Hierarchical Reinforcement Learning for Multi-modal Continuous Control Environments with 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

TODO 2018, 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

TODO 2018

Hierarchical Reinforcement Learning for Multi-modal Continuous Control Environments with Sparse Rewards

by

Zeng Cancheng

This is to certify that I have examined the above PhD 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.

Prof. Dit-Yan Yeung, Thesis Supervisor
Department of Computer Science and Engineering

TODO head

Department of Computer Science and Engineering, HKUST

TODO 2018

TODO Somebody

Acknowledgments

Contents

List of Figures

List of Tables

Hierarchical Reinforcement Learning for Multi-modal Continuous Control Environments with Sparse Rewards

by Zeng Cancheng

Department of Computer Science and Engineering, HKUST

The Hong Kong University of Science and Technology

Abstract

Chapter 1

Introduction

1.1 Overview

Deep reinforcement learning methods have gained much attention from researchers and the public since the recent advances such as DQN [3] and AlphaGo [4]. The method of deep reinforcement learning combine deep neural network models and reinforcement learning algorithms, enabling the potential for solving general control problems in fully autonomous manner. However, the application of contemporary deep reinforcement learning methods are still limited to a finite class of problems. The reinforcement learning problem either has a fully known model like the Go game, or has to be problems with low complexity like Atari game [3] or primary robot control tasks [5]. The real-world robot control problems are far from being solved without labour-intensive domain specific engineering.

In this study, we propose a hierarchical reinforcement learning method that can learn to solve increasingly complex tasks in several realistic robot control environments.

1.2 Background

1.2.1 Reinforcement Learning (RL)

Reinforcement learning [6] is learning a policy (map from situations to actions) in order to maximize a numerical reward signal. The learner must discover the optimal policy by trying them. Reinforcement learning is different from supervised learning that the learner is not provided any labelled examples by any knowledgeable external supervisor. Reinforcement learning is also different from unsupervised learning, whose objective is finding hidden structures in unlabeled data.

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:

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

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

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

1.2.2 Hierarchical reinforcement learning based on transferred policies from easy tasks

This study focus on a class of reinforcement learning problems, where the objective is to solve a *target task* given a set of trained *source tasks*. The policies that can solve the source tasks, namely the *actuator policies*, are available for the agent. It is known that target task can be solved by a sequential execution of a subset of the source-task policies. The agent needs to learn a *scheduler policy*, which chooses the source-task policy to be executed and decides when to switch the currently running source-task policy.

This paradigm is related to hierarchical reinforcement learning (HRL) [7], 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 where the agent needs to extract the useful closed-loop partial policy from the original task. However, we focus on extracting and reusing partial policies from other tasks.

This paradigm is also related to transfer learning in reinforcement learning [8], because the source-task policies are reused by the high-level scheduler agent. However, conventional transfer learning paradigms usually don't assume a hierarchical relationship between the target task and the source tasks.

1.3 Scope of study

Real-world robot control problems involves many sources of complexity. However, we only focus on some critical properties of real-world robot control problems. This study focus on the reinforcement learning problems with multi-modality state space, continuous action space and sparse reward in a continual learning setting.

1.3.1 Multi-modality state space

The reinforcement learning problems with multi-modality state space refers to the environments whose state space is a combination of signals with different modality.

Specifically, the study focus on realistic robot control environments whose state space is a combination of a low-dimensional state sensor and a image sensor observation.

The low-dimensional state input provides accurate information about the agent's physical system. The image input provides information about the external environment and has a datatype that is in much higher dimension.

Most recent works in reinforcement learning for continuous control focus on the RL tasks with low-dimensional state input only [5]. There are also works [9] trying to handle some tasks with image input only, but these image-based tasks are very simple to solve.

However, the proposed multi-modality environment is the most common case in real-world robotic problems, and is considerably more complex and realistic than solving a low-dimensional state input only task or an image only task.

1.3.2 Sparse reward function

The problem of sparse / delayed reward signal has been one of the major problems to be tackled in reinforcement learning. Many methods are proposed to solve the sparse reward problem in environments with discrete action spaces. However, the problem is much more difficult for the environments with continuous action spaces, and few works have focused on the class of system. By the time of writing, we haven't seen any method that has been proposed to effectively solve continuous control tasks with sparse rewards.

Studies have shown that 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 good enough.

However, the reward function of a realistic robot control environment is usually discontinuous and sparse. Reward shaping techniques may improve the reward function, but it usually relies on domain-specific engineering. In our study, we try to solve some realistic robot problems without reward shaping, which essentially indicates the problems have sparse reward functions.

1.3.3 Assumption on hierarchical relationship of tasks

We focus on solving the reinforcement learning setting described in section 1.2.2 . We assume that the agent has already learned the optimal policies for a set of source tasks before solving the target problem, and we also assume that the target task can be solved by executing a subset of the source task policies in according to some "high level policy".

However, we don't pose any other assumption on the relationship between the source tasks and the target task. We don't know whether any specific source task is useful. A source task policy might be useful in some cases, or it could be completely useless. We also don't make assumption on how long an actuator policy should be executed in its term. The scheduler agent need to learn it instead. An actuator policy might be executed for only 1 time-step before the agent switch to another one, or it could be executed for a longer time, or until a specific condition.

This assumption is the only assumption that is involved in domain-specific design. However, we observe that many real-world complex robot control problems can usually be decomposed to primary tasks. Classical robot control methods also organize the abstract problems (such as trajectory planning) and low level problems (such as model control) in hierarchical structures.

If this hierarchical task relationship assumption was removed, the problem becomes the fully autonomous hierarchical reinforcement learning problem, where the agents need to learn not only a task structure but also all actuator policies from scratch. This problem is considered in-feasible to solve for general robot control problems [7].

1.4 Research Questions

We will propose several reinforcement learning environments for continuous control in section 3.1 according to the stated scope and assumption. The study will mainly focus on solving the proposed problems. The primary research question that we focus on is:

Can we develop an agent that can achieve good performance (in terms of total reward) in the reinforcement learning environments defined in section 3.1 within reasonable 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)?

We will first answer the question on whether the previous work already solve the problem: *Can the contemporary flat reinforcement learning methods achieve good performance in the proposed environments?*

We will then answer the question on whether our proposed model can solve this: A model that solves the primary 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 a good performance (in terms of the expected total return of the original problem), with or without fine-tuning methods, in a target environment which has different initial state distribution?*".

To address the second issue, we will answer the question of "*Can the target scheduler agent learn the optimal policy given the trained sub-task policies, without any domain-specific design defined previously?*".

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, where 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 [10]:

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

Ψ_t may be one of the following:

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 [10]

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

2.1.1 Trust Region Policy Optimization

The method proposed by [11], 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 gradientupdate 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 primary robot control tasks in [12] within a few number of training samples. However, both the computation of the Hessian matrix and the conjugate gradient algorithm are too expensive for deep neural network models.

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 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 proposed method further speeds up the optimization by reusing recently computed statistic matrices in the previous steps and performing asynchronous computation, and finally achieves a computation time comparable to ordinary gradient decent algorithms. The method manages to computationally efficiently perform trust region natural policy gradient optimization without sacrificing the reinforcement learning performance.

2.1.3 Proximal Policy Gradient Method

The author of [13] proposed a first-order policy optimization method, namely proximal policy gradient with clipping (PPO). The method clips the surrogate objective according to the following equation:

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

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 batch 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.

The method can achieve a relatively fast improvement rate in the early phase (within 1 million training steps) of training in several continuous control environments. However, the method is more difficult to tune compared to trust region methods, and might become increasingly unstable in the late phase of training. The method might also converge too early and stuck at a local minimum point, because updating the parameter too many times on a single batch might lead to overfitting.

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) [14], Actor-Critic with Experience Replay (ACER) [15], and Trust-PCL [16]. These methods are first order off-policy algorithm, and are not the focus of our study because they're not reported to significantly outperform the above mentioned methods in terms of training time, final performance and stability.

2.2 Hierarchical Reinforcement Learning Methods

2.2.1 Option framework

The work of [17] 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 2.2.2. The hierarchical agent is

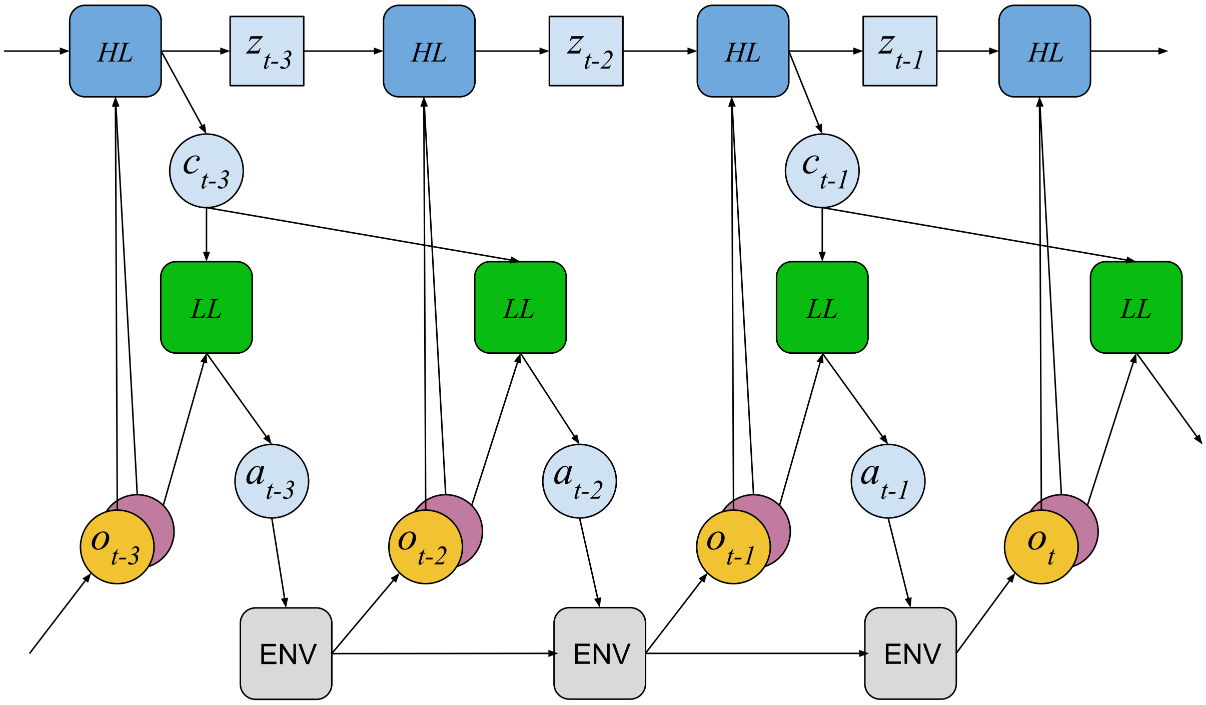

Figure 2.1: The hierarchical structure of the modulated controller [1], which consists of the recurrent high-level controller (HL, dark blue) and feedforward low-level controller (LL, green). The high-level controller has access to all observations (yellow and red). While both controllers observe sensory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

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:

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

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

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 2.2.3. The architecture is basically a two-level hierarchical reinforcement learning agent, the components $\phi_1, \phi_2, \dots$ 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 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,

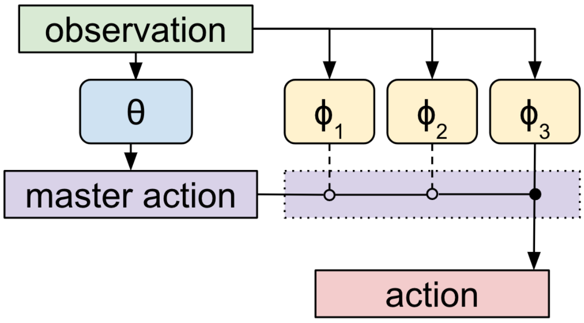

Figure 2.2: The agent setup of the modulated controller [2],θ represents the master policy, which selects a sub-policy to be active. In the diagram, ϕ_3 is the active sub-policy, and actions are taken according to its output.

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 [18], [19] have tried to solve sparse-reward robot control environments through the learning and scheduling of auxiliary policies. The method of [18], 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 $A = \{A_1, \dots, A_k\}$ 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 [20], FuN [21], HRL with Stochastic Temporal Grammar [22], policy sketch method [23] and strategic attentive writer [24]. 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 TODO: Target Environments

3.1.1 Ant-based environments

Summary of Environments

The source environments and target environments are summarized in table 3.1.1.

Task name	Goal	Reward	Description
move0	velocity: $g_0 = (1, 0)$	$v_g + 1 - c_p - c_c$	move toward east direction
move1	velocity: $g_1 = (-1, 0)$	$v_g + 1 - c_p - c_c$	move toward west direction
move2	velocity: $g_2 = (0, 1)$	$v_g + 1 - c_p - c_c$	move toward north direction
move3	velocity: $g_3 = (0, -1)$	$v_g + 1 - c_p - c_c$	move toward south direction
move1d	velocity: $\{g_0, g_1\}$	$v_g + 1 - c_p - c_c$	each episode has a random sampled goal direction
move2d	velocity: $\{g_0, g_1, g_2, g_3\}$	$v_g + 1 - c_p - c_c$	there are 4 possible goal directions compared to move1d
dynamic2d	velocity: $\{g_0, g_1, g_2, g_3\}$	$v_g + 1 - c_p - c_c$	the goal direction is reset with probability 0.01
reach2d	position: $\{g_0, g_1, g_2, g_3\}$	I(The agent's distance to the goal position is within 0.5)-0.01	move to a specific location instead of a direction

Table 3.1: Summary of Ant-based environments

Dynamic2D environment

Reach2D environment

3.1.2 TODO: Robot manipulation environments

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

As is reported in [25], 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 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:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & J(\theta) \\ \text{subject to} \quad & \overline{W_2}(\pi_{\theta_{old}}, \pi_\theta) \leq \delta_W \end{aligned} \tag{3.1}$$

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 [26]:

$$W_2(P,Q) = \inf_{\Gamma \in \mathbb{P}(X \sim P, Y \sim Q)} \mathbb{E}_{X,Y \sim \Gamma} \left[\|X - Y\|_2^2 \right]^{1/2} \tag{3.2}$$

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 [27] :

$$W_2^2(P,Q) = \|m_1 - m_2\|_2^2 + \operatorname{Tr}\left(\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}\right) \tag{3.3}$$

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

$$s = H_\theta \left(\overline{W_2^2}(\pi_{old}, \pi)\right)^{-1} g := A^{-1} g \tag{3.4}$$

We use the approximation of the Rianmanian metric tensor A:

$$A \approx \mathbb{E}_{s_t} \left[\nabla_\theta W_2^2(\pi_{old}, \pi) \left(\nabla_\theta W_2^2(\pi_{old}, \pi) \right)^T \right] \tag{3.5}$$

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 [28]. 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:

$$g' = g + \beta_{ent} \nabla_\theta \mathbb{E}[H(\pi_\theta(a|s))] \tag{3.6}$$

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:

$$H(\pi) = \frac{1}{2} \ln \det(2\pi e \Sigma) \tag{3.7}$$

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:

$$g'_\Sigma = g_\Sigma + \beta_{exc} \mathbb{E}\left[\max\left(0, I(\hat{A}^{GAE} > 0)\hat{A}^{GAE} \nabla_\Sigma \log \pi(a|s)\right)\right] \tag{3.8}$$

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:

$$\pi(a|s) = (1-\alpha_{ex})\mathcal{N}(\mu,\Sigma) + \alpha_{ex}\mathcal{N}(\mu, q_{ex}\Sigma) \tag{3.9}$$

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:

$$\begin{aligned} &W_2(\pi_0(a|\mu_0,\Sigma_0), \pi_1(a|\mu_1,\Sigma_1) = \\ &\quad (1-\alpha_{ex})W_2\Big(\mathcal{N}(\mu_0,\Sigma_0), \mathcal{N}(\mu_1,\Sigma_1)\Big) + \alpha_{ex}W_2\Big(\mathcal{N}(\mu_0, q_{ex}\Sigma_0), \mathcal{N}(\mu_1, q_{ex}\Sigma_1)\Big) \end{aligned} \tag{3.10}$$

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

3.5 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 switch to another policy based on the current state. The agent

The network θ_d, namely decider network, outputs an n_a-discrete action space, that select the acting agent

The selected leaf agent executes the corresponding sub-policy and computes the primary actions the agent should take for the original environment.

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

Algorithm 1 The decider agent mechanism

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

3.6 Generalized advantage estimation for hierarchical reinforcement learning agents

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:

$$\bar{r}_{t_i} := \sum_{l=0}^{t_{i+1}-t_i-1} \gamma^l r_{t_i+l} \tag{3.11}$$

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

$$\delta_{t_i}^V := -V(s_{t_i}) + \bar{r}_{t_i} + \gamma^{t_{i+1}-t_i} V(s_{t_{i+1}}) \tag{3.12}$$

Then the k-step advantage estimation is given by:

$$\begin{aligned}
\hat{A}_{t_i}^{(1)} &:= \delta_{t_i}^V &&= -V(s_{t_i}) + \bar{r}_{t_i} + \gamma^{t_{i+1}-t_i} V(s_{t_{i+1}}) && (3.13)\\
\hat{A}_{t_i}^{(2)} &:= \delta_{t_i}^V + \gamma^{t_{i+1}-t_i}\delta_{t_{i+1}}^V &&= -V(s_{t_i}) + \bar{r}_{t_i} + \gamma^{t_{i+1}-t_i}\bar{r}_{t_{i+1}} + \gamma^{t_{i+2}-t_i} V(t_{i+2}) && (3.14)\\
\hat{A}_{t}^{(3)} &:= \delta_{t}^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V &&= -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3}) && (3.15)
\end{aligned}$$

$$\begin{aligned}
\hat{A}_{t_i}^{(k)} &:= \sum_{d=0}^{k-1} \gamma^{t_{i+d}-t_i} \delta_{t_{i+d}}^V \\
&= -V(s_t) + \bar{r}_{t_i} + \gamma^{t_{i+1}-t_i}\bar{r}_{t_{i+1}} + \cdots + \gamma^{t_{i+k-1}-t_i}\bar{r}_{t_{i+k-1}} + \gamma^{t_{i+k}-t_i} V(s_{t_{i+k}})
\end{aligned} \tag{3.16}$$

We can define the unnormalized generalized advantage estimator as a exponentially-weighted sum of these k-step advantage estimators [10]:

$$\begin{aligned}
\hat{A}_{t_i}^{GAE_{unnorm}(\lambda)} &:= \hat{A}_{t_i}^{(1)} + \lambda^{t_{i+1}-t_i}\hat{A}_{t_i}^{(2)} + \lambda^{t_{i+2}-t_i}\hat{A}_{t_i}^{(3)} + \cdots + \lambda^{t_{i+k-1}-t_i}\hat{A}_{t_i}^{(k)} \\
&= \delta_{t_i}^V + \lambda^{t_{i+1}-t_i}(\delta_{t_i}^V + \gamma^{t_{i+1}-t_i}\delta_{t_{i+1}}^V) && (3.17)\\
&\qquad + \lambda^{t_{i+2}-t_i}(\delta_{t}^V + \gamma^{t_{i+1}-t_i}\delta_{t_{i+1}}^V + \gamma^{t_{i+2}-t_i}\delta_{t_{i+2}}^V) + \ldots \\
&\qquad + \lambda^{t_{i+k-1}-t_i}\sum_{d=0}^{k-1}\gamma^{t_{i+d}-t_i}\delta_{t_{i+d}}^V && (3.18)\\
&= (\delta_{t_i}^V \sum_{b=0}^{k-1}\lambda^{t_{i+k-1}-t_i} + \gamma^{t_{i+1}-t_i}\delta_{t_{i+1}}^V \sum_{b=1}^{k}\lambda^{t_{i+b}-t_i} \\
&\qquad + \gamma^{t_{i+2}-t_i}\delta_{t_{i+2}}^V \sum_{b=2}^{k}\lambda^{t_{i+b}-t_i} + \ldots && (3.19)\\
&\qquad + \gamma^{t_{i+k-1}-t_i}\delta_{t_{i+k-1}}^V \lambda^{t_{i+k-1}-t_i}) \\
&= \sum_{d=0}^{k-1}\delta_{t_i}^V \gamma^{t_{i+d}-t_i}\sum_{b=0}^{k-1}\lambda^{t_{i+b}-t_i} && (3.20)
\end{aligned}$$

The normalized generalized advantage estimater is then given by:

$$\hat{A}_{t_i}^{GAE(\lambda)} = \frac{\hat{A}_{t_i}^{GAE_{unorm}(\lambda)}}{\sum_{b=0}^{k-1}\lambda^{t_{i+b}-t_i}} \tag{3.21}$$

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.

Chapter 4

Experiments

4.1 Solution to the basic environment: *move0*

In this section, we compare the proposed W-KTR method with other state-of-art algorithms on the basic *move0* environment.

The performance of W-KTR algorithm is shown in Figure 4.1:

It can be shown that the W-KTR agent has a stable final performance, which is not much dependent on the hyper-parameter δ_W^2.

The performance of ACKTR agents is shown in Figure 4.1: It can be shown that the ACKTR agent can achieve a faster rate of improvement at the first 10 million steps, but the improvement rate becomes slow as the training proceeds. The agents tend to stuck at sub-optimal policies since the policies converge too fast and cannot effectively adjust once the STD becomes small.

We also run experiments on tuning a PPO agent for the task move0, which is shown in Figre 4.1. The original PPO algorithms has 40 minibatch updates in each batch, which converges too early. We only performa one gradient update over the whole batch in this experiment. The performance of the PPO agent appears to be sensitive to the hyper-parameter, and the agent can't avoid the drop of performance in the late phase of training. The method also needs to trade-off between the early improvement rate and the final performance. In conclusion, the PPO method is not a robust algorithm.

All the method above use a batch size of 4000 timesteps produced by 20 parallel agents.

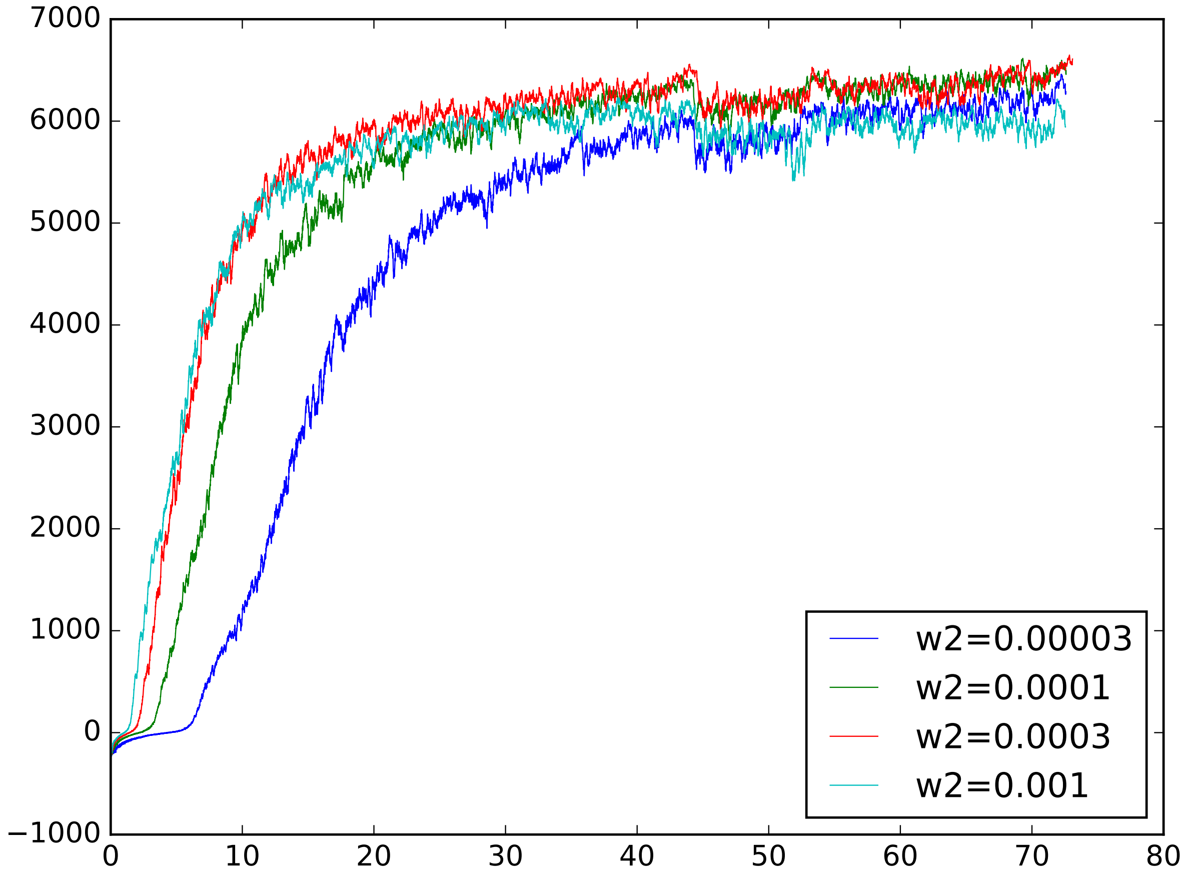

Figure 4.1: Performance of move0 W-KTR agent with diferent hyper-parameter δ_W^2, the x-axis is the number of million timesteps and the y-axis is the total episode reward averaged over the last 200 episodes

4.2 Flat solution to a simple Multi-Modality environment: *move1d*

In the move1d environment, the agent needs to learn from the image input to decide whether to move forward or backward, as well as control according to the states. We have tried to solve the problem using ACKTR method, presented in Figure 4.2, shows that the agent can easily get stuck at the score of 1000, and cannot improve much even if the agent breaks the bottleneck because the average STD parameter already drops to below 0.04.

We also did a single experiment using the proposed W-KTR method, and is shown in Figure 4.2. The W-KTR agent manages to achieve a fast improvement rate and high final performance after getting stuck at the bottleneck score 1000 for a while.

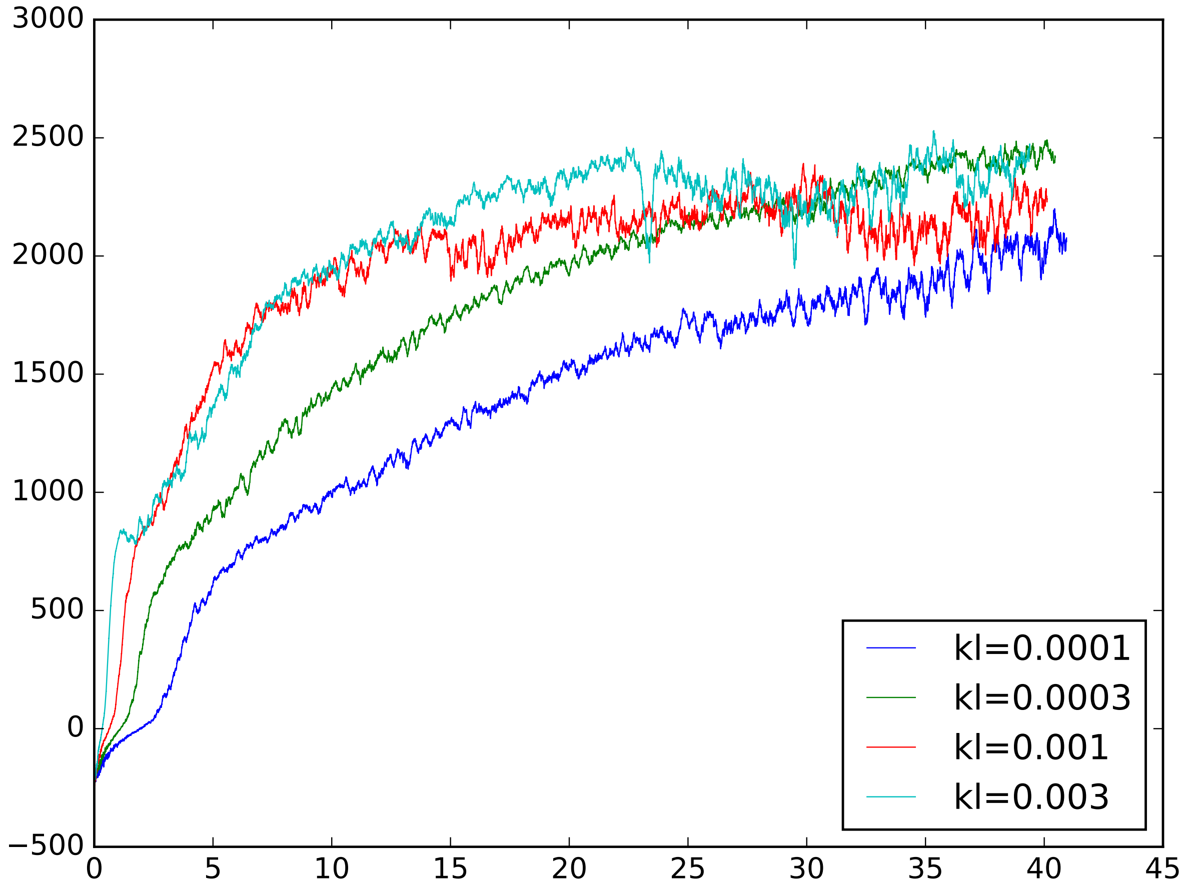

Figure 4.2: Performance of move0 PPO agent with diferent hyper-parameter δ_{kl}, the x-axis is the number of million timesteps and the y-axis is the total episode reward averaged over the last 200 episodes

This proves that the W-KTR agent is better at handle local minimum and re-adjust the agent's policy even when the policy STD becomes low.

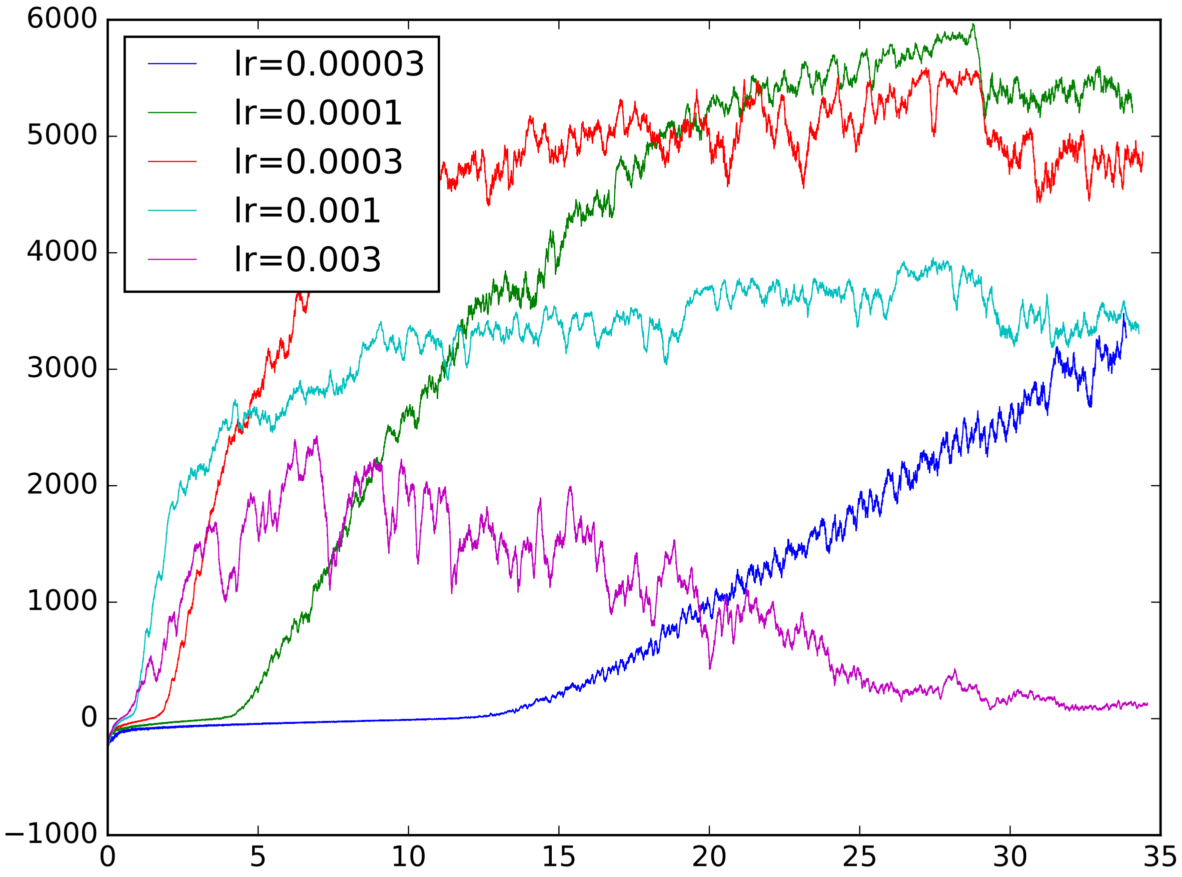

Figure 4.3: Performance of move0 PPO agent with diferent hyper-parameter *learningrate*, the x-axis is the number of million timesteps and the y-axis is the total episode reward averaged over the last 200 episodes

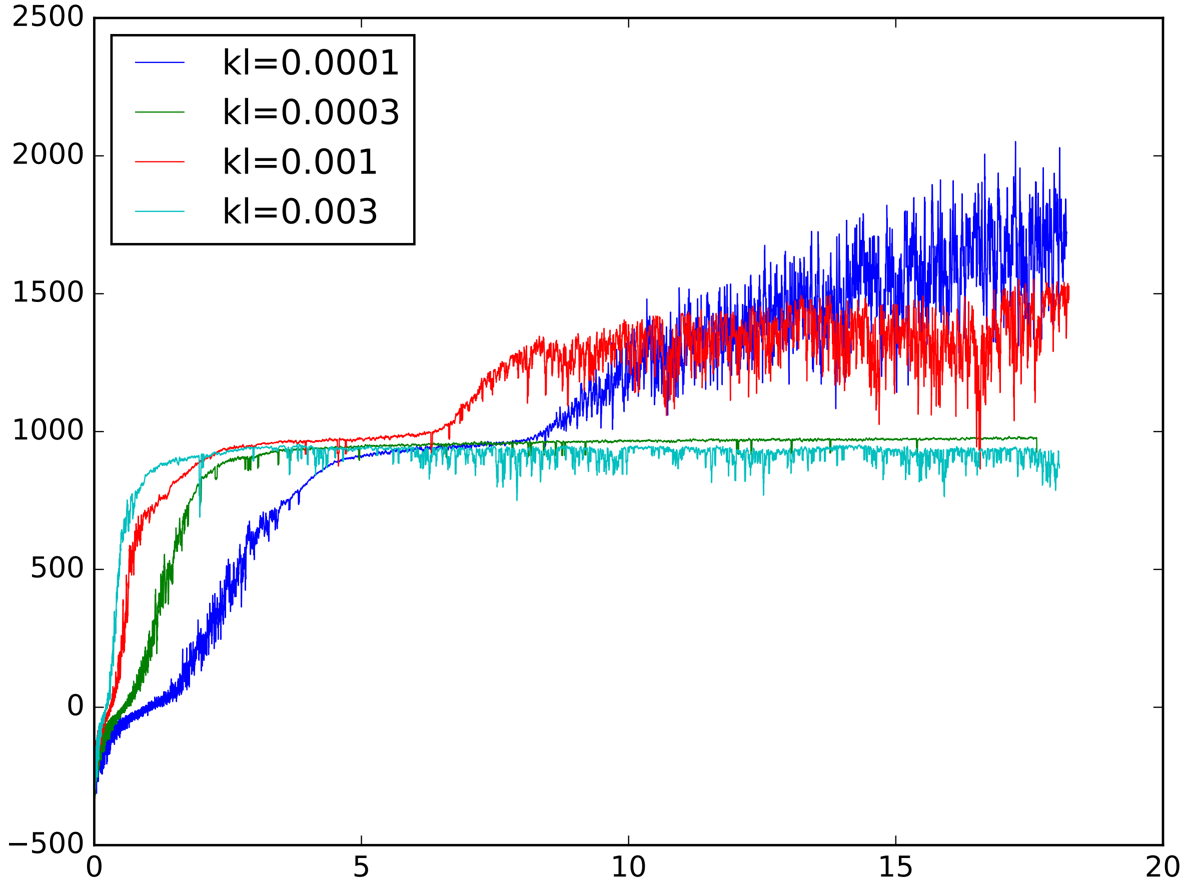

Figure 4.4: Performance of move1d ACKTR agent with diferent hyper-parameter *learningrate*, the x-axis is the number of million timesteps and the y-axis is the total episode reward averaged over the last 20 episodes

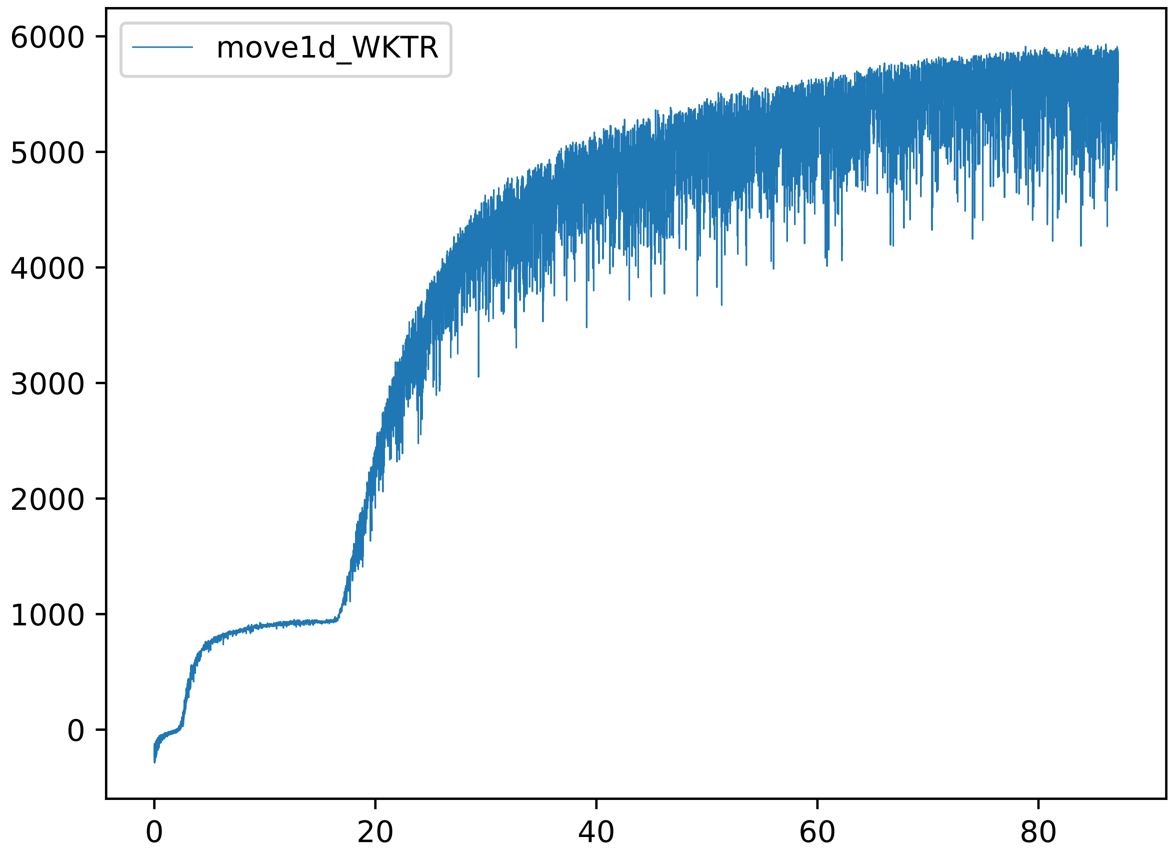

Figure 4.5: Performance of move1d W-KTR agent with hyper-parameter $\delta_W^2 = 0.0001$, the x-axis is the number of million timesteps and the y-axis is the total episode reward averaged over the last 20 episodes

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

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

[8] 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.

[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] 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.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

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

[24] 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.

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

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

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

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

Appendix A

Appendix 1