Deep Reinforcement Learning for Continuous Action Control

Zhaoyang Yang

A thesis in partial fulfilment of the requirements for the degree of

Master of Philosophy

UNSW
AUSTRALIA

School of Engineering and Information Technology

University of New South Wales, Canberra

July 2017

Abstract

Recent advances in reinforcement learning have successfully combined the strong generalization and feature extraction ability of deep learning models with the bootstrapping nature of reinforcement learning. Many works in this area have achieved record breaking performance, especially on tasks in a discrete action space. However, much less work has been done to deal with robotic control in a continuous action space. This is more challenging as unlimited action choices may be available. In fact, all single-thread based algorithms in this domain can only control a robot to solve basic tasks, which are tasks that can be achieved by simply choosing actions to move in a single movement pattern, and only one task at a time.

In this thesis, we are taking continuous control deep reinforcement learning one step further to address the existing limitations discussed above. Three main contributions are introduced that finally enable agents to learn compound tasks, which are tasks that can only be achieved by combining different movement patterns, in a continuous action space. We first propose a novel deep reinforcement learning network architecture that can reduce the number of parameters needed for learning single basic skill in a continuous action space by more than 70%. We then propose a novel multi-task deep reinforcement learning algorithm to learn multiple basic tasks simultaneously. It makes use of the proposed network architecture to reduce the number of parameters needed for learning multiple tasks by more than 80%. Based on this, we finally propose a novel hierarchical deep reinforcement learning algorithm which consists of two levels of hierarchy. It adapts the proposed multi-task learning algorithm in its first level of hierarchy to learn multiple basic skills, and then learns to reuse these skills in its second level of hierarchy to solve compound tasks.

We conducted several sets of experiments to test both the proposed network architecture and the proposed algorithms. The experiments were conducted with a simulated Pioneer 3AT robot with front-view camera and range sensor in Gazebo 2 in a

ROS Indigo environment. Results show that agents built with the proposed network architecture can learn skills that are as good as the ones learned by agents built with traditional convolutional neural networks. Also, all basic skills learned by the proposed multi-task learning algorithm achieve comparable performance to the skills learned one after another by a single-task learning algorithm. Results also show that the proposed hierarchical learning algorithm can learn both high performance basic skills and compound skills within the same learning process. The performance of the proposed algorithm on solving compound tasks outperforms both a state-of-the-art single-thread based continuous action control algorithm and a well-known discrete action control algorithm.

In summary, the works in this thesis extends continuous control deep reinforcement learning algorithms to multi-task and hierarchical learning scenarios. While the proposed multi-task learning algorithm can learn multiple tasks simultaneously, the proposed hierarchical learning algorithm can solve compound tasks that cannot be solved with existing single-thread learning algorithms. These works have further potential to bring people new inspirations for building deep reinforcement learning agents in a continuous action space.

Keywords

Deep Learning, Reinforcement Learning, Multi-Task Learning, Hierarchical Learning, Continuous Action Control, Neural Network Architecture.

Acknowledgement

Along this one year journey of finishing my MPhil, I am grateful to many people without whom the completion of this project would not be possible. I would like to thank all individuals that have helped me with this study.

I would first offer my special appreciation to my supervisor Prof. Kathryn Merrick, who has gave me continued guidance, expertise and support along the way of finishing this project. I wish to express my thanks to her for the time and energy she has spent. She has taken the supervisor's role to the very limit by being constantly available for consultations. I would also offer my appreciation to my co-supervisor Prof. Hussein Aly Abbass, who took time out of his pressured academic life to support me with his invaluable assistance, advice and ideas on my work. His insights and inspirations have always been an indispensable part of my research.

This work would not exist without the enormous support of all members in our big family of the Trust Autonomy Group. I am especially indebted to Jiangjun Tang, Xuejie Liu and Min Wang who warmly embraced me into the group and helped me get immersed into the life in Canberra; also to Duy Tung Nguyen and Hung Nguyen, the discussions with them have been the source of innovations. Special thanks to George Leu for sharing me his knowledge and resources, which all of my works were built on.

My heartfelt gratitude to Prof. Joseph Lai for his immeasurable contributions in associating international cooperation. His kind and patient support was my first picture of this university. He was the person who has made this double master degree project and my dream real. I would also like to express my gratitude to my host supervisor in China, Prof. Lianwen Jin, who trusted me enough to give so freely of himself in my study abroad.

I would never be able to thank enough my beloved parent. They have always been on my side to support me with their tireless encouragements. Their trust, love and

prayers have, and will continue to guide my life.

All in all, I have learnt and been given a lot in this unique journey. I am filled with emotions at this special moment of completing this thesis that I am feeling such a deep and quiet satisfaction of being a part of my group, the university and the peaceful world. I am so appreciated to own all these love and support, which definitely will go into my memory and live in my future life.

List of Publications Arising from this Thesis

1. Zhaoyang Yang, Kathryn Merrick, Hussein Abbass, Lianwen Jin, Multi-Task Deep Reinforcement Learning for Continuous Action Control. Proceedings of International Joint Conference on Artificial Intelligence, 2017. [Accepted on April 23, 2017].

2. Zhaoyang Yang, Kathryn Merrick, Hussein A. Abbass, Lianwen Jin, Hierarchical Deep Reinforcement Learning for Continuous Action Control. [Under Review].

Contents

Abstract……..I

Keywords……. III

Acknowledgement V

List of Publications Arising from this Thesis VII

Contents…….. IX

Chapter 1 Introduction 1

1.1 Research Objectives 3

1.2 Contributions and Significance 4

1.3 Research Methodology 5

1.4 Thesis Overview 6

Chapter 2 Literature Review 9

2.1 Reinforcement Learning Overview 9

2.1.1 Reinforcement Learning Setups 10

2.1.2 Classical Algorithms 12

2.1.3 The Actor-Critic Architecture 14

2.2 Deep Learning Overview 15

2.2.1 Recent Advances in Deep Learning 16

2.2.2 Recent Advances in Training Optimization 19

2.2.3 The Multi-Layer Perceptron Convolutional Layer 21

2.3 Deep Reinforcement Learning 23

2.3.1 The New Generation of Intelligent Agents 24

2.3.2 Deep Q Network and Deep Deterministic Policy Gradient 26

2.3.3 Multi-Task and Hierarchical Deep Reinforcement Learning 29
2.4 Summary 30
Chapter 3 A Novel Deep Reinforcement Learning Network Architecture 33
3.1 Inspirations 34
3.2 General Architecture of the Proposed Network 35
3.3 Implementation Variations 37
3.3.1 Parameter Sharing Implementation 37
3.3.2 Image Data and Sensor Data Fusing Implementation 38
3.4 Network Architecture Summary 39
3.5 Experiment Setup 40
3.6 Performance of Different Implementation Variations 42
3.6.1 Parameter Sharing Implementation 42
3.6.2 Image Data and Sensor Data Fusing Implementation 44
3.7 Comparison to Traditional Convolutional Neural Networks 45
3.8 Experiment Summary 47
Chapter 4 A Novel Multi-Task Deep Reinforcement Learning Algorithm 49
4.1 Algorithm Architecture 50
4.2 Algorithm Learning Process 52
4.3 Algorithm Summary 53
4.4 Experiment Setup 55
4.5 Multi-Task Learning Performance 56
4.6 Learning with an Incompetent Actor 59
4.7 Experiment Summary 60
Chapter 5 A Novel Hierarchical Deep Reinforcement Learning Algorithm 63
5.1 Algorithm Architecture 64
5.2 Algorithm Learning Process 66

5.3 Implementation Details 67
5.3.1 Rewards and Punishments 67
5.3.2 Replay Memory and Batch Sampling 68
5.3.3 Exploration 69
5.4 Algorithm Summary 69
5.5 Experiment Setup 71
5.6 Hierarchical Learning Performance 74
5.6.1 Basic Critic Performance 74
5.6.2 Meta Critic Performance 74
5.7 Learning with an Incompetent Actor 77
5.8 Comparisons with Other Algorithms 79
5.9 Experiment Summary 80
Chapter 6 Conclusion and Future Work 83
6.1 Conclusion 84
6.2 Limitations 88
6.3 Future Work 89
References…. 91

Chapter 1

Introduction

Intelligent computer agent design is a topic that has been long discussed in the field of artificial intelligence. Much work has been done to investigate different aspects of the intelligence of artificial agents. For example, swarm intelligence [1] investigates agents' intelligence in multi-agent scenarios, and human-computer interaction [2], which mainly focuses on agents' intelligence when communicating with humans. These topics are all active research areas on how to achieve intelligent control and how intelligent robots will react.

Another important branch of research investigates how to design agents that can solve specific tasks. A number of different methods have been proposed to achieve this. For example, there are dynamic programming [3] methods which rely on human knowledge of the task to program agents' decisions under different situations. There are also imitation learning [4] methods, which achieve learning by imitating humans or well-designed agents. While these methods can achieve high performance in some scenarios, they are highly dependent on past knowledge and experience.

Departing away from these methods, reinforcement learning methods focus on designing algorithms that can enable agents to learn to achieve goals in given environments by themselves. A reinforcement learning agent can bootstrap skills or policies for solving tasks throughout training and exploration, and this process mainly depends on the interaction between the environment and the agent itself. Classical algorithms such as Q-learning [5] and SARSA [6] are well-known reinforcement learning algorithms that can make use of sparse reward feedback from the environment to learn desired skills or policies. What is more, many classical works introduced linear function approximations to enhance the generalization of the algorithms [7, 8]. This helps agents to handle tasks in more complex environments and tasks with more action choices.

However, the generalization of the algorithms are still limited because of the limited capacity of linear function approximations.

In recent years, the growing accessibility of high performance computation devices and big data have propelled deep learning [9] research. Deep learning based algorithms have achieved record-breaking performance in several applications and research topics. With hundreds of thousands of auto-learned parameters in the model, deep neural networks have shown unprecedented feature extraction and generalization abilities. Moreover, available choices for network architectures such as convolutional neural networks (CNN) [10] and long-short term memory (LSTM) networks [11] further help deep learning fit in applications with different requirements. These advances have been combined to make deep learning a powerful model in applications such as computer vision [12-14] and semantic analysis [15-17]. These successes have inspired interest in combining reinforcement learning with deep learning to further improve the performance of agents.

However, it is generally believed that non-linear function approximators like deep neural networks are not suitable for reinforcement learning. This is mainly because of the correlations between data and the sparsity of supervision signals in reinforcement learning scenarios [18]. The combination of deep learning and reinforcement learning has not been done until recent advances have addressed these challenges and brought deep reinforcement learning great success. Some of the successful deep reinforcement learning based agents have recently outperformed humans in playing Atari games [19] and Go games [20].

Unlike games or other decision making processes that contain only a limited number of legal actions, robotic control usually involves potentially unlimited action choices, as the control is in a continuous action space. This is challenging as the agent needs not only to find solutions to tasks, but also to take other physical factors into account to keep the robot moving smoothly. The problem becomes even more difficult when trying to handle more complex scenarios such as multi-task learning or compound skill learning that, to the best of our knowledge, no existing work can do [21]. This thesis will explore ways to make up for these limitations.

The reminder of this chapter is organized as follows. The research aims of the thesis will be presented in Section 1.1. This will be followed by a summary of the contributions of the thesis. The research methodology will be presented in Section 1.3 and Section 1.4 will give an overview of the thesis.

1.1 Research Objectives

Unlike game or other decision making processes that have a discrete action space, there is less work done to handle continuous control tasks with deep reinforcement learning. In this thesis, we will explore ways of achieving continuous control in more complex scenarios. Specifically, the aims of our work are:

- To find ways to reduce the number of parameters needed for multi-task learning. This is based on the fact that while multi-task learning usually saves time for learning different tasks, multi-task learning agents usually need much more parameters in the system than single-task learning agents. Reducing the number of parameters needed is beneficial for further extension of the agents.

- To investigate ways to achieve multi-task learning in continuous action spaces with a single agent. In other literature, multi-task learning is generally achieved with transfer learning [22] methods or multi-thread learning schemes, which involve at least two agents. In addition, most of this work is solving tasks in discrete action spaces. We aim to find alternatives to transfer learning appropriate for continuous action spaces.

- To find ways to extend continuous control algorithms to learn more complex, compound tasks in a one-thread training based context. Most of the existing work on continuous control tasks only focuses on learning basic tasks, which are generally different kinds of behaviours or locomotion tasks that can be achieved by simply choosing actions to move in a same movement pattern (We will keep this definition for the rest of the thesis). In contrast, compound tasks need a combination of different movement patterns to achieve them (we will also keep

this definition for the rest of the thesis). This is much more challenging as it requires that the agent not only learn basic skills, but also compound skills during exploration.

1.2 Contributions and Significance

Our works focus on designing agents that can learn to control robots equipped with range sensors and a front view camera.

Corresponding to each of our research objectives presented in the last section, the contributions of this thesis are that:

- We propose and validate a novel deep neural network architecture for deep reinforcement learning. The network architecture makes use of the multi-layer perceptron convolutional (mlpconv) layer [23] to reduce the number of parameters in the agent. Analyses and validations show that the proposed novel network architecture can:
  - Reduce the number of parameters originally for a single task agent by 75%. For a multi-task learning algorithm, the reduction becomes more significant as the number of tasks increases.
  - Learn skills that are as good as the skills learned by original big networks and keep the performance when multiple tasks are introduced.

- We propose and validate a novel deep reinforcement learning algorithm that can learn multiple continuous control tasks concurrently. The algorithm is called multi-DDPG (DDPG is for Deep Deterministic Policy Gradient). The validations of the proposed multi-DDPG algorithm show that:
  - It is, to the best of our knowledge, the first multi-task deep reinforcement learning algorithm for continuous action control.
  - It learns multiple tasks within a single agent and within the same one-thread training process. This is different from most multi-task learning algorithms

that are based on transfer learning or multi-thread learning, as they always involve at least two agents or two learning stages.
  - The multi-task learning scheme does not result in any loss in the performance of each of the single tasks.

- We propose and validate a novel deep reinforcement learning algorithm that can learn complex compound skills to solve compound tasks in fully observable scenarios. The proposed algorithm is based on the multi-DDPG algorithm in our first contribution. It is called h-DDPG as it has a hierarchical architecture. The validations of the algorithm demonstrate the following significance:
  - It is, to the best of our knowledge, the first hierarchical deep reinforcement learning algorithm for continuous action control.
  - It can learn compound skills to solve compound tasks that other one-thread training based continuous control deep reinforcement learning algorithm cannot solve.
  - The hierarchical architecture allows the agent to learn basic skills and compound skills within the same training process, while the compound skills are made up of the basic skills it learns.

1.3 Research Methodology

Three main contributions have been made in this thesis, each follows one of the objectives we assigned in our work. Our methodology analyses existing works and extends them to new applications. It has a sequence of developing stages. Firstly, the new network architecture proposed in the thesis has made use of the mlpconv layer [23] for image classification problems. Secondly, the proposed multi-DDPG algorithm shares some basic concepts proposed by Lillicrap, et al. [24] for learning single continuous control task. Lastly, a part of the proposed h-DDPG algorithm is based on the new network architecture and the multi-DDPG algorithm.

The development and evaluation of our work are dependent on robot control simulations built in Gazebo 2 in a Robot Operating System (ROS) Indigo environment.

The evaluations are mainly conducted by comparing different algorithms. Therefore we reproduced all the algorithms used for comparison and applied them in the same simulations. We use measurements that are commonly used in deep reinforcement learning, such as average rewards collected per action, to show the performance of different algorithms. Both the average values and standard deviations across multiple instances are considered in analyses, which is a standard and systematic way for analysing statistics.

1.4 Thesis Overview

The reminder of this thesis is organized as follows:

Chapter 2: Literature Review

This chapter includes a thorough review of existing work that has been done in related research areas. It starts with a literature review of classical reinforcement learning research, including introductions to general setup and important architectures, especially the one being selected in the thesis. Then we move on to deep learning research and review the development of deep learning in recent years. We also introduce the recent advances in deep learning that have been applied in the thesis. The chapter ends with a literature review of deep reinforcement learning. Past multi-task and hierarchical deep reinforcement learning algorithms are also introduced in this section.

Chapter 3: A Novel Deep Reinforcement Learning Network Architecture

This chapter introduces the first contribution of the thesis: the novel network architecture. We first introduce the underlying ideas of the network architecture and then present the keys of implementation. We will also show theoretically how this network architecture will reduce parameters. Finally, we will present experimental results for evaluating the proposed network architecture. DDPG [24] is used for comparison in this chapter.

Chapter 4: A Novel Multi-Task Deep Reinforcement Learning Algorithm

This chapter introduces the second contribution of the thesis: the proposed multi-task deep reinforcement learning algorithm. The chapter includes detailed information on the algorithm architecture and learning process as well as experimental results for evaluating the algorithm.

Chapter 5: A Novel Hierarchical Deep Reinforcement Learning Algorithm

This chapter introduces the third contribution of the thesis: the proposed hierarchical deep reinforcement learning algorithm, h-DDPG. The chapter is also divided into three sections, one each for algorithm architecture, learning process and implementation details. We also present the experimental results for evaluating the proposed algorithm. This includes the performance of each level of hierarchy in the algorithm and a comparison to DDPG and Deep Q Network (DQN) [18].

Chapter 6: Conclusions and Future Work

This chapter summarizes the conclusions of the thesis, the limitations of the contributions and the directions for future work.

Chapter 2

Literature Review

Deep reinforcement learning algorithms have combined recent advances in deep learning [9] and reinforcement learning [25]. An effective combination of deep learning and reinforcement learning has both strong generalization capability inherited from its deep learning parts, and the ability to bootstrap skills or policies inherited from its reinforcement learning parts. Before deep reinforcement learning came into being, it was generally believed that non-linear techniques such as deep learning are hard to apply in reinforcement learning scenarios. This concern was first addressed when deep reinforcement learning was used to play Atari games [18]. The success of this work [18] has inspired people's interests in further improving deep reinforcement learning algorithms and applying it in different applications.

In this chapter, a thorough literature review of the related topics is presented. This includes an overview of classical reinforcement learning research in Section 2.1 and an overview of recent developments in deep learning techniques in Section 2.2. A review of previous works on deep reinforcement learning is presented in Section 2.3. Some key techniques that this thesis is based on are also introduced in these three sections, depending on their corresponding topic branches. Section 2.4 is a summary of this chapter.

2.1 Reinforcement Learning Overview

Reinforcement learning [25] algorithms aim to address the problem of how agents should learn to take actions to solve tasks in a given environment. Different from other methods such as dynamic programming and imitation learning, which require past

experience or knowledge from a human or well performing agents, reinforcement learning algorithms rely on agents' learning by trial-and-error to bootstrap skills or policies for solving tasks.

The following three subsections review literature on classical reinforcement learning. This includes a review of typical reinforcement learning setups, of well-known classical reinforcement learning algorithms and of a reinforcement learning architecture which is important for the work presented in this thesis.

2.1.1 Reinforcement Learning Setups

A standard reinforcement learning setup always consists of an agent and an environment $\boldsymbol{E}$ with which the agent interacts. The learning of the agent is dependent on the interaction between it and the given environment, as shown in Figure 2.1.

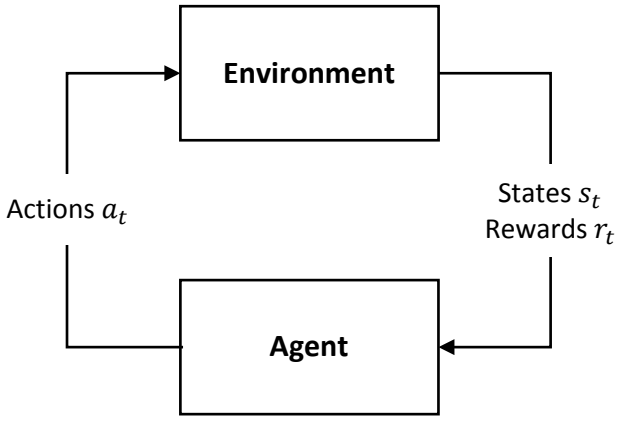

Figure 2.1: The interaction between the agent and the environment in a standard reinforcement learning setup.

Specifically, in each timestep t, the agent first receives a state $s_t \in \boldsymbol{S}$ from the environment, and chooses and executes an action $a_t \in \boldsymbol{A}$ according to the current policy π: $\boldsymbol{S} \rightarrow \boldsymbol{A}$. Then the agent will receive a reward r_t for taking a_t and transition to the next state s_{t+1}, where the process starts again.

Note that, the state is an observation of the environment and contains information to describe the current situation that the agent is in. In most partially observable

environments where agent cannot observe the entire environment in each timestep, the state that the agent receives from the environment degrades to an observation $o_t \in \boldsymbol{O}$, which indicates that what the agent receives cannot fully describe the current state.

Also note that the definition of timestep t may vary. For discrete time, each timestep contains an equal length of time, while for continuous time, the length of timesteps varies. What is more, the policy π may be stochastic, where $\pi: \boldsymbol{S} \rightarrow P(\boldsymbol{A})$, and P indicates a probability. However, for the rest of the thesis, we only focus on deterministic policies where, as shown above, $\pi: \boldsymbol{S} \rightarrow \boldsymbol{A}$.

Besides the standard setup, there are also some other setups being introduced for special reinforcement learning implementations. For example, there is hierarchical reinforcement learning [26] setup which contains additional hierarchies inside of the agent. There is also intrinsically motivated reinforcement learning [27] setup where agents do not need rewards from the environment. These two setups are shown in Figure 2.2.

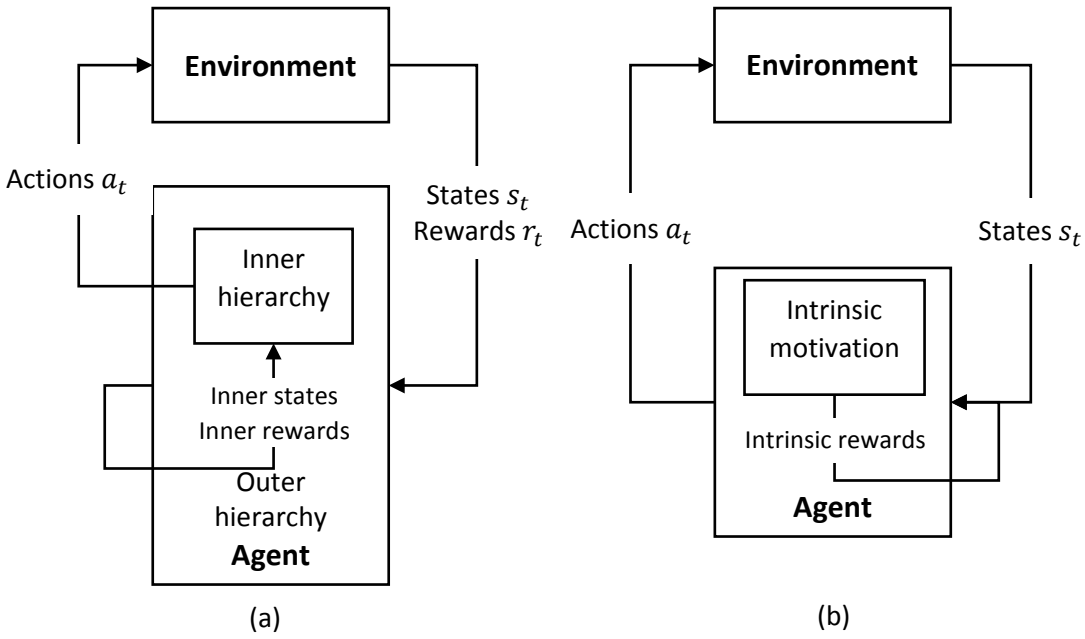

Figure 2.2: Interaction between the agent and the environment in (a) hierarchical reinforcement learning setup and (b) intrinsically motivated reinforcement learning setup.

We can see from Figure 2.2(a) that, in a hierarchical reinforcement learning setup, the decision of which action to execute is made by the inner level hierarchy of the agent

and may be influenced by any of the hierarchies in the agent. Different levels of states and rewards may be introduced for each level of the hierarchy. Some of the hierarchies may be able to be reused by the agent when the environment or the goal has changed. With regards to intrinsically motivated reinforcement learning setups, the agent will not receive any rewards from the environment. Instead, the rewards will be replaced by some intrinsic motivation techniques inside of the agent which generate intrinsic rewards to guide the agent. This allows the agent to make decision depending on its own motivations.

2.1.2 Classical Algorithms

Generally, there are two kinds of reinforcement learning algorithm: value based algorithms and policy based algorithms. Value based algorithms mainly focus on learning estimations of the value of the states or actions. Then the agent can choose actions according to these estimations with some implicit policies such as $\epsilon - greedy$ policy. On the contrary, policy based algorithms focus on learning an explicit policy function that will output the action decisions directly.

In fact, value based algorithms have been dominant in the past decades. A kind of well-known value based algorithm is called Temporal Difference Learning (TD-learning) [28]. TD-learning aims to learn how to predict a quantity that depends on future values of a given signal.

Q-learning [5] is one of the most commonly used TD-learning algorithms. Q values are central to Q-learning. They are estimations of values of actions in given states, which are also called state-action values. A Q value of each action under a given state represents an estimation of the expected future return of choosing that action under that state. The expected future return is defined as:

$$R_t = \sum_{t\prime=t}^{T} \gamma^{t'-t} r_{t\prime} \tag{2.1}$$

Where γ is the discount factor that indicates to what extent future reward will be taken into account.

Q-learning will bootstrap the estimation of the expected future return by updating the estimation using:

$$Q'(s_t, a_t) = Q(s_t, a_t) + \mu(r_t + \gamma \cdot \max_a\{Q(s_{t+1}, a)\} - Q(s_t, a_t))) \quad (2.2)$$

where μ is the step size, also known as learning rate. The term inside the bracket aside α is known as TD error. It is the error that provides the update direction of TD-learning algorithms. The aim is to make the estimation more accurate as updates continue so that after several updates, we can have:

$$Q(s_t, a_t) = E[R_t|s_t, a_t] \quad (2.3)$$

Then, in each timestep, the agent can choose an action that has the maximum Q value. As the estimation gets more accurate, the chosen action can lead the agent to collect more rewards.

Another well-known TD-learning algorithm is State-Action-Reward-State-Action (SARSA) [6]. Different from Q-learning, SARSA updates the estimation using:

$$Q'(s_t, a_t) = Q(s_t, a_t) + \mu(r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))) \quad (2.4)$$

The main difference is in the TD error term, specifically the way of inferring future reward. While Q-learning uses the maximum Q value across all available actions as the estimation of the future reward in state s_{t+1}, SARSA uses exactly the Q value of the chosen action in state s_{t+1}. As a result, while Q-learning updates the estimation as soon as the agent gets to state s_{t+1}, SARSA updates it after the action a_{t+1} has been chosen in s_{t+1}, which is one step later than Q-learning. This can make the learning more stable in some tasks.

Many value based algorithms introduce linear function approximations to approximate the Q values [7, 8]. These linear functions can add to the generalization ability of the algorithm, especially in complex environments or tasks. However, most value based algorithms can only solve discrete action control tasks where only a limited number of legal actions are available.

With regards to policy based algorithms, the Reward Increment Nonnegative Factor Offset Reinforcement Characteristic Eligibility (REINFORCE) algorithm [29] is one representative. The REINFORCE algorithm generally uses an explicit policy function to

bootstrap the task-solving policies [30]. As a result, without any value estimation, the agent can choose appropriate actions by directly using the policy function which maps each individual state to a certain action output. In this way, the REINFORCE algorithm extends reinforcement learning to scenarios that involve a more complex control spaces, such as continuous control spaces, where there are potentially unlimited action choices.

However, the REINFORCE algorithm learns much more slowly than value based algorithms. It is then believed that the assistance of a value function is essential for rapid learning [31]. Following this belief, a number of ways have been proposed to combine policy based methods with value based methods. The actor-critic architecture [32] and policy-iteration architectures [33-35] are successful architectures that can fit both value and policy functions into the same system.

2.1.3 The Actor-Critic Architecture

Different from other value based or policy based algorithms, actor-critic architecture [32] based algorithms have the characteristics of both value based and policy based algorithms. A typical actor-critic architecture is shown in Figure 2.3.

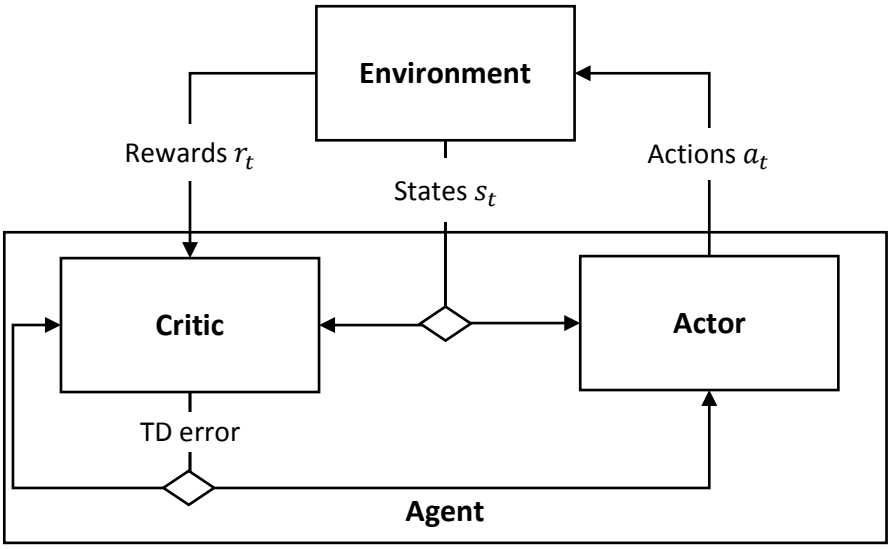

Figure 2.3: Typical actor-critic architecture.

We can see that the actor and critic are two key components in the architecture. The critic is functioning like the value function. It updates according to an update rule that is

similar to Q-learning or SARSA. The actor is functioning like the policy function. Different from pure policy based algorithms like REINFORCE, the actor updates based on information provided by the critic. Note that, the way that the actor and critic are realized may vary in different algorithms. The way the actor uses the information from the critic may also vary depending on different implementations [36, 37]. This shows the flexibility of the architecture.

Research shows that actor-critic algorithms can have strong capability in learning policies for complex action spaces and can also be fast in learning. Therefore, it is considered a very suitable architecture for learning in a continuous action space. Many actor-critic algorithms [38-40] in different domains have achieved very good results.

2.2 Deep Learning Overview

In 1943, a computational model based on threshold logic was created, which is seen as the ancestor to neural network models [41]. This model was then developed in a two-layer computer learning network in 1958 by Frank Rosenblatt, who also created early concepts of the perceptron [42]. However, training these neural networks is not easy [43], and neural network research stagnated for some time.

The challenge with neural network algorithms were first addressed by the backpropagation algorithm proposed in 1975 [44]. Based on the backpropagation algorithm, a successful neural network model was proposed in 1998 to recognize handwritten digits in documents [45]. This model is also seen as a very classical deep learning model. Unfortunately, deep learning research again stagnated because of the limit of computational resources and training data in those days. Finally, in recent years, advances of high performance computational devices and the development of big data have revitalised deep learning research and made it one of the most important techniques in artificial intelligence.

In this section, a review of deep learning is presented. The following subsection is a wrap up of recent advances in deep learning research. Then the rest of the subsections

are reviews on the training strategies of deep learning models and an important type of neural network layer for this thesis.

2.2.1 Recent Advances in Deep Learning

In recent years, the presence of high performance Graphics Processing Units (GPUs) and the accessibility of big data have pushed deep learning to become one of the most influential topics in academia. The first breaking news came in 2012, when Hinton won first place in the LSVRC-2012 competition with his deep convolutional neural networks (CNN) model [10]. The competition is about an image recognition task with an image dataset called ImageNet [46], which contains 1,000 different categories. The proposed model, which is shown in Figure 2.4, broke the record at that time by improving the accuracy by more than 10% compared to traditional methods.

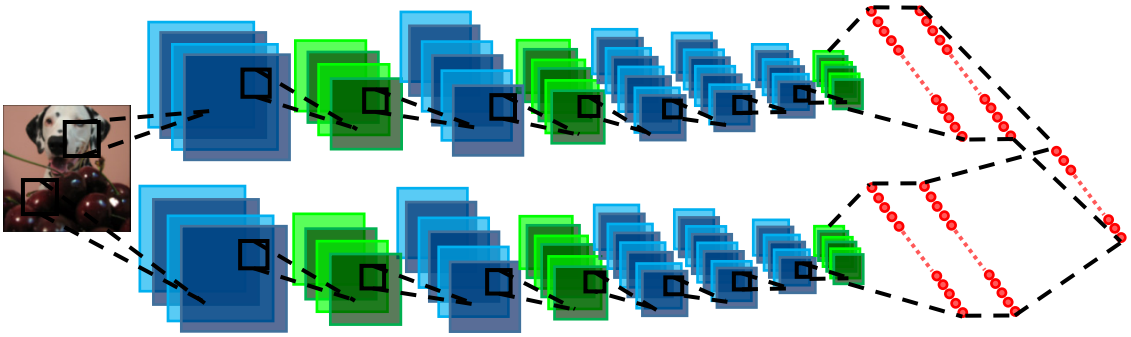

Figure 2.4: The architecture of the deep convolutional neural network proposed by Krizhevsky et al. [10] for image classification problems. The blue squares are convolutional layers while the green ones are max pooling layer.

The parameter sharing nature of the convolutional layer and the max pooling operation that follows it can not only reduce the number of parameters needed for extracting features from raw image data, but also add to the robustness of the model against shifting or rotating distortions in the images. With hundreds of thousands of auto-learned parameters, these deep learning models have shown strong generalization capability and unprecedented feature extraction ability. Deep learning has freed people from designing hand-crafted features for each new application.

The success of Krizhevsky et al. [10] quickly attracted people's attention and a series of record-breaking works on classifying ImageNet dataset have been proposed in the following years. These include a refinement of Krizhevsky et al. 's model in [10] that won the LSVRC-2013 title [47], two even deeper neural network models that won first [48] and second [49] place in LSVRC-2014, and a model that contains 152 layers that won the LSVRC-2015 title [50].

While the networks are getting deeper and deeper, there are also many works that focus on finding more effective network architectures. The long-short term memory (LSTM) network [11] is one of the successful innovations on designing new network architectures. LSTM networks can effectively solve problems that need to combine sequential or past information, such as optical character recognition (ORC) [51], speech recognition [11] and so on.

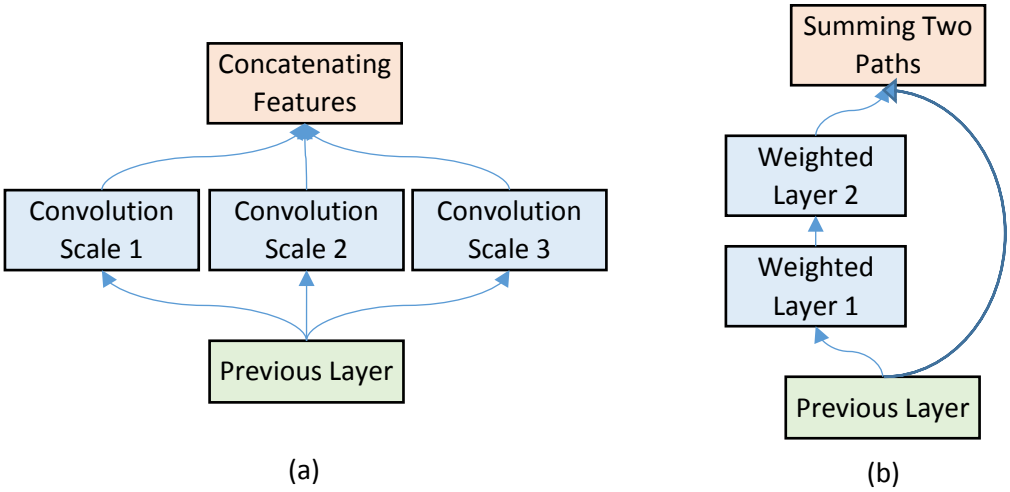

Figure 2.5: A local layer rollout of (a) the Inception architecture and (b) the ResNet architecture. These figures just show the basic concepts of these two architecture.

The Inception architecture [48] (Figure 2.5(a)) is also an effective deep learning architecture. It uses parallel inputs with different reception fields within the same layer to enable the network to combine features from different scales. Another successful architecture is the residual network (ResNet) [52] (Figure 2.5(b)). By alternatively switching between a short path and a long path in a residual block, ResNet can effectively train very deep networks. Both the Inception architecture and ResNet architecture have received considerable attention in academia and are considered as

important forms of future deep learning models. At lot of work has been done to improve these two architectures [50, 53-55].

What is more, there are works that aim to apply deep learning in other fields rather than image classification. One of the important applications is object detection. A breakthrough in this topic was the presence of Region with CNNs algorithm (R-CNN) [56]. It uses CNN to first detect several candidate regions and then to select regions that contain specific objects. Following, several works have improved the speed and accuracy of R-CNN, most famously Fast R-CNN [57], Spatial Pyramid Pooling (SPP) [58], Fully CNN [59] and Faster R-CNN [12]. These new algorithms have not only achieved state-of-the-art results that can be applied in our daily life, but also have led us to a better understanding of deep learning.

However, deep leaning still has some major drawbacks. First, its strong generalization capability is based on the availability of huge amounts of data. Otherwise, it will suffer from overfitting on the training data and fail to work on data that it has not seen. Secondly, training deep learning models requires huge amounts of computational resources, while a well-trained model may also need a lot of memory space that some embedded systems cannot afford. This has confined deep learning to be applied only on high performance computers. A lot of work has been done to address these two challenges.

To solve overfitting problems, a number of regularization techniques have been proposed. The most common way is to add regularizations on the loss function of the network. In addition, data augmentations are also very practical strategies. Traditional augmentation techniques including horizontally flipping, random rotations and shifting, and fancy PCA [10] can help prevent overfitting by considerably increasing the amount of data by adding pseudo-data into the original datasets. DropOut [60] and DropConnect [61] are also very successful techniques in dealing with overfitting. They achieve this by reducing the interdependence between neurons during training. In practice, these regularization techniques are usually combined to achieve the best generalization result.

With regards to the heavy memory and computational resource requirement, there are also many works done to achieve lighter parameterized network and training. Low-

rank filters decomposition [62] and connection pruning [63] are two successful techniques to reduce the number of parameters in the network after training. By dropping connections or transforming the dimension of the convolutional kernels, these two techniques also considerably speed up the feedforward computation of the network. In the meantime, techniques such as hashing trick [64] and binary connect [65] have been proposed to significantly compress the memory space needed to record a trained network. These compression techniques improve the efficiency of parameter saving and allow deep networks to be applied in low performance devices.

After a rapid growth in recent years, deep learning methods now are becoming more and more recognized in many research fields. New deep learning development is coming to make it more generalized and efficient.

2.2.2 Recent Advances in Training Optimization

Training optimization is central to any deep learning algorithm. Finding effective ways to update the network is the prerequisite of applying any deep learning models. All the training optimizations nowadays are based on the backpropagation algorithm proposed in 1975 [44]. As deep learning models are getting deeper and more architecture variations have been proposed, simply applying backpropagation seems to be insufficient as bigger models are more likely to suffer from training instability or gradient problems (such as gradient explosion).

Generally speaking, the update methods for adaptive machine learning algorithms can be grouped into three classes: (1) stochastic gradient descent (SGD) methods, which uses one single sample per update; (2) batch gradient descent methods, which uses the whole dataset for updating the model in each update iteration; and (3) mini-batch gradient descent methods, which sample a mini-batch from the dataset to update the network in each iteration. As mini-batch gradient descent is more stable than SGD and also more data-efficient than batch gradient descent methods, it has become the dominant gradient update rule in deep learning. Combined with the backpropagation algorithm, the update rule is:

$$\theta' = \theta - \mu\nabla_\theta \frac{1}{M}\sum_{i=1}^{M} L(X_i, y_i) \tag{2.5}$$

where M is the mini-batch size, ∇ indicates gradient calculation, θ is the network parameter, X_i is the final output of the network of the i^{th} sample in the mini-batch and y_i is its corresponding supervising signal (also known as the label or ground truth). $L(\cdot)$ is the loss function that the training is trying to minimize.

In order to accelerate training, a technique called momentum [66] has been proposed to accumulate past updates to push future update towards the dominant direction. With momentum, the update rule becomes:

$$v' = \delta v + \mu\nabla_\theta \frac{1}{M}\sum_{i=1}^{M} L(X_i, y_i) \tag{2.6}$$

$$\theta' = \theta - v' \tag{2.7}$$

where δ is the momentum term that indicates the speed of the accumulation.

Nesterov Accelerated Gradient (NAG) [67] is also a training optimization to accelerate training. It is similar to momentum, except for taking the second derivative (or called the Hessian) of the gradient into account:

$$v' = \delta v + \mu\nabla_\theta [\frac{1}{M}\sum_{i=1}^{M} L(X_i, y_i) - \delta v] \tag{2.8}$$

$$\theta' = \theta - v' \tag{2.9}$$

In recent years, new training optimizations have been proposed. These new optimizations include RMSProp [68], Adagrad [69], Adadelta [70] and Adam [71]. The basic idea of these optimizations is to find a way to make the network less sensitive to the value of hand-crafted learning rates. Each of these optimization methods has its own way to adjust the value of the learning rate during training according to some rules or analysis of past updates. Therefore, they are also called adaptive learning rate optimizations.

While all these adaptive learning rate optimizations can significantly accelerate and stabilize training, Adam optimization stands out as the most general training optimization that can fit in different deep learning models. The update rules are:

$$m' = \beta_1 v + (1 - \beta_1)\nabla_\theta \frac{1}{M}\sum_{i=1}^{M} L(X_i, y_i) \tag{2.10}$$

$$v' = \beta_2 v + (1 - \beta_2)[\nabla_\theta \frac{1}{M}\sum_{i=1}^{M} L(X_i, y_i)]^2 \tag{2.11}$$

$$\widehat{m} = \frac{m'}{1 - \beta_1^{iter+1}} \tag{2.12}$$

$$\hat{v} = \frac{v'}{1 - \beta_2^{iter+1}} \tag{2.13}$$

$$\theta' = \theta - \frac{\mu\widehat{m}}{\sqrt{\hat{v}} + \varepsilon} \tag{2.14}$$

where $iter$ indicates the number of updates taken. β_1 and β_2 are called exponential decay factors which are close to 1. ε is a small constant for numerical stability.

By adaptively adjusting the learning rate according to the momentum of the squared gradients, Adam optimization achieves a training optimization that is insensitive to any hand-crafted hyper-parameters.

2.2.3 The Multi-Layer Perceptron Convolutional Layer

Another important topic in deep learning research is about network layers. Unlike developments in network architecture or training optimization that take care of the big picture of learning, network layer developments focus on designing new types of layers that can boost the network's ability in a certain aspect.

For example, a new type of convolutional layer was proposed by Sun, et al. [72], which is called a local convolutional layer. Different from a traditional convolutional layer that uses a shared convolution kernel for each of the individual feature maps, a local convolutional layer can have more than one convolution kernel for a single feature map, each one responsible for a specific region in the map. This layer can add to the network's ability to apply different extracting parameters for different regions in the image. Another example is a frictional max pooling layer [73]. Unlike a traditional pooling layer that has a fixed stride size, a frictional max pooling layer can change stride size in

each pooling step. This can prevent information being missed because of the down-sampling nature of the pooling layer.

The multi-layer perceptron convolutional (mlpconv) layer [23] is also a new type of network layer. The mlpconv layer was first proposed for image classification problems. A classical network with mlpconv layers is shown in Figure 2.6.

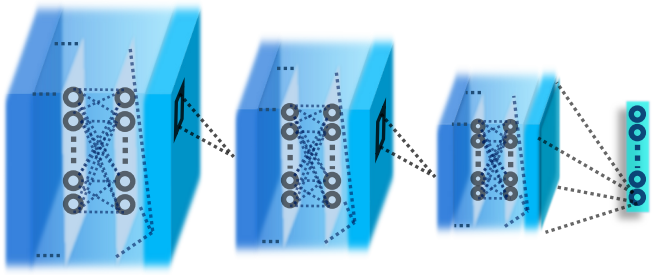

Figure 2.6: A classical network with mlpconv layers. A global average pooling layer is applied in the last mlpconv layer. The network is aimed to solve image classification problems.

A traditional convolutional layer simply activates feature maps with an activation function $F(\cdot)$ using:

$$f_{i,j,k} = F(\omega_k^T x_{i,j} + b_k) \tag{2.15}$$

where (i, j) indexes the pixels (also known as neurons or nodes) in the k^{th} feature map. ω_k and b_k are the weight and bias, which are the learning parameters of the convolution kernel for the k^{th} feature map.

As the activation function $F(\cdot)$ is usually chosen to be a linear function in a deep neural network to accelerate training and avoid gradient vanishing, in a traditional convolutional layer, the feature maps of a layer are usually a linear rendering of the feature maps of the previous layer. However, good abstractions are generally achieved by applying non-linear functions to the data. Therefore, in order to compensate with the linear activation function, a mlpconv layer applies a recombination of feature maps across different channels with some multi-layer perceptron operations before feeding the feature maps into the next layer. As a result, the recombined feature map outputs of the mlpconv layer becomes:

$$f_{i,j,k_n}^n = F\left(\omega_{k_n}^{n\,T} f_{i,j}^{n-1} + b_{k_n}\right) \tag{2.16}$$

where n is the number of multi-layer perceptron operations used and $f_{i,j}^{0} = x_{i,j}$. In practice, it is set to $n = 2$ [23].

For classification problems in [23], the final classification output can be obtained by simply applying a global average pooling layer on the feature maps of the last mlpconv layer. This network has achieved state-of-the-art results for several classification problems, while it also reduces the number of parameters needed compared to traditional CNNs by removing all fully-connected layers from the network. This is feasible owing to the mlpconv layers in the network that can provide features that are representative enough for the network to infer the results even with a non-parameterized layer.

2.3 Deep Reinforcement Learning

The success of deep learning in generalizing big datasets and automatically extracting features from raw image inputs has inspired interest in combining reinforcement learning with deep learning to further improve the performance of the agents, especially in case that complex environments or controls are involved.

However, it is generally believed that non-linear approximations like deep neural networks are difficult to converge in reinforcement learning scenarios. This is especially the case for deep learning models that are very likely to get overfitting and require well-distributed labelled data. The correlations between consecutive states, the tendency of dropping in biased exploration and the sparse reward signals in reinforcement learning are challenges that need to be considered for combining deep learning and reinforcement learning. All these challenges were finally addressed by Mnih, et al. [18], which has also brought deep reinforcement learning great success. It proved that deep learning can be used as an approximation function that is much more powerful than any other former linear approximations used in reinforcement learning.

The reminder of this section starts with a subsection of overview of deep reinforcement learning developments in recent years. It is followed by an introduction

of two well-known deep reinforcement learning algorithms and a more focused literature review on the main topics of this thesis.

2.3.1 The New Generation of Intelligent Agents

The first successful deep reinforcement learning algorithm was deep Q network (DQN) [18] proposed in 2013. It combined advances in reinforcement learning such as a replay memory mechanism [74] to address the challenges in training deep learning models in reinforcement learning scenarios. Game agents trained with DQN achieved state-of-the-art performance in playing a variety of Atari games. The unprecedented performance of DQN and its ability to learn from raw image frames of the games shocked the artificial intelligence community.

Deep reinforcement learning quickly gained popularity and lots of new deep reinforcement learning works emerged out. Many works have focused on applying deep reinforcement learning in different types of control tasks. For example, Deep Deterministic Policy Gradient (DDPG) [24] is the first deep reinforcement learning algorithm to handle continuous control scenarios. Following it, the Stochastic Value Gradients (SVG) [75] algorithm was proposed to handle continuous control scenarios with stochastic policy gradients, which allows on-policy training (training online). In addition, Normalized Advantage Functions (NAF) [76] and Trust Region Policy Optimization (TRPO) [77] are also continuous control deep reinforcement learning algorithms which introduce deep learning models in the algorithms. In cases where large discrete action spaces are involved, Wolpertinger [78] outperformed traditional methods by combining DDPG with embedded mapping. What is more, works also have been done in solving parameterized action space control problems with deep reinforcement learning [79]. Other contributions [80-83] are all successful works in fitting deep reinforcement learning to specific types of control tasks.

There are also many works that focus on improving existing deep reinforcement learning algorithms by fusing different techniques. For example, in order to make training more stable, work has been done to introduce priority in the replay memory so

that the agent can see important transitions more frequently [84]. Other work [85] also improves learning stability by increasing the gap between different actions. The introduction of target networks can also stabilize learning. It helps DQN achieve human-level performance in playing all types of Atari games [19]. Some other works [86, 87] also contribute to this topic. Another way to improve existing algorithms is to improve the learning efficiency. This is generally achieved by some pre-learning techniques that help the agent find updating direction earlier during formal training [88, 89].

Moreover, there are works that aim to add new ability to the agent trained by deep reinforcement learning algorithms. A good example is the introduction of recurrent networks in the agent [90-92]. These recurrent networks such as LSTM [11] can give the agent the ability to memorize past experience, so that the agent can make decisions that are based on not only the current state, but also the previous states. This enables the agent to solve partially observable tasks, where the final goals are hidden or only partially observable in the environment. Another way to add memory ability is through episodic control [93]. In scenarios where the task cannot be solved with a single agent, the ability of cooperating with other agents needs to be added. This can be achieved by either introducing communication skills to the agent system [94-96] or by finding ways to balance gradient propagation among all available agents when updating networks for each of the agents [97].

It is worth mentioning that the performance of deep reinforcement learning agents is still improving even though they have already achieved significantly better performance compared to classical reinforcement learning agents. For example, asynchronous methods [98] have freed deep reinforcement learning from the replay memory mechanism. This is achieved by implementing multi-thread training schemes that can make use of transitions of multiple learning agents. While this method can have some similar functions to the replay memory, it can also make deep reinforcement learning algorithms become on-policy, which is more data-efficient. Another example is the work that introduces the ResNet network architecture in the algorithm to significantly improve the agent's capability in extracting information from raw image data, which achieves target-driven control where the target is given in the form of pictures [99].

Deep reinforcement learning algorithms have also been successfully applied in many other applications. These include application such as semantic analysis [100-103], computer vision [104-106], management systems [107-109] and others [110-113]. While deep reinforcement learning agents can help improve performance in all these applications, a breakthrough was in playing the Go game, which is regarded as the most complex intellectual game in human history. A Deep reinforcement learning based agent, called AlphaGo [20], has beaten the legend and champion Go player, Lee Sedol, in a five-set match. This announced that a new generation of artificial intelligence is about to begin.

2.3.2 Deep Q Network and Deep Deterministic Policy Gradient

Deep Q Network (DQN) [18] was the first successful algorithm that combines deep learning with reinforcement learning. The algorithm was proposed to handle discrete control tasks. It uses deep learning models as an approximation of the Q values and can learn and infer Q values using raw image data. An overview of DQN is shown in Figure 2.7.

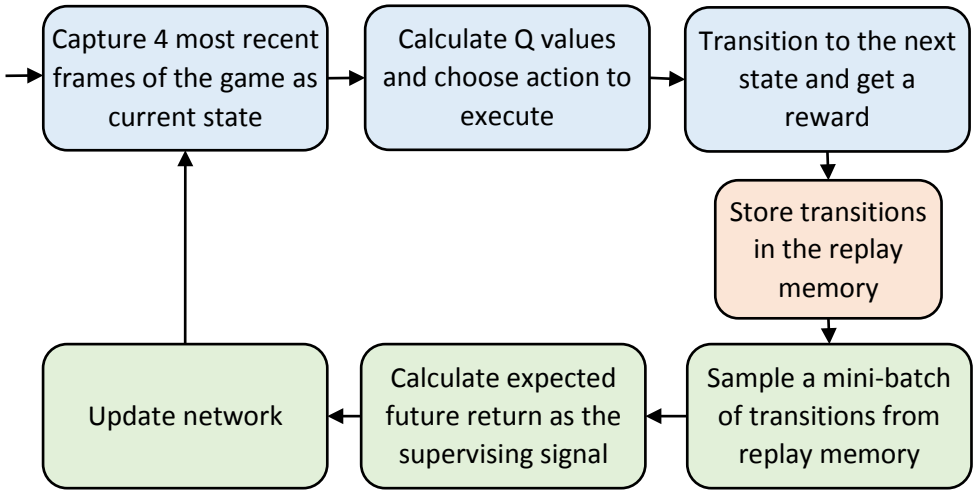

Figure 2.7: An overview of DQN [18]. The replay memory (in the yellow box) breaks learning into two parts, exploration (in blue boxes) and training (in green boxes).

There are mainly two techniques in DQN which are fundamental to address the

challenges for combining deep learning and reinforcement learning. The first one is replay memory, which is a memory pool that stores all past transitions the agent has experienced. This is necessary because on-policy learning is not suitable for training deep learning models. The correlations between consecutive inputs in on-policy learning methods can easily drive deep learning models to get overfitting. DQN addresses this by randomly sampling a mini-batch of transitions from the replay memory. This de-correlates data as the transitions sampled in each iteration may come from different exploration periods. The introduction of replay memory actually splits learning into two separate phases: exploration and learning. Even though these two phases process alternately, which may make it look like a learning-while-exploring method, it is actually an off-policy algorithm (training offline).

The second key technique is called target networks [19]. In addition to a Q network, another target Q network is introduced in the algorithm. The parameters of the target Q network θ^t is first initialized to the same values of the Q network. But unlike the Q network that is updated every training iteration, the target Q network will only be updated at certain times during training by being synchronized to the current Q network. It can be regarded as a slowly updated version of the Q network and is used when calculating the supervising signal. It is necessary because, unlike image classification problems where every data has its own label as its supervising signal, the supervising signal in deep reinforcement learning scenarios, which is the expected future return, needs to be calculated during training. As a result, the supervising signal is actually a changing value because the estimation of the Q value of a certain state will be changed every time the network has been updated. The slowly updated target Q network here is therefore introduced to provide more stable supervising signals to stabilize the learning process.

What is more, the exploration of DQN is always governed by a $\epsilon - greedy$ policy. This helps balance the exploration with exploitation and improve the quality of the transitions in the replay memory, which can further prevent networks overfitting. This is especially important in the early stage of learning, when the agent can easily get stuck at a local optima.

However, even though DQN is a successful algorithm, the nature of Q-learning has limited DQN to discrete action control scenarios. When facing continuous control problems where there are unlimited action choices, DQN is easily overloaded by the high dimension of the action space.

Deep Deterministic Policy Gradient (DDPG) [24] was the first deep reinforcement learning algorithm that can handle a continuous action space. DDPG is based on the actor-critic architecture [32] that is more capable in dealing with complex action space. The introduction of deep learning models in the actor-critic architecture also enable the agent to learn from raw image data. An overview of DDPG is shown in Figure 2.8.

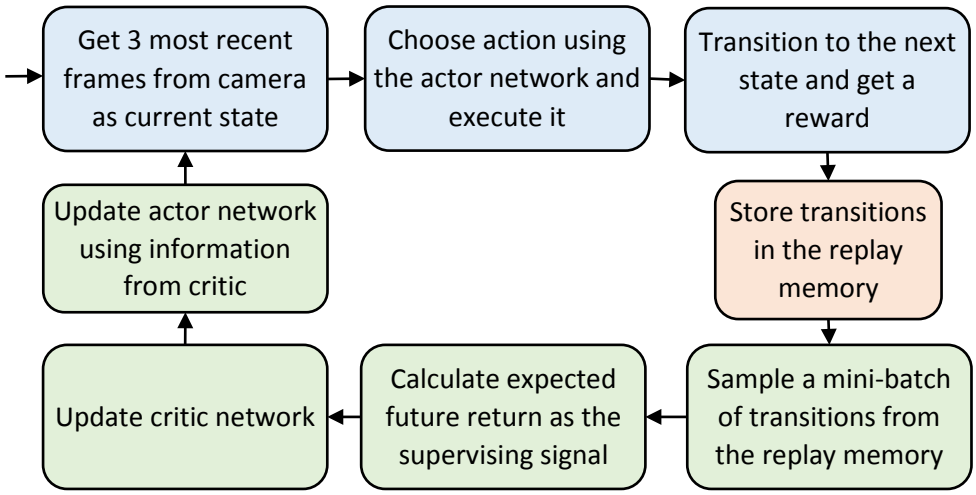

Figure 2.8: An overview of DDPG [24]. Similar to DQN, the replay memory (in the yellow box) breaks learning into two parts, exploration (in blue boxes) and training (in green boxes).

DDPG inherits the two key techniques mentioned above from DQN. The main difference is that there are two deep neural networks in DDPG: the actor network, which can also be regarded as a policy network (θ^{π}), and the critic network, which is also known as a value network or Q network (θ^{Q}). Like the case in any other actor-critic based algorithms, the actor is a complex policy that can provide actions and the critic is consistently estimating the value of the action chosen by the actor to guide the actor to learn.

As a result, after a mini-batch of transitions have been sampled from the replay memory, the critic network will be first updated to minimize the loss function:

$$L(\theta^Q) = (Q(s_t, a_t|\theta^Q) - y_t)^2 \quad (2.17)$$

where the supervising signal y_t is calculated before updating using the target critic network ($\theta^{Q,t}$) and the target actor network ($\theta^{\pi,t}$) as:

$$y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}|\theta^{\pi,t})|\theta^{Q,t}) \quad (2.18)$$

And the update rule of the critic network is:

$$\theta^{Q'} = \theta^Q - \mu_Q \cdot \nabla_{\theta^Q} L(\theta^Q) \quad (2.19)$$

After the critic network has been updated, the actor network will be updated using gradient information from the critic network. Specifically, this is achieved by using the deterministic policy gradient calculating rule [114]:

$$\theta^{\pi'} = \theta^\pi - \mu_\pi \cdot \nabla_a Q(s_t, , \pi(s_t|\theta^\pi)|\theta^Q) \cdot \nabla_{\theta^\pi} \pi(s_t|\theta^\pi) \quad (2.20)$$

Note that the above update rules are based on the mini-batch gradient descent training optimization. The exact update rules can be changed according to the form of training optimization applied.

Similar to the exploration of DQN, the exploration of DDPG is always governed by an Ornstein-Uhlenbeck process [115]. This process can randomize the action choosing process to allow maximum exploration in the continuous action space. It is important for the agent especially in early stage of learning.

2.3.3 Multi-Task and Hierarchical Deep Reinforcement Learning

Although a lot of work has been done to improve deep reinforcement learning algorithms over single tasks, there is much less work done for multi-task scenarios, although some exists [116-118]. Bangaru, et al. [116] mostly focused on the exploration phase of deep reinforcement learning. They aimed to find ways to generalize several well-learned models of different tasks to one model. Borsa, et al. [117] learn universal abstractions of different tasks so that they can be reused when learning. Zhang, et al. [118] aim to learn successor features that the agent can transfer from one task to another. Although all these works involve multiple tasks, the tasks are not learned at the same time. Actually, the work of Bangaru, et al. is close to model compression work and the work of Borsa, et al. and Zhang, et al. are similar in nature to transfer learning [22].

Another topic that may share some common points with multi-task learning is multi-agent learning [94, 95]. However, even though multiple tasks are involved, these works need at least two different agents, while multi-task learning permits one agent to learn multiple tasks.

With regards to hierarchical deep reinforcement learning, work also exists found in [119, 120]. Kulkarni, et al. [119] achieved hierarchical learning by embedding an additional DQN structure into the agent. Therefore is limited to handle discrete action control tasks. Krishnamurthy, et al. [120] mostly focused on finding the hierarchical structure of the tasks with clustering methods. Their contribution is on analyzing task structure and decomposing complex tasks prior learning.

Another related topic to hierarchical learning is intrinsically motivated learning which also may involve multiple levels of hierarchy in the agent. However, only a few works use deep reinforcement learning to achieve intrinsic motivation. Recent work is the algorithm proposed by Mohamed and Rezende [121] that has successfully achieved an intrinsically motivated agent by replacing reward functions with a maximization of the mutual information during learning. But this work is again limited to discrete action control.

Multi-task and hierarchical learning are very time and computationally efficient ways for the agent to learn. Therefore, it is worthwhile making use of the capability of deep reinforcement learning to achieve multi-task and hierarchical learning, especially in continuous action space scenarios, where limited work has been done.

2.4 Summary

Deep reinforcement learning algorithms combine deep learning techniques with reinforcement learning methods. Although it is generally believed that non-linear function approximations like deep learning models are difficult to converge in reinforcement learning scenarios, recent advances in reinforcement learning have successfully addressed these challenges and brought deep reinforcement learning success. The first success in deep reinforcement learning was Deep Q Network [18].

Inspired by this [18], a lot of work appeared to improve deep reinforcement learning algorithms or adapt it to different types of control tasks. Moreover, many works apply deep reinforcement learning in other areas. The most successful ones are deep reinforcement learning agents that outperformed humans in playing Atari games [19] and Go games [20].

However, much less work has been done to use deep reinforcement learning to achieve multi-task and hierarchical learning. This is especially the case for learning in a continuous action space. This has inspired our work in this thesis.

In this thesis, we propose two novel deep reinforcement learning algorithms for multi-task and hierarchical learning in continuous action spaces. A novel deep reinforcement learning network architecture is also proposed to reduce the number of parameters needed. This is introduced in the next chapter.

Chapter 3

A Novel Deep Reinforcement Learning Network Architecture

The work presented in this chapter has been partially published in the following paper:

Zhaoyang Yang, Kathryn Merrick, Hussein Abbass, Lianwen Jin, Multi-Task Deep Reinforcement Learning for Continuous Action Control. Proceedings of International Joint Conference on Artificial Intelligence, 2017. [Accepted on April 23, 2017].

Specifically, Section 3.1 and experimental results in Section 3.7 were written based on texts or/and results in the above paper.

As the first author of the above paper, I designed the proposed network architecture, built the simulation and implemented the architecture on it, and wrote the majority of the paper on my own.

Recent advances in reinforcement learning have addressed the challenges of combining deep learning with reinforcement learning and brought deep reinforcement learning great success. Some work also successfully applied recent deep learning techniques, such as the ResNet architecture [52], to further enhance the capability of deep reinforcement learning agents [99]. This has inspired our interest in fusing other deep learning methods with reinforcement learning.

Although deep reinforcement learning agents can handle very complex environments and action spaces owing to the strong generalization ability of deep learning models, they are very heavily parameterized (one of the characteristics of deep learning models) compared to classical reinforcement learning agents that only use linear function approximations. The number of parameters introduced in the agent will turn out to be a significant problem when the agent needs to learn multiple tasks within the same learning process. This is generally the case in multi-task learning or hierarchical learning scenarios.

In this chapter, we propose a novel network architecture for deep reinforcement learning agents. The proposed network architecture makes use of the multi-layer perceptron convolutional (mlpconv) layer to significantly reduce the number of parameters needed in a reinforcement learning agent. We first introduce the basic ideas behind this novel network architecture in Section 3.1. Then we introduce the network architecture and ways of implementing it in Section 3.2 and Section 3.3. Section 3.4 is a summary of the network architecture. The validation of the proposed network architecture is in Sections 3.5 to Section 3.8.

3.1 Inspirations

The unprecedented feature extraction ability of deep learning models have enabled deep reinforcement learning agents to learn from raw image data such as camera photos or game frames. However, this ability is based on a large number of parameters in the deep CNN networks. With a closer insight into the network architecture, we can understand that most of the parameters in a CNN are introduced when all feature maps of the last convolutional layer are flattened into a long vector to connect to the first fully connected layer. This is especially the case for CNNs in deep reinforcement learning agents, which have relatively smaller fully connected layers compared to the larger networks used in image classification problems.

The large number of parameters in between the last convolutional layer and the first fully connected layer becomes even more significant when the agent needs to handle multiple tasks. For example, at least 1,600,000 new parameters need to be introduced for a DQN agent to handle each additional task. For DDPG agents that have two deep networks, the number of new parameters needed for learning a new task-solving policy is at least 800,000, regardless of whether convolutional layers are shared between the actor network and the critic network. This huge number of parameters is not only tricky to train as it may tend to make the network overfit, but also redundant to some extent, especially when more tasks are involved. Therefore, it is important to find ways to reduce the parameters in the network.

The multi-layer perceptron convolutional (mlpconv) layer is a relatively new network layer [23] for image classification problems. In addition to a traditional convolutional layer, two perceptron layers are added to reconstruct the feature maps rendered by that particular convolutional layer, before these feature maps are fed into the next layer. This reconstruction of the feature maps aims to compensate with the linear activation function used in CNNs to make the final feature maps a non-linear rendering of the inputs. It can merge information across different channels and enable the network to learn more abstract features. With the help of this reconstruction, the network achieves excellent classification performance by simply implementing an average pooling layer on the last mlpconv layer.

In our proposed network architecture, we make use of the mlpconv layer to extract more features in a more abstract level, while also reducing the parameters needed in-between the convolutional layers and fully connected layers. The basic idea is that compared to other image processing problems, such as image classification and object detection, the information the agent needs to infer from images is less complex, especially when the agent is learning basic skills. Thus, we do not need pixel level information in the feature maps.

Instead, by implementing mlpconv layers, the feature maps can reach a more abstract level. Then with a global average pooling layer, we can obtain a shorter feature vector. This shorter vector may lose some detailed information about the environment, but, with the help of the previous mlpconv layer, the short vector can contain more refined information about the environment. This is beneficial for the agent as it can be more focused on learning skills without being confused with irrelevant information in the environment. The short vector can also help us reduce the number of parameters needed when it is connected to the fully connected layers of the network, compared to the long vector from flattened feature maps of traditional convolutional layers.

3.2 General Architecture of the Proposed Network

The proposed network architecture is shown in Figure 3.1.

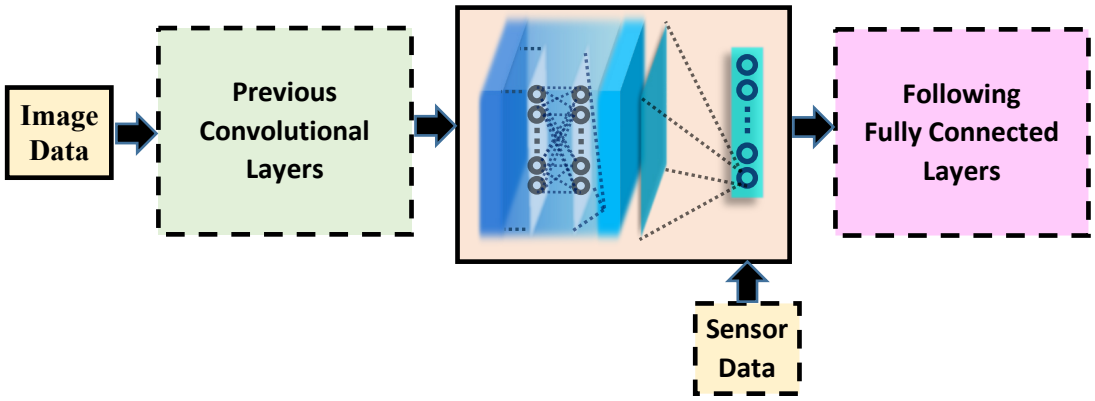

Figure 3.1: The general architecture of the proposed network. The text inside the boxes with dashed borders can be replaced by corresponding network layers or data.

Note that the architecture shown in Figure 3.1 is a general architecture that does not contain any detailed information about the network parameters. The box with solid borders is the part that will always be present in the proposed network architecture, while the boxes with dashed borders are changeable and/or removable parts and can be filled with the specified layers or data according to different implementations presented in Section 3.3 and evaluated in Section 3.6.

As shown in Figure 3.1, the key part of the proposed network architecture is the mlpconv layer and the global average pooling layer between the convolutional part and the fully connected part of the network. Different from the existing implementation that used a mlpconv and global average pooling layer as the output combination of the network, we use them in the middle of the network as a connection between two parts of the network [23]. The mlpconv will increase the abstraction level of the feature maps extracted from previous convolutional layers and the global average pooling layer will summarize these feature maps into a short vector, in which each element represents a feature map channel.

By doing so, we not only allow the agent to learn from more refined features, but also significantly reduce the number of parameters we need. For example, suppose that we have two more convolutional layers before the mlpconv layer and two more hidden fully connected layers before the output with 400 and 300 nodes respectively, a comparison of the number of parameters needed for a DDPG agent with traditional CNN and the proposed network architecture is shown in Figure 3.2.

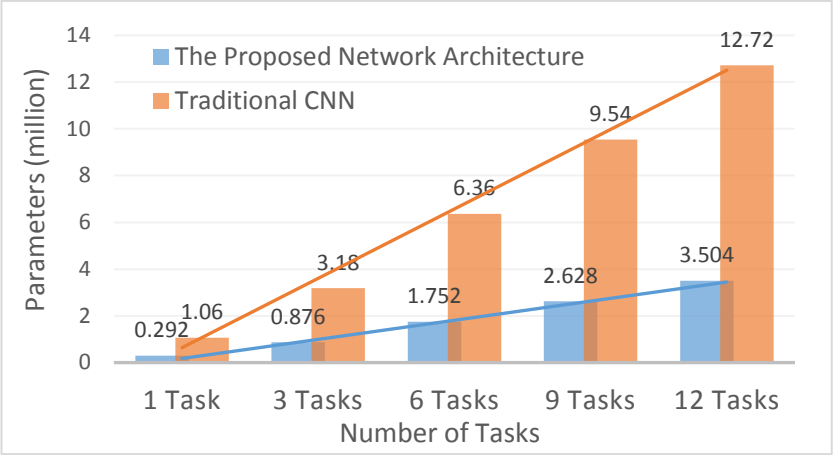

Figure 3.2: A comparison between the numbers of parameters needed for a DDPG agent with CNN and the proposed network architecture.

We can see that, for a DDPG agent that can only solve a single task, the proposed network architecture can reduce the number of parameters needed from around 1,060,000 to around 292,000, which is a 72.5% relative reduction. This reduction will become more and more significant in a sheer volume aspect as the number of tasks increases, because each additional task needs one new critic network and one new actor network.

3.3 Implementation Variations

Besides the advantages of using the proposed network architecture discussed in the last section, some good properties of the short vector formed by the mlpconv and global average pooling layers enable the architecture to be expanded to the following two implementation variations naturally.

3.3.1 Parameter Sharing Implementation

As discussed above, the output feature vector of the global average pooling layer will contain information at a higher abstraction level compared to the feature vector obtained from flattened feature maps of traditional convolutional layers. This makes it

natural to share the output feature between the fully connected layer parts of different networks. This is owing to the fact that features with higher levels of abstraction are more generalized features that can contain more information, even though this information is not as detailed as features with lower level of abstraction. As a result, the following fully connected layer can use these features and decode them in different ways according to different objectives (or loss functions).

Generally, there are two ways to share the output feature vector. The first way is to co-train the shared convolutional part among different networks. This way allows all networks to update the convolutional layers. The second way is to only allow one network (usually the one that is taking hold of the learning, such as the critic network in an actor-critic architecture) to update the convolutional part. In this way, the rest of the networks will use the output feature vector as their inputs and only need to update their own fully connected layers. We will illustrate in Section 3.6.1 that both ways are suitable for the proposed network architecture.

This parameter sharing implementation is especially suitable for deep reinforcement learning algorithms that have more than one network. For example, we can share the convolutional part of the actor network and the critic network in a DDPG agent. What is more, it is also very suitable in cases where multiple tasks are involved, such as multi-task learning and hierarchical learning, because parameters can be shared among different tasks or levels of hierarchy.

3.3.2 Image Data and Sensor Data Fusing Implementation

We will show in Section 3.6.2 that it is beneficial to include sensor data in the algorithm. While images can enable the agent to learn high level representations by providing the agent with vivid and rich information about the environment, information from raw sensors is more straightforward and sometimes can help the agent to learn more efficiently, especially when the agent is learning basic skills. For example, while raw sensors can directly send the agent distance information, this information is implicit and hard to infer from images.

However, image data and sensor data should be treated differently in neural networks. While image data needs to be processed and extracted by the convolutional layers, sensor data can be processed directly by fully connected layers. In fact, the proposed network architecture is suitable for combining image data and sensor data. This is another benefit we can get from having a feature vector that is at a higher level of abstraction, as the abstraction level of the sensor data is higher than that of the image data. What is more, the short feature vectors of images in the proposed network architecture can make it easier for the network to infer information from sensor data. This is because long vectors of flattened feature maps can easily drown the short vector of sensor data. Short feature vectors can help alleviate this effect.

In practice, the fusion of image data and sensor data can be achieved by including the sensor data before the fully connected layers, concatenating it with the feature vector from the global average pooling layer, as shown in Figure 3.1.

3.4 Network Architecture Summary

We proposed a novel network architecture for deep reinforcement learning agents. The proposed network architecture uses the mlpconv and global max pooling layer between the convolutional part and fully connected part of the network. It can reduce the number of parameters needed in the agent by 72.5% compared to traditional CNNs.

The proposed network architecture can also make the feature vector of the convolutional layers shorter and more abstract. This enables the proposed architecture to have more implementation variations such as parameter sharing and fusing sensor and image data, in a way that is more natural and efficient compared to traditional CNNs.

The reminder of this chapter will focus on evaluating the proposed deep reinforcement learning network architecture. To do so, we reproduced the Deep Deterministic Policy Gradient (DDPG) algorithm [24]. We compared the performance of a DDPG agent which is built with the original network proposed by Lillicrap, et al. [24] and a DDPG agent which is built with the proposed network architecture. We also conducted several experiments to show the impact of different implementations of the

network architecture on the learning performance of the algorithm. All networks are built and trained in TensorFlow [122].

The following sections will start with an introduction of the experimental setup in Section 3.5. Then we discuss the two implementation variations of the proposed network architecture in Section 3.6. The comparison between the proposed network architecture and the traditional CNNs will be presented in Section 3.7. Section 3.8 is the summary of the experiments.

3.5 Experiment Setup

All experiments were conducted in an environment built in Gazebo 2 in a ROS Indigo environment. The environment is an obstacle-free, walled space where a robot can move. The robot we use is a Pioneer 3AT robot. We set a camera on the front of it to give it front view image observations. We also set a range sensor on the top of it which can give it distance information from four angles (left, right, back and front). Furthermore, motors of the wheels can give us speed readings, which are another kind of sensor data we collected. The robot has two degrees of freedom, which are the speeds of a pair of wheels on each side of it. In each timestep, an action which is made up of two values chosen from a continuous space from -5 to 5 (corresponding to moving speed from -1.1 m/s to 1.1 m/s), each indicating the speed of wheels in one degree of freedom, will be executed by the robot. The task of agent is to use the image data or/and the sensor data to learn the actions for the robot to solve specific basic tasks. A screenshot of the environment and robot is shown in Figure 3.3.

During learning, the agent will receive a reward with value 1 when the action it chooses achieves the assigned task. When the robot turns over or crashes into walls, the agent will receive a punishment with value -1. Also in case of a crashing or turning over the current episode will terminate and the robot will be reinitialized at a random position and orientation in the space to start a new exploration episode. Note that in practice, we considered the robot is turning over when the angle between its baseboard

and the ground is bigger than 45° (this setting is kept for all the experiments in the rest of the thesis).

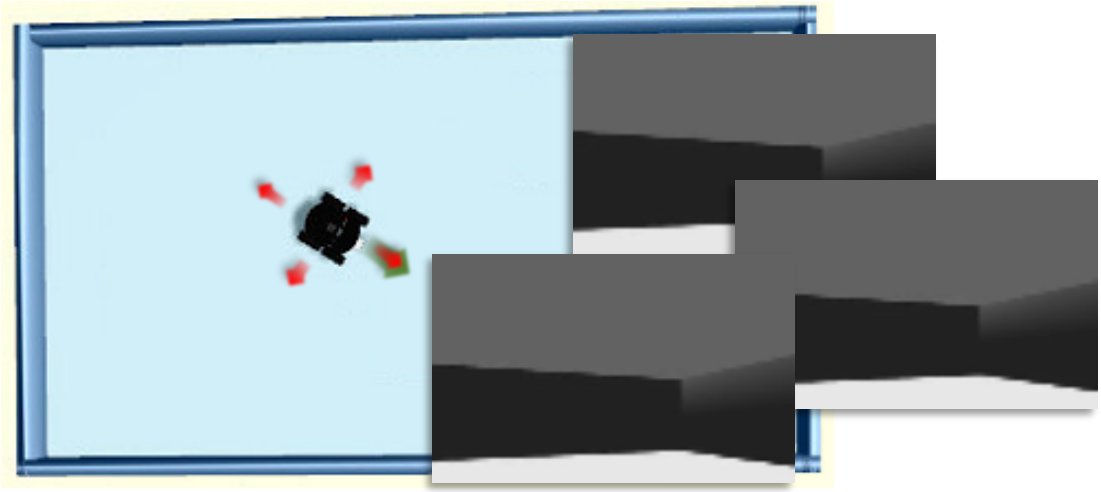

Figure *3.3*: A screenshot of the experiment environment. *The red arrows indicate the reading directions of the range sensor and the green arrow indicates the reading direction of the camera. Some images captured by the camera are shown on the bottom right.*

We tested the performance of the proposed network architecture using the DDPG algorithm. For comparison proposes, we also built an agent with the original network architecture used in DDPG. Specifically, the networks consisted of three convolutional layers. The first one has 32 kernels of size 8×8 with stride 4. The other two all have 64 kernels, the ones for the second layer are of size 4×4 with stride 2 and the ones for the third layer are of size 3×3 with stride 1. These three convolutional layers are followed by two hidden fully connected layers with 400 nodes and 300 nodes respectively. The network architecture is the same for the critic and actor except for the output layer. While the output layer for the critic network is a single state-action value, the output layer for the actor network is the values for the chosen action, each for one degree of freedom. When building the DDPG agent with our proposed network architecture, we simply replace the last convolutional layer with a mlpconv layer with same number of kernels with the same size and stride. The mlpconv layer is followed by a global max pooling layer to compress feature maps into a short vector with 64 elements, which is fed into the rest of the network. In case that sensor data is used, a vector of sensor data will be concatenated with this short vector. Before being fed into the networks, the images captured by the camera will first be resized to 64×64 grey scale images and the

three most recent images from the camera will be bundled together. Samples of image and sensor data can be found in Figure 3.3 and Figure 3.4 respectively.

		Back distance (m)	Left distance (m)	Right distance (m)	Front distance (m)	Left wheel speed (π rad/s)	Right wheel speed (π rad/s)	
Timestep 1	[	1.47	2.64	8.93	1.48	1.02	1.05	]
Timestep 2	[	2.86	2.49	4.51	1.62	1.17	1.19	]
Timestep 3	[	6.87	2.37	2.04	1.75	1.18	1.19	]
Timestep 4	[	1.99	2.32	8.09	1.79	1.24	1.25	]
Timestep 5	[	3.45	2.18	3.82	1.93	1.34	1.36	]
Timestep 6	[	7.52	2.06	1.58	2.05	1.26	1.26	]
Min	[	0	0	0	0	-5	-5	]
Max	[	10	10	10	10	5	5	]

Figure 3.4: Samples of sensor data. We show six sensor data vectors that have been fed into the networks consecutively. We also illustrate their corresponding maximum and minimum values.

For updating the networks, we use all the hyper-parameters proposed by Lillicrap, et al. [24] for both training with the proposed network architecture and with the original networks.

3.6 Performance of Different Implementation Variations

We first conducted two sets of experiments to investigate the performance of the two implementation variations introduced in Section 3.3. The tasks we assigned to the agent in these two sets of experiments are three basic locomotion tasks: going straight, turning left and turning right.

3.6.1 Parameter Sharing Implementation

In the first set of experiments, we investigated the impact of the parameter sharing

implementation on the performance of the agent.

As introduced in Section 3.3.1, we can have three implementation variations with different parameter settings or training schemes. They are: (1) parameter setting without any shared parameters; (2) parameter setting that shares the convolutional parts of the actor and critic with a co-training scheme; and (3) parameter setting that shares the convolutional parts of the actor and critic with only one leading network (in this case, the critic) being responsible for updating the shared convolutional layers.

We built the agent with all these three variations and compared their performance on the three basic locomotion tasks introduced above. We trained each agent for 2,000 episodes and tested its performance every 100 episodes to get a clear vision into the performance before the training converged. We ran all tasks 3 times and the result is shown in Figure 3.5.

We can see that there is no statistical significance difference between the average performances of the three parameter settings as the curves are very close and the shadow areas mostly overlapped.

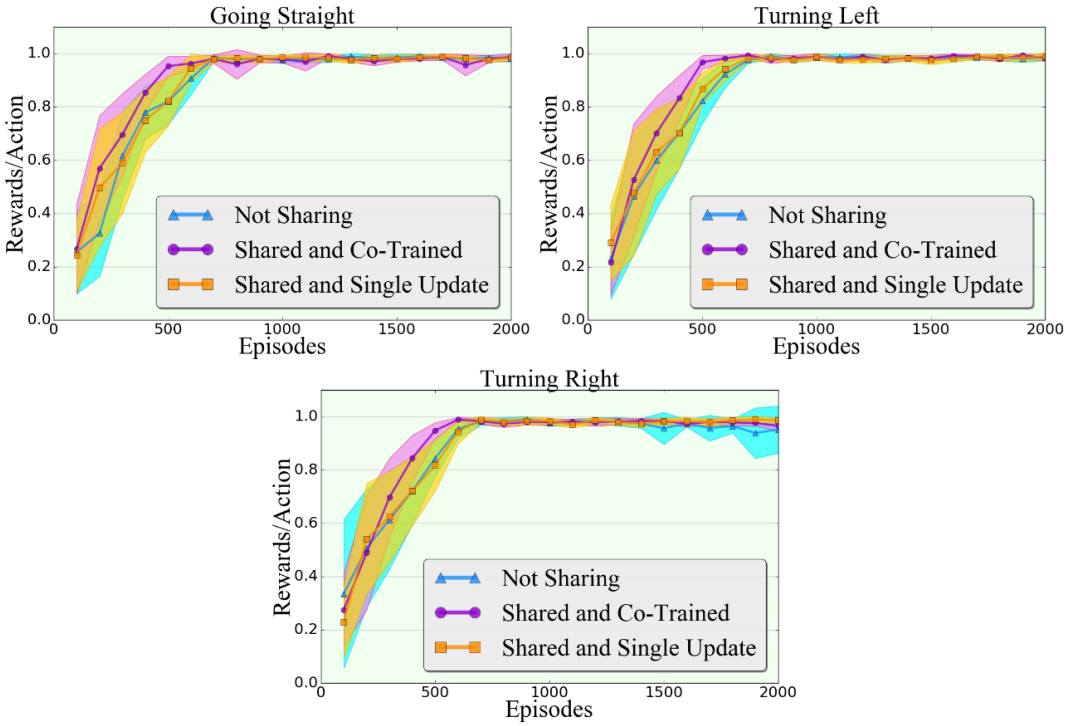

Figure 3.5: Performance comparison of different parameter settings of the proposed network architecture on three basic locomotion tasks. The curves are the average performance and the shadows indicate the standard deviations.

However, we should note that, while the performances are comparable, the parameter sharing implementations are actually using parameters more efficiently. Specifically, the parameter sharing implementation with co-training scheme saves the parameters needed for another three convolutional layers. And the other parameter sharing implementations have further reduced the computational operations needed for calculating the gradients of convolutional layers, as they are only updated once. From this perspective, we suggest that the parameter sharing implementations are more efficient, especially when only one network is updating the shared convolutional part. For the rest of the experiments in this chapter, we fixed this variation to a parameter sharing implementation where only the leading network is updating the shared part.

3.6.2 Image Data and Sensor Data Fusing Implementation

The second set of experiments focuses on investigating the performance of the agent when using different input data.

As introduced in Section 3.3.2, the proposed network architecture can have an implementation variation that fuses sensor data with image data by concatenating sensor data with the feature vector before the fully connected layers. We built the agent with this implementation variation and compared its performance to the agent that uses only image data. The sensor data we gave to the agent is the distance information from the range sensor and the speeds of the wheels on both side of the robot (which have been shown in Figure 3.4). Similar to the first experiment set, we trained each agent for 2,000 episodes and tested its performance every 100 episodes. The results are shown in Figure 3.6.

We can see that while the confidence intervals of the two implementations have less overlapping areas in the early stage of learning, they mostly overlapped after training for around 700 episodes. This is consistent with the t-test results as while some results in the early stage of training indicate the exist of difference between the two implementations (p-values are less than 0.05), these differences can no longer be

observed after 700 episodes. Therefore, we can conclude that the agent that uses both image data and sensor data is generally learning around 100 episodes faster than the one uses only image data on all three tasks in the early stage of training. This indicates that the sensor data can give the agent more straightforward information about the environment that can help it learn faster, especially when learning basic locomotion tasks. For the rest of the experiments in this chapter, we kept using the implementation variation that includes both two kinds of data.

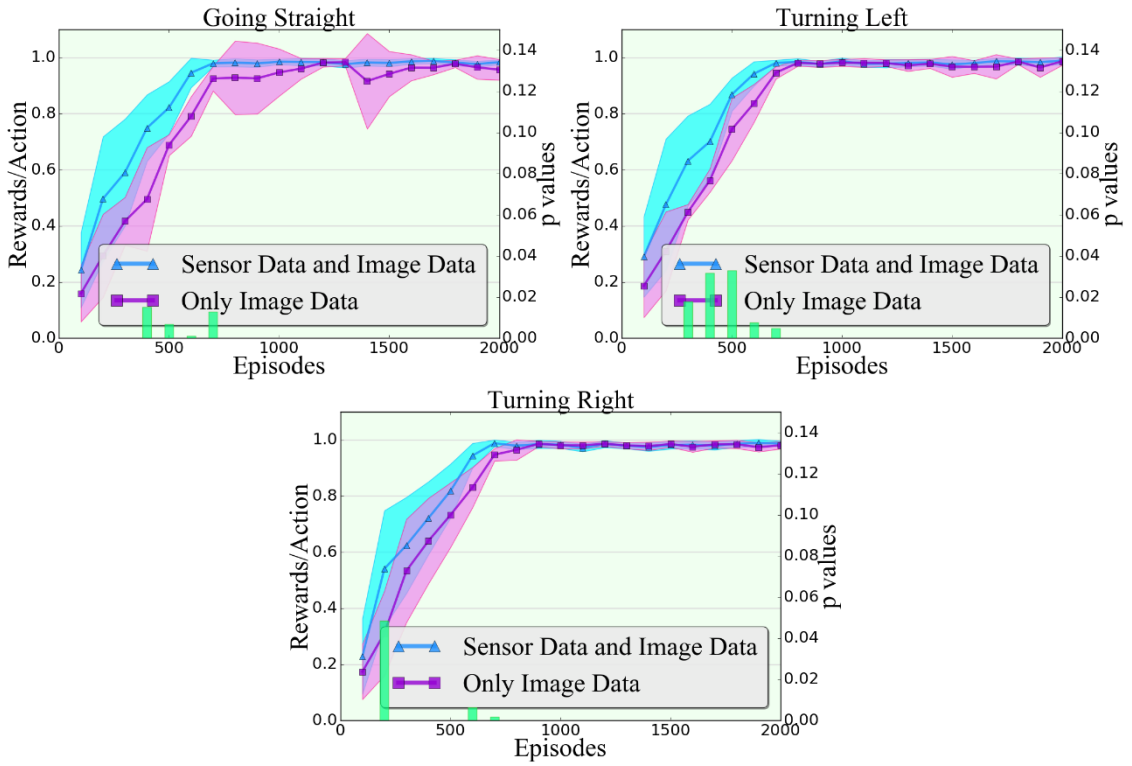

Figure 3.6: Performance comparison between agent that only uses image data and agent that uses both image data and sensor data. The curves are the average performance and the shadows indicate the 95% confidence intervals. The t-test results that have a p-value less than 0.05 have been shown as green bars in the figures.

3.7 Comparison to Traditional Convolutional Neural Networks

Based on the experimental results in Section 3.6, we conducted a final set of experiments to compare the proposed architecture with the CNNs used in DDPG.

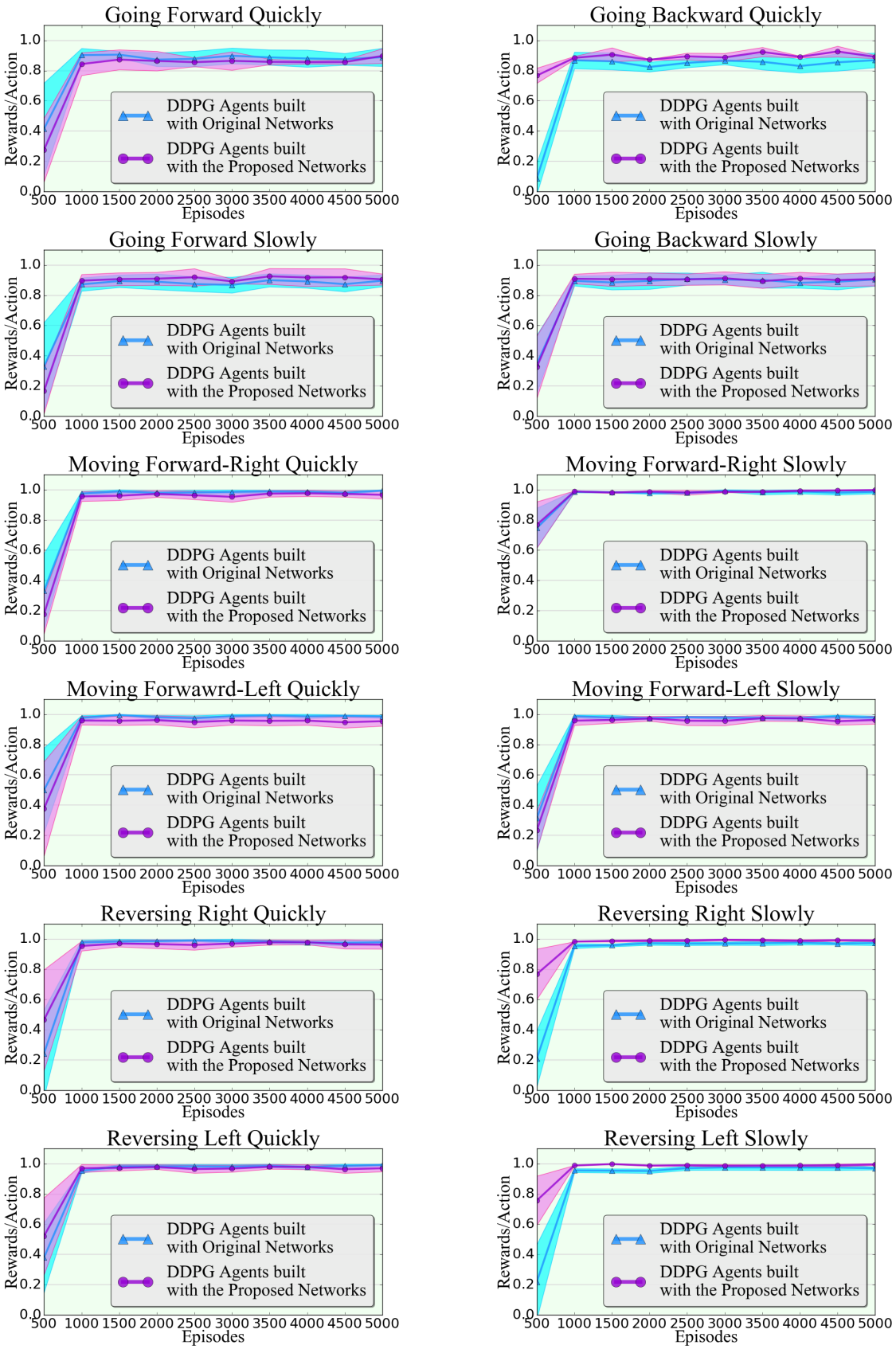

Figure 3.7: Performance comparison between the agent built with the proposed network architecture and the agent built with original CNNs in DDPG. The curves are the average performance and the shadows indicate the standard deviations.

In this set of experiments, the agents are assigned to learn 12 highly-constrained basic locomotion tasks that are more difficult than the ones in Section 3.6. They are going forward and backward at high and low speed (absolute moving speed more than 0.18 m/s and less than 0.22 m/s respectively), moving forward-left and forward-right slowly and quickly (turning speed less than 25°/s and more than 25°/s respectively, same for reversing) and reversing-left and reversing-right slowly and quickly. We trained each agent for 5,000 episodes on each task and tested its performance every 500 episodes. The results are shown in Figure 3.7.

We can see that the agent built with the proposed network architecture achieved a comparable performance to the agent built with original CNNs [24] on all 12 tasks, as the reward curves are close and shadows are mostly overlapped. Both the agents can achieve very good performance on these tasks. They can receive a reward for almost every actions they take. Also, when the training converged, these high performances can be retained throughout the learning process. This indicates that the proposed network architecture can enable the agent to achieve comparable performance to agent that uses traditional CNNs.

Note that, while the performances are comparable, the proposed network architecture is using many fewer parameters than traditional CNNs. A comparison of the parameters introduced can be found in Figure 3.2. This means the proposed network architecture is more parameter efficient.

3.8 Experiment Summary

We investigated the performance of the proposed network architecture with a simulated Pioneer 3AT robot in an obstacle-free, walled space built in Gazebo 2 in a ROS Indigo environment. We first investigated the performance of the implementation variations of the network architecture by training agent to learn 3 basic locomotion skills. We found that the proposed network architecture can achieve the best performance and the highest efficiency when applying the parameter sharing implementation and data fusing implementation.

We then trained the agent to learn 12 highly-constrained basic locomotion tasks and compared its performance with the traditional CNNs. Results showed that the agent can achieve comparable performance to traditional CNNs while uses many fewer parameters in the networks.

In the next chapter, we will introduce a novel multi-task deep reinforcement learning algorithm that is based on the proposed network architecture in this chapter.

Chapter 4

A Novel Multi-Task Deep Reinforcement Learning Algorithm

The work presented in this chapter has been partially published in the following paper:

Zhaoyang Yang, Kathryn Merrick, Hussein Abbass, Lianwen Jin, Multi-Task Deep Reinforcement Learning for Continuous Action Control. Proceedings of International Joint Conference on Artificial Intelligence, 2017. [Accepted on April 23, 2017].

Specifically, Sections 4.1, 4.2 and experimental results in Sections 4.4, 4.5 were written based on texts or/and results in the above paper.

As the first author of the above paper, I designed the proposed algorithm, built the simulation and implemented the algorithm on it, and wrote the majority of the paper on my own.

Deep reinforcement learning has achieved considerable success in recent years in many control tasks. However, most deep reinforcement learning agents can only learn one task at a time. Work that focuses on dealing with multi-task learning scenarios mainly achieves this using transfer learning or generative model methods, which are still methods that need to learn tasks one by one. Moreover, little work can be found considering multi-task learning in a continuous action space. Nonetheless, multi-task learning should still be considered a very promising and efficient way to learn multiple tasks concurrently.

In this chapter, based on the Deep Deterministic Policy Gradient (DDPG) algorithm [24], we propose a novel multi-task deep reinforcement learning algorithm, which we call multi-DDPG, to learn multiple tasks concurrently in continuous action spaces. In the algorithm, we apply the network architecture proposed in Chapter 3 to significantly reduce the number of parameters needed and combine images and sensor data as input. Compared to DDPG, which has only one actor and one critic, the proposed algorithm has a single-critic, multi-actor architecture. While each actor learns a different task, all

actors are trained concurrently within the same training process. The tasks we are considering in this chapter are basic movement tasks that can be solved directly with basic skills.

In the reminder of this chapter, we first introduce the architecture of the proposed multi-DDPG algorithm in Section 4.1. This will be followed by a detailed introduction of the learning process of the algorithm in Section 4.2. Section 4.3 is the algorithm summary. The validations will start from Section 4.4.

4.1 Algorithm Architecture

Similar to DDPG, the proposed multi-DDPG algorithm has two kinds of networks, a critic network and actor networks. However, unlike DDPG, which has only one actor and one critic, multi-DDPG has multiple actors with one critic. An overview of the architecture of the proposed algorithm is shown in Figure 4.1.

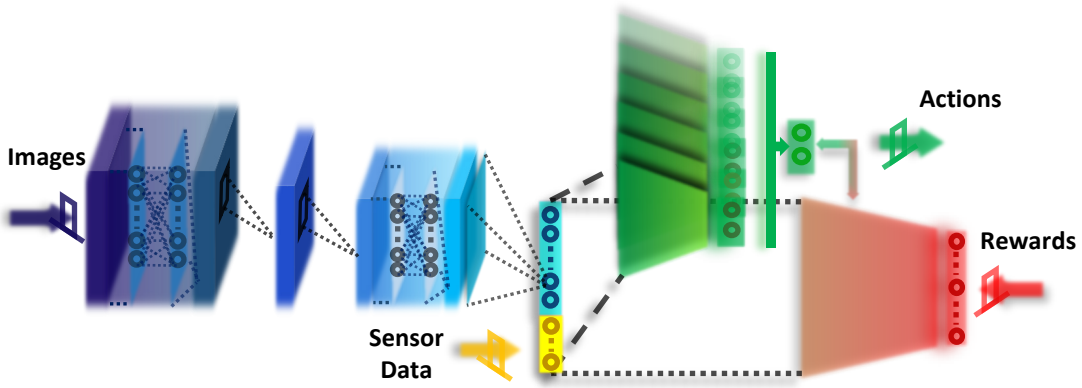

Figure 4.1: An overview of multi-DDPG architecture. The trapeziums in the picture represent fully connected layers, the green ones (in the middle) for actors and red one (on the right) for critic. The square-dotted lines indicates that there is back-propagation between two layers, while dashed lines do not involve back-propagation.

The way that multi-DDPG achieves multi-task learning is to have multiple actors in its architecture, with each actor responsible for one specific task (see the multiple actors in the middle of Figure 4.1). These actors are learned from the same inputs and also within the same training process.

Note that while multiple actors are introduced, no new critic has been added in our algorithm (see the single critic on the right of Figure 4.1). This means that the single critic in the algorithm must be able to guide all actors to update properly. Therefore, instead of a single output state-action value, our critic has to output multiple state-action values, one for each actor.

We can also see from Figure 4.1 that the network architecture proposed in Chapter 3 is applied in multi-DDPG by feeding the convolutional part before the mlpconv layer with one mlpconv layer and one traditional convolutional layer, and the fully connected part with several layers for each network. What is more, sensor data has been included in the algorithm, as they can be effective in training basic skills we are considering for the proposed algorithm. Finally, we applied the parameter sharing implementation that shares parameters of the convolutional part among all existing networks (see the left part of Figure 4.1).

The single-critic, multi-actor architecture as well as the parameter sharing implementation of the network architecture proposed in Chapter 3 can actually further reduce the number of parameters needed for learning multiple tasks compared to the DDPG algorithm. Suppose that all fully connected parts consist of two hidden layers with 400 nodes and 300 nodes respectively, a new comparison of the number of parameters is shown in Figure 4.2.

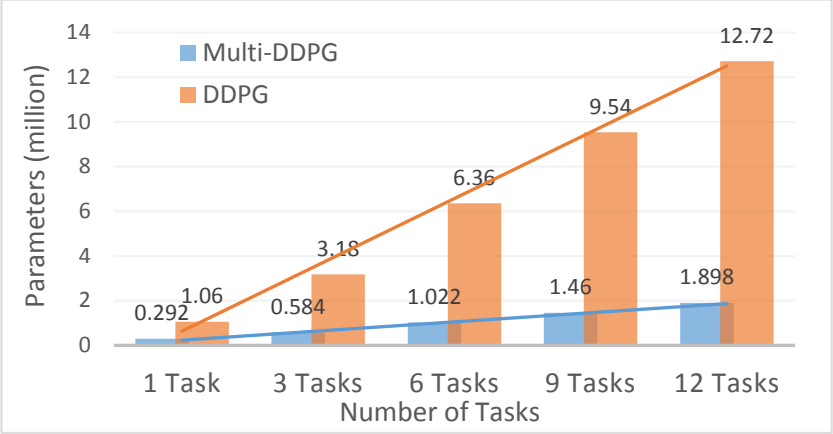

Figure 4.2: A comparison between the numbers of parameters needed for multi-task learning with multiple DDPG agents and with agent based on the proposed multi-DDPG algorithm.

We can see that the reduction reaches 81.6% when having 3 tasks and 85.1% when having 12 tasks.

4.2 Algorithm Learning Process

In order to train a critic that can output multiple state-action values at the same time, each for a different task, we need to correspondingly get multiple reward values for different tasks at the same time. To achieve this, the agent will assess the executed action according to all rewarding criteria we have for different tasks to form a vector of rewards, regardless of which actor produced that action. In case that the chosen action caused an exploration termination, all the actors will receive the same punishment as actions that could cause a termination are undesired actions system-wide.

Then instead of the loss function (2.17) and (2.18) used for the critic network in DDPG, the loss function for the critic network in multi-DDPG is:

$$L(\theta^Q) = \sum_{g=1}^{G} (Q_g(s_t, a_t|\theta^Q) - y_{g,t})^2 \tag{4.1}$$

where g is the identity number of the task and G is the total number of tasks we have. Correspondingly, the supervising signal becomes:

$$y_{g,t} = r_{g,t} + \gamma Q_g(s_{t+1}, \pi(s_{t+1}|\theta^{\pi,t})|\theta^{Q,t}) \tag{4.2}$$

Note that only one actor will be activated to choose actions in each timestep. During exploration, actors will be activated iteratively. We do not distinguish actions produced by different actors and all transitions will be stored in the same replay memory.

Also note that as we do not distinguish actions produced by different actors, we can simply iteratively choose a target actor network to calculate $y_{g,t}$ for all input data in a training iteration. This is benefited by the fact that critic training and actors' training are not synchronous. It turns out that the critic will not be trained to be task specific and it will always be able to infer state-action values of all tasks we have for any input (s_t, a_t) pairs, whichever actors produced the a_t.

After the critic is updated, we update all actors one after another. For each individual actor, the updating gradient can be obtained by:

$$\nabla\theta^{\pi_g} = \mu_{\pi_g} \cdot \nabla_a Q_g\left(s_t, \pi_g(s_t|\theta^{\pi_g})\middle|\theta^Q\right) \cdot \nabla_{\theta^{\pi_g}} \pi_g(s_t|\theta^{\pi_g}) \tag{4.3}$$

Note that here, the action gradient $\nabla_a Q_g$ is task specific. For each individual actor, it is the gradient with respect to the corresponding state-action value output in the critic. This can also be understood as only one output neuron of the critic network will be activated when inferring gradients for a specific actor.

Finally, after the critic network and actor networks are updated, their corresponding target networks will be updated according to a soft updating rule as follows:

$$\theta_{t+1}^t = (1-\varphi)\theta_t^t + \varphi\theta_t \tag{4.4}$$

where φ is the soft update factor.

With regards to the exploration phase, actors will be selected iteratively to choose actions in each episode. What is more, the exploration is governed by an Ornstein-Uhlenbeck process [115], while this process will be removed during testing.

By the end of learning, we will have a single critic that outputs state-action values for all tasks and multiple actors each producing actions to achieve a different task.

4.3 Algorithm Summary

Based on the DDPG algorithm, we proposed a novel multi-task deep reinforcement learning algorithm called multi-DDPG. The proposed algorithm can learn multiple tasks concurrently within the same training process. This is achieved by adapting a single-critic, multi-actor architecture. While each actor is responsible for a different task, all actors are trained by the same critic.

Also, by applying the network architecture proposed in Chapter 3 as well as its parameter sharing implementation, the proposed algorithm further reduces the number of parameters needed for learning multiple tasks. As the algorithm aims to learn basic skills, the image and sensor data fusing implementation is also applied to help the agent

learn faster. We summarize the proposed multi-DDPG algorithm in Algorithm 4.1.

In the reminder of this chapter, we will test the performance of the proposed multi-DDPG algorithm. To do so, we trained the multi-DDPG agent to learn skills in 3-task, 6-task and 12-task scenarios and compared its performance on each individual task to the performance of a DDPG agent. We also conducted an experiment to investigate the performance of multi-DDPG in the case that one incompetent actor exists. All networks are built and trained with TensorFlow [122].

The reminder of this chapter is organized as follows. Section 4.4 will introduce the experimental setup for this chapter. Section 4.5 and 4.6 are experimental results for the multi-task learning scenarios and the learning with incompetent actor scenario respectively. Section 4.7 summarizes of the experiments.

Algorithm 4.1 Multi-DDPG

Input: maximum training episode E_{max}, maximum steps in each episode S_{max}, mini-batch size M, replay memory **P.**

Initialization: randomly initialize networks weights θ^{Q}, θ_1^{π}, … … , θ_G^{π} and target networks weights $\theta^{Q,t} \leftarrow \theta^{Q}$, $\theta_g^{\pi,t} \leftarrow \theta_g^{\pi}$.

while episode **<** E_{max}
  Initialize random noise *N* for exploration
  Iteratively select activated actor
  Get initial state s_1
  while step **<**S_{max} **and** episode **not** terminated
    Select action a_t using selected actor and add *N*
    Execute a_t and get reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in **P**
    Randomly sample a batch of M transitions from **P**
    Update θ^{Q} according to (4.1) and (4.2)
    Update $\theta^{Q,t}$
    for g **<** *G*
      Update θ_g^{π} according to (4.3)
      Update $\theta_g^{\pi,t}$
    end for
  end
end

4.4 Experiment Setup

The experimental environment for this chapter is the same to the one used in Chapter 3, which is an obstacle-free, walled space built in Gazebo 2 in a ROS Indigo environment, which is shown in Figure 3.3. The robot we use is also the Pioneer 3AT robot used in Chapter 3, which has front view camera, range sensor on the top that reads distance information from four angles (left, right, front, back) and readings of the speeds of the wheels.

For all the experiments, the agent was given reward of value 1 for achieving a task. On the contrary, the agent was penalized a value -0.5 if the robot made dangerous movements such as crashing into walls or turning over (in practice, it is when the angle between the baseboard of the robot and the ground is larger than 45°). Otherwise, the reward was 0. When the agent receives a punishment, the current episode will terminate and the robot will be initialized at a random position and orientation in the space to start a new exploration episode. In each individual experiment, only one robot is spawned in the environment. The agent will learn to control this robot to achieve the multiple tasks we assigned for that experiment.

The network parameters we used for the multi-DDPG agent are as follows. The first layer is a mlpconv layer with 32 kernels of size 8×8 with stride 4. It is followed by a traditional convolutional layer with 64 kernels of size 4×4 with stride 2. The last layer of the convolutional part is another mlpconv layer with 64 kernels of size 3×3 with stride 1. Then after a global average pooling layer, the feature vector as well as the concatenated sensor data are fed in two hidden fully connected layer. For the critic network, the size of these two layers are 400 and 300 respectively, while for the actor networks, they are 300 and 200 respectively. The output of the critic network is a vector in which each element indicates the state-action value of a specific task. The outputs of all actor networks are the speed values of the wheels on each side of the robot they choose. The images captured by the camera will first be resized to 64×64 grey scale images and the three most recent images from the camera will be bundled together and fed into the network.

Description	Symbol (if has)	Value
Critic base learning rate	μ_Q	0.001
Critic discount factor	γ	0.9
Actors base learning rate	μ_π	0.0001
l_2 penalty	/	0.01
Mini-batch size	M	64
Ornstein-Uhlenbeck process intensity	σ	0.12
Maximum training episode	E_{max}	5000
Maximum step in each episode	S_{max}	100
Soft update factor	φ	0.001

Table 4-1: Hyper-parameters for the multi-DDPG algorithm.

With regards to training strategy, we use Adam optimization [71] for all the networks. At the same time, l_2 regularization is introduced to prevent overfitting. All the other hyper-parameters we used in this chapter are listed in Table 4-1.

4.5 Multi-Task Learning Performance

We first tried the proposed multi-DDPG algorithm to learn 3 tasks and 6 tasks concurrently. As the number of tasks increases, the tasks are more constrained. The performance of the algorithm in these two learning scenarios are shown in Figure 4.3.

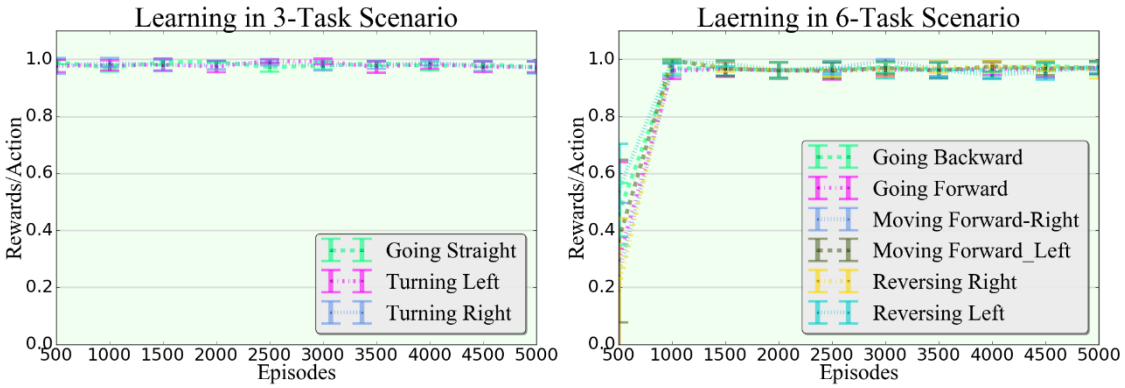

Figure 4.3: Performance of multi-DDPG in 3-task and 6-task learning scenarios. The curves are the average performance and the error bars indicate the standard deviations.

The tasks used in the 3-task learning scenario are identical to the ones used in

Chapter 3, which are going straight, turning left and turning right. The tasks in the 6-task scenario are going forward and backward, moving forward-left and forward-right, reversing-left and reversing-right.

The results shown in Figure 4.3 show that the proposed multi-DDPG algorithm can deliver robust multi-task training of these less constrained tasks, in which actors started to act according to their corresponding reward signals in early stages of learning and the performance stayed high throughout the rest of the learning.

Finally, we tested the performance of multi-DDPG in a 12-task scenario. Some samples of the movement trajectories the actors performed for the 12 tasks during testing are shown in Figure 4.4. We also trained 12 agents, in which each agent learn one of these 12 tasks, with the DDPG algorithm to have a comparison. All the learning results are shown in Figure 4.5.

As we can see in Figure 4.4, the 12 tasks in this scenario are highly constrained basic locomotion tasks the same as the ones used in Chapter 3, which are going forward and backward at high and low speed, moving forward-left and forward-right slowly and quickly, and reversing-left and reversing-right slowly and quickly.

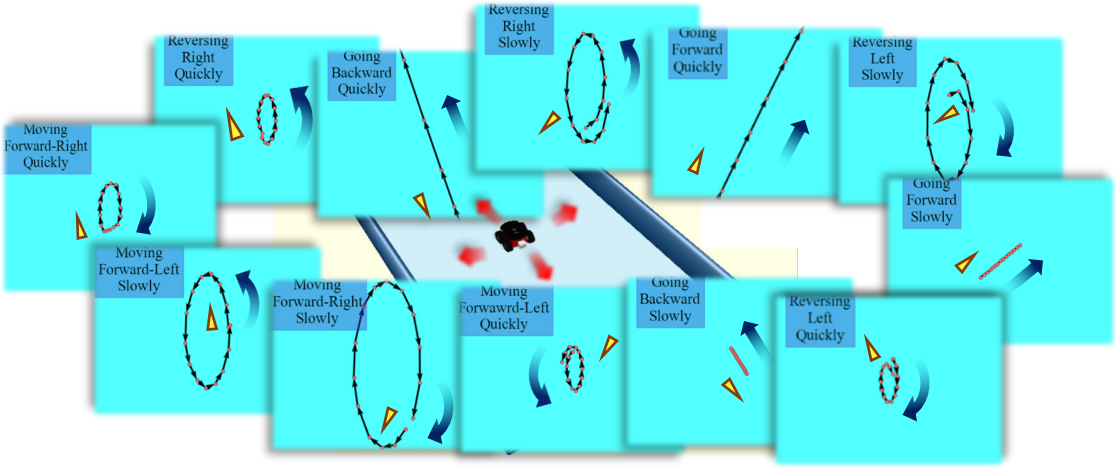

Figure 4.4: A collection of movements of the 12 basic locomotion tasks. At most 15 action steps are shown in each picture. The red nodes are the location of the robot, and black arrows in between are its movement trajectories. In each picture, a yellow triangle indicates the initial orientation of the robot, while the blue arrow shows the overall movement direction.

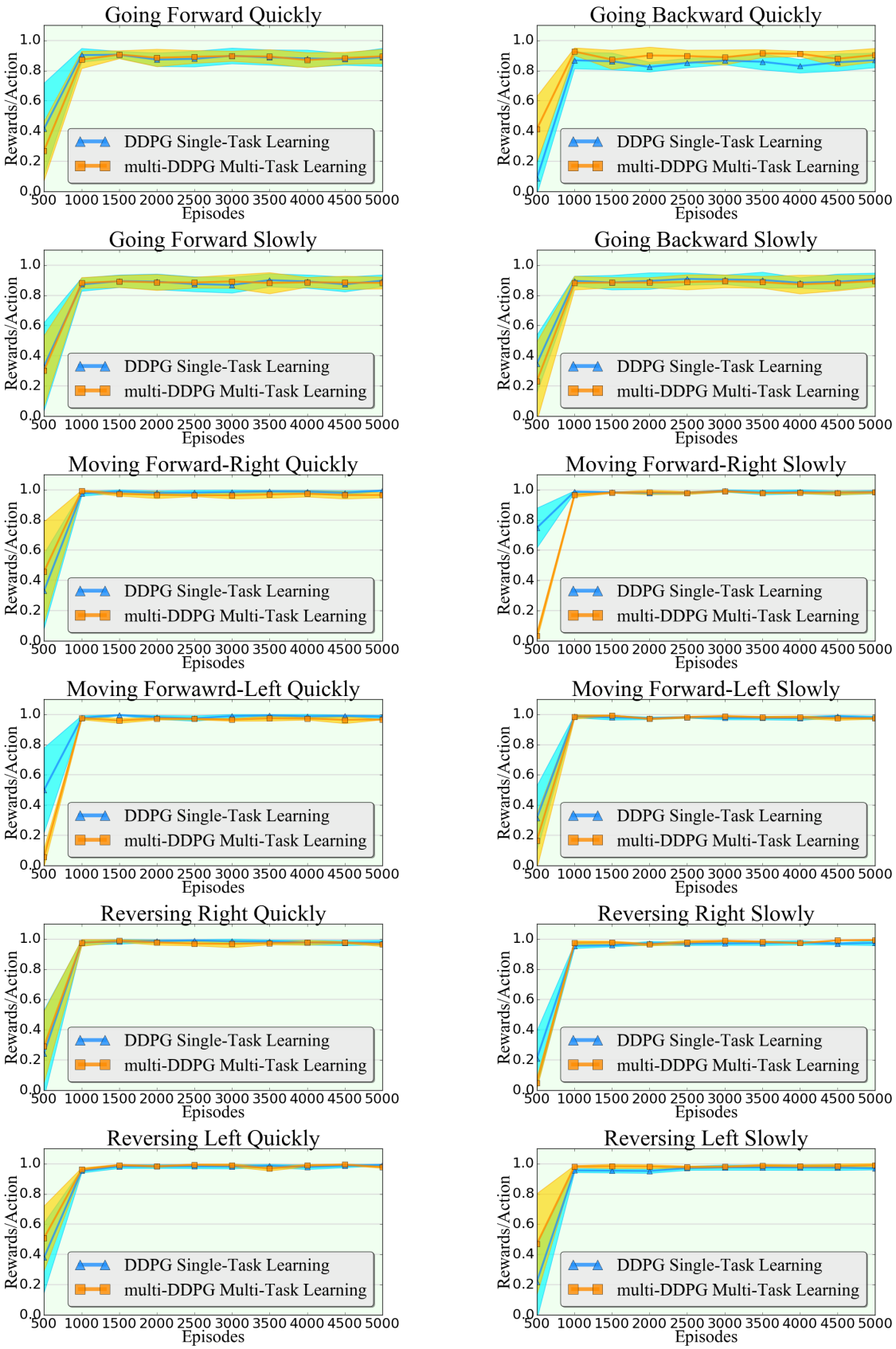

Figure 4.5: Performance comparison between the DDPG agent and multi-DDPG agent on 12 highly constrained basic locomotion tasks. The curves are the average performance and the shadows indicate the standard deviations.

We can see from Figure 4.5 that the performances of multiple actors trained by multi-DDPG are comparable to actors trained by DDPG which is a single task scheme. The narrow shadow areas and mild vibrations in the graphs demonstrate the robustness of the algorithm and the stability of the performances of trained actors. We can also see from Figure 4.4 that the robot was moving as expected when testing each actor. Although the trajectories of those turning tasks are not in perfect circles, they are good enough to collect rewards.

Note that comparing results in Figure 4.3 and Figure 4.5, we can know that the increased number of tasks and their constraints do not increase the number of episodes needed to stabilize training. This may be owing to the parameter and replay memory sharing amongst all tasks, which helps the agent to avoid dangerous actions and increase the chance of collecting rewards during exploration. The high average rewards collected in every action across all individual tasks have further demonstrated the effectiveness of the multi-DDPG algorithm.

These results not only suggest that the proposed multi-DDPG algorithm can learn high performance actors to handle basic locomotion tasks, but also demonstrate that the performance of the algorithm can stay consistent either when the number of tasks or the constraints of tasks increase. What's more, its light-weight architecture as well as the parameter sharing strategy also make it flexible enough to be expanded to learn more actors and tasks.

4.6 Learning with an Incompetent Actor

In this section, we conducted an additional set of experiments to investigate the performance of the proposed multi-DDPG algorithm when one of the actors of included in the agent is having very poor performance. The result of this experiment is shown in Figure 4.6.

In this experiment, we included an actor with a meaningless reward function. This actor received rewards randomly (which is a random generation of 0 and 1). This means, even though it can receive rewards sometimes, it would never learn. In this set of

experiments, the rest of the actors were assigned to learn the tasks we used in the 6-task scenario we introduced above.

We can see that the agent can still learn high performance actors for the other 6 tasks even when one of the actors failed. By comparing the results in Figure 4.3, we can see the performance of the rest of the actors are as they were trained without the failed actor. This may be owing to the actor update rule in (4.3) that only activates one output neuron of the critic when updating a specific actor. It makes the update of each actor independent. This adds to the robustness of the proposed algorithm as individual failed actors will not interfere with the learning of other actors.

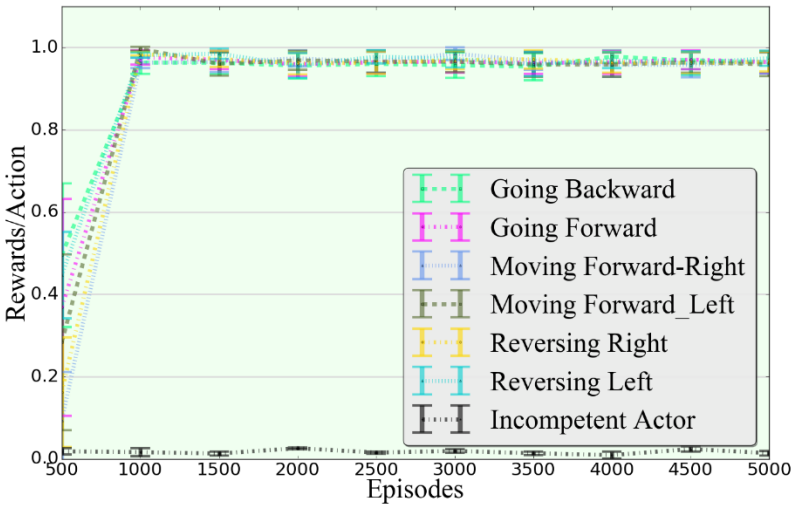

Figure 4.6: Performance of multi-DDPG algorithm when training with an incompetent actor in the 6-task scenario. The curves are the average performance and the error bars indicate the standard deviations.

4.7 Experiment Summary

We evaluated the proposed multi-DDPG algorithm introduced with a simulated Pioneer 3AT robot in an obstacle-free, walled space which was built in Gazebo 2 in a ROS Indigo environment. The robot can read the speeds of its wheels and was equipped with a camera that can give it front view images as well as a range sensor that can give it

distance readings from four directions (left, right, front and back). We trained multi-DDPG agents to learn in 3-task, 6-task and 12-task scenarios.

The results show that multi-DDPG can learn high performance basic locomotion skills in all these scenarios. The performance stays consistent when the number of tasks and the constraints on the tasks increase.

By comparing the learned skills with the corresponding skills learned by DDPG agents (each skill is learned with one DDPG agent), we can see that the skills learned by multi-DDPG are comparable to the skills learned by DDPG, which learned these skills one after another.

What is more, results also show that when one of the actors of the agent is having poor performance, the rest of the actors can still learn high performance skills. This means that individual failed actors will not interfere with the learning of other actors, which adds to the robustness of the algorithm.

In the next chapter, we will introduce a novel hierarchical deep reinforcement learning algorithm which adapts multi-DDPG in one of its levels of hierarchy to permit learning of compound tasks.

Chapter 5

A Novel Hierarchical Deep Reinforcement Learning Algorithm

The work presented in this chapter has been partially submitted to the following paper:

Zhaoyang Yang, Kathryn Merrick, Hussein A. Abbass, Lianwen Jin, Hierarchical Deep Reinforcement Learning for Continuous Action Control. [Under Review].

Specifically, Sections 5.1, 5.2, 5.3 and experimental results in Section 5.5, 5.6, 5.8 were written based on texts or/and results in the above paper.

As the first author of the above paper, I designed the proposed algorithm, built the simulation and implemented the algorithm on it, and wrote the majority of the paper on my own.

As discussed in Chapter 2, most existing continuous control deep reinforcement learning algorithms seem to be better at learning basic movement skills and perform worse when facing complex compound tasks. Even though the A3C algorithm [98] can solve some compound tasks, it needs a multi-thread learning scheme, which allows multiple agents to explore the environment at the same time. This indicates that it is challenging to learn compound skills in continuous action spaces with a single agent. Therefore, hierarchical learning can be a potential way to enable the agent to learn compound skills by decomposing them into several basic skills that it can learn directly.

In this chapter, based on the multi-DDPG algorithm proposed in Chapter 4, we propose a novel hierarchical deep reinforcement learning algorithm, which we call h-DDPG, to learn compound skills and basic skills simultaneously. The proposed algorithm has a duel-critic, multi-actor architecture, in which each critic is taking control of a level of hierarchy. Specifically, there is one basic critic that is responsible for training multiple actors that each handles a different basic skills, and one meta critic that is responsible

for learning compound skills that are made up of the basic skills learned by the actors. The multi-DDPG algorithm is adapted to create the basic critic.

In the reminder of this chapter, we introduce the architecture, the learning process and implementation details of the proposed h-DDPG algorithm respectively in Section 5.1, 5.2 and 5.3. Section 5.4 is the algorithm summary. The validation of the algorithm is in Section 5.5 to Section 5.9. In the rest of the thesis, we will use subscript letters b and m to distinguish basic critic components and meta critic components in equations.

5.1 Algorithm Architecture

The proposed h-DDPG algorithm has two levels of hierarchy, which are achieved with a duel-critic, multi-actor architecture. The duel-critic comprises a basic critic in the first level of hierarchy that is responsible for training multiple actors that learn different basic skills and a meta critic in the second level of hierarchy that learns to reuse actors to solve compound tasks. An overview of the architecture of the algorithm is shown in Figure 5.1.

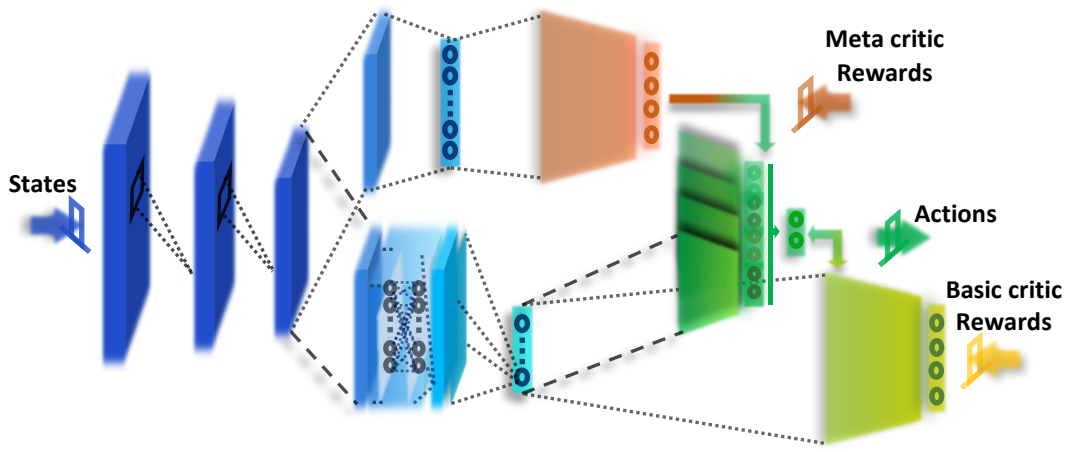

Figure 5.1: An overview of h-DDPG architecture. The trapeziums represent fully connected layers. These fully connected layers are layers from meta critic, actors and basic critic respectively from top to bottom. The square-dotted lines are connections with back-propagation while the dashed lines are not.

In order to achieve a level of hierarchy that learns multiple basic skills simultaneously, we adapt the multi-task deep reinforcement learning algorithm we proposed in Chapter

4. Specifically, we kept the basic concepts multi-DDPG and made some changes to its network architecture to fit it in the proposed h-DDPG algorithm. We can see from Figure 5.1 (left part) that the main difference is that the first mlpconv layer is replaced by a traditional convolutional layer.

Different from the basic critic, the meta critic focuses on learning compound skills to solve compound tasks. The meta critic can access a set of basic skills provided by actors in the first level of hierarchy. The goal of the meta critic is to choose a basic skill that will help it solve the attempted compound tasks in each timestep. Therefore, similar to discrete action scenarios, the meta critic will choose the basic skill with the highest value from a given set of basic skills.

This can be achieved by bootstrapping estimation of Q values of each basic skill. Thus, the meta critic will be a network with G output neurons that give estimations of Q values of all available actors.

Note that as learning of the basic critic and actors happens in the same process as the meta critic learns, there is no guarantee that all actors have good performance at their corresponding basic skills. However, the meta critic has already taken this into consideration. This is because the way that the meta critic understands the actors is to understand the transitions from s_t to s_{t+1} after a particular actor has been chosen. The meta critic does not know what basic skills the actors are assigned to learn prior to the training starting.

Similar to multi-DDPG, in the proposed algorithm, we include sensor data after the convolutional part of the networks and concatenate it with a feature vector of the image data. In addition, we extract two levels of image feature abstractions to keep the whole hierarchical architecture consistent and concise.

Specifically, for the meta critic network that needs a thorough understanding of the environment to infer proper choice of basic skills to solve compound tasks, image features are a long vector flattened from the feature maps of the last convolutional layer. This vector is then fed into the rest of the fully-connected layers. In this way, every pixel in the feature maps of the last convolutional layer will contribute to the final decision of the meta critic. Abstractions at this level of hierarchy are intended to give a more

detailed description of the states so that the critic can learn and make decisions according to observations of the environment.

For the basic critic as well as the actors that focus on basic skill learning, image features are a much shorter vector rendered from a global max pooling. This is achieved by applying mlpconv layer operations on the feature maps of the last convolutional layer. Each reconstructed feature map from the mlpconv layer will then be averaged globally to form an element in the abstraction vector. Abstractions at this level of hierarchy are intended to give a more abstract and less detailed description of the states so that the critic can learn basic skills without being disturbed by noise or irrelevant information in the environment.

Moreover, the implementation of the mlpconv layer in the basic critic helps reduce the number of parameters needed significantly compared with using a traditional convolutional layer. Figure 4.2 gave a comparison of the number of parameters introduced.

5.2 Algorithm Learning Process

The whole learning process of the proposed h-DDPG algorithm follows an ordering of hierarchy priority. The basic critic and actors in the first level of hierarchy are updated first, and then the meta critic in the second level of hierarchy. Each training iteration will start right after an action has been executed in a timestep of exploration.

The training of the basic critic and actors are similar to multi-DDPG which can be found in Chapter 4. For the meta critic, in order to bootstrap estimation of Q values of each basic skill, the network will be optimized by minimizing the loss function:

$$L(\theta^{Q_m}) = (Q_m(s_t, a_t|\theta^{Q_m}) - y_{m,t})^2 \tag{5.1}$$

where the supervision signal $y_{m,t}$ is:

$$y_{m,t} = r_{m,t} + \gamma_m \max_{a_{t+1}} Q_m\left(s_{t+1}, a_{t+1}\middle|\theta^{Q_m^t}\right) \tag{5.2}$$

Similar to multi-DDPG, after the update of the meta critic, all target networks will be updated following the soft update rule in (4.4).

Note that, we also applied a parameter-sharing scheme on convolutional layers across different networks. Specifically, all traditional convolutional layers will be updated by the meta critic and fixed when updating other networks. The mlpconv layer will be updated by the basic critic and actor networks. We also implement an annealing based learning rate scheme on the actor networks to decrease the learning rate as the performance of the actors improves.

5.3 Implementation Details

The co-existence of two levels of hierarchy in the same architecture demands adjustments in several aspects of the algorithm compared to the multi-DDPG algorithm in Chapter 4.

5.3.1 Rewards and Punishments

In this thesis, we are considering scenarios where all reward functions are pre-defined. Specifically, two kinds of rewards are necessary: reward for the meta critic, which can only be received when the final compound task is achieved, and reward for the basic critic, which can be received as soon as the action chosen by the actor is achieving its corresponding basic skill. As the frequency of receiving these two kinds of rewards is different, different values may be chosen. Having suitable reward values for the meta critic is especially important as it is much more sparse than the one for the basic critic. We investigate this in Section 5.6.2.

Note that in order to keep consistent with the loss function of the basic critic in (4.1), the reward for the basic critic will be a reward vector in which each element represents whether this action is what is desired for the corresponding actor. This means that

whichever actor is providing the action, the action will always be evaluated by the reward functions of all available actors.

Similar to the rewards, two kinds of punishments, one for the meta critic and one for the basic critic, are introduced in the algorithm. Only the component that has caused undesired actions will receive punishments. In our case, in order to make the basic critic and actors focus on learning basic skills, we only punish the basic critic when the robot turns over, as making the robot move stably is a prerequisite of having good basic skills. Note that actions that could cause the robot to turn over are undesired actions for all actors, regardless of what basic skill the actor is assigned to learn. Therefore, the punishment is universal to all actors. The meta critic will receive a punishment when the robot crashes into obstacles, as avoiding collision should be considered as a part of the compound skills. What is more, we punish the meta critic with a small value every step before the episode terminates. This is mainly to push the meta critic to find the optimal solution to the task.

5.3.2 Replay Memory and Batch Sampling

As a consolidated system, all levels of hierarchy in this algorithm share a single replay memory. However, when sampling transitions from the replay memory, a balance among transitions made by different actors is required. Specifically, the final batch of transitions will always consist of the same number of transitions from different actors. This sampling strategy makes the sampling more controllable and ensures that the critics can see transitions of different actors evenly.

Note that, similar to multi-DDPG, when calculating supervision signals in (4.2), only one target actor network will be used. In practice, we select target networks iteratively. This is feasible owing to the actor-unspecific nature of the basic critic during self-updating as explained in Chapter 4.

5.3.3 Exploration

Exploration is a critical aspects for all reinforcement learning algorithms. For deep reinforcement learning, exploration needs to be balanced to prevent getting stuck in a local optimum.

In the proposed algorithm, the exploration of the meta critic is governed by a $\epsilon - greedy$ policy while the exploration of the basic critic is governed by an Ornstein-Uhlenbeck process [115]. Moreover, the value of ϵ will be annealed throughout the training process to allow more exploitation. Similarly, we change the intensity of the Ornstein-Uhlenbeck process according to the testing performance of actors during training according to:

$$\sigma_g \leftarrow \max\{\sigma_{min}, (1 - p_g)\sigma_{init}\} \tag{5.3}$$

where σ is the parameter that controls the intensity of the process and subscript $init$ and min denotes its initial and minimum value respectively. p_g is the performance of actor g in the latest testing.

5.4 Algorithm Summary

In this chapter, based on multi-DDPG algorithm proposed in Chapter 4, we proposed a novel hierarchical deep reinforcement learning algorithm called h-DDPG. The proposed algorithm can learn basic skills and compound skills simultaneously. It consists of two levels of hierarchy, which are achieved by a duel-critic, multi-actor architecture.

In the architecture, a basic critic, which is in the first level of hierarchy, is responsible for training multiple actors that are each responsible for a different basic skill. The other critic in the second level of hierarchy, which is called meta critic, is responsible for learning compound skills by reusing basic skills learned by the actors. At the same time, a number of adjustments in reward function, replay memory and exploration are made to allow the learning of all components in both level of hierarchy to fit in the same one-thread learning process. We summarize the h-DDPG algorithm in Algorithm 5.1.

From the next section, we evaluate the performance of the proposed h-DDPG algorithm. To do so, we built three scenarios with different compound tasks. The tasks in these scenarios are designed to examine the agent capability in observing and distinguishing objects and moving accurately. We test both the performance of the basic critic and the meta critic and also examine the agent's performance when one of the actors of the agent failed to provide a basic skill. Finally, we compare the performance of h-DDPG with DQN [18] and DDPG [24] on solving the compound tasks in the three scenarios.

The rest of this chapter will start with an introduction of the experimental setup used in this chapter in Section 5.5. Then the performance of the algorithm is presented in Section 5.6. Section 5.7 will present the performance of the agent when an incompetent actor exists while Section 5.8 will present a comparison between h-DDPG and other algorithms. Section 5.9 is the summary of the experiments.

Algorithm 5.1 H-DDPG

Input: maximum training episode E_{max} , maximum steps in each episode S_{max}, mini-batch size M, replay memory **P.**

Initialization: randomly initialize networks weights θ^{Q_b}, θ^{Q_m}, θ^{π_1}, … … , θ^{π_G} and target networks weights $\theta^{Q_b^t} \leftarrow \theta^{Q_b}$, $\theta^{Q_m^t} \leftarrow \theta^{Q_m}$, $\theta^{\pi_g^t} \leftarrow \theta^{\pi_g}$.

while episode **<** E_{max}
  Initialize random noise *N* for exploration
  Get initial state s_1
  while step **<**S_{max} **and** episode **not** terminated
    Get Q values of actors using meta critic
    Select actor i according to $\epsilon-$ greedy policy
    Select action a_t using selected actor and add *N*
    Execute a_t and get reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in **P**
    Randomly sample $\frac{M}{G}$ transitions of each actor from **P** and make up a mini-batch with M transitions
    Update θ^{Q_m} according to (5.1) and (5.2) and update $\theta^{Q_m^t}$
    Update θ^{Q_b} according to (4.1) and (4.2) and update $\theta^{Q_b^t}$
    for g **in** G:
      Update θ^{π_g} according to (4.3) and update $\theta^{\pi_g^t}$
  end
end

5.5 Experiment Setup

The experiments in this chapter were also conducted in Gazebo 2 in a ROS Indigo environment, and we also use the Pioneer 3AT robot with a camera, range sensor and wheel speed readings the same to the one used in Chapter 3 and Chapter 4. To test the performance, we built three scenarios with different tasks in a walled space. We have tried to make them as observable as possible when building them. These scenarios are:

- **Scenario 1: Approaching an Object**

  The walled space is obstacle-free with only a single target object. The task that the robot needs to finish in this scenario is to approach the target object, starting from a random position and orientation, without crashing into walls.

- **Scenario 2: Approaching a Specific Target**

  The walled space contains two different objects, one target and one decoy. The task that the robot needs to solve in this scenario is to approach the target object, starting from a random position and orientation, without confusing it with the decoy or crashing into walls. To achieve this, the robot has to distinguish between the objects and apply different strategies to them, which makes this task more difficult than the task in scenario 1.

- **Scenario 3: Doorway Escape**

  The walled space is obstacle-free with four doorways one on each side of the space. The task that the robot needs to finish in this scenario is to go through one of the doorways, starting from a random position and orientation, without crashing into walls. This task is even more difficult than tasks in scenario 1 and 2 as the robot has to avoid collision with the sides of a door when going through it.

Top-down screenshots of these three scenarios are shown in Figure 5.2. We bundled the four most recent frames of the camera as well as the sensor data in the last frame to form a state. Frames captured by the camera will be converted to 64×64 grey scale images before being fed into networks. The agent will use these states to decide values of wheel speeds on both sides of the robot. The basic skills we assigned to the actors in the first level of hierarchy are going forward, going backward, turning left and turning right. Note that the actions of the robot are executed in continuous time, which means there is no gap between two consecutive actions and the length of time an action will be executed depends on the processing time needed before the next action has been decided. Similar to the experimental setups in Chapter 3 and Chapter 4, the episodes were terminated when the robot turns over or crashes into walls or decoy. In addition, in the experiments in this chapter, we also terminated the episode when the robot get stuck in the environments. This happened mainly because that the environments in this chapter are more complex and sometimes the robot may get stuck at corners or edges of the environment. But in this case, we simply re-initialized the episode without giving any punishments to the meta critic or the basic critic.

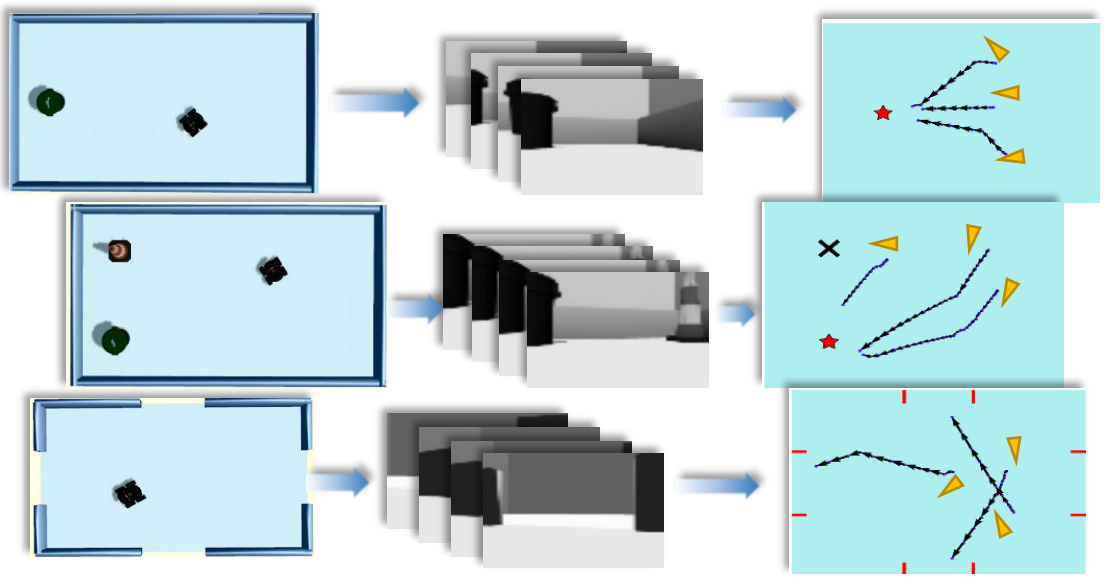

Figure 5.2: An introduction of the three simulation scenarios. They are, from top to bottom, the approaching object scenario (Scenario 1), the approaching specific target scenario (Scenario 2) and the doorway scenario (Scenario 3). Pictures from left to right in each row are top-down views of scenarios, image captures of the camera and the samples of solutions made by the agent. In the samples of solution, the yellow triangles indicate initial orientation, and stars indicate target objects.

As described in Section 5.2, we applied a parameter sharing learning scheme among different networks in the algorithm. The shared convolutional part consists of three layers. The first layer has 32 kernels with size 8×8 and stride 4, followed by the second layer which has 64 kernels with size 4×4 and stride 2. The last convolutional layer has 64 kernels with size 3×3 and stride 1. For the meta critic, the fully connected layer part consists of two hiddern layers with 512 nodes and 256 nodes respectively, while for the basic critic, the two hidden fully connected layers both consists of 300 nodes. Moreover, the last convolutional layer of the basic critic is a mlpconv layer. It takes the shared feature maps from the earlier convolutional layers and further processes them with mlpconv operations. The basic critic also includes the actions in its second fully-connected layer. All fully-connected parts of the actors consist of two hidden layers each, with 200 and 150 nodes respectively. In all networks, range sensor data and wheel speed readings is included right before the fully connected part of the networks. All the networks were built and trained in TensorFlow [122]. With regards to training strategy, we use Adam optimization [71] for all the networks. At the same time, l_2 regularization is introduced to prevent overfitting.

In each experiment, we tested the performance of both the meta critic and the basic critic at several points during training. Each time we tested the model, we ran 10 independent testing episodes. In each episode, we initialized the robot at a random position in the scenarios with a random orientation. All the other hyper-parameters we used in this chapter are listed in Table 5-1.

Description	**Symbol (if has)**	**Value**
Meta critic base learning rate	μ_{Q_m}	0.0025
Meta critic discount factor	γ_m	0.99
Basic critic base learning rate	μ_{Q_b}	0.001
Basic critic discount factor	γ_b	0.9
Basic critic reward	r_b	1
Actors base learning rate	μ_{π_g}	0.0001
l_2 penalty	/	0.01
Mini-batch size	M	64
Ornstein-Uhlenbeck process initial intensity	σ_{init}	0.12
Ornstein-Uhlenbeck process minimum intensity	σ_{min}	0.021

Initial ϵ	/	1
Minimum ϵ	/	0.1
Maximum step in each episode	S_{max}	50
Maximum training episode	E_{max}	5000
Soft update factor	φ	0.001

Table 5-1: Hyper-parameters for h-DDPG algorithm.

5.6 Hierarchical Learning Performance

We tested the performance of both the basic critic and the meta critic in all three scenarios. The performance of the basic critic is reflected by the performance of the actors. The results are presented below.

5.6.1 Basic Critic Performance

We first examined the performance of the basic critic by testing the performance of the actors during training. Results are shown in Figure 5.3.

We can see that all four actors achieved very good performance after training for around 1500 episodes. After the actors reached their best performance, the performance remains stable until the end of training. Samples of movement trajectories of each actor shown in Figure 5.3 also demonstrate that all actors are achieving good basic skills. High performance basic skills provided by the basic critic and actors will help the meta critic to understand the movement patterns of each actor better and learn how to reuse them to achieve the final compound task.

5.6.2 Meta Critic Performance

We first conducted a set of experiments in Scenario 1 to find an appropriate value of the reward for the meta critic. We compared the performance of the algorithm after 5000

episodes of training. The results are shown in Figure 5.4.

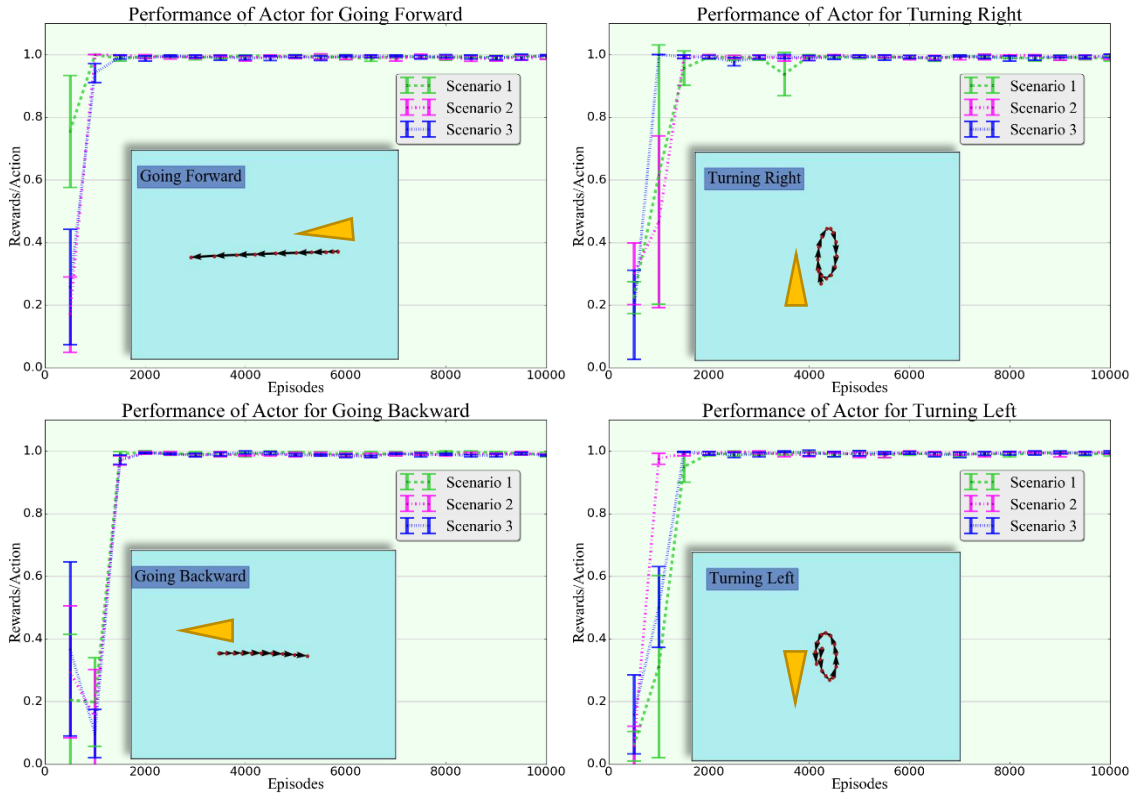

Figure 5.3: The performance of each actor in the three scenarios. The curves are the average performance and the error bars indicate the standard deviations. The pictures at the bottom-right of each curve are samples of moving trajectories of the actor. The yellow triangles indicate the initial orientation.

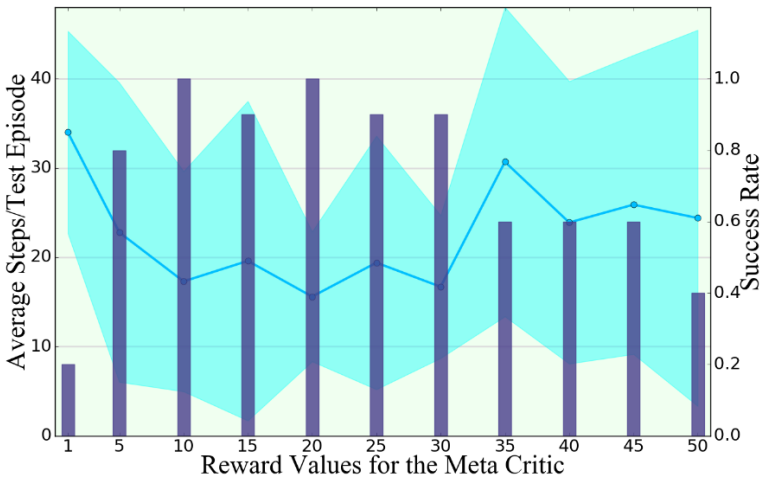

Figure 5.4: Performance comparison at 5,000 episode between training with different meta critic reward values in Scenario 1. The bars are success rate of finishing the task. The curves are average steps taken to finish the task in each test episode and the shadows indicate the standard deviation.

We can see that, the performance is poor when the reward value for the meta critic is close to the reward for the basic critic. The agent generally failed to learn when the reward value was 1. The performance improves when the value increases and tops when reward value gets to 10, as the average steps in one episode becomes fewer and success rate reached 100%.

After this, the performance remains high in a range of reward values. However, it starts to drop when the value gets to 35. The performance gets worse when the value gets higher. This may be because of the unstable gradient updates caused by big loss values, as the reward signal in deep reinforcement learning is also a part of the supervising signal for the networks. The results in Figure 5.4 demonstrate that a reward value between 10 and 30 is most suitable for the meta critic in learning compound skills in our scenarios. We then chose the reward value for the meta critic to be 10 and fixed it for the rest of the experiments in this chapter.

The results of the final performance of the proposed h-DDPG algorithm in all three scenarios are shown in Figure 5.5.

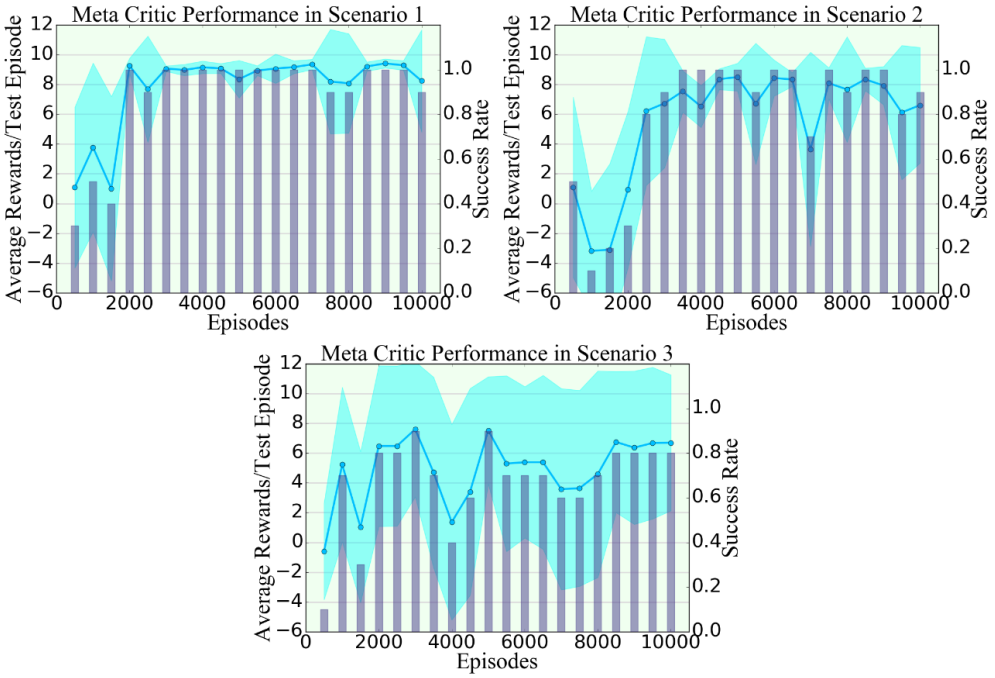

Figure 5.5: The performance of the meta critic. In each curve, the bars are success rate of finishing the task. The curves are average rewards in each test episode and the shadows indicate the standard deviation.

We can see that, the agent started to find solutions to the tasks after training for around 2000 episodes. Note that, we can know from Figure 5.3 that most actors achieve stable basic skills at around 1500 episodes. This means the meta critic actually learned better compound skills for the tasks right after stable basic skills became available. This also explains the instability of the performance before 2000 episodes as it would be hard for the meta critic to infer the basic skills of unstable actors.

For Scenarios 1 and 2, the agent successfully achieved the final goal in more than 90% of test cases with random initialization. This statistic is lower for Scenario 3, which is at around 80%. We observed that most failures in Scenario 3 were caused by collisions with the sides of a door when the robot tried to go through it. This may be caused by the fact that when the robot is near the door, it becomes harder to infer the orientation and position as what it can sense from camera and range sensors there is extremely similar (it is blank outside of the door). Sometimes, we observed the robot tried to solve the task by reversing out of the doors. This may be the way the agent learned to infer orientation and position near the door.

In all three scenarios, the agent was able to solve the tasks within around 18 action steps. Note that this is highly relevant to the position and orientation of the random initialization, which also partially causes the high deviation in Figure 5.5. We can see from samples of task solving trajectories given in Figure 5.2 that the agent was actually solving the tasks with near optimal solutions from different initializations. The overall success rate of the proposed algorithm in solving the tasks is 87.6%.

5.7 Learning with an Incompetent Actor

In addition, we conducted a set of experiments in Scenario 1 to investigate the impact of an incompetent actor on the performance of other actors. Similar to Chapter 4, this experiment is achieved by including an actor with a meaningless reward function. This actor received rewards randomly, so it would never learn. The performance of the actors in this situation is shown in Figure 5.6.

We can see that the basic critic can still learn several high performance actors even when one of the actors has bad performance. This owes to the robustness of the multi-DDPG algorithm discussed in Chapter 4, as this level of hierarchy of h-DDPG is mainly achieved by adapting the multi-DDPG algorithm. Finally, the performance of the meta critic in this scenario is shown in Figure 5.7.

We can see that the meta critic still has high performance on solving the compound task in Scenario 1. By comparing Figure 5.5 and Figure 5.7, we can see that the incompetent actor has no influence on the final performance of the algorithm. This adds to the robustness of the h-DDPG algorithm as individual failed actors will not interfere with the learning of other actors and the meta critic can still learn high performance compound skills by reusing actors that have high performance.

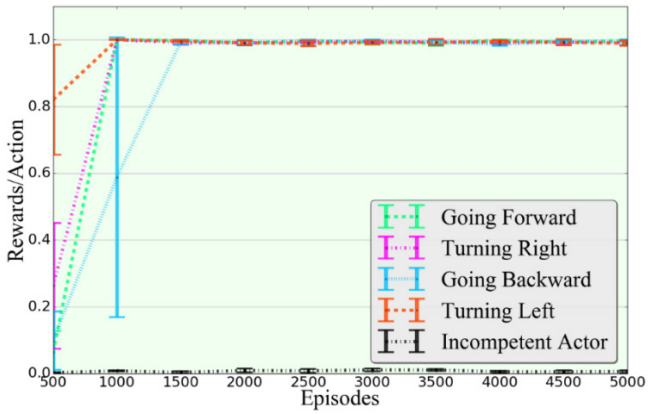

Figure 5.6: Performance of actors in Scenario 1 when training with an incompetent actor. The curves are average performance and the error bars indicate the standard deviations.

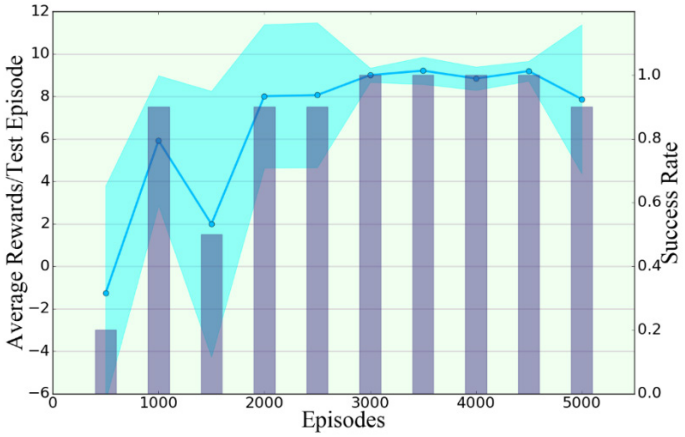

Figure 5.7: Performance of the meta critic in Scenario 1 when training with an incompetent actor. The bars are success rate of finishing the task. The curves are average rewards in each test episode and the shadows indicate the standard deviation.

5.8 Comparisons with Other Algorithms

We compared the proposed h-DDPG algorithm with two well-known deep reinforcement learning algorithms: DQN [18] and DDPG [24]. As introduced in Chapter 2, the first algorithm is a discrete action algorithm, so we fixed the speed values of the wheels in each basic skill so that the agent can get access to the four basic skills we used in our algorithm with equal quality to the best quality skills learned by our h-DDPG actors. Specifically, the action value pair ([left wheel, right wheel]) sent to the robot was fixed at [1.18, 1.18] for going forward (resulted in moving speed at around 0.2606 m/s), [-1.18, -1.18] for going backward (resulted in moving speed at around -0.2606 m/s), [-0.29, 0.89] for turning left (resulted in turning speed at around 27.69°/s) and [0.89, -0.29] for turning right (resulted in turning speed at around 27.69°/s). We use the hyper-parameters and network architecture proposed in their original papers when implementing these two algorithms. Both algorithms are one-thread training based, so suitable for comparison with the proposed h-DDPG algorithm. We compared the three algorithms in all three scenarios. The best performances obtained by learning with these algorithms are shown in Table 5-2.

	Scenario 1		Scenario 2		Scenario 3	
	Average reward ± standard deviation	Success rate	Average reward ± standard deviation	Success rate	Average reward ± standard deviation	Success rate
DQN	5.34 ± 5.17	80%	3.91 ± 5.78	70%	2.03 ± 5.38	50%
DDPG	-0.97 ± 6.21	30%	-3.45 ± 4.05	10%	-4.03 ± 3.80	10%
h-DDPG	**9.88 ± 0.74**	**100%**	**8.86 ± 0.90**	**100%**	**7.73 ± 2.01**	**90%**

Table 5-2: Comparison between best performance of different algorithms.

We can see that, the proposed h-DDPG outperformed DQN and DDPG in solving tasks in all three scenarios. DDPG frequently failed to solve the tasks, as the success rates are very low. This implies it may be hard to learn compound skills without knowing any basic skills.

Although DQN can solve all three tasks, it is less capable to do so compared to h-DDPG as the success rates are lower. Moreover, it took more action steps to solve the tasks compared to h-DDPG. We also observed that when using DQN, the robot usually suffered from instability when the chosen action in a timestep is different from the one chosen in its previous timestep. Sometimes this instability may result in wobbles that are big enough to cause terminations (causing the angle between the baseboard of the robot and the ground to be larger than 45°). This is mainly caused by the abrupt changes in speed when changing from one skill to another, as actions are executed in continuous time space and the length of time an action will be executed may vary. This may have influenced the performance of DQN. In contrast, when using h-DDPG, the robot merely wobbled and the testing episodes have never been terminated by a turnover of the robot. This is owing to the actors that could adjust the speed when performing basic skills to avoid sharp changes in speed. This may have allowed the meta critic to learn better compound skills, which is also one of the advantages of h-DDPG for handling robot control in a continuous action space.

These comparisons show that by introducing the hierarchical architecture and decomposing compound and basic skill learning, the proposed algorithm can not only learn better compound skills compared to other continuous action control algorithms, but also achieve smoother movement to support more stable compound skill learning compared to discrete action control algorithms.

5.9 Experiment Summary

In conclusion, we tested the h-DDPG algorithms in three scenarios with different compound tasks built in Gazebo 2 in a ROS Indigo environment. The tasks in these three scenarios were designed to examine the agent capability in observing and distinguishing objects and moving accurately.

The results show that the h-DDPG algorithm successfully learns both high performance basic skills and compound skills. In total, it successfully solved the tasks with a rate of 87.6% among all test cases with random position and orientation

initialization in different scenarios. Results also show that in cases that even when some of the actors fail to learn, other actors can still learn high performance basic skills that can be reused by the meta critic to solve the compound task. In comparison with other algorithms, the proposed h-DDPG outperforms other one-thread training based algorithms while also achieving comparable performance against other discrete action based algorithms in solving compound tasks.

In the next chapter, we will conclude the thesis.

Chapter 6

Conclusion and Future Work

Reinforcement learning is a kind of algorithms that can make use of sparse reward signals to bootstrap skills or policies to solve tasks in a given environment. Classical reinforcement learning algorithms are generally incapable of learning in complex environments or action spaces due to limited generalization capability. The success of deep learning models in dealing with computer vision problems that need strong generalization and feature extraction ability inspired interest in combining deep learning with reinforcement learning to improve the performance of the agent. Major challenges for combining these two techniques have finally been addressed by the Deep Q Network (DQN) algorithm [18], which brought deep reinforcement learning great success. Following DQN, more work has been done in recent years to improve the performance of deep reinforcement learning algorithms and apply them in other control problems or applications.

Compared to control tasks that have a discrete action space, control tasks that involve a continuous action space are more challenging as there are potentially unlimited action choices. Less work has been done in this area, especially when considering learning in multi-task or hierarchical learning scenarios. In addition, all one-thread based deep reinforcement learning algorithms for continuous action control can only solve basic tasks. Our works in this thesis aims to address some of these issues.

In the reminder of this chapter, we will conclude the whole thesis. This includes the conclusions of the contributions and experimental results in Section 6.1, the major limitations of the contributions in Section 6.2 and possible directions of future work in Section 6.3.

6.1 Conclusion

In this thesis, we further explored ways to achieve continuous action control with deep reinforcement learning. As introduced in Chapter 1, the final goal of our work is to develop an algorithm that can learn compound skills to make up for the limitations of existing one-thread learning based algorithms that can only learn basic skills.

This final goal has led to three main contributions in this thesis:

- We first proposed a novel network architecture for deep reinforcement learning in Chapter 3. The proposed network architecture made use of the multi-layer perceptron convolutional (mlpconv) layer [23] and the global average pooling layer in-between the convolutional part and the fully connected part of the network. It can reduce the number of parameters needed for deep reinforcement learning agents that need to use image data by around 72%. While the proposed network architecture is flexible enough to fit in different network settings for the convolutional part and fully connected part, we also proposed two implementation variations of the network architecture that can allow us to share parameters between networks and fuse sensor data into the agent.

- We then proposed a novel multi-task deep reinforcement learning algorithm that can learn multiple basic continuous action control tasks concurrently in Chapter 4. The proposed algorithm is called multi-DDPG, as it is based on the Deep Deterministic Policy Gradient (DDPG) algorithm [24]. The algorithm has a single-critic, multi-actor architecture, in which each actor is responsible for a particular task. We applied the sensor fusing and parameter sharing implementation of the proposed network architecture in Chapter 3 to further reduce the number of parameters needed for learning multiple tasks. Unlike other works on multi-task deep reinforcement learning that mainly rely on transfer learning or generative model techniques, multi-DDPG produces one agent that can do multiple tasks within the same learning process.

- Finally, we proposed a novel hierarchical deep reinforcement learning algorithm that can learn compound continuous action control tasks by reusing basic skills in Chapter 5, which is called h-DDPG. The algorithm has two levels of hierarchy which is achieved by a duel-critic, multi-actor architecture. Specifically, a basic critic is in the first level of hierarchy which is responsible for training multiple actors to learn multiple basic skills. A meta critic is in the second level of hierarchy which is responsible for learning compound skills by reusing the basic skills learned by the actors in the first level of hierarchy. The single-critic, multi-actor architecture in the first level of hierarchy was realized by adapting the multi-DDPG algorithm in Chapter 4. While the two levels of hierarchy are responsible for different types of learning, they will learn within the same process.

To test the performance of the proposed network architecture and algorithms, we built environments where a simulated robot could move in Gazebo 2 in a ROS Indigo environment. For the experiments in Chapter 3 and Chapter 4 that focus on evaluating the performance of the proposed algorithm and multi-DDPG, the environment is an obstacle free, walled space. For experiments that evaluate h-DDPG in Chapter 5, the environments were three scenarios with different compound tasks designed to test the agent's ability for observing and distinguishing objects and moving accurately. The simulated robot we used was a Pioneer 3AT robot. We set a camera on the front of it to give it first person vision of the environment. We also set a range sensor on the top of it to give it distance readings from four different angles (left, right, back and front).

From the experimental results we have shown in the last three chapters, we can see that:

- The parameter sharing variations of the proposed network architecture can achieve comparable performance to implementations that have not shared any parameters. By fusing sensor data into the network, the learning of the agent is sped up on these basic skill learning scenarios, as discussed in Section 3.3. A DDPG agent that is built on the proposed network architecture can achieve comparable performance to a DDPG agent that is built on traditional CNNs on all the 12 highly constrained basic locomotion tasks, with many fewer parameters

being introduced in the networks.

- The performance of the proposed multi-DDPG algorithm on learning multiple tasks concurrently is consistent when the number of tasks and the constraints on the tasks increase. On learning the 12 highly constrained basic locomotion tasks, multi-DDPG has achieved comparable performance to DDPG, which learns these 12 tasks by training 12 individual agents, one for each task, with many more parameters introduced during learning. What is more, when one of the actors of the multi-DDPG agent is incompetent, the rest of the actors can still learn high performance skills that are as good as in the scenarios where all the actors learn skills successfully.

- Both levels of hierarchy of h-DDPG can learn efficiently. The basic critic and actors in the first level of hierarchy achieved comparable performance to multi-DDPG on learning the four basic skills we assigned to the agent (going forward, going backward, turning left and turning right). The meta critic in the second level of hierarchy successfully learned all the three compound skills needed to solve the compound tasks in the three scenarios. The performance of h-DDPG on solving the compound tasks in the three scenarios outperformed DDPG, which generally failed to learn any of these tasks, and DQN, which is a discrete action control algorithm that is not able to make the robot move as smoothly as h-DDPG does in our continuous time control scenarios. Moreover, when one of the actors in the first level of hierarchy failed to provide a good basic skill, the basic critic can still train other actors to learn high performance basic skills, which can be reused by the meta critic to prevent the compound skill learning being interfered with by the failed actor.

These results support the conclusion that the proposed network architecture and algorithms all achieved good performance on learning the given tasks. These results and discussions in the thesis also permit us to conclude that:

- The mlpconv and global average pooling layer applied in-between the convolutional part and fully connected part of the network can significantly reduce the number of parameters needed, while also holding enough features

extracted from the raw image data to provide high performance learning of deep reinforcement learning agents, even when the convolutional part of the networks are shared among several networks in the agent.

- The combination of the loss function provided by (4.1) and (4.2) and the gradient inferring method provided by (4.3) is successful in providing an agent that can learn multiple tasks concurrently. Specifically, by using the proposed loss function and gradient inferring method, the single critic in the architecture can balance among all evaluations of state-action values for existing actors during self-updating, while it can also separate information to help update each individual actor to learn different tasks.

- The introduction of a new hierarchy in the agent can enable the agent to learn compound skills in a one-thread based learning scheme by reusing basic skills in the lower level of hierarchy. While the two levels of hierarchies are cooperating with each other on providing the final compound skills and have shared some of their components (network layers and replay memory), unsuccessfully learned basic skills in the lower level of hierarchy do not affect the overall performance of the algorithm.

- Even though discrete action control methods are easier to learn, especially with the introduction of deep learning models, they are less capable than continuous action control methods in a continuous time scenario, where the length of time that an action last is uncertain. The fixed legal actions provided prior to learning may also impose restrictions on the flexibility of the agent.

In summary, the work in this thesis has addressed the lack of multi-task and hierarchical learning algorithms for continuous action spaces by proposing two novel algorithms, one for each scenario. A novel deep reinforcement learning network architecture is proposed to reduce the huge amount of parameters needed in these two algorithms. The final h-DDPG algorithm can successfully learn compound skills in a continuous action space with a one-thread learning scheme, which cannot yet be achieved by other deep reinforcement learning algorithms.

6.2 Limitations

While the final outcome of this thesis, which is the h-DDPG algorithm, can successfully learn compound skills in a continuous action space with a one-thread learning scheme, we still found three main limitations:

- During learning, we found the agent needs sufficient free moving space in the environment to explore the continuous action space to learn to choose actions to achieve the movement patterns to build basic skills. When learning in crowded environments where a lot of objects or obstacles exist, the agent may failed to learn the basic skills needed for the compound skill learning.

- The algorithm can only learn compound skills to solve compound tasks in fully observable scenarios. This means the final goal of the task must not be hidden in the environment. In scenarios involve partially observable tasks such as maze problems or sequential tasks, the algorithm generally fails to understand the tasks and provide any compound skills.

- All the algorithms proposed in this thesis require predefined reward functions for each task assigned, including the basic tasks and the compound tasks. The reward function is central to these algorithms and a badly defined reward function can directly cause failure. This to some extent has limited the flexibility of the algorithms.

Future works can be done to fix these limitations and further improve the performance of the algorithm.

Nonetheless, the proposed h-DDPG algorithm has proved its capability of learning compound skills in a continuous action space to solve fully observable compound tasks as well as its ability at providing smoother movement compared to discrete action control algorithms. These advantages should still make it a competitive algorithm compared to other deep reinforcement learning algorithms.

6.3 Future Work

Considering the limitations we discussed in the last section and the properties of the works in this thesis, future work could focus on two directions.

The first direction is to find ways to further improve the performance of the algorithms in this thesis. It should focus on solving the limitations of the algorithm we discussed in the last section:

- Improvements on the replay memory could be made to help reduce the free moving space needed for learning basic skills. This is because the transitions in the replay memory may have different contributions to the learning. There may also be transitions that are not good for learning. Ways could found to classify different kinds of transitions and highlight those that are beneficial for learning. This may also help make the whole learning process faster as compound skills are based on the basic skills.

- New levels of hierarchy can be introduced in the algorithm to enable the algorithm to use multi-level hierarchy to decompose complex tasks in a more detailed way. This may give the agent more potential to solve partially observable tasks, as each of the decomposed tasks can be fully observable.

- It is also beneficial to find ways to enable the agent to have memory ability. It can help the agent combine sequential information in exploration to further improve its intelligence. This is also a potential way to reduce the free moving space needed and to allow the agent to understand partially observable tasks.

- Ways could be found to introduce intrinsic motivations in the algorithm (in some or all levels of hierarchy). This is a very promising way to eliminate the need of predefined reward functions in the algorithm, which could further add to the generalization and flexibility of the algorithm.

The second direction of future way is to find entirely new methods to handle continuous action control with deep reinforcement learning. Possible future work in this direction could be:

- Finding new network architectures or applying other network architectures in deep reinforcement learning algorithms. The new network should be able to learn more features of the environment and analyse them more thoroughly. Some existing network architectures may be suitable for this such as Inception Networks [48] and SSP [58].

- Finding new gradient policies to train a more capable actor. The new update gradient policy must be able to deliver more detailed information from the critic so that it can maximally alleviate information loss when connecting actor and critic through gradients.

Besides these two directions, more analysis could be carried out to better assess the capability of the proposed algorithms. This includes:

- Analysis of the robustness of the algorithms against noises in the input data. It would be interesting to see whether removing the noise in the input data using some data pre-processing techniques could help improve the performance of the agent.

- Analysis of the capability of the algorithms in extreme conditions. It would be interesting to see how much 'uncertainty' in the reward function the algorithm can sustain before being unable to learn and what the overall performance would be if more than one, or even most of, the actors are having poor performance.

References

[1] S. Dehuri, A. K. Jagadev, and M. Panda, "*Multi-Objective Swarm Intelligence,"* vol. 592. Berlin, Heidelberg: Springer, 2015.

[2] M. G. Helander, T. K. Landauer, and P. V. Prabhu, "*Handbook of Human-Computer Interaction,"* 2 ed. New York, United States: Elsevier, 2014.

[3] M. L. Puterman, "*Markov Decision Processes: Discrete Stochastic Dynamic Programming."* Hoboken, New Jersey: John Wiley & Sons, 2008.

[4] B. D. Argalla, S. Chernovab, M. Velosob, and B. Browning, "A Survey of Robot Learning from Demonstration," *Robotics and Autonomous Systems,* vol. 57, pp. 469-483, 2009.

[5] C. J. C. H. Watkins and P. Dayan, "Q-Learning," *Machine Learning,* vol. 8, pp. 279-292, 1992.

[6] G. A. Rummery and M. Niranjan, "*On-Line Q Learning using Connectionist Systems."* Cambridge, England: University of Cambridge, Department of Engineering, 1994.

[7] M. Grounds and D. Kudenko, "Parallel Reinforcement Learning with Linear Function Approximation," in *Proceedings of International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007, pp. 60-74.

[8] G. Konidaris, S. Osentoski, and P. Thomas, "Value Function Approximation in Reinforcement Learning using the Fourier Basis," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2011, pp. 380-385.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature,* vol. 521, pp. 436–444, 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Network," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097-1105.

[11] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," in *International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645-6649.

[12] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 91-99.

[13] R. Zhao, W. Ouyang, H. Li, and X. Wang, "Saliency Detection by Multi-Context Deep Learning," in *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1265-1274.

[14] W. Ouyang, X. Wang, X. Zeng, S. Qiu, P. Luo, Y. Tian*, et al.*, "DeepID-Net: Deformable Deep Convolutional Neural Networks for Object Detection," in *Proceedings of Conference on Computer Vision and Pattern Recognition* 2015, pp. 2403-2412.

[15] A. Kumar, O. Irsoy, P. Ondruska, M. Iyyer, J. Bradbury, I. Gulrajani*, et al.*, "Ask Me Anything: Dynamic Memory Networks for Natural Language Processing," in *Proceedings of International Conference on Machine Learning*, 2016, pp. 1378-1387.

[16] Z. Liu, X. Li, P. Luo, C. C. Loy, and X. Tang, "Semantic Image Segmentation via Deep Parsing Network," in *International Conference on Computer Vision*, 2015, pp. 1377-1385.

[17] H. Noh, S. Hong, and B. Han, "Learning Deconvolution Network for Semantic Segmentation," in *Proceedings of International Conference on Computer Vision* 2015, pp. 1520-1528.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra*, et al.*, "Playing Atari with Deep Reinforcement Learning," presented at the Conference on Neural Information Processing Systems Deep Learning Workshop, 2013.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare*, et al.*, "Human-level Control through Deep Reinforcement Learning," *Nature,* vol. 518, pp. 529-533, 2015.

[20] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. v. d. Driessche*, et al.*, "Mastering the Game of Go with Deep Neural Networks and TreeSearch," *Nature,* vol. 529, pp. 484-489, 2016.

[21] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking Deep Reinforcement Learning for Continuous Control," in *Proceedings of International Conference on Machine Learning*, 2016, pp. 1329-1338.

[22] A. Lazaric, "Transfer in Reinforcement Learning: a Framework and a Survey," in *Reinforcement Learning*. vol. 12, ed Berlin, Heidelberg: Springer, 2012, pp. 143-173.

[23] M. Lin, Q. Chen, and S. Yan. "Network in Network." *arXiv preprint arXiv:1312.4400*, 2013.

[24] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa*, et al.*, "Continuous Control with Deep Reinforcement Learning," in *International Conference on Learning Representations*, 2016.

[25] R. S. Sutton and A. G. Barto, "*Reinforcement Learning: An Introduction,"* vol. 1. Cambridge, United States: MIT Press, 1998.

[26] A. G. Barto and S. Mahadevan, "Recent Advances in Hierarchical Reinforcement Learning," *Discrete Event Dynamic Systems,* vol. 13, pp. 341-379, 2003.

[27] G. Baldassarre and M. Mirolli, "*Intrinsically Motivated Learning in Natural and Artificial Systems."* Berlin, Heidelberg: Springer, 2013.

[28] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences," *Machine Learning,* vol. 3, pp. 9-44, 1988.

[29] R. J. Williams, "*Toward a Theory of Reinforcement-Learning Connectionist Systems."* Boston, United States: Northeastern University, 1988.

[30] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning,* vol. 8, pp. 229-256, 1992.

[31] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Proceedings of International Conference on Neural Information Processing Systems*, 1999, pp. 1057-1063.

[32] J. Peters, S. Vijayakumar, and S. Schaal, "Natural Actor-Critic," in *European Conference on Machine Learning*, 2005, pp. 280-291.

[33] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike Adaptive Elements that Can Solve Difficult Learning Control Problems," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 13, pp. 834 - 846, 1983.

[34] R. S. Sutton, "Temporal Credit Assignment in Reinforcement Learning," Ph.D, University of Massachusetts Amherst, Amherst, United States, 1984.

[35] H. Kimura and S. Kobayashi, "An Analysis of Actor/Critic Algorithms Using Eligibility Traces: Reinforcement Learning with Imperfect Value Function," in *Proceedings of International Conference on Machine Learning* 1998, pp. 278-286.

[36] V. R. Konda and V. S. Borkar, "Actor-Critic--Type Learning Algorithms for Markov Decision Processes," *SIAM Journal on Control and Optimization,* vol. 38, pp. 94-123, 1999.

[37] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 42, pp. 1291-1307, 2012.

[38] K. G. Vamvoudakis and F. L. Lewis, "Online Actor–Critic Algorithm to Solve the Continuous-Time Infinite Horizon Optimal Control Problem," *Automatica,* vol. 46, pp. 878-888, 2010.

[39] Y. Nakamura, T. Mori, M.-a. Sato, and S. Ishii, "Reinforcement Learning for a Biped Robot based on a CPG-Actor-Critic Method," *Neural Networks,* vol. 20, pp. 723-735, 2007.

[40] V. S. Borkar, "An Actor-Critic Algorithm for Constrained Markov Decision Processes," *Systems & Control Letters,* vol. 54, pp. 207-213, 2005.

[41] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *The Bulletin of Mathematical Biophysics,* vol. 5, pp. 115–133, 1943.

[42] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.," *Psychological Review,* vol. 65, pp. 386-408, 1958.

[43] M. L. Minsky and S. Papert, "*Perceptrons: An Introduction to Computational Geometry."* Cambridge, United States: Mit Press, 1969.

[44] P. J. Werbos, "*Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences."* Cambridge, United States: Harvard University, 1975.

[45] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE,* vol. 86, pp. 2278 - 2324, 1998.

[46] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proceedings of Computer Vision and Pattern Recognition*, 2009, pp. 248-255.

[47] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *European Conference on Computer Vision*, 2014, pp. 818-833.

[48] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov*, et al.*, "Going Deeper with Convolutions," in *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1-9.

[49] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations*, 2015.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770-778.

[51] Z. Zhang, C. Zhang, W. Shen, C. Yao, W. Liu, and X. Bai, "Multi-Oriented Text Detection with Fully Convolutional Networks," in *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4159-4167.

[52] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, "Deep Networks with Stochastic Depth," in *European Conference on Computer Vision*, 2016, pp. 646-661.

[53] C. Szegedy, V. Vanhoucke, S. Ioffe, and J. Shlens, "Rethinking the Inception Architecture for Computer Vision.," in *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818-2826.

[54] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[55] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," in *European Conference on Computer Vision*, 2016, pp. 630-645.

[56] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580-587.

[57] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1440-1448.

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," in *European Conference on Computer Vision*, 2014, pp. 346-361.

[59] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431-3440.

[60] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. "Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. ." *arXiv preprint arXiv:1207.0580*, 2012.

[61] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of Neural Networks using DropConnect," in *Proceedings of International Conference on Machine Learning*, 2013, pp. 1058-1066.

[62] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding Up Convolutional Neural Networks with Low Rank Expansion," in *British Machine Vision Conference*, 2014.

[63] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both Weights and Connections for Efficient Neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135-1143.

[64] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing Neural Networks with the Hashing Trick," in *Proceedings of International Conference on Machine Learning*, 2015, pp. 2285-2294.

[65] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations," in *Advances in Neural Information Processing Systems*, 2015, pp. 3123-3131.

[66] N. Qian, "On the Momentum Term in Gradient Descent Learning Algorithms," *Neural Networks,* vol. 12, pp. 145-151, 1999.

[67] Y. Nesterov, "A Method of Solving a Convex Programming Problem with Convergence Rate O (1/k2)," *Soviet Mathematics Doklady,* vol. 27, pp. 372-376, 1983.

[68] G. Hinton, N. Srivastava, and K. Swersky. "Lecture 6a: Overview of Mini-Batch Gradient Descent." *Coursera Lecture slides* *https://class.coursera.org/neuralnets-2012-001/lecture*, Online.

[69] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *The Journal of Machine Learning Research,* vol. 12, pp. 2121-2159, 2011.

[70] M. D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method." *arXiv preprint arXiv:1212.5701*, 2012.

[71] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations*, 2015.

[72] Y. Sun, X. Wang, and X. Tang, "Deep Convolutional Network Cascade for Facial Point Detection," in *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2013.

[73] B. Graham. "Fractional Max-Pooling." *arXiv preprint arXiv:1412.6071*, 2014.

[74] L.-J. Lin, "Reinforcement Learning for Robots using Neural Networks," Ph.D, School of Computer Science, Carnegie Mellon University, Pittsburgh, United States 1993.

[75] N. Heess, G. Wayne, D. Silver, T. Lillicrap, Y. Tassa, and T. Erez, "Learning Continuous Control Policies by Stochastic Value Gradients," in *Advances in Neural Information Processing Systems*, 2015, pp. 2944-2952.

[76] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous Deep Q-Learning with Model-Based Acceleration," in *Proceedings of International Conference on Machine Learning*, 2016, pp. 2829-2838.

[77] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust Region Policy Optimization," in *Proceedings of International Conference on Machine Learning*, 2015, pp. 1889-1897.

[78] G. Dulac-Arnold, R. Evans, H. v. Hasselt, P. Sunehag, T. Lillicrap, J. Hunt*, et al.*, "Deep Reinforcement Learning in Large Discrete Action Spaces," presented at the International Conference on Machine Learning Abstraction in Reinforcement Learning Workshop, 2016.

[79] M. H. P. Stone, "Deep Reinforcement Learning in Parameterized Action Space," in *International Conference on Learning Representations*, 2016.

[80] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria*, et al.*, "Massively Parallel Methods for Deep Reinforcement Learning," presented at the International Conference on Machine Learning Deep Learning Workshop, 2015.

[81] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal Value Function Approximators," in *Proceedings of International Conference on Machine Learning*, 2015, pp. 1312-1320.

[82] Z. Wang, T. Schaul, M. Hessel, H. v. Hasselt, M. Lanctot, and N. d. Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," in *Proceedings of International Conference on Machine Learning*, 2016, pp. 1995-2003.

[83] H. v. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2016, pp. 2094-2100.

[84] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," in *International Conference on Learning Representations*, 2016.

[85] M. G. Bellemare, G. Ostrovski, A. Guez, P. S. Thomas, and R. e. Munos, "Increasing the Action Gap: New Operators for Reinforcement Learning," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2016, pp. 1476-1483.

[86] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare, "Safe and Efficient Off-Policy Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 1046-1054.

[87] A. Altahhan, "Towards a Deep Feature-Action Architecture for Robot Homing," in *IEEE International Conference on CIS & RAM*, 2015, pp. 205-209.

[88] C. W. Anderson, M. Lee, and D. L. Elliott, "Faster Reinforcement Learning after Pretraining Deep Networks to Predict State Dynamics," in *International Joint Conference on Neural Networks*, 2015.

[89] Y. Yang, X. Li, and L. Zhang, "Task-specific Pre-Learning to Improve the Convergence of Reinforcement Learning Based on a Deep Neural Network " in *World Congress on Intelligent Control and Automation*, 2016, pp. 2209-2214.

[90] M. Hausknecht and P. Stone, "Deep Recurrent Q-Learning for Partially Observable MDPs," presented at the AAAI Fall Symposium, 2015.

[91] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based Control with Recurrent Neural Networks," presented at the Conference on Neural Information Processing Systems Deep Reinforcement Learning Workshop, 2015.

[92] I. Sorokin, A. Seleznev, M. Pavlov, A. Fedorov, and A. Ignateva, "Deep Attention Recurrent Q-Network," presented at the Conference on Neural Information Processing Systems Deep Reinforcement Learning Workshop, 2015.

[93] C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo*, et al.* "Model-Free Episodic Control." *arXiv preprint arXiv:1606.04460*, 2016.

[94] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru*, et al.*, "Multiagent Cooperation and Competition with Deep Reinforcement Learning," *PLoS ONE,* vol. 12, p. e0172395, 2017.

[95] J. N. Foerster, Y. M. Assael, N. d. Freitas, and S. Whiteson, "Learning to Communicate with Deep Multi-Agent Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 2137-2145.

[96] J. N. Foerster, Y. M. Assael, N. d. Freitas, and S. Whiteson, "Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks," presented at the International Joint Conference on Artificial Intelligence Deep Reinforcement Learning: Frontiers and Challenges Workshop, 2016.

[97] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning Multiagent Communication with Backpropagation," in *Advances in Neural Information Processing Systems*, 2016, pp. 2244-2252.

[98] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap*, et al.*, "Asynchronous Methods for Deep Reinforcement Learning," in *Proceedings of International Conference on Machine Learning*, 2016, pp. 1928-1937.

[99] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei*, et al.* "Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning." *arXiv preprint arXiv:1609.05143*, 2016.

[100] H. Cuayáhuitl, S. Keizer, and O. Lemon, "Strategic Dialogue Management via Deep Reinforcement Learning," presented at the Conference on Neural Information Processing Systems Workshop on Deep Reinforcement Learning, 2015.

[101] H. Cuayáhuitl, "SimpleDS: A Simple Deep Reinforcement Learning Dialogue System," in *Dialogues with Social Robots*. vol. 427, ed Berlin, Germany: Springer, 2016, pp. 109-118.

[102] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky. "Deep Reinforcement Learning for Dialogue Generation." *arXiv preprint arXiv:1606.01541*, 2016.

[103] J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng*, et al.*, "Deep Reinforcement Learning with a Natural Language Action Space," in *Annual Meeting of the Association for Computational Linguistics*, 2016, pp. 1621-1630.

[104] J. C. Caicedo and S. Lazebnik, "Active Object Localization With Deep Reinforcement Learning," in *Proceedings of International Conference on Computer Vision*, 2015, pp. 2488-2496.

[105] F. Abtahi, Z. Zhu, and A. M. Burry, "A Deep Reinforcement Learning Approach to Character Segmentation of License Plate Images," in *International Conference on Machine Vision Applications*, 2015, pp. 539-542.

[106] D. Zhao, Y. Chen, and L. Lv, "Deep Reinforcement Learning with Visual Attention for Vehicle Classification," *IEEE Transactions on Cognitive and Developmental Systems,* vol. PP, 2016.

[107] J. He, M. Ostendorf, X. He, J. Chen, J. Gao, L. Li*, et al.*, "Deep Reinforcement Learning with a Combinatorial Action Space for Predicting Popular Reddit Threads," in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1838-1848.

[108] L. Li, Y. Lv, and F.-Y. Wang, "Traffic Signal Timing via Deep Reinforcement Learning," *Automatica Sinica,* vol. 3, pp. 247 - 254, 2016.

[109] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep Direct Reinforcement Learning for Financial Signal Representation and Trading," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 28, pp. 653 - 664, 2017.

[110] C.-K. Yeh and H.-T. Lin, "Automatic Bridge Bidding Using Deep Reinforcement Learning," in *European Conference on Artificial Intelligence*, 2016, pp. 1362-1369.

[111] M. Lai. "Giraffe: Using Deep Reinforcement Learning to Play Chess." *arXiv preprint arXiv:1509.01549* 2015.

[112] K. Narasimhan, T. Kulkarni, and R. Barzilay, "Language Understanding for Text-based Games Using Deep Reinforcement Learning," in *Proceedings of Conference on Empirical Methods in Natural Language Processing* 2015, pp. 1-11.

[113] X. B. Peng, G. Berseth, and M. v. d. Panne, "Terrain-adaptive Locomotion Skills Using Deep Reinforcement Learning," in *Proceedings of ACM SIGGRAPH*, 2016.

[114] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *Proceedings of International Conference on Machine Learning*, 2014, pp. 387-395.

[115] G. E. Uhlenbeck and L. S. Ornstein, "On the Theory of the Brownian Motion," *Physical Review,* vol. 36, p. 823, 1930.

[116] S. P. Bangaru, J. Suhas, and B. Ravindran, "Exploration for Multi-task Reinforcement Learning with Deep Generative Models," presented at the Neural Information Processing Systems Deep Reinforcement Learning Workshop, 2016.

[117] D. Borsa, T. Graepel, and J. Shawe-Taylor. "Learning Shared Representations in Multi-Task Reinforcement Learning." *arXiv preprint arXiv:1603.02041* 2016.

[118] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard. "Deep Reinforcement Learning with Successor Features for Navigation across Similar Environments." *arXiv preprint arXiv:1612.05533*, 2016.

[119] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical Deep Reinforcement Learning Integrating Temporal Abstraction and Intrinsic Motivation," in *Advances in Neural Information Processing Systems*, 2016, pp. 3675-3683.

[120] R. Krishnamurthy, A. Lakshminarayanan, P. Kumar, and B. Ravindran, "Hierarchical Reinforcement Learning using Spatio-Temporal Abstractions and Deep Neural Networks," presented at the International Conference on Machine Learning Abstraction in Reinforcement Learning Workshop, 2016.

[121] S. Mohamed and D. J. Rezende, "Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 2125-2133.

[122] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro*, et al.* "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems." *arXiv preprint arXiv:1603.04467*, 2016.