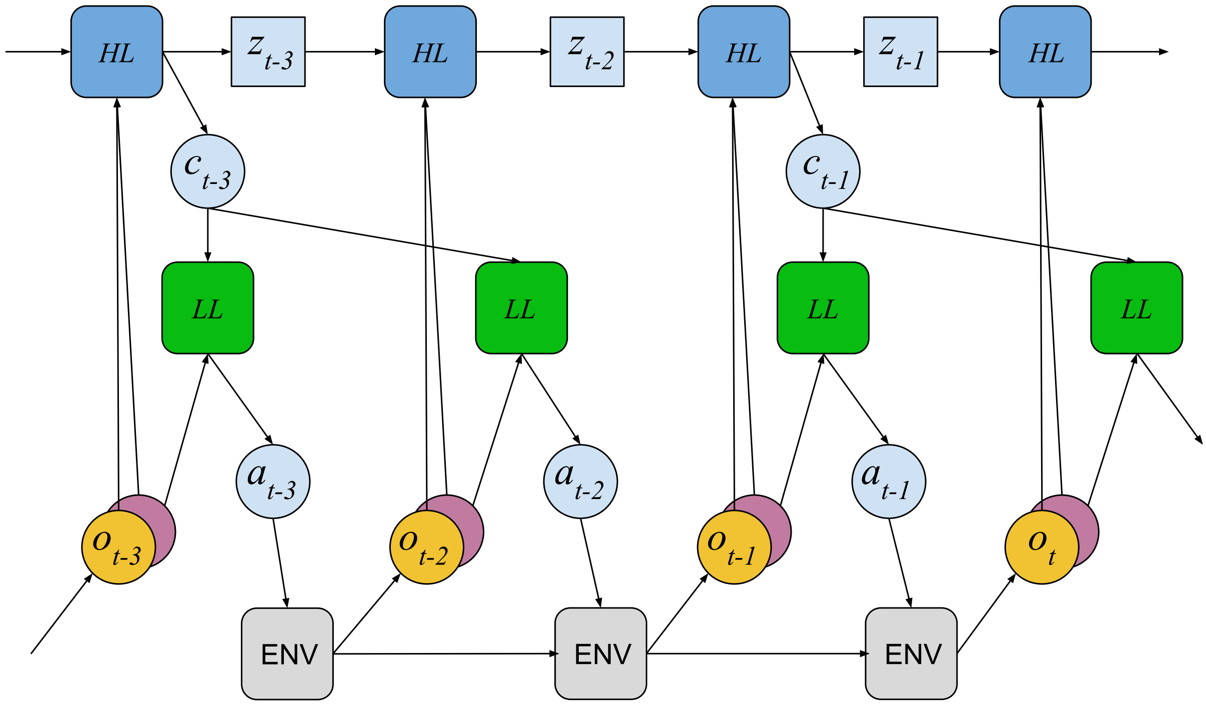

Figure 2.1: The hierarchical structure of the modulated controller [1], which consists of the recurrent high-level controller (HL, dark blue) and feedforward low-level controller (LL, green). The high-level controller has access to all observations (yellow and red). While both controllers observe sensory 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 controller are frozen. The high-level controller is then trained in the transfer task, which has a sparse reward function. This method achieves better performance than flat agents on a few realistic robot control tasks. The limitation is that the architecture is highly engineered and domain-specifc. The observation of the low-level agent $o(s)$ need to be designed specifically for each task, and the However, the hierarchical relationships between the two agent, including the pre-training task definition and modulation model, need to be manually predefined. Apart from that, the design of the output modulated signal the high-level agent is limited to several locomotion tasks. This involves the agent.

The parameter K is also predefined, and thus the time scale of the high level agent lacks the flexibility.

2.2.3 Learning a hierarchical model by meta-learning

Another work [2], namely meta learning shared hierarchies (MLSH), proposes a metalearning 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:

$$\text{maximize}_{\phi} \mathbb{E}_{M \sim P_M}[R] \tag{2.9}$$

The detailed structure of the agent is shown in Figure 2.2. The architecture is basically 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) shared across different tasks and θ is the parameters of the high-level agent of the current task. The master policy samples the master action at a fixed frequency which selects the acting sub-policies. The

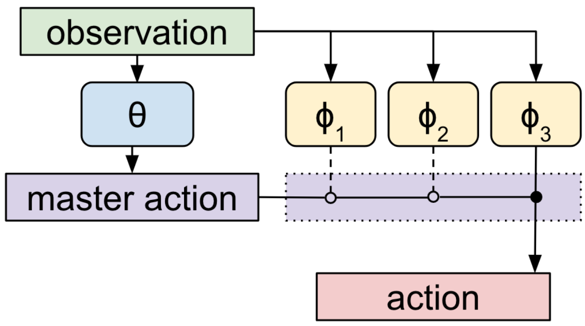

Figure 2.2: The agent setup of the modulated controller [2],θ represents the master policy, which selects a sub-policy to be active. In the diagram, ϕ_3 is the active sub-policy, and actions are taken according to its output.

agent is trained in the following manner: First a task is sampled M is sampled and the parameter θ is re-initialized randomly. Then there is a warmup phase, where only θ is trained to achieve optimal. Then the agent enters the joint update phase, where both θ and ϕ are updated simutaneously. The method achieves good performance in several control environments. Including a moving-bandit Ant-robot task, a Ant-robot maze navigation task and an obstacle course task. However the tasks are actually simplified version of the control tasks in [5], where the state of the Ant-robot is periodically reset to prevent episode termination due to the falling over of Ant. That largely reduces the difficulty on learning the locomotion skills of agents. It is not clear whether the method will perform well on the original version of the Ant-robot environment. The major limitation is that the method relies on a warm up period to learn a high-level agent's policy. However, the authors didn't provide a method to learn the high-level agent's policy θ at first task, where the parameters ϕ are also randomly initialized. The method also predefines the temporal relationship between the high level policy and low level policy.

2.2.4 Goal-directed learning method

Several works [25], [26] have tried to solve sparse-reward robot control environments through the learning and scheduling of auxiliary policies. The method of [25], namely Scheduled Auxiliary Control (SAC-X), is based on several main principles. First, the original MDP is modified that every state-action pair has a vector rewards, consisting 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 in this context. Third, there is a high-level scheduler agent that selects and executes the intentions. Fourth, the learning of intentions is performed off-policy on the same experience. The definitions of internal auxiliary rewards are based on predefined goal states. The internal auxiliary reward with goal state g is defined as:

$$r_g(s,a) = \begin{cases} \delta_g(s), & \text{if } d(s,g) \\ 0, & \text{else} \end{cases} \tag{2.10}$$

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

$$\mathcal{L}(\theta) = \mathcal{L}(\theta; M) + \sum_{k=1}^{|A|}(\theta; \mathcal{A}_{\mathcal{K}}) \tag{2.11}$$

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 an robot-arm environment. The major limitations of SAC-X is that the method relies on predefined auxiliary MDPs, which is infeasible to obtain in realistic environments.

2.2.5 Other methods targeting at sparse environments

Apart from the above mentioned works, several other methods have also been proposed to solve environments with sparse reward recently, including Unicorn [27], FuN [28], HRL with Stochastic Temporal Grammar [29], policy sketch method [30] and strategic attentive writer [31]. These methods have only been verified in discrete environments that have relatively low complexity, and involve domain-specific engineering components.

Chapter 3

Methodology

3.1 Target Environments

3.1.1 Summary of environment design

We propose a set of reinforcement learning tasks that are suitable for studies in our target problem of multi-modality and sparse environments. The agents' physical systems are consistent with the 3D robot control environments in OpenAI Gym [15], while the external environments and reward functions of the target tasks are different from the original environments in OpenAI Gym. There are three kinds of 3D robot agents in 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-dimensinoal action space. The three agents are showed in Figure 3.1, Figure 3.2 and Figure 3.3.

We will mainly focus on target environments based on the "Ant" agent, because the agent has the capability of solving a rich variety of high-level tasks, while the agent's primary control tasks are reasonably challenging. Compared to the "Ant" agent, the "Swimmer" agent is too simple and its capability of solving more challenging tasks is limited. The "Humanoid" agent's on the other hand, has a much more unstable physical system compared to "Ant".

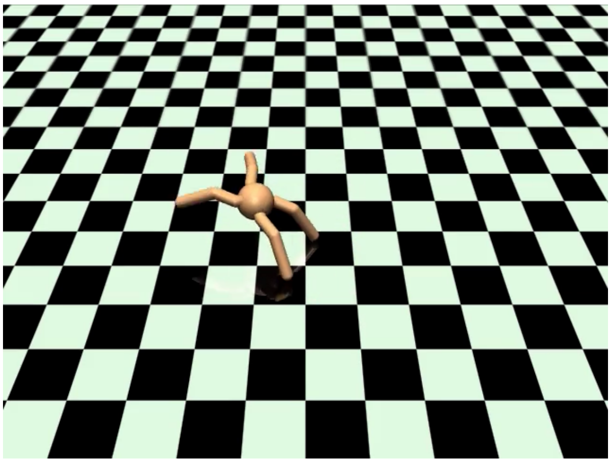

Figure 3.1: The ”Ant” agent

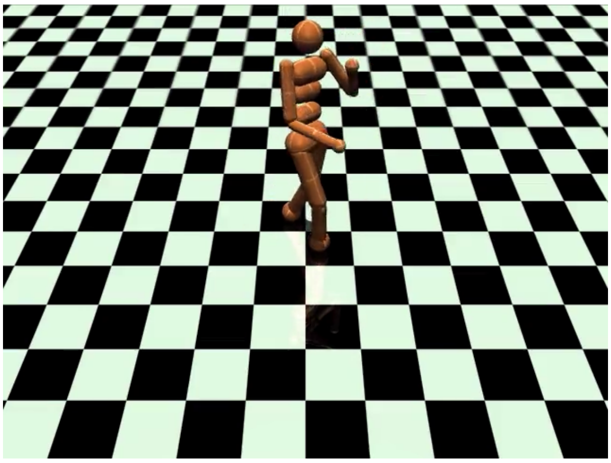

Figure 3.2: The ”Humanoid” agent

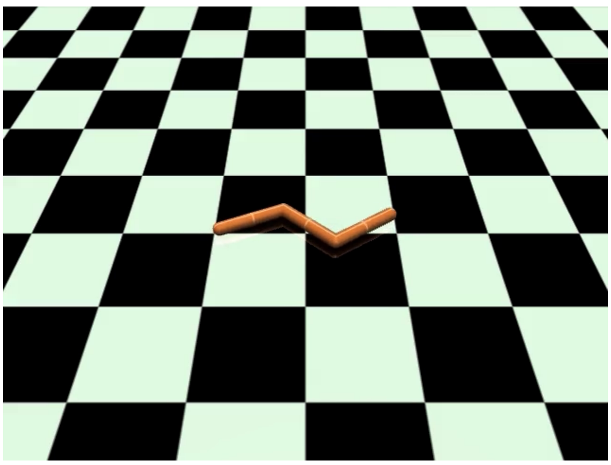

Figure 3.3: The ”Swimmer” agent

3.1.2 Detailed environment specification

We provide a detailed description of the experiment environments in this section. All of the environments are based on the "Ant" task [15]. In the original "Ant" task, the agent receives a 111-dimensional state input and outputs a 8-dimensional action output. The agent's state consists of a 13-dimensional vector that represents its pose, a 14-dimensional vector that represents its velocity, and a 84-dimensional vector that represents related contack force.

In the settings of the proposed environments, the agent receives not only the 111-dimensional state, but also a $64 \times 64 \times 1$ dimensional grayscale image observation. A sample image observation is shown in Figure 3.1.2.

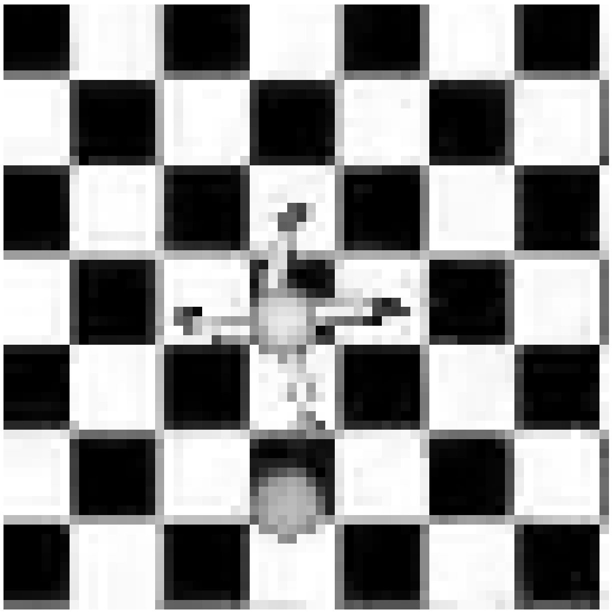

Figure 3.4: A sample image observation of the target environments

A basic task, namely "move0", is the same as the definition of "Ant" in OpenAI Gym [15] except an extra image observation as state. 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 goal 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 strong collision. The episode terminates when the agent enters a unrecoverable state, such as being turned over, or if the episode length reaches 1000 time-steps.

Other basic environments that might be used as source tasks are ”move1”, ”move2”, ”move3”, ..., ”move7”. This tasks are the same as ”move0” except that the goal direction is different. The image observation is redundant for all these low-level tasks, because the agent only needs to move toward one specific direction.

The target tasks are design to be suitable for the definition of multi-modality and sparse-reward. We propose several multi-modality environments, including ”moveg2”, ”moveg4”, ”moveg8”, ”movecont”, ”dynamicg4”, ”dynamiccont”. These tasks requires the agent to learn not only from the state representation but also the image representation. We also propose sparse multi-modality environments: ”reachg4” and ”reachcont”. The agent receives very sparse rewards in these environments compared with the previous environments, which have smooth reward function. The goal direction or location is represented by a sphere object in the image observation, thus the image observation plays a critical role in these tasks. The set of all the proposed environments are described in table 3.1.2.

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

As is reported in [32], the result of the current state-of-art deep reinforcement learning methods for continuous control, including TRPO, ACKTR, PPO and DDPG are hard to be reproduced, due to they’re heavily influenced by a variety of factors including random seed, neural network architecture, activation functions in neural network and software implementation. This means that the state-of-art methods are lack of stability in the agent’s final performance and robustness to minor change in parameters. That has greatly diminishes the reliability of contemporary deep reinforcement learning methods.

We propose a new policy optimization method, namely Wasserstein Actor Critic Kronecker-factored Trust Region Policy Optimization (W-KTR). We claim that the proposed method outperform other state-of-art methods in terms of agent’s final performance, total training time and reproducibility.

The proposed W-KTR method focus on the problem scope of deep reinforcement

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 goal direction
moveg4	velocity samples from: $\{g_0, g_1, g_2, g_3\}$	$v_g + 1 - c_p - c_c$	each episode has a random sampled goal direction
moveg8	velocity samples from: $\{g_0, g_1, \ldots, g_7\}$	$v_g + 1 - c_p - c_c$	each episode has a random sampled goal direction
movecont	velocity samples from a continuous range of all unit directions	$v_g + 1 - c_p - c_c$	each episode has a random sampled goal direction
dynamicg8	velocity samples from: $\{g_0, g_1, \ldots, g_7\}$	$v_g - c_p - c_c$	the goal direction is re-sampled with probability 0.005 at each time-step
dynamiccont	velocity samples from a continuous range of all unit directions	$v_g - c_p - c_c$	the goal direction is re-sampled with probability 0.005 at each time-step
reachg4	position samples from $\{g_0, g_1, g_2, g_3\}$	$I(\lVert x - g \rVert_2^2 < 0.5) - 0.01$	The agent is only rewarded when reaching a goal position
reachcont	position samples from the unit circle	$I(\lVert x - g \rVert_2^2 < 0.5) - 0.01$	The agent is only rewarded when reaching a goal position

Table 3.1: Summary of Ant-based environments

learning for continuous control, specifically robot control problems. The 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 algorithms are not suitable for these problems. Because the KL-divergence cannot represent the "deviation of policy" of the problem well, because a small perturbation in the mean value of a policy will lead to a large KL-divergence when the variance is small.

Therefore, we propose to use another criteria, which is the Wasserstein metric, to measure the deviation of the policy updates. 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 [33]:

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

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 [34] :

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

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

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

We use the approximation of the Rianmanian metric tensor A:

$$A \approx \mathbb{E}_{s_t}\left[\nabla_\theta W_2^2(\pi_{old}, \pi)\left(\nabla_\theta W_2^2(\pi_{old}, \pi)\right)^T\right] \tag{3.6}$$

The solution of s is then computed using Kronecker-factor approximation technique [9].

The policy is then updated following the gradient direction s with ADAM stochastic optimization algorithm [35]. 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 major reasons that multi-modality robot environments are challenging is because the features in different modalities have different complexities.

In the target case, where there are two modality: the locomotion state vector and the image input, the agent is very likely to get stuck at a local minimum after having learned the features of the locomotion state vectors while haven't learned the image features. However, the policy is at a much lower entropy at the phase when the agent starts to learn image features.

As far as we know, the distribution type of the continuous stochastic policy in reinforcement learning is always chosen to be normal distribution with diagonal covariance matrices in the works on policy gradient methods . We have observed a phenomenon in the training process of policy gradient methods that, the variance of the policy is extremely unlikely to be increased, even when the agent has just escaped from a local minimum. Therefore, a regulation method becomes necessary to encourage the exploration of the agent when it finds that its policy has converged too fast at a sub-optimal point.

A conventional regularization method for encouraging exploration is entropy regularization:

$$g' = g + \beta_{ent} \nabla_\theta \mathbb{E}[H(\pi_\theta(a|s))] \tag{3.7}$$

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:

$$H(\pi) = \frac{1}{2} \ln \det(2\pi e \Sigma) \tag{3.8}$$

When the policy distribution is a normal distribution with diagonal covariance matrix, the entropy regularization basically introduces a constant bias in the gradient of the logarithm of variance parameters. As a result, the entropy regularization method is usually hard to tune, and leads to degradation on learning performance.

Here we propose a novel method that can efficiently encourage exploration without large penalty on the agent's learning performance. We add a loss, namely Exceptional Advantage Regularization loss (EAR), to the original gradient of the variance parameters:

$$g'_\Sigma = g_\Sigma + \beta_{exc}\mathbb{E}\left[\max\left(0, I(\hat{A}^{GAE} > 0)\hat{A}^{GAE}\nabla_\Sigma \log \pi(a|s)\right)\right] \tag{3.9}$$

where β_{exc} is the weight controlling the bias on exploration, and $I(.)$ is the indicator function, which returns 1 if the expression is true, otherwise 0.

The EAR loss add weights for the positive gradients of the variance parameters of the samples with positive advantage values. The reason behind the loss is that we want to add more importance to the samples which produce exceptionally positive advantage value, but with low sampling likelihood. Therefore we first only focus on points with positive advantage values. The problem remaining is to identify the points with low likelihood, namely "exceptional" points. We simply choose these points that produce positive gradients any of the variance parameters.

3.4 Efficient Exploration Through Robust Concentric Mixture Gaussian Policy

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

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

$$\pi(a|s) = (1 - \alpha_{ex})\mathcal{N}(\mu, \Sigma) + \alpha_{ex}\mathcal{N}(\mu, q_{ex}\Sigma) \tag{3.10}$$

where the constant α_{ex} is the weight of second distribution, which for example can be set at 0.05, and the constant $q_{ex} > 1$, for example $q_{ex} = 5$, controls the standard deviation of the second deviation.

Although the KL divergence between the two RCMG policy doesn't have a analytical form, the Wasserstein-2 distance between two RCMG policy can be given by:

$$W_2(\pi_0(a|\mu_0, \Sigma_0), \pi_1(a|\mu_1, \Sigma_1) = \\ (1 - \alpha_{ex}) W_2\Big(\mathcal{N}(\mu_0, \Sigma_0), \mathcal{N}(\mu_1, \Sigma_1)\Big) + \alpha_{ex} W_2\Big(\mathcal{N}(\mu_0, q_{ex}\Sigma_0), \mathcal{N}(\mu_1, q_{ex}\Sigma_1)\Big) \tag{3.11}$$

The RCMG policy is much more robust than an ordinary Gaussian policy because it has much longer tails.

3.5 Efficient training of actuator policies

We have found that the straight-forward reuse of source task policies in the target task may not work.

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" fails to perform properly, because the initial state is sampled from the states generated by "move1" policy, rather than the original initial state distribution.

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 makes the training much slower and much more ineffective.

There is a study [36] that proposes a method to improve the robustness of policies. The method basically artificially adds variance to the environments, so that the trained policy will be able to deal with a larger variety of circumstances. However, this method doesn't work in our case. Because simply increasing the variance of initial states will unnecessarily lead the agent to start with many unstable states that would otherwise not be encountered, and it is infeasible to manually design the generative model for all the possible states in the target task.

3.5.1 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 a experience buffer that store its state history. At the beginning of each episode, the actuator environment randomly selects the initial state from the set 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. The proposed method is also more efficient than the direct joint training method, and doesn't require domain-specific engineering.

One of the problems related to the joint training of multiple tasks is 'strategy collapse'. This phenomenon refers to the status where the agents converge to sub-optimal policies due to their policy collectively form a bad equilibrium. We propose a method, namely "synchronous scheduling of actuator learning ", to solve this problem. The method actively control the training schedule of actuator agents. In our setting, we pause the training of an actuator agent if it largely outperforms the agent with lowest performance until all the other agents have reached the performance level. This method requires a universal definition of performance level for all the source tasks, which becomes 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 different specifications.

3.6 Hierarchical reinforcement learning architecture

We propose to solve the reinforcement learning problem by a two-level hierarchical model.

The hierarchical model consists of a top-level decider agent and a set of bottom-level actuator agents. The actuator agents' policies are trained im the source task environments.

The decider agent takes an action at every time-step. It may either decide which actuator-policy should be executed, or simply skip and continue current actuator-policy. Therefore, assume there are n_a sub-policies, the action space of the root agent is an $(n_a + 1)$-discrete action space.

The observation space of the root agent consists of 2 parts, original statn (motion-sensor observation, image observation) and the meta state. The meta observation state the current sub-policy being executed and the number of time-steps since the last decision has been made.

Empirically, the decider agent is parameterized by two policy networks, θ_s and θ_d. The network θ_s, namely switcher network, outputs a binary action that decides whether the agent should simply continue using the current acting actuator policy, or make the full decision based on the current state. The network θ_d, namely decider network, outputs an n_a-discrete action space, that select the acting actuator agent.

The overalll decision-making process of the decider agent is shown in Algorithm 1.

Algorithm 1 The decider agent mechanism

```
function DECIDERACT(self, s_t)
    a_decider ~ π_decider(s_t)
    if a_decider ≠ 0 then
        self.currentActuator ← self.allActuators[a_decider − 1]
    a_actuator ← self.currentActuator.act(s_t)
    return a_actuator
```

3.7 Generalized advantage estimation for the decider agent

We propose a generalized advantage estimation method for decider agents hierarchical reinforcement learning agents.

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

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

$$\bar{r}_{t_i} := \sum_{l=0}^{t_{i+1}-t_i-1} \gamma^l r_{t_i+l} \tag{3.12}$$

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

$$\delta_{t_i}^V := -V(s_{t_i}) + \bar{r}_{t_i} + \gamma^{t_{i+1}-t_i} V(s_{t_{i+1}}) \tag{3.13}$$

Then the k-step advantage estimation is given by:

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

$$\hat{A}_{t_i}^{(2)} := \delta_{t_i}^V + \gamma^{t_{i+1}-t_i}\delta_{t_{i+1}}^V = -V(s_{t_i}) + \bar{r}_{t_i} + \gamma^{t_{i+1}-t_i}\bar{r}_{t_{i+1}} + \gamma^{t_{i+2}-t_i} V(t_{i+2}) \tag{3.15}$$

$$\hat{A}_{t}^{(3)} := \delta_{t}^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V = -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3}) \tag{3.16}$$

$$
\begin{aligned}
\hat{A}_{t_i}^{(k)} &:= \sum_{d=0}^{k-1} \gamma^{t_{i+d}-t_i} \delta_{t_{i+d}}^V \\
&= -V(s_t) + \bar{r}_{t_i} + \gamma^{t_{i+1}-t_i} \bar{r}_{t_{i+1}} + \cdots + \gamma^{t_{i+k-1}-t_i} \bar{r}_{t_{i+k-1}} + \gamma^{t_{i+k}-t_i} V(s_{t_{i+k}})
\end{aligned} \tag{3.17}
$$

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

$$
\begin{aligned}
\hat{A}_{t_i}^{GAE_{unnorm}(\lambda)} &:= \hat{A}_{t_i}^{(1)} + \lambda^{t_{i+1}-t_i} \hat{A}_{t_i}^{(2)} + \lambda^{t_{i+2}-t_i} \hat{A}_{t_i}^{(3)} + \cdots + \lambda^{t_{i+k-1}-t_i} \hat{A}_{t_i}^{(k)} \\
&= \delta_{t_i}^V + \lambda^{t_{i+1}-t_i} (\delta_{t_i}^V + \gamma^{t_{i+1}-t_i} \delta_{t_{i+1}}^V) \qquad (3.18) \\
&\qquad + \lambda^{t_{i+2}-t_i} (\delta_t^V + \gamma^{t_{i+1}-t_i} \delta_{t_{i+1}}^V + \gamma^{t_{i+2}-t_i} \delta_{t_{i+2}}^V) + \ldots \\
&\qquad + \lambda^{t_{i+k-1}-t_i} \sum_{d=0}^{k-1} \gamma^{t_{i+d}-t_i} \delta_{t_{i+d}}^V \qquad (3.19) \\
&= (\delta_{t_i}^V \sum_{b=0}^{k-1} \lambda^{t_{i+k-1}-t_i} + \gamma^{t_{i+1}-t_i} \delta_{t_{i+1}}^V \sum_{b=1}^{k} \lambda^{t_{i+b}-t_i} \\
&\qquad + \gamma^{t_{i+2}-t_i} \delta_{t_{i+2}}^V \sum_{b=2}^{k} \lambda^{t_{i+b}-t_i} + \ldots \qquad (3.20) \\
&\qquad + \gamma^{t_{i+k-1}-t_i} \delta_{t_{i+k-1}}^V \lambda^{t_{i+k-1}-t_i}) \\
&= \sum_{d=0}^{k-1} \delta_{t_i}^V \gamma^{t_{i+d}-t_i} \sum_{b=0}^{k-1} \lambda^{t_{i+b}-t_i} \qquad (3.21)
\end{aligned}
$$

The normalized generalized advantage estimater is then given by:

$$
\hat{A}_{t_i}^{GAE(\lambda)} = \frac{\hat{A}_{t_i}^{GAE_{unorm}(\lambda)}}{\sum_{b=0}^{k-1} \lambda^{t_{i+b}-t_i}} \tag{3.22}
$$

However, 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 of the advantages. This pratical method usually lead to large advantage scales for experience data at the beginning of the episodes and small scales for the experience data near episode ends.

3.8 Training of the switcher agent

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

$$
J'(\theta_s) = J'(\theta_s) + \mathbb{E}[\pi_s(s)] \tag{3.23}
$$

Where π_s is the output of the switcher network. The second term is the expected frequency that the switcher network produces the output 1, which indicates the termination of the current actuator policy.

Chapter 4

Experiments

4.1 Experiments on the basic source tasks

We discuss and compare the methods in solving the basic task ”move0” in this section.

As is observed by many studies [32], one of the disadvantages with the traditional KL-divergence based is that the performance result could be inconsistent across different runs, due to the agent getting stuck at local minimum. One example is the performance of the ACKTR [9] method on the ”move0” task. An example result is shown in Figure 4.1, where we train the same ACKTR agent on the task ”move0” with batch-size 8000, KL-divergence 0.0001 and 20 parallel agents. It appears that one agent gets stuck at the a score below 300 while another continues to improve after reaching the score of 4000.

We observe that the conventional KL-divergence based trust region policy gradient methods in general could get stuck easily at some local minimum before the policy converges to the optimal solution. An experiment showing the performance of the original ACKTR algorithm is shown in Figure 4.2. We observe that the agents tend to get stuck around the total return of 3000. The reason might be that while the KL-divergence based trust-region method provides a theoretical guarantee on the monotonic improvement of performance, the agent might converge too early at local minimums and fails to make efficient exploration.

We propose that the proposed W-KTR method is less prune to the local minimum problem on the task ”move0”. The experiment comparing the performance of W-KTR agents with different parameters is shown in Figure 4.3. We can see that none of the agents get stuck at the local minimum around 3000, and their final performance can reach a total return of 6000. However, the W-KTR agent appears to have a much slower

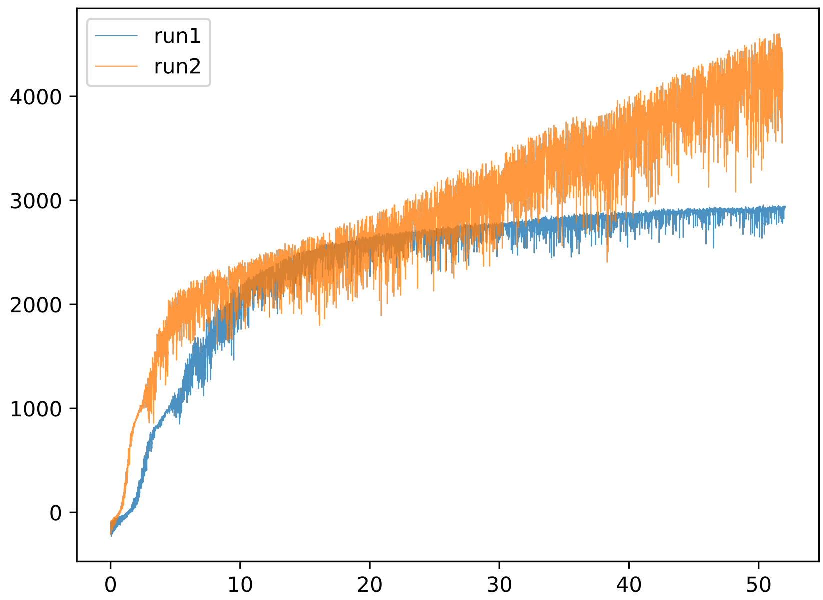

Figure 4.1: Inconsistent performance produced by ACKTR agents with the same parameters but different runs. The vertical axis is the total return averaged over the recent 20 episodes and the horizontal axis is the number of million time-steps

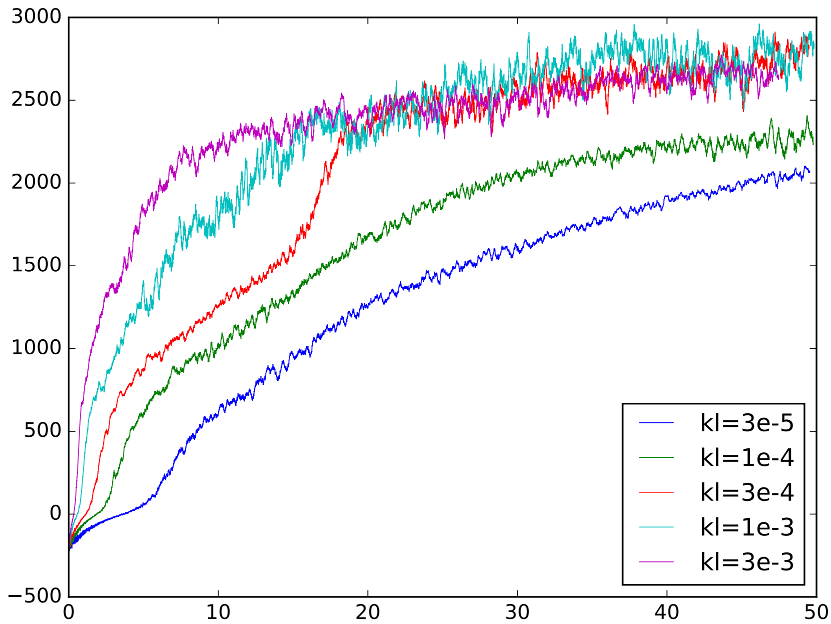

Figure 4.2: Performance of ACKTR agents with different KL-divergence constraints. All the agents are trained with batch-size 4000 and 20 parallel agents. The vertical axis is the total return averaged over the recent 200 episodes and the horizontal axis is the number of million time-steps

improvement rate before 50 million time-steps.

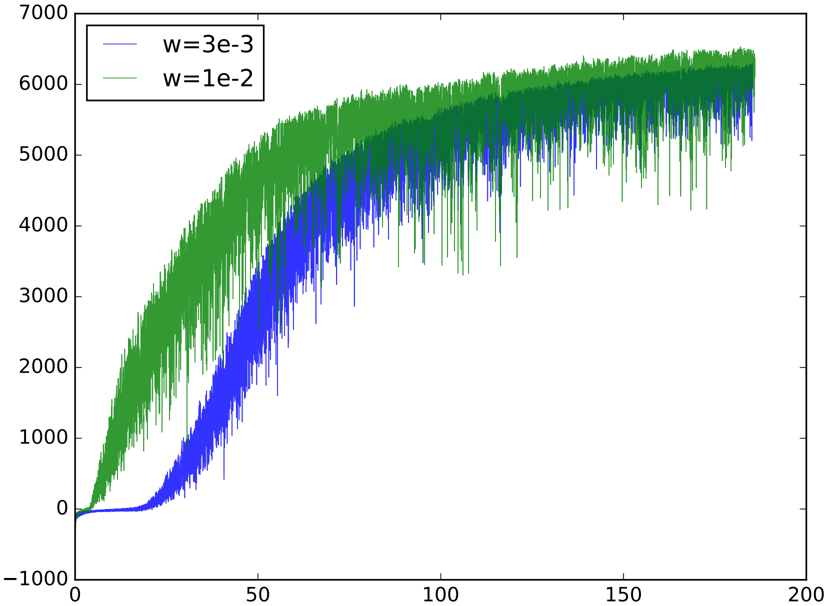

Figure 4.3: Performance of W-KTR agents with different W2-metric constraints. All the agents are trained with batch-size 4000 and 20 parallel agents. The vertical axis is the total return averaged over the recent 200 episodes and the horizontal axis is the number of million time-steps

One proposal to solve the slow initial improvement rate problem of W-KTR method is to apply a decaying Wasserstein constraint at the beginning. The performance of a proposed W-KTR agent with decaying Wasserstein constrainet is shown in Figure 4.4. The agent can achieve much faster improvement rate in the first 50 million time-steps. The limitation is that there are more hyper-parameters to be tuned in this algorithm.

In conclusion, we have verified that the proposed W-KTR method can achieve the state-of-art final performance in the basic ”move0” task, and is able to produce consistent final results across different runs.

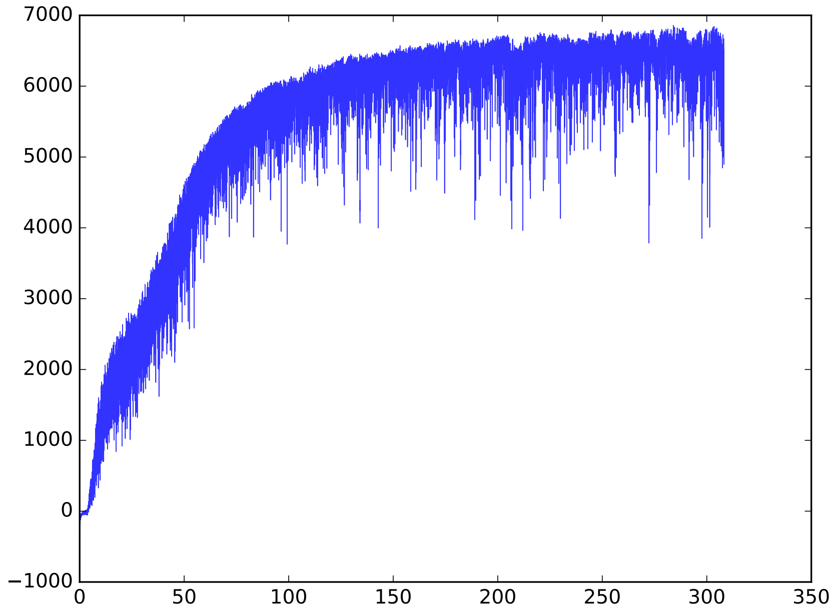

Figure 4.4: Performance of a W-KTR agent with a decaying W2-metric from 0.02 to 0.00003 in the first 15 million time-steps. The agent is trained with batch-size 4000 and 20 parallel agents. The vertical axis is the total return averaged over the recent 200 episodes and the horizontal axis is the number of million time-steps

4.2 Experiment on the flat reinforcement learning solution to multi-modality tasks

4.2.1 Discussion on conventional flat reinforcement learning methods

We observe contemporary end-to-end flat reinforcement learning methods fail to achieve good performance. We suggest that one of the important reason is the image features are much harder to learn than the state. The agent would tend to stuck at a local minimum when it has already learned the state features well, but has not been able to extract any useful information from the image. However, when the agent has finally learned the image features, the policy already converge to a relatively low-entropy distribution, and contemporary reinforcement learning methods cannot perform exploration again at this phase.

We 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 indicate the goal direction from the image, where there is a sphere object at the corresponding position.

A conventional method for encouraging exploration of reinforcement learning agents with continuous action space is entropy regularization. As we discussed in the section 3.3, the entropy regularization method is not an effective method in the general case because it simply adds constant biases on the policy parameters and may dominate the learning of the original task.

An experiment on the effectiveness of entropy regularization is shown in Figure 4.5. The agents are trained using ACKTR algorithm with 32 parallel agents, minibatch-size 2560 and KL-divergence constraint 0.0003. The result shows that the agent easily fails to learn the original task when the weight on entropy term is too large.

The change of average Standard Deviation of the policy distributions is shown in Figure 4.6. The agents' policy distributions appear to get stuck at certain level of standard deviation when the entropy term is not optimal.

4.2.2 Experiment on exceptional advantage regularization

We discuss the effectiveness of exceptional advantage regularization method on multi-modality tasks in this section.

We first demonstrate the general patterns of the distribution on the advantage values on the simple multi-modality task "moveg2". In this task, the goal direction is sampled from 2 opposite directions. The task is much more simple than the "movecont" task because the agent only needs to learn from the image whether it needs to go forward or backward.

The performance on the total return of an ACKTR agent is shown in Figure 4.7. The performance in terms of average reward per time-step is shown in Figure 4.8, and the average standard deviation is shown in Figre 4.9.

The distribution of advantage values at the first batch (batch 0) is shown in Figure 4.10. It can be seen that the advantage values are distributed uniformly across different values of log-likelihood because the critic model has not been trained. After the critic model has been trained for a reasonable amount of time, the marginal distribution advantage value tend to follow a normal distribution with zero mean. An example is shown in Figure 4.11, which is the distribution at batch 3000. However, the advantage values are likely to have higher values at low log-likelihood samples when the agent has just escaped from a local

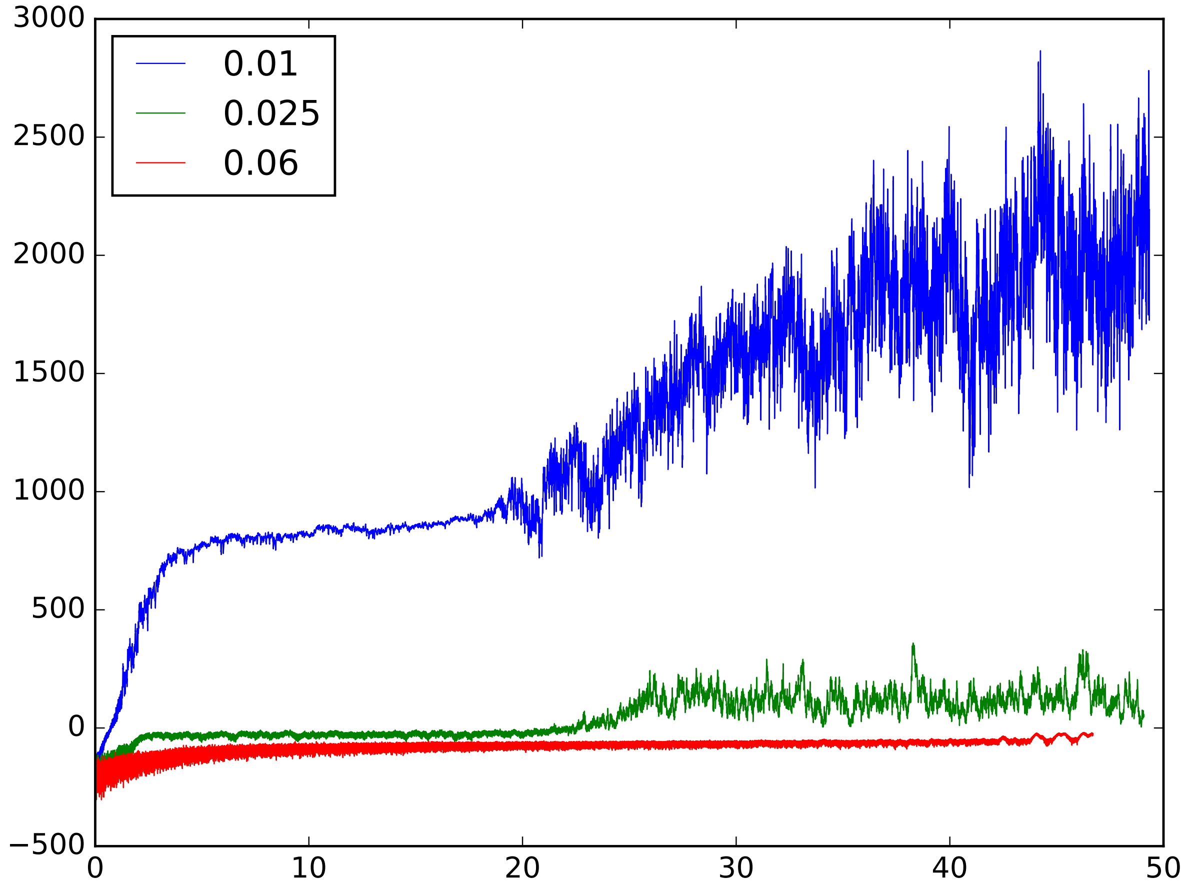

Figure 4.5: Performance of agents with different weight on entropy regularization term, the horizontal axis is the number of million timesteps and the vertical axis is the total episode reward averaged over the last 32 episodes

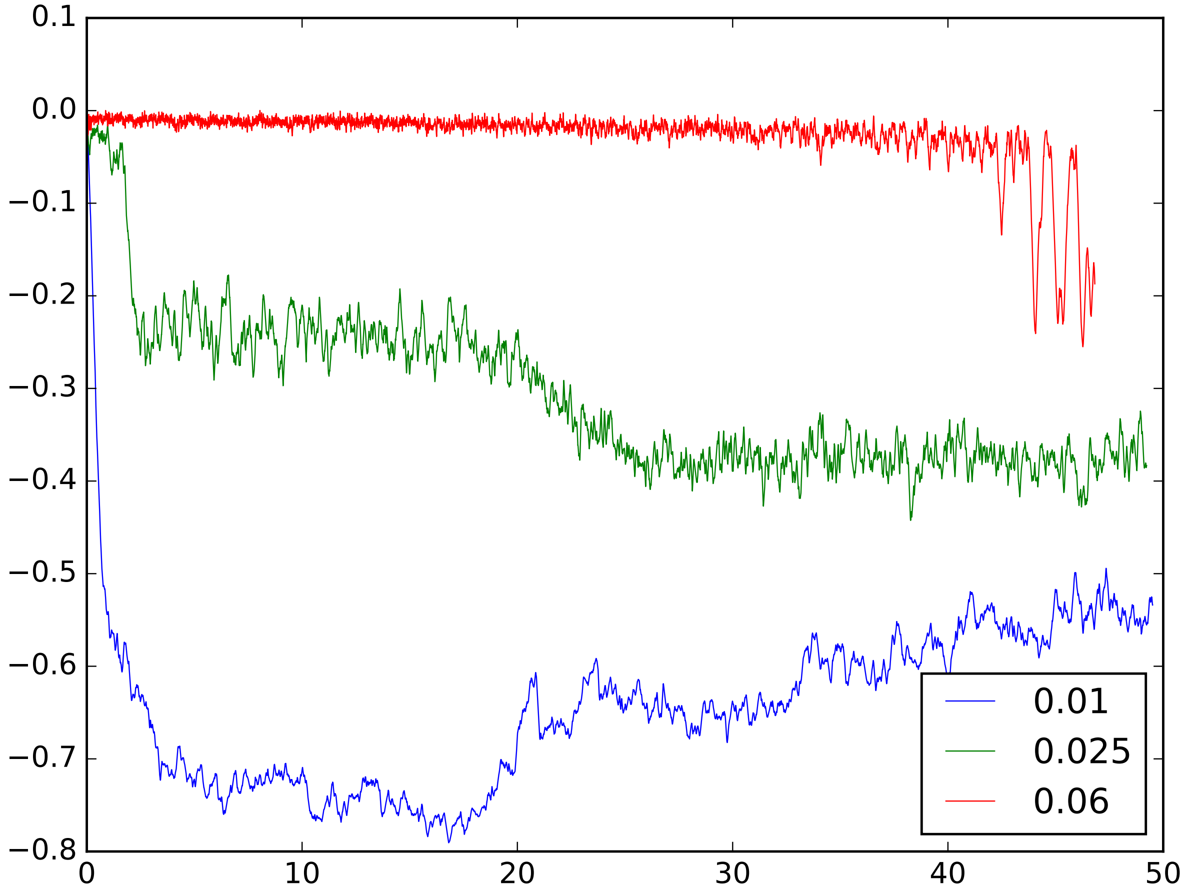

Figure 4.6: Logarithm of average standard deviation of the policy of agents in Figure 4.5, the horizontal axis is the number of million timesteps.

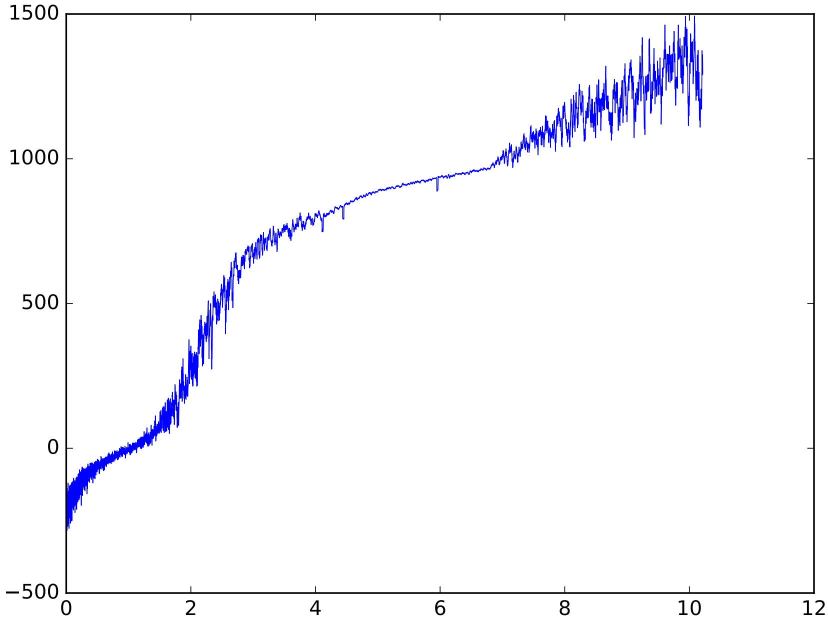

Figure 4.7: Performance of ACKTR agent on the "moveg2" task, the horizontal axis is the number of million timesteps and the vertical axis is the total episode reward averaged over the last 20 episodes

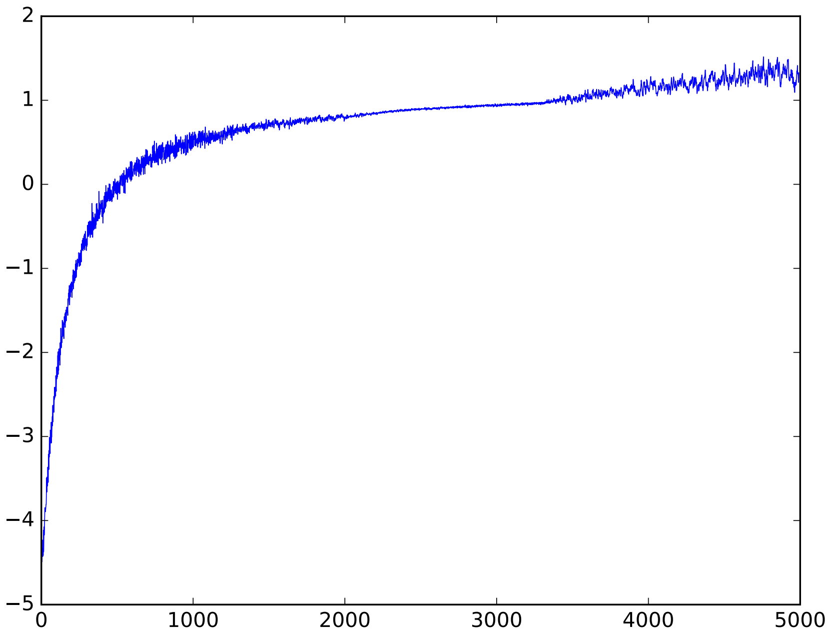

Figure 4.8: The avearge reward per time-step of ACKTR agent on the "moveg2" task, the horizontal axis is the number of training batches

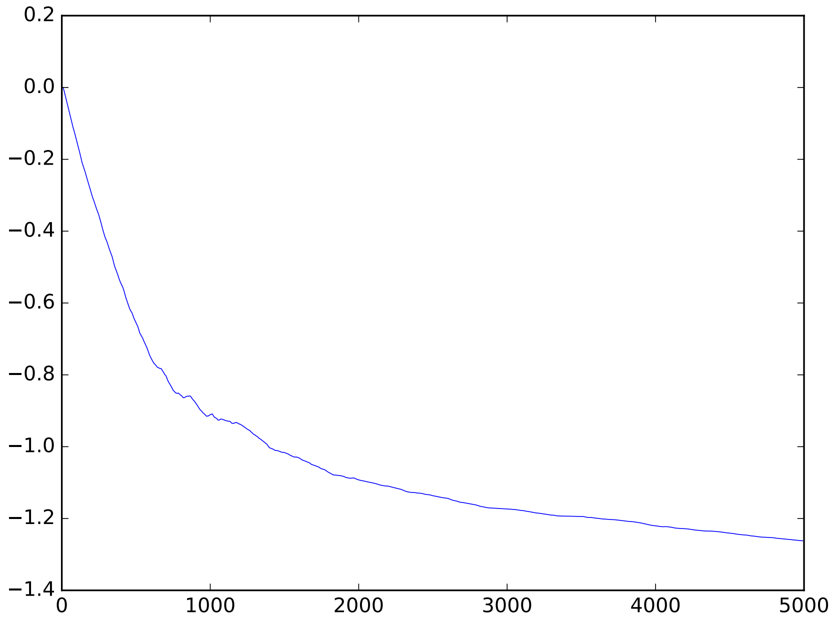

Figure 4.9: The logarithm of avearge standard deviation of ACKTR agent's policy on the "moveg2" task, the horizontal axis is the number of training batches

minimum. An example is Figure 4.12, which shows the the distribution of advantage values at batch 4900. It can be seen that the distribution becomes significantly different because many positive advantage points are spread across different log-likelihoods.

Intuitively, a large number of points with low log-likelihood but high advantage value indicates that the agent needs to increase the degree of exploration. However, the figure 4.9 on the change of standard deviation shows that the agent actually still keep decreasing the standard deviations of its policy even at this state. Therefore, the application of a exceptional advantage regularization term, could be useful in encouraging exploration in these cases.

We then verify the performance of exceptional advantage regularization on the 'movecont' task. The performance of an ACKTR agent with different weights exceptional advantage regularization on the in Figure 4.13, and the average standard deviation parameter of their policies are shown in Figure 4.14. 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

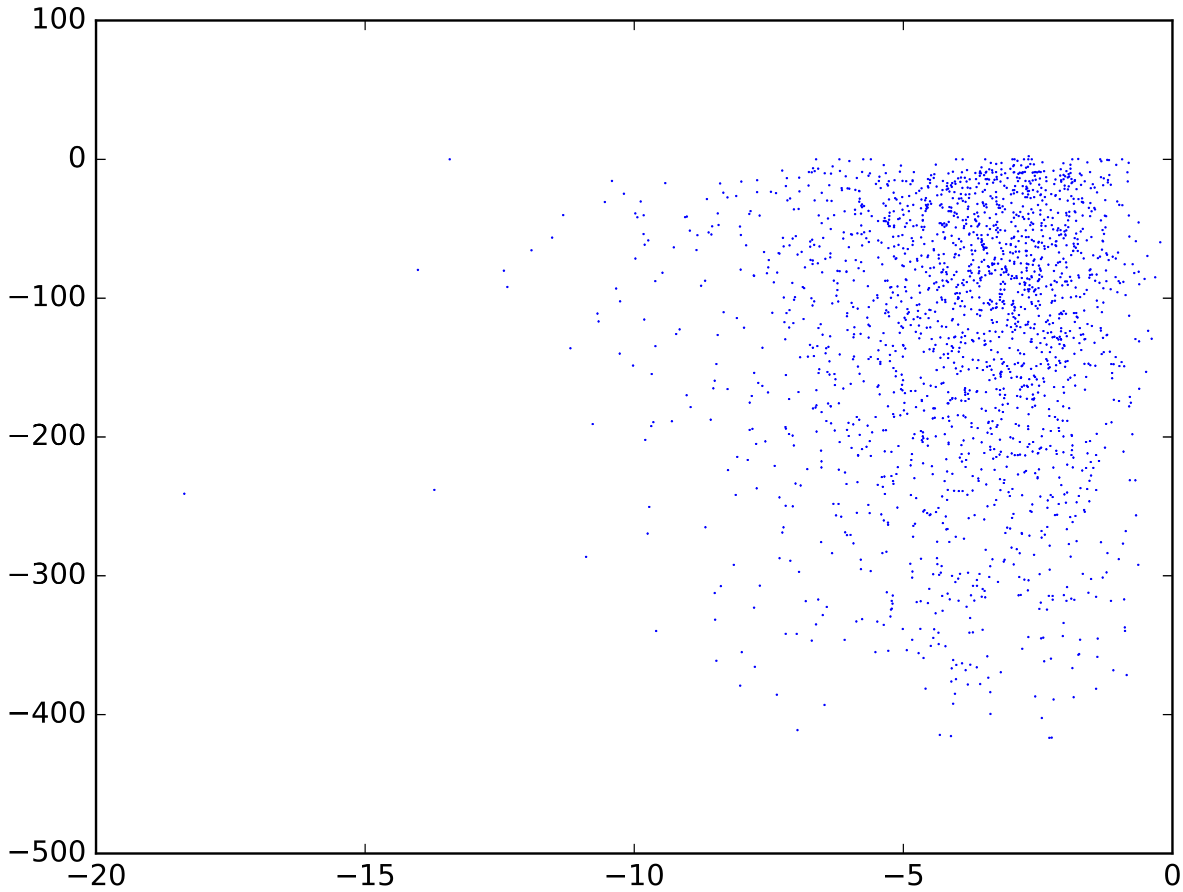

Figure 4.10: The distribution of advantage values of the ACKTR agent at batch 0 on the ”moveg2” task, the horizontal axis is the log-likelihood value

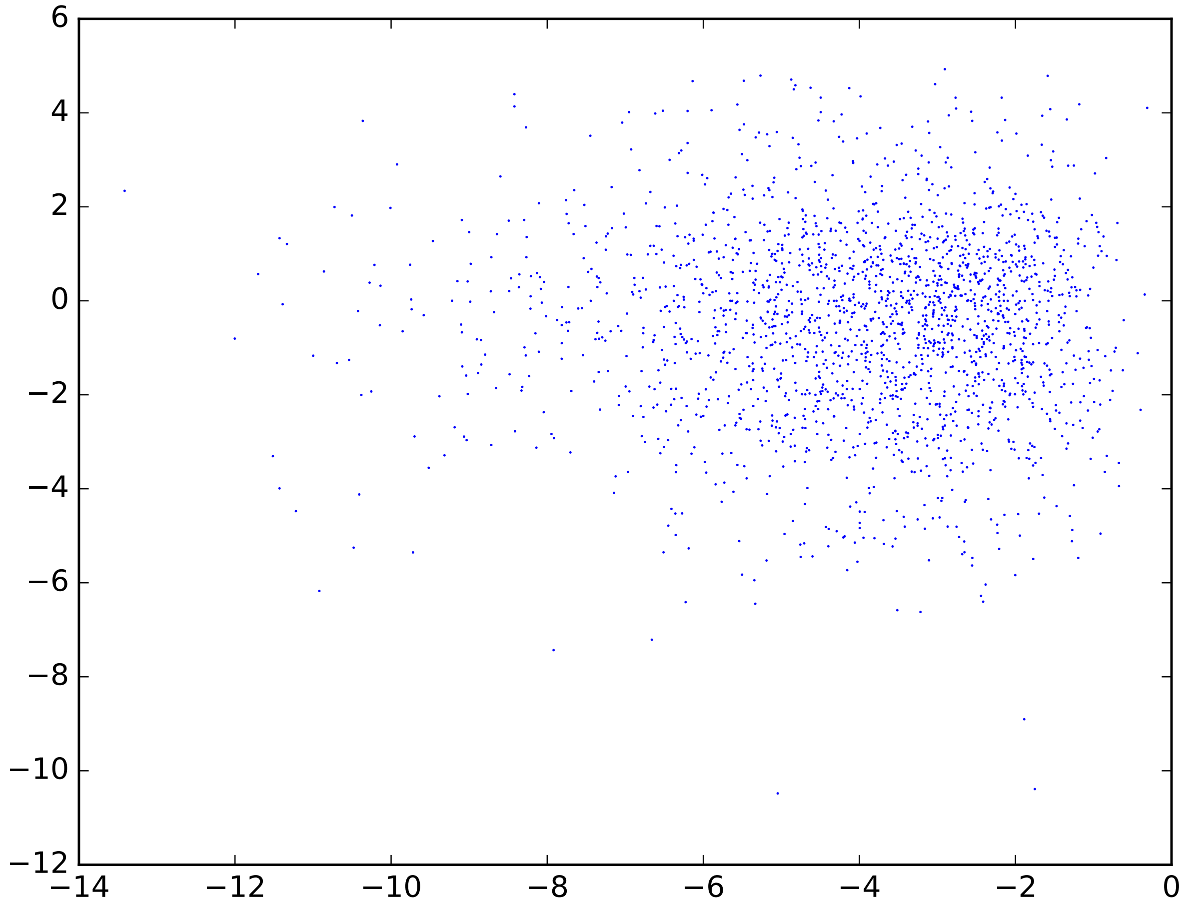

Figure 4.11: The distribution of advantage values of the ACKTR agent at batch 300 on the ”moveg2” task, the horizontal axis is the log-likelihood value

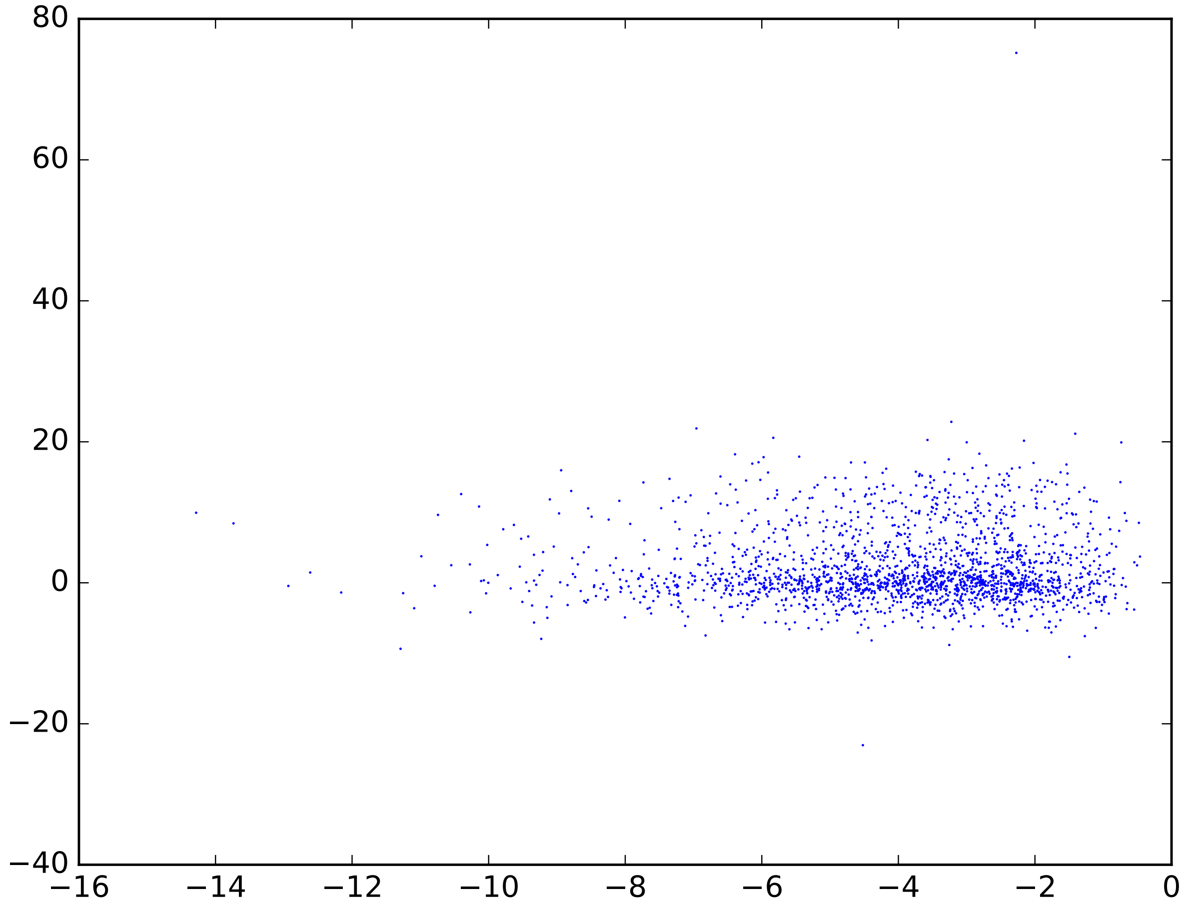

Figure 4.12: The distribution of advantage values of the ACKTR agent at batch 4900 on the "moveg2" task, the horizontal axis is the log-likelihood value

method could have adverse effect on the convergence of policy in the late phase of training, which is also indicated in by average std curve. 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.

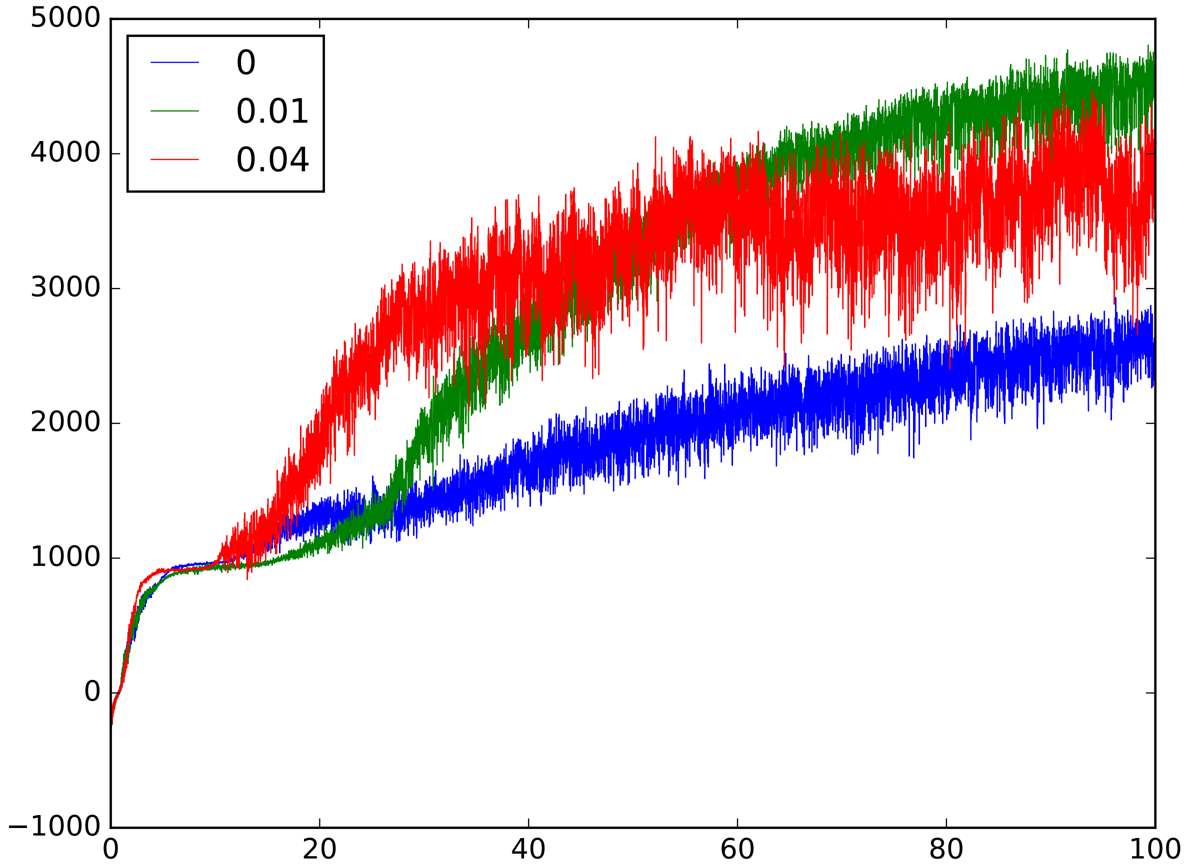

Figure 4.13: Performance of agents with different exceptional advantage regularization weights, the x-axis is the number of million time-steps and the y-axis is the total episode reward averaged over the last 32 episodes

4.2.3 Experiment on the Robust Concentric Mixture Gaussian Policy

The effectiveness of robust concentric mixture Gaussian policy agent in exploration of task ”movecont” is verified in this section.

The performance of an ACKTR mixture Gaussian policy agent is shown in Figure 4.15. The results shows that the mixture Gaussian policy agents have slower learning rates

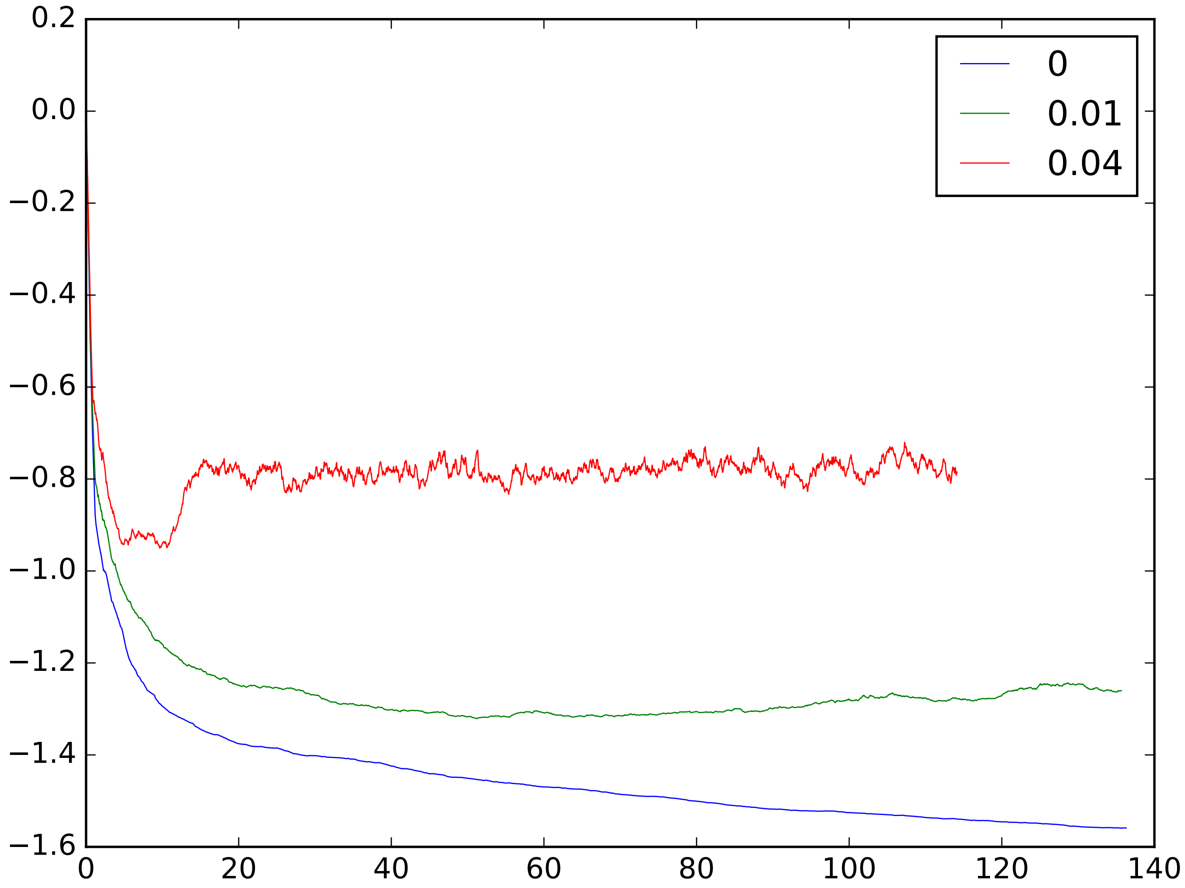

Figure 4.14: The logarithm of avearge standard deviation of agents with different exceptional advantage regularization weights, the horizontal axis is the number of million time-steps and the vertical axis is the total episode reward averaged over the last 32 episodes

compared to pure Gaussian policy agents. However, the agents are able to achieve a good final performance, with a total reward of around 4000 when the KL-divergence is set properly.

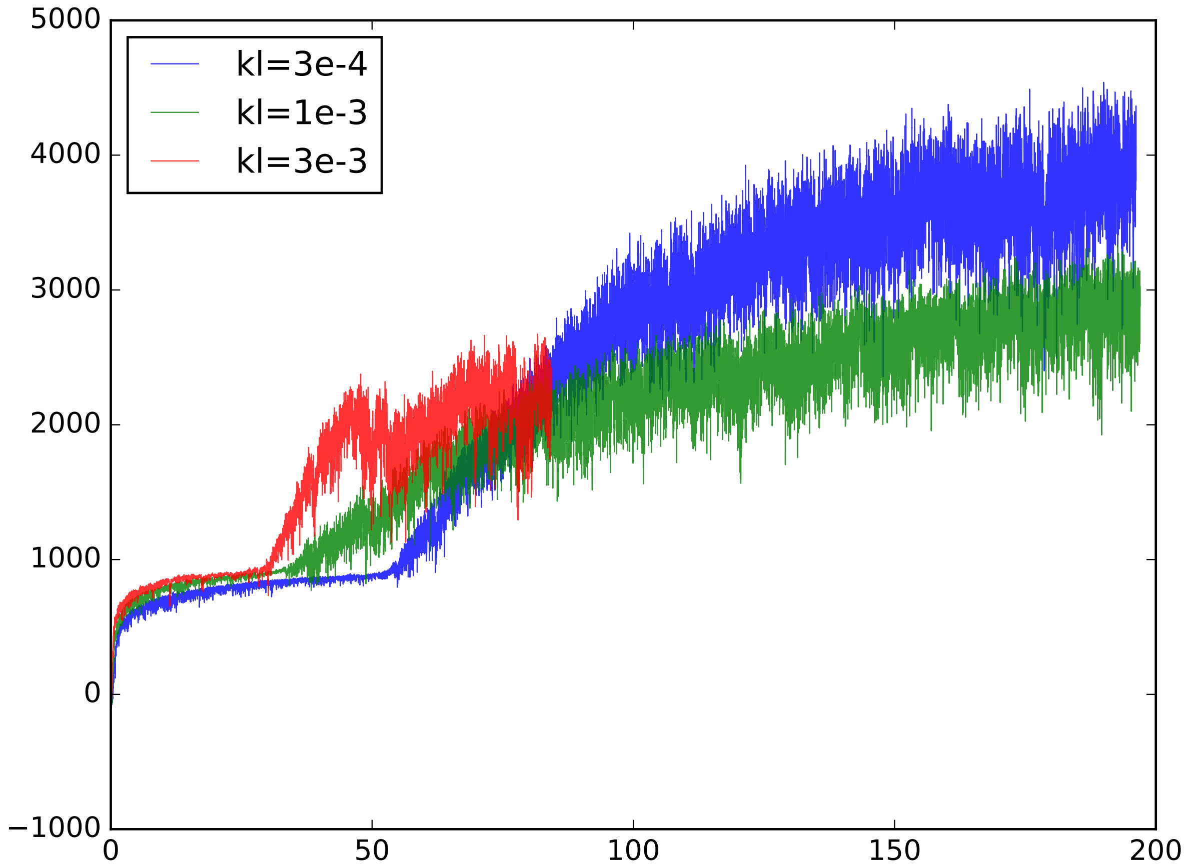

Figure 4.15: Performance of ACKTR agents with different KL-divergence constraints, the x-axis is the number of million time-steps and the y-axis is the total episode reward averaged over the last 32 episodes

4.3 Hierarchical reinforcement learning methods for multi-modality and sparse environments

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

There are two target tasks to be solved in our experiment setting: ”dynamicg8” and ”reachcont”. The target task ”dynamicg8” is a representative problem that has multi-modality state-space, 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 $\{move0, move1, \ldots, move7\}$ for both target tasks.

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 the following sections.

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

We proposed that the method "domain randomization by cross-sampling initial states" can handle the problem of novel initial states. The performance of this method is experimented in this section. We train the set of source tasks $\{move0, move1, \ldots, move7\}$ simultaneously in this experiment.

The result is plotted in Figure 4.16, which shows that the actuator agents for some tasks get stuck at sub-optimal scores and fail to solve the corresponding source tasks. As is discussed previously, this problem could be due to the interference between actuator agents.

Therefore, we proposed the "synchronous scheduling of actuator learning" method to prevent the learning progress of any actuator agents from lagging behind too far from others. The experiment results of the proposed "synchronous scheduling of actuator learning" is shown in Figure 4.17. In this experiments, the policy training of 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 reach a final performance of around 6000 although their performance diverge initially. This has verified that the proposed method can successfully prevent the trapped sub-optimal actuator agents.

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

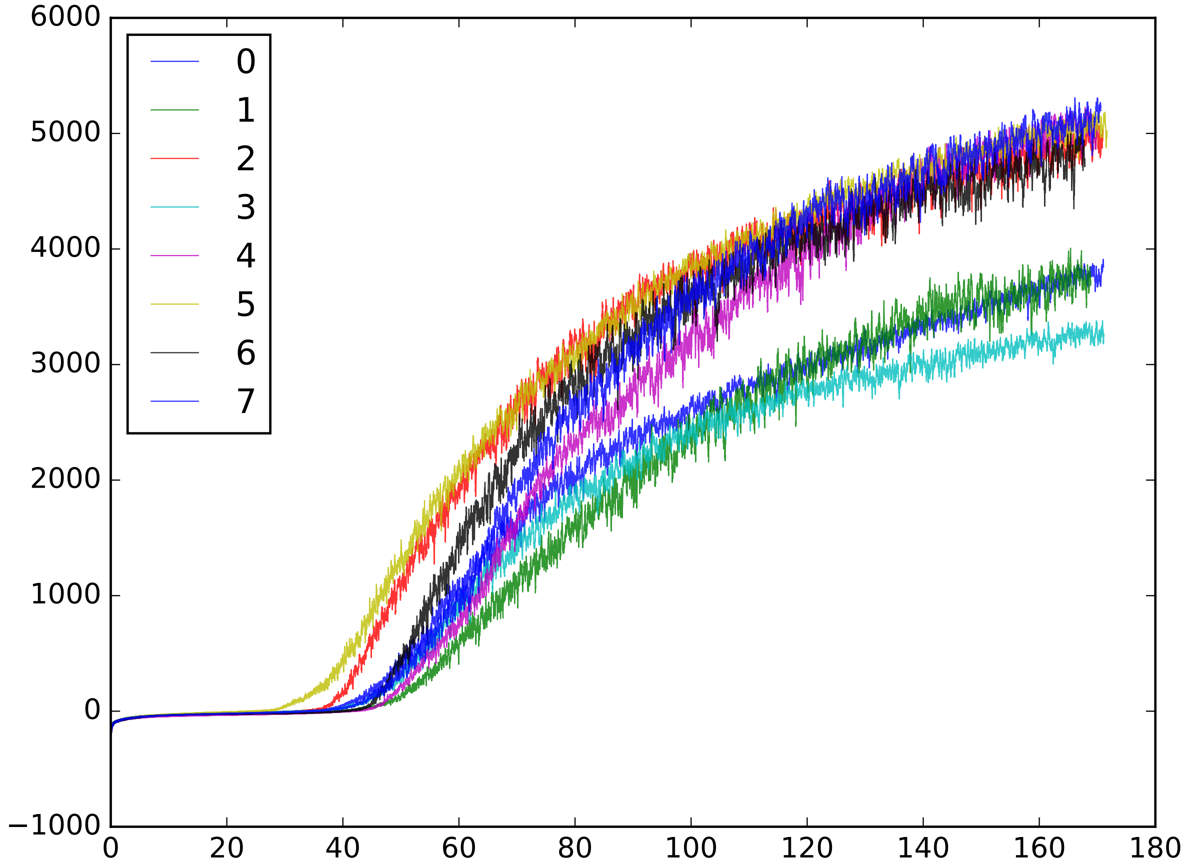

Figure 4.16: Performance of actuator agents with domain randomization by cross-sampling initial states, the x-axis is the number of million time-steps and the y-axis is the total episode reward averaged over the last 200 episodes

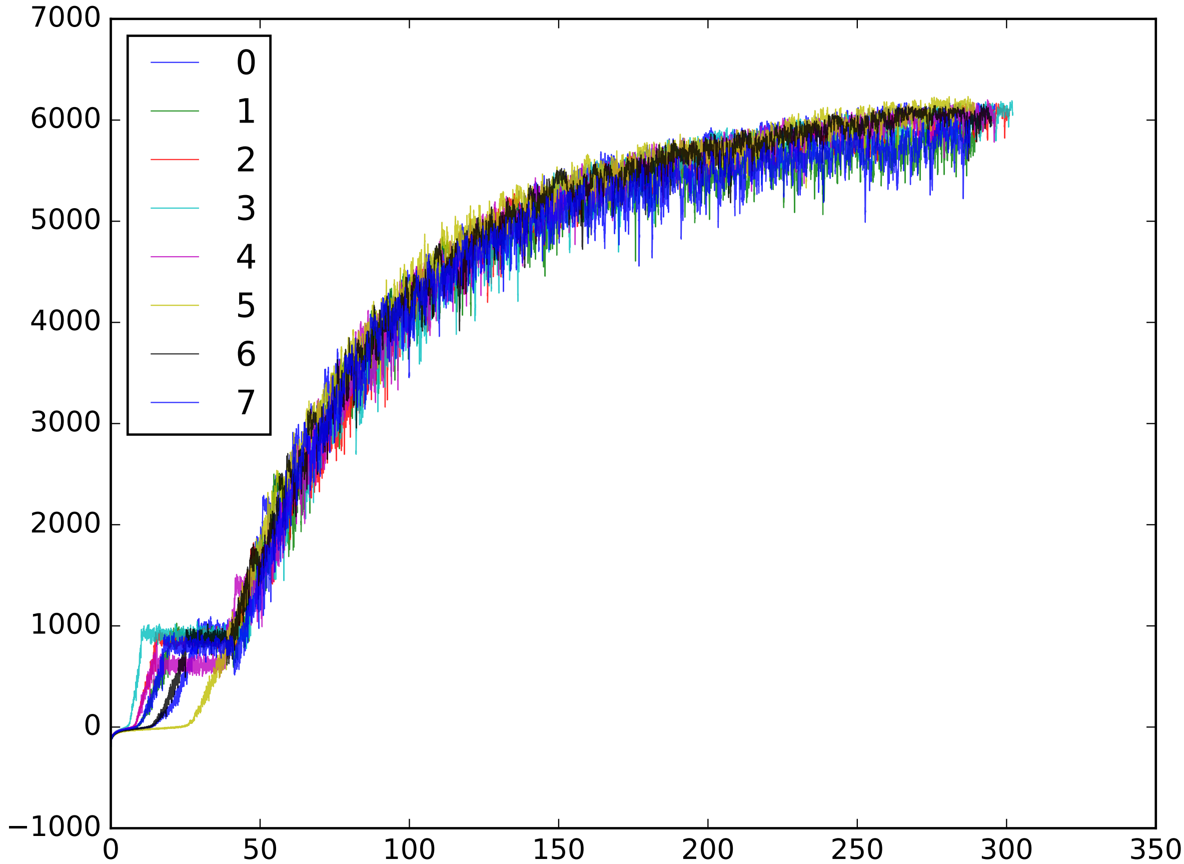

Figure 4.17: Performance of actuator agents using "synchronous scheduling of actuator learning" , the x-axis is the number of million time-steps and the y-axis is the total episode reward averaged over the last 200 episodes

4.3.2 Training the decider agent

The training of the decider agent consists of two phases: the training of the decision policy and the the switcher policy. We train the two parts in 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 no less 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 the performance of the decision policy becomes stable, the switcher policy is then trained with the decision policy fixed.

Phase 1: Decision policy training

The performance of the experiment on the training of decision policy on ”dynamicg8” is shown in Figure 4.18. The result shows that the agent is able to achieve a reasonable performance of around 2500, while it is not considered optimal.

The performance of the experiment on the training of decision policy on task ”reach-cont” is shown in Figure 4.19. The agent achieves a final score of around 0.75 and is considered.

The result of the training of the decision policy shows that the hierarchical reinforcement learning agent is able to successfully learn the decision policy. The result is as expected because previous works in SMDP have studied intensively on this problem.

Phase 2: Switcher policy training

The parameters of the decision policy is fixed in the second phase of the training, while the switcher policy is trained. The switcher policy is re-initialized randomly at the beginning of this phase, so that it outputs a termination signal with a probability 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 reinforcement learning performance during the training of switcher policy on the task ”dynamicg8” is shown in Figure 4.20. The decider agent appears to achieve a relatively stable performance during the training of the switcher policy, without any significant degradation from the final performance of phase 1.

The average execution length of the decision policies is plotted in Figure 4.21. The

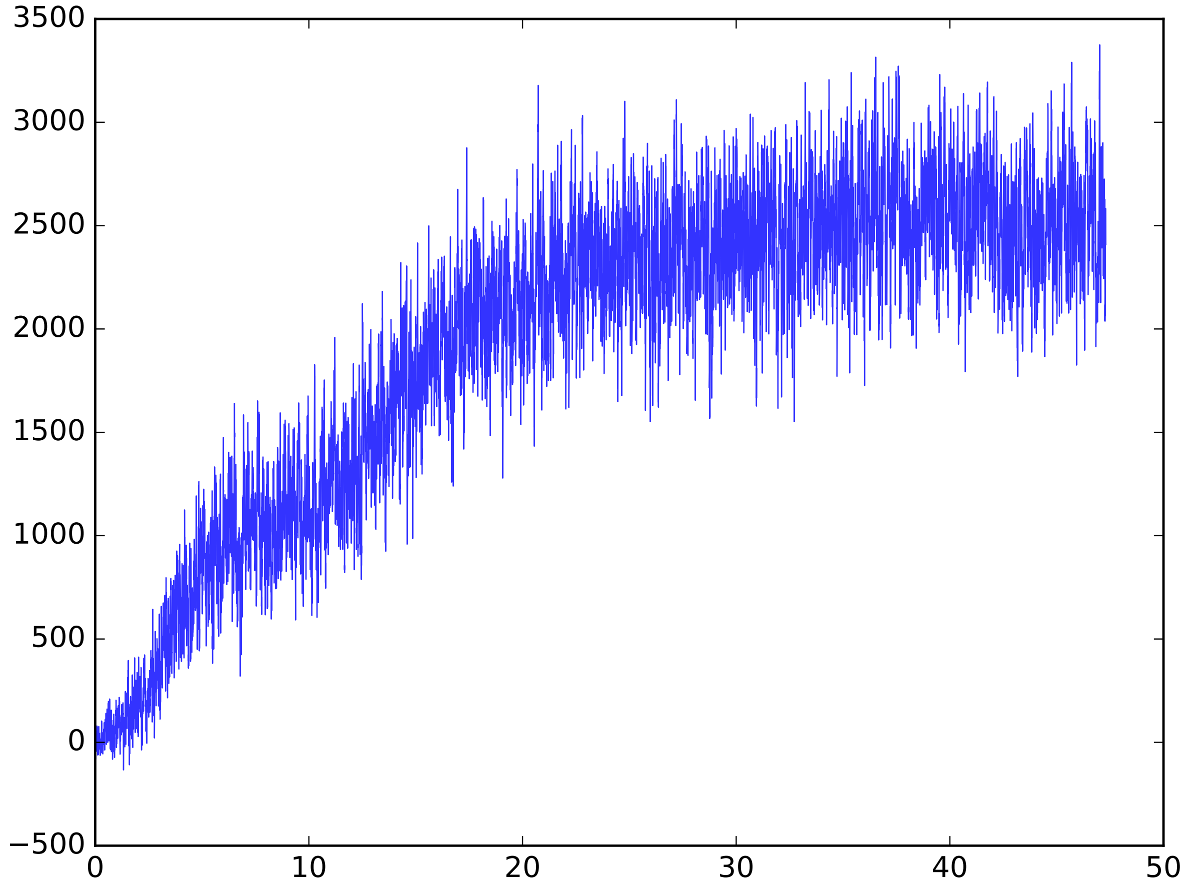

Figure 4.18: Decision policy training performance of the task "dynamicg8", the x-axis is the number of million time-steps and the y-axis is the total episode reward averaged over the last 32 episodes

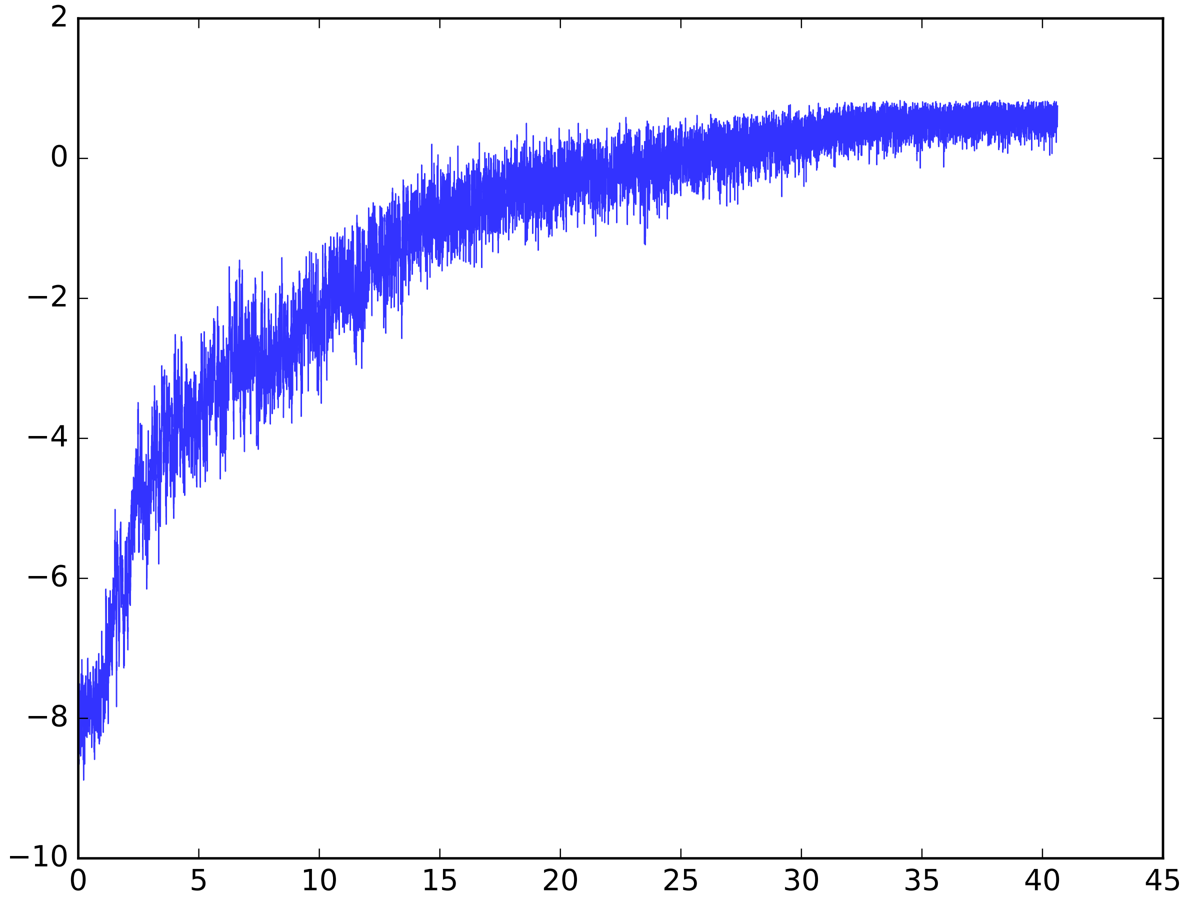

Figure 4.19: Decision policy training performance of the task ”reachcont”, the x-axis is the number of million time-steps and the y-axis is the total episode reward averaged over the last 200 episodes

agent manages to find a better switcher policy since the final average execution length becomes around 10 while it was around 5 in phase 1. As the result, the decision policy does not need to compute its output as frequently as in phase 1.

Therefore, the training of switcher policy as considered to be useful on the task ”dynamicg8”, in the sense that the agent can maintain a good performance as well as minimize the number of decisions being made by the decider agent.

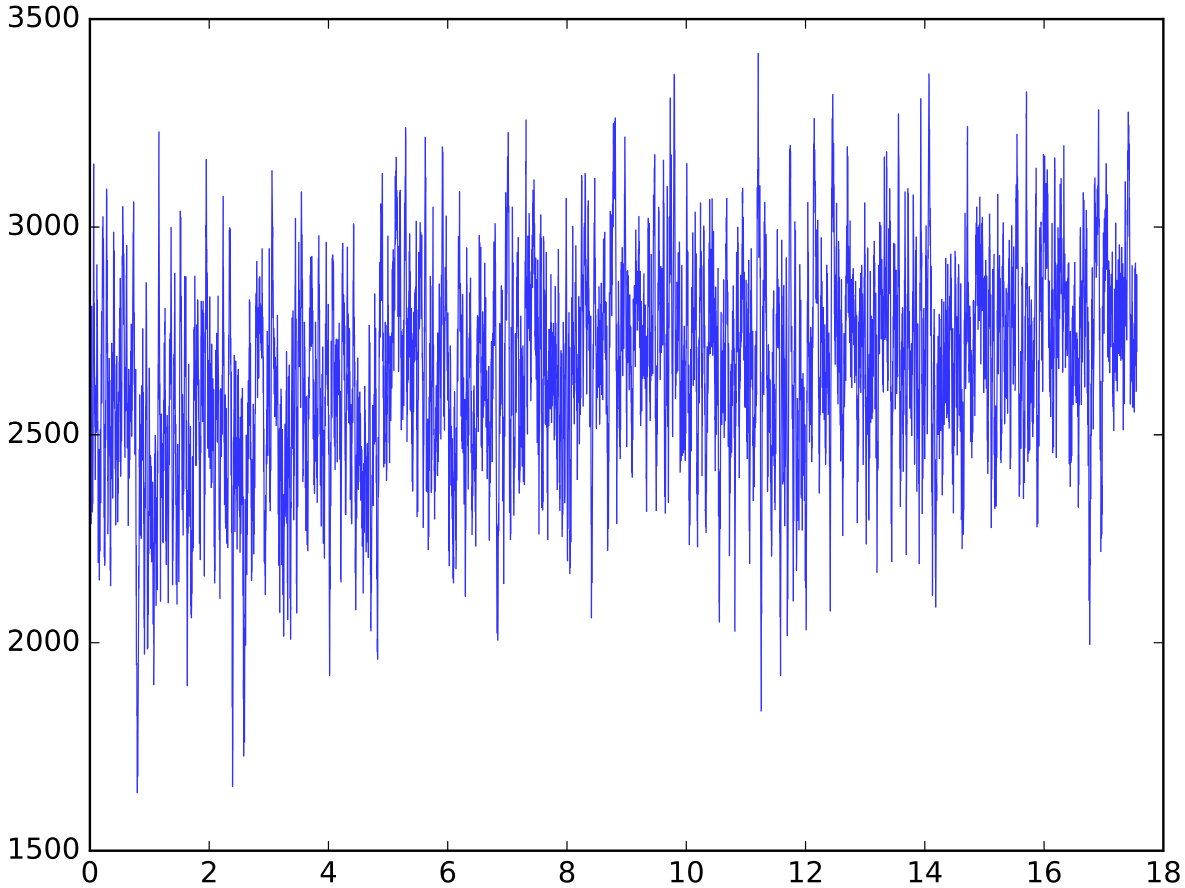

Figure 4.20: Switcher policy training performance of the task ”dynamicg8”, the x-axis is the number of million time-steps and the y-axis is the total episode reward averaged over the last 32 episodes

The reinforcement learning performance during the training of switcher policy on the task ”dynamicg8” is shown in Figure 4.22. 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 4.21. The final average value of the execution length appears to be reduced to 1.6, compared to 5

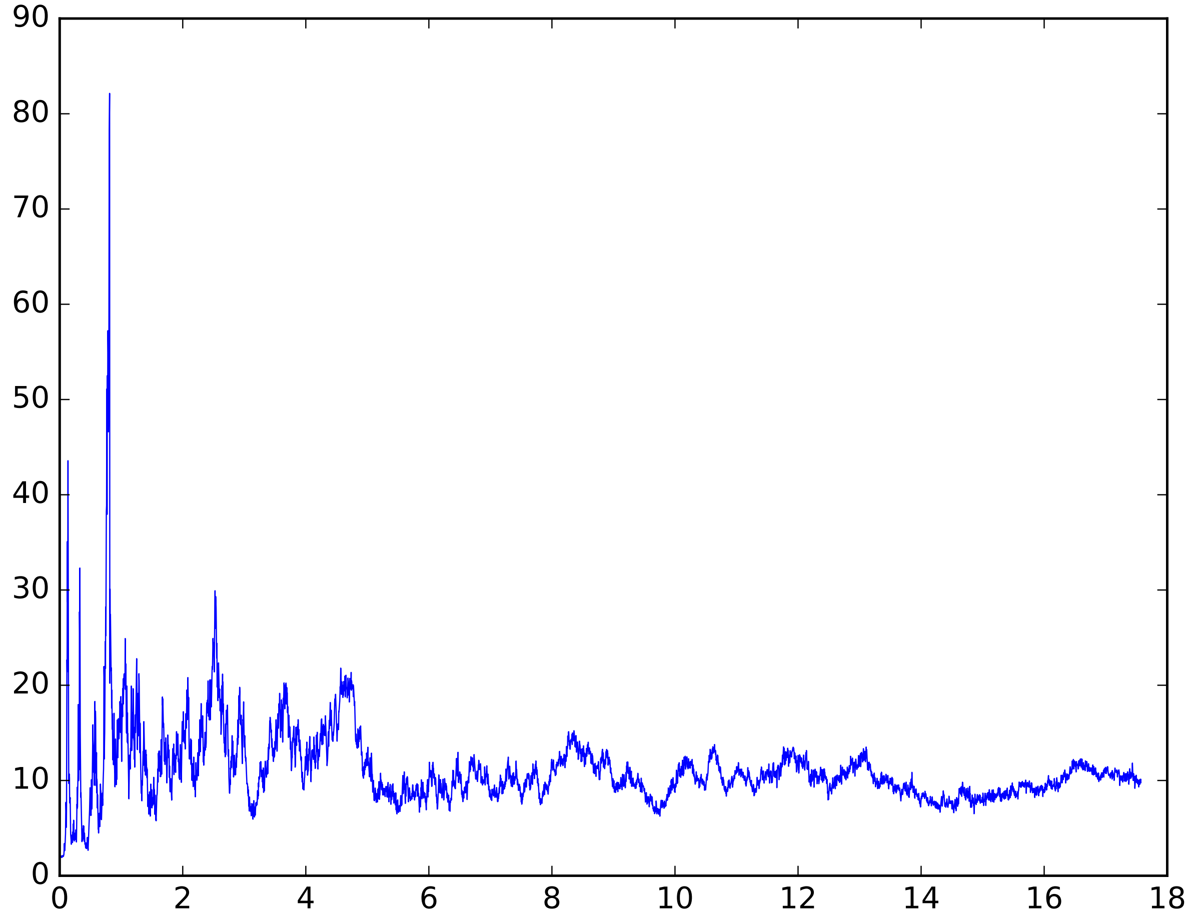

Figure 4.21: Average execution length of the actions of decision policies of the task "dynamicg8", the x-axis is the number of million time-steps and the y-axis is the decision policy's average execution length of the last batch.

in the decision policy training phase.

Therefore, the switcher policy is able to improve the reinforcement learning performance at the expense of the computation spent on the decision policy being increased.

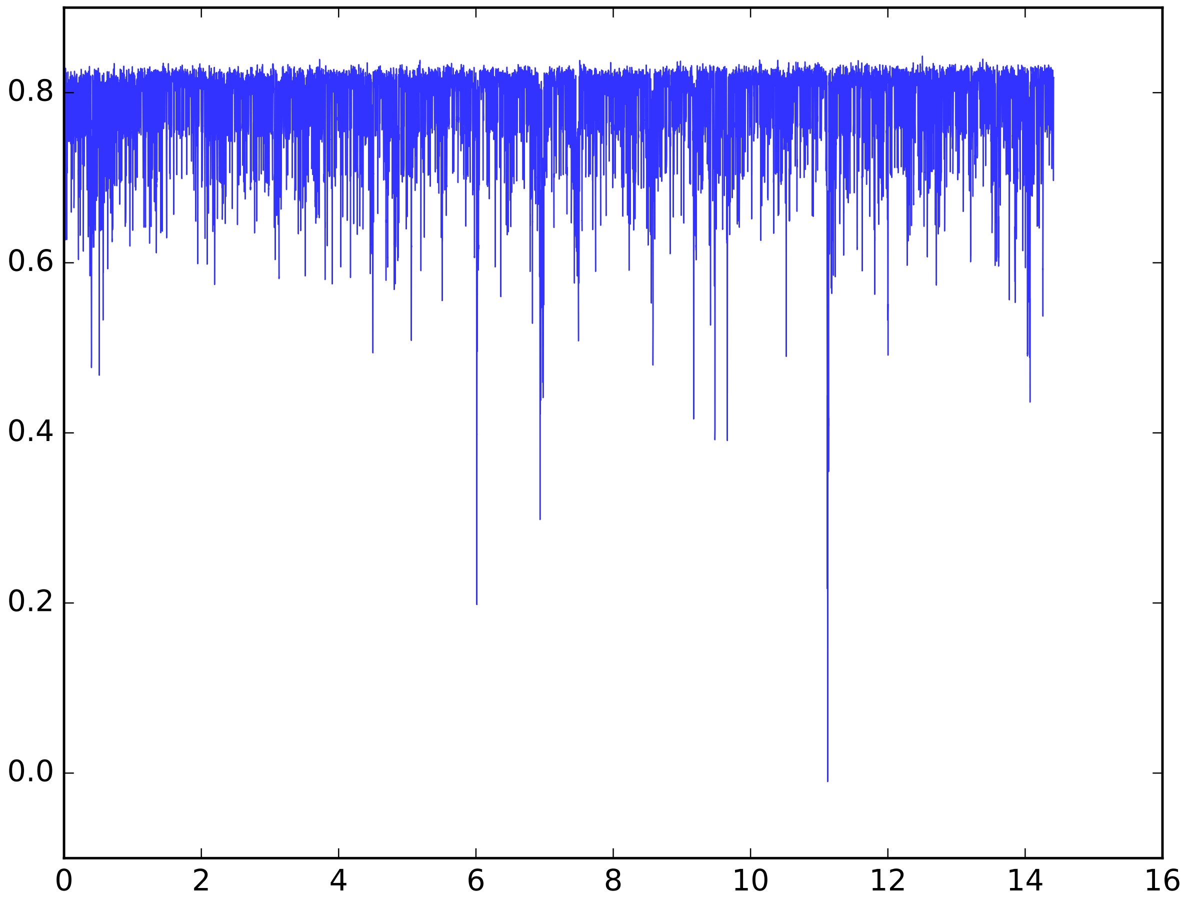

Figure 4.22: Switcher policy training performance of the task ”reachcont”, the x-axis is the number of million time-steps and the y-axis is the total episode reward averaged over the last 32 episodes

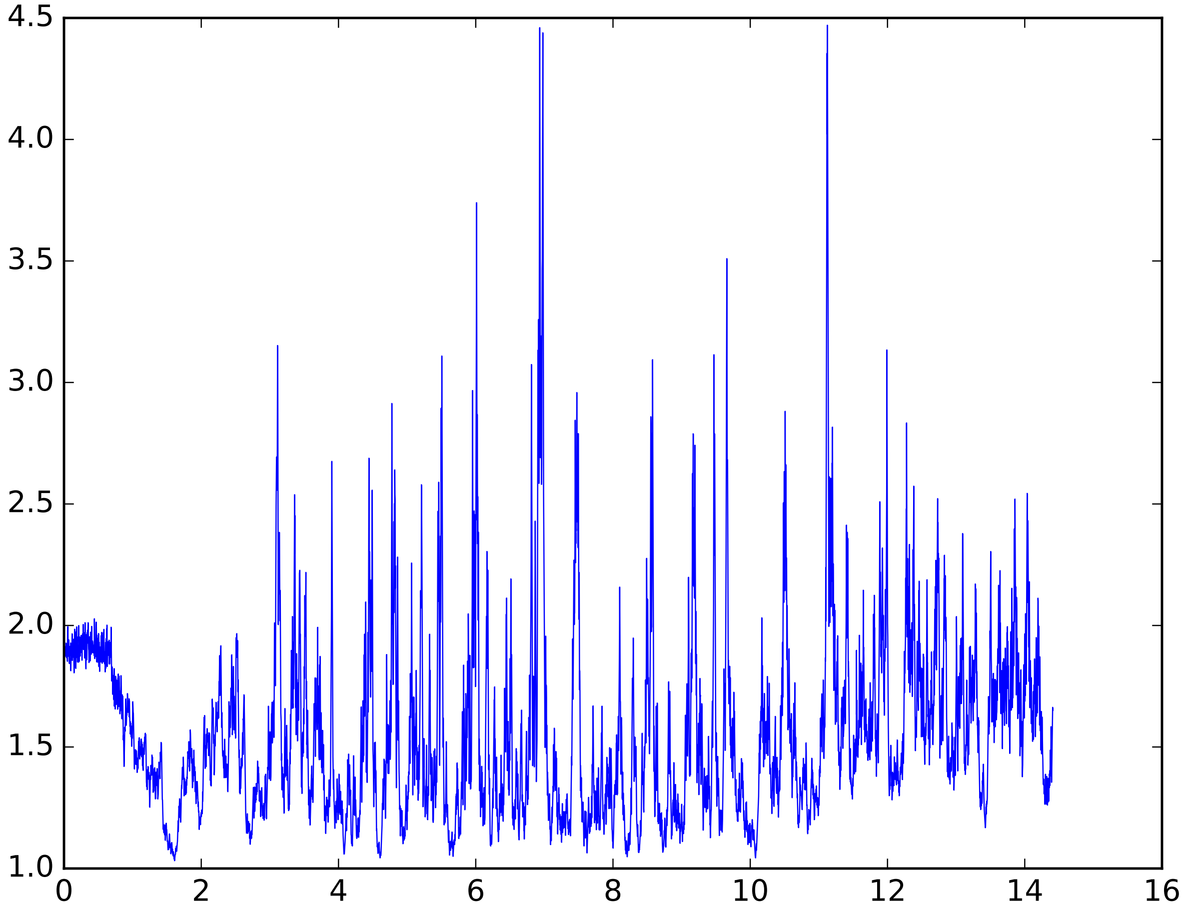

Figure 4.23: Average execution length of the actions of decision policies of the task ”reach-cont”, the x-axis is the number of million time-steps and the y-axis is the decision policy’s average execution length of the last batch.

Chapter 5

Discussion

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] OpenAI, “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] 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.

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

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

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

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

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

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

[31] 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.

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

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

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

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

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