Fall 12-22-2020

Multi-Agent Deep Reinforcement Learning for Walkers

Inhee Park
San Jose State University

Multi-Agent Deep Reinforcement Learning for Walkers

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Inhee Park

December 2020

The Designated Project Committee Approves the Project Titled

Multi-Agent Deep Reinforcement Learning for Walkers

by

Inhee Park

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

December 2020

Teng Moh, Ph.D.	Department of Computer Science
Katerina Potika, Ph.D.	Department of Computer Science
Mike Wu, Ph.D.	Department of Computer Science

ABSTRACT

Multi-Agent Deep Reinforcement Learning for Walkers

by Inhee Park

This project was motivated by seeking an AI method towards Artificial General Intelligence (AGI), that is, more similar to learning behavior of human-beings. As of today, Deep Reinforcement Learning (DRL) is the most closer to the AGI compared to other machine learning methods. To better understand the DRL, we compares and contrasts to other related methods: Deep Learning, Dynamic Programming and Game Theory.

We apply one of state-of-art DRL algorithms, called Proximal Policy Optimization (PPO) to the robot walkers locomotion, as a simple yet challenging environment, inherently continuous and high-dimensional state/action space.

The end goal of this project is to train the agent by finding the optimal sequential actions (policy/strategy) of multi-walkers leading them to move forward as far as possible to maximize the accumulated reward (performance). This goal can be accomplished by finding the tuned hyperparameters of the PPO algorithm by monitoring the performances for the multi-agent DRL (MADRL) settings.

At the end, we can draw three conclusions from our findings based on the various MADRL experiments: 1) Unlike DL with explicit target labels, DRL needs larger minibatch size for better estimate of values from various gradients. Therefore, a minibatch size and its pool size (experience replay buffer) are critical hyperparameters in PPO algorithm. 2) For the homogeneous multi-agent environments, there is a mutual transferability between single-agent and multi-agent environments to be able to reuse the tuned hyperparameters. 3) For the homogeneous multi-agent environments with a well tuned hyperparameter set, the

parameter sharing is a better strategy for the MADRL in terms of performance and efficiency with reduced parameters and less memory.

To conclude, reward-driven, sequential and evaluative learning, the DRL, would be closer to AGI if multiple DRL agents learn to collaborate to capture the true signal from the shared environment. This work provides one instance of implicit cooperative learning of MADRL.

ACKNOWLEDGMENTS

I truly appreciate my research advisor, Professor Teng Moh for providing the razor-sharp insights into my weekly research presentations. With his finer feedback, I've been able to improve this project to this final stage. Most of all, I could resurrect the genuine interest in CS research by learning from him how to articulate beneficial aspects from my findings.

I am grateful to my committees, Professor Katerina Potika and Professor Mike Wu for their insightful questions during my defense presentation. I could polish this report more clearly based on their feedback.

My husband, Gwangho deserves much kudos for this completion. Unreserved critiques on my time management & Evergreen cheers on my new research! I'm excited to have similar CS career path with him here in Silicon Valley.

Lastly, thank you Mom and family in Korea for their unconditional support.

#SJSU, #CS, #DRL, #MADRL, #AGI, #Study-from-Home, #Work-from-Home, #COVID-19, #2020, #Silicon Valley, #San Jose

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Motivation

The end goal of this project is how to find the optimal sequential actions (walking and carrying a package) for the multi-objectives (walkers with robot-legs) to take leading them to move forward as far as possible to maximize the accumulated score (reward) using Deep Reinforcement Learning.

1.1.1 Why Deep Reinforcement Learning (DRL)? It's Closer to Artificial General Intelligence.

This project was motivated by seeking an AI method more towards Artificial General Intelligence (AGI), i.e. more similar to learning behavior of human-beings. As of today, Deep Reinforcement Learning (DRL) is the most closer to the AGI compared to other machine learning methods. As shown in Figure 1 Supervised Learning (SL) is one-shot learning, i.e. its prediction is either true or false with respect to the target label of training data; Without target labels, Unsupervised Learning (UL) is able to extract complex patterns from the given data. Both SL and UL are passive and static leaning by minimizing the prediction errors from the offline static data. In contrast, DRL (DL + RL = DRL) is active and dynamic learning method owing to an **agent** that interacts with its **environment** (i.e. takes actions to environment and incorporates the feedback from environment) through generating online experience data while learning.

Human-beings are rather active and dynamic learners by **interacting with environment**, not just passively learning the given information at that moment, but also **exploring** more information by taking the previous experiences (ups and

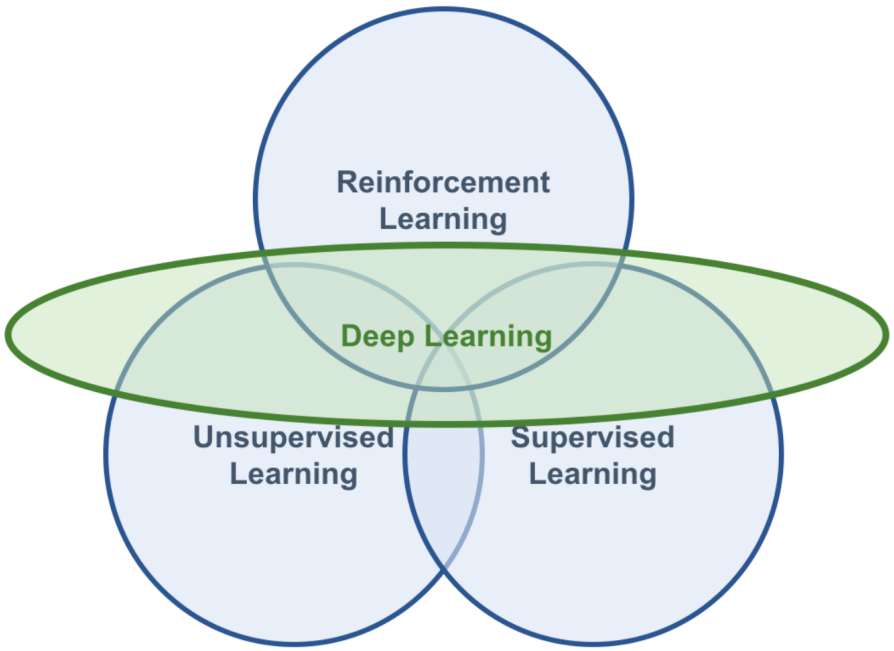


Figure 1: Reinforcement Learning is neither SL nor UL. Deep Learning + Reinforcement = DRL

downs envisaged as rewards or penalties) into consideration relentlessly. Also, to achive a specific goal, human-beings learn **sequentially** (not just one time) by **evaluating own actions** followed by accumulating experience in their own learning **strategy (aka policy)** to **maximize the return** (not just immediate reward, but also long-term accumulated rewards). Such description provides a high-level analogy to the technical explanation of the DRL in Chapter 2.

1.1.2 Why Walkers (Robot-Legs) Environment? It's Very Intuitive DRL Application to Real-World Robotics.

AlphaGo has gotten public attention as a prototype of advanced AI technology to realization exceeded human intelligence. Core driving force of AlphaGo is ''Deep'' RL, which uses neural network (like DL) in conjunction to RL. Since then

DRL has been a popular AI method applied to various fields, namely, robotics for manipulating locomotion, healthcare for protein folding prediction [3], operational research for adaptive scheduling [4], resource management for heterogeneous 5G network [5], etc.

Here for the thesis project, we select a simple walker system (a part of robot system, consisting of several joints of legs and thighs) as an environment for the DRL based on the following two reasons:

Firstly, robot locomotion is straightforward to understand what we want to achieve without prior domain knowledge (robotics environments are collected for the DRL researchers [6, 7]), hence I can focus on the algorithmic aspect of DRL without distraction of application specific knowledge.

Secondly, learning/training the walkers environment requires high-dimensional and continuous action space (not discrete action space) thus challenging to train. Yet, walkers environment has reduced degrees of freedom from legs/thighs instead of whole body, thus it's feasible to train given the available time and computing resource allowed for this thesis project. Chapter 4 provides a detailed specification of the walkers environment.

1.1.3 Why Multi-Agent? Challenging Problems Can be Solved Cooperatively by Sharing Information.

Table 1 shows that DRL is closely related to Dynamic Programming in the single-agent settings; also related to the Game Theory in the multi-agent settings.

In single-agent settings, for both DRL and Dynamic Programming (DP) as optimization methods, DP enumerates all the possible states hence limited to small scale problems, whereas DRL uses sampling and neural network as a function approximator thus can be can scalable to larger problems.

Table 1: Comparison: DRL vs. Dynamic Programming vs. Game Theory

Suitable Scale	Single-Agent	Multi-Agent
Large-Scale Problems	**DRL**	**Multi-Agent DRL**
Small-Scale Problems	Dynamic Programming	Game Theory

Table 2: Comparison: DRL vs. Game Theory

DRL	Game Theory
Agent	Player
Environment	Game
Policy	Strategy
Reward	Payoff
State	Information State
Greedy Policy	Best Response

Aa listed in Table 2 Multi-Agent DRL (MADRL) and Game Theory share similar concept almost one-on-one conceptual matching between them just represented by different terminologies. That is, both MADRL and Game Theory behave similarly with common goal to maximize accumulated reward and total payoff, respectively.

In multi-agent settings, multiple agents share a single environment with each other agent, thus the optimal policy of single-agent depends not only on the environment, but on the policies of the other agents. That's why multi-agent DRL is challenging than single-agent settings suffering two problems: 1) non-stationarity and 2) high variance of estimated values. Theses challenges in MADRL can be

mitigated by **cooperative** learning strategies such as **parameter sharing** and other training strategies in a way to improve the stationarity as well as to reduce the variance.

1.2 Aims & Research Questions

At the end of this work, we'd like to address the following questions based on the observations from the DRL runs with **Agent** (Proximal Policy Optimization) and **Environment** (single and multi-(2-, 3-, and 4-) walkers).

Three **broad conceptual questions** are: 1) How to capture the true reward signal of environment to achieve a goal? 2) Mutual transfer learning is feasible in DRL between single- and multi-agent settings? and 3) In multi-agent settings, parameter sharing is better than independent learning?

In addition, two **specific technical questions** will be answered: 1) How to find the tuned hyperparameters in DRL to maximize the performance? 2) How to balance training bias vs. variance in DRL?

The answers to those questions listed above would contribute to not only enhancing the conceptual understanding of DRL problems, but also providing technical information with DRL training strategies (little different from DL) together with an algorithm-specific tuned hyperparameter set.

1.3 Summary of Accomplishments & Overall Structure

Figure 2 highlights major achievements from this study. Overall structure of this report is following: Chapter 2 will provide background information on conceptual aspect of DRL, component of DRL, classification of DRL methods, and key equations of DRL. Learning strategies and architectures of Multi-Agent DRL

will be described in Chapter 3. Experimental settings for DRL framework and specifications of walkers environments will be given in Chapter 4. Mainly six observations will be described and corresponding supporting information will be discussed in Chapter 5. We will conclude with lessons learned from this project and suggest future direction in Chapter 6.

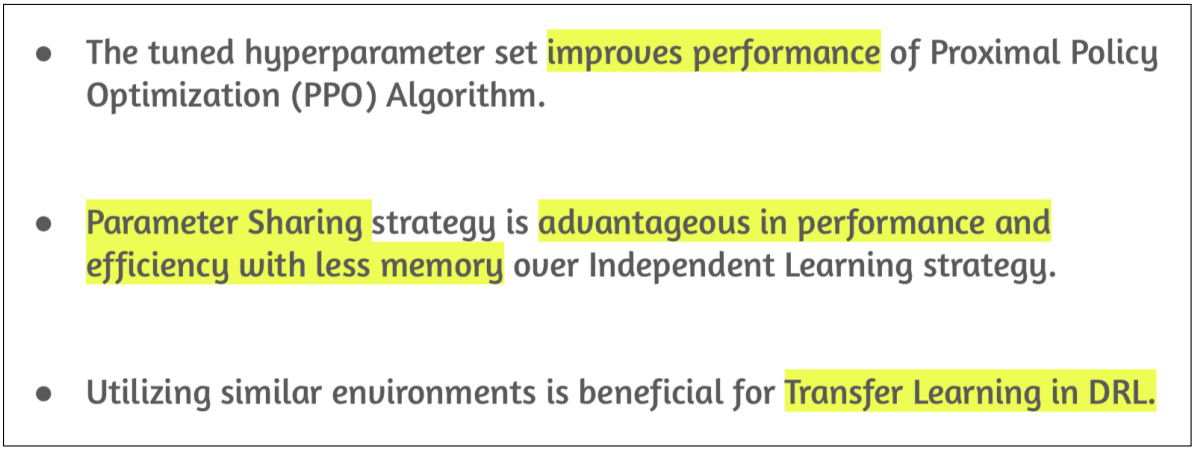


Figure 2: Accomplishments of This Study

CHAPTER 2

Background on Deep Reinforcement Learning

The purpose of providing background information in this chapter is to aid understanding of the Advantage Actor-Critic Proximal Policy Optimization (PPO), which is used throughout this study as a main DRL algorithm. At the end of this chapter the PPO architecture is shown in Figure 7.

2.1 Concept of Reward-Driven Learning DRL

Outside of AI/ML domain, ''reward function'' has been examined as a means of reinforcement, motivation or stimulus of learning. [2]

Similarly in AI/ML domain, core driving force of DRL is ''reward'', thus most research questions in DRL is all about ''reward''. For example, how to assign the ''reward'' to a series of actions to achieve a goal by value-function? how to estimate ''reward'' by value-function-based or policy-gradient-based methods? how to improve the policy to maximize the accumulated ''reward''?

As depicted in Figure 3, DRL is a combination of deep learning and reinforcement learning. Core component of DRL is Agent and Environment (everything else other than an agent is considered as en environment). An abridged description of DRL is that it's a reward-driven, sequential learning (opposed to one-shot learning), evaluative learning (opposed to supervised learning). [1] DRL is learning through interaction with environment by generating experiences (state, action, reward and new state), where agent is taking an action to influence environment, where one state is transitioned into new state as a result of action, its corresponding reward is passed onto the agent.

Reward concept and prediction of true reward signal are therefore foundations

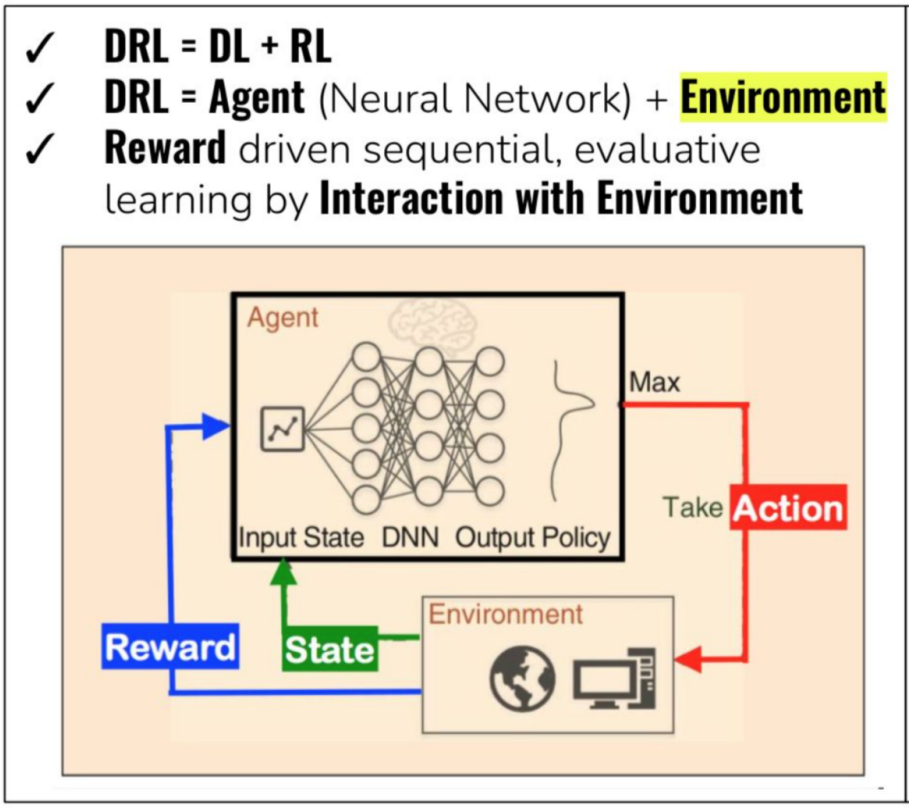


Figure 3: DRL Concept

of DRL. Ultimate goal of DRL is to maximize the accumulative, discounted (not just immediate reward but considering future reward) reward as depicted in the inset figure of Figure 4 describing time-span view of an agent interacting with environment. Hence the end goal of this project is to improve the performance of DRL evaluated by constructing a performance graph shown Figure 4.

2.2 Notations and Terms Used in DRL

Table 3 provides a convenient lookup for notations, terms and corresponding definitions used in DRL. More comprehensive list can be found in [8].

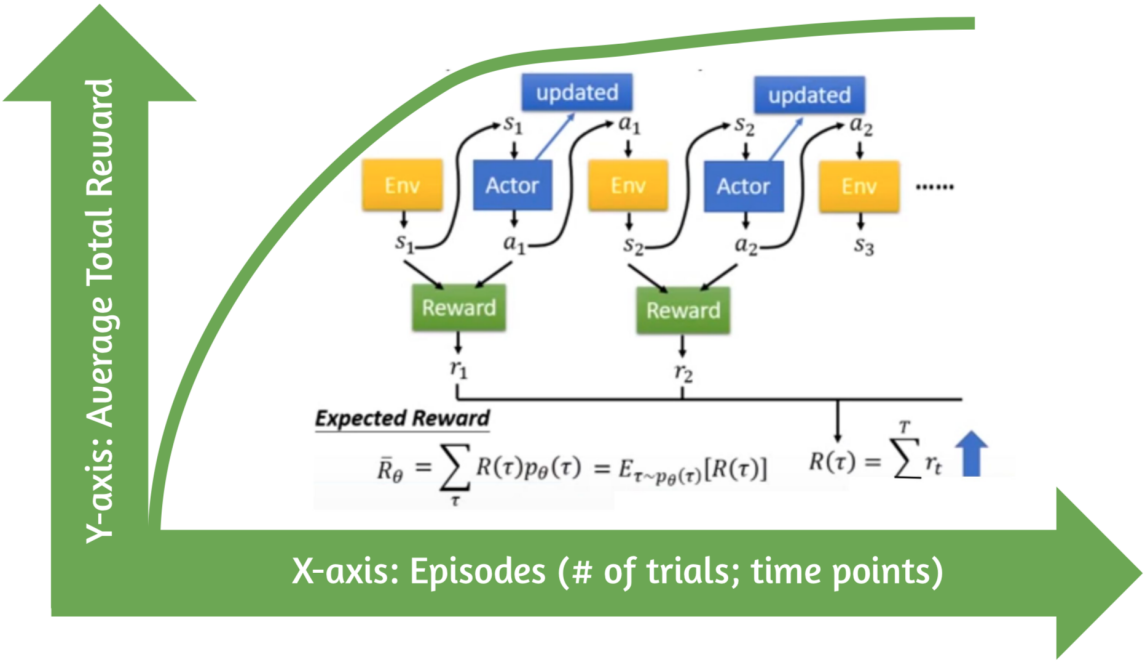


Figure 4: DRP Performance Graph

2.3 Evolution of DRL

For more holistic view of the DRL features, we compares the DRL with other related AI methods. Firstly, the core framework of Markov Decision Process (MDP) as a **reward-driven sequential decision making process** hasn't been changed since the old-fashioned RL era (exhaustive, explicit information required) the RL has been advanced to more powerful AI as DRL (combination of DL + RL) as compared in Table 4. Secondly, another comparison of DRL with relatively well known DL is tabulated in Table 5.

2.4 Components of DRL

2.4.1 Core Components for RL: Agent + Environment

Overall, the components of DRL, **environment** and **agent**, are interconnected with each other via **state/observation**, **reward**, **action** and **policy**. as depicted and annotated in Figure 5.

One of core components of RL is **Agent**, also called **Actor**. Agent is learning by interacting with environment. Main tasks of an agent at time t are: executing action A_t to influence environment; receiving observation S_t from environment; and receiving reward R_t from environment. Overall agent role is to update the policy while training on the experiences, τ, which is also generated by the policy, specifically called ''behavioral policy'', using value-function network (with parameters θ) and policy-network (parameterized with ϕ). **Value-function**, $Q_\pi(s, a)$, is a method of estimate future reward from action a in state s under policy π.

$$Q_\pi(s, a) = \sum_a \pi(a|s) \sum_{s',r} P(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right] \tag{1}$$

Policy, $\pi(s)$, is agent's behavior of selecting action a, which is updated while learnig.

$$\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s',r} P(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right] \tag{2}$$

Another core components of RL is **Environment**, which is composed of three components: **Observation-Space** $\mathcal{S}$, **Action-Space** $\mathcal{A}$, and **Reward-function**.

$$r(s, a) = \mathbb{E}\left[R_t | S_{t-1} = s, A_{t-1} = a\right] \tag{3}$$

All the components in the environment are pre-defined specified by the end-goal. None of the components in the environment can be directly controlled by the agents. Agent can only interact and influence environmen via actions. Main tasks of environment at time t are: receiving action A_t from agent, emitting observation S_{t+1}, and emitting reward R_{t+1}.

2.4.2 Additional Components for Deep RL: Neural Network + Experience Replay

For the DRL, in which neural network is used to approximate the value-function, we should consider conventional strategy used in DL, i.e. trade-off between variance and bias. In DL as a supervised learning, **high variance** results in **over-fitting** to the training data, whereas **high bias** results in **under-fitting**. In DRL, **high variance** indicates **noisy value prediction** but more accurate values, whereas **high bias** indicates **stationary target** but inaccurate values.

DRL is not supervised (no target labels) but evaluative (value-based prediction). To obtain temporal target similar to supervised learning, two separate neural networks are considered: **target network** and **evaluation network**. Then the ''target'' network is cloned from the evaluation network to imitate supervised learning target by freezing the parameters of network with occasional update from value-prediction network. Then the objective function for the value-based DRL is to minimize the loss function.

$$L(\theta) = \mathop{\mathbb{E}}_{\tau}\left[r + \gamma \max_{a'} Q(s', a'; \theta^{-}) - Q(s, a; \theta))^2\right] \tag{4}$$

Another important additional component of DRL is **Experience Replay Buffer**. In DRL, experience samples are generated online, thus samples are correlated. To avoid correlations among samples and to reduce the variance, experience replay buffer holds a broad set of past experiences generated by the behavioral policy of the agent through interacting with the environment.

Table 3: Notations and terms in DRL

Notations	Terms and Definitions
$s; S_t \in \mathcal{S}$	- states or observations; - state/observation at time t in observation-space $\mathcal{S}$
$a; A_t \in \mathcal{A}$	- actions; - action at time t in action-space $\mathcal{A}$
$r; R_t$	- (scalar) rewards $\in \mathbb{R}$
$r(s, a)$	- reward function; - $r(s, a) = \mathbb{E}[R_t \vert S_{t-1} = s, A_{t-1} = a]$
$p(s', \vert\, s, a)$	- in short $P_{ss'}$; - transition function getting to the next state s' by taking action a in current state s
τ	- episodes ~ experiences ~ trajectories ~ roll-outs; - sequence of (states/observations, actions and rewards) generated by behavioral policy; - $\tau = (S_1, A_1, R_1, \cdots, A_{t-1}, S_t, R_t)$
γ	- discount factor ($0 < \gamma < 1$); - if $\gamma = 1$, no penalty to uncertainty of future reward
G_t	- return = sum of discounted accumulated reward from step t to final step $T \in \mathbb{Z}^+$; - $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T$
$\pi(a \mid s)$	- (stochastic) policy, i.e. $P(a \mid s)$; agent's behavior function
$V^\pi(s)$	- value-function = V-function under π; - measure of goodness of in each state s; - value of expected return of state s; - $V^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi}[G_t \mid S_t = s]$
$Q^\pi(s, a)$	- state-action-value function = Q-function under π; - measure of goodness of in each state s and action a; - value of expected return of state s and action a; - $Q^\pi(s, a) = \mathop{\mathbb{E}}_{a\sim\pi}[G_t \mid S_t = s, A_t = a]$;
$A(s)$	- A-function; - advantage function with reduced variance using state-value function as baseline; - $A(s, a) = Q(s, a) - V(s)$
$\mathop{\mathbb{E}}_{\tau\sim\pi}[G(\tau)]$	- expected return G from experienced samples τ generated by policy π; - $= \int \pi(a \mid s) G(\tau) d\tau$

Table 4: Comparison: MDP, RL and Deep RL

Methods	Comparative Features
1st Generation: MDP	- basis framework for RL in terms of agent/environment state/action/reward; - requires full **model** information (entire state-space, action-space and explicit state-transition probabilities $P_{ss'}$); - limited to small-scale; unrealistic to have full state information
2nd Generation: RL	- model-free (i.e. no more need explicit $P_{ss'}$), rather sampling experience; - selects action from the Q-table, which is updated while learning; - not scalable as growing the size of Q-table
3rd Generation: Deep RL	- model-free (i.e. no more need explicit $P_{ss'}$), suitable for high dimensional, continuous state-/action-space, - use **Deep** Learning (neural network) to parameterize value-function, such as state-in-action-out network, which is updated while learning; - scalable owing to use Neural Network

Table 5: Comparison: Deep Reinforcement Learning (DRL) vs. Deep Learning

	DRL	DL
Requirements	Environment (State-space, Action-Space and Reward) + Agent (Policy, Value Function, Neural Network as Function Approximator)	Dataset & Neural Network Architecture
Input Data	Online SARS (S_t, A_t, R_t, S_{t+1}) tuples for agent experienced through environment	Offline training dataset (X) with target labels (Y)
Tasks	Data learning + Data generating	Data learning (optimizing NN weights and biases)
End Goal	How well the reinforcement signal reflects the true reward structure of the environment to maximize mean accumulated rewards	How well the model fits the training data
Who's Learning	Agent is learning by interacting with Environment	Model is learning by optimizing weights
Learning Style	Sequential learning; adaptive learning by taking an action to Environment and learning from the feedback (reward)	One-shot learning; passive learning from the given data
Objective Function	Loss function to minimize (Value-function) + Performance function to maximize (Policy-gradient)	Loss function to minimize
Performance Metric	Mean accumulated rewards over episodes/trials	Prediction accuracy

DRL Components

Agent (brain; policy; DRL algorithm)

- "Parameterized" Policy (action distribution)
- Value-function (policy evaluator)

Neural Networks

- Target Network (mimicking target label)
- Evaluation Network

Experience Replay Buffer

- Mini-batch (assorted experience)
- Training-batch (generated experiences)

Environment (pre-defined)

- Observation-Space (state representation)
- Action-Space (change of configuration)
- Reward-function (driving force; -100 if fallen)

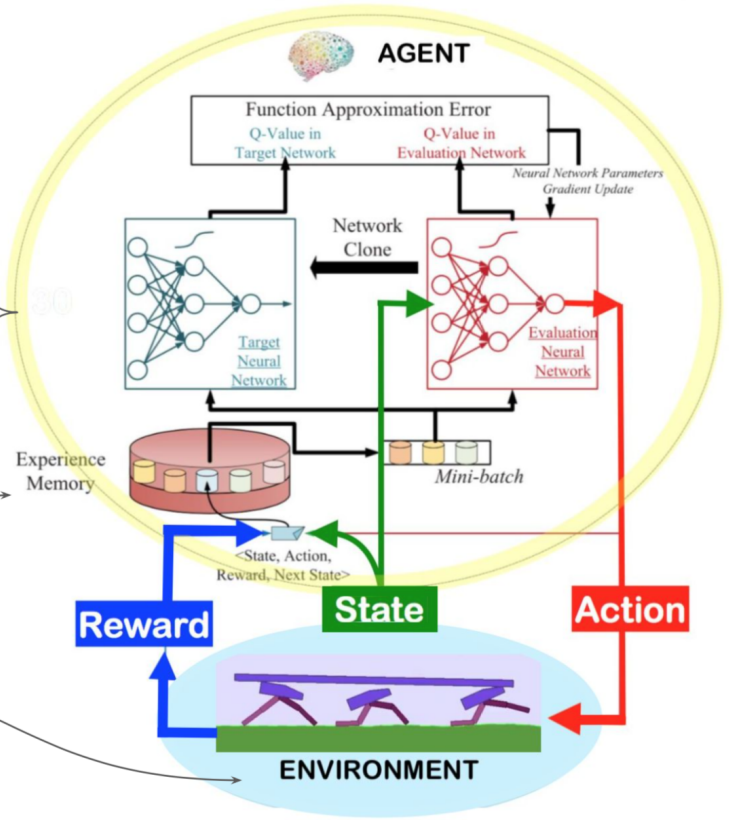


Figure 5: Components of DRL

2.5 Key Equations in DRL

DRL is a reward-driven learning to find an optimal policy. The value-function is a method of assigning reward to a series of actions that agent takes till episode termination time at T. But agent should keep updating its policy based on the value-function estimate at some time interval, instead of delaying the update of the policy till termination time at T. The value-function can be decomposed per time step t by **Bellman Equation**, so that we agent can evaluate reward per experience sample.

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{s',a'}\left[R_t + \gamma Q^{\pi}(s',a') \mid s,a\right] \tag{5}$$

Value function is crucial metric to find an optimal policy. There are mainly three types of value functions, namely V-function, Q-function and A-funciton. **In this work, we use the A-function** as a reduced variance value-function owing to use V-function as a baseline.

$$\text{V-function}: \quad V^{\pi}(s) = \mathop{\mathbb{E}}_{a\sim\pi}[G_t \mid S_t = s] \tag{6}$$

$$\text{Q-function}: \quad Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{a\sim\pi}[G_t \mid S_t = s, A_t = a] \tag{7}$$

$$\text{A-function}: \quad A(s,a) = Q(s,a) - V(s) \tag{8}$$

In DRL, there are **three different approaches** toward computing the optimal policy: 1) **value-based methods**; 2) **policy-gradient-based methods**; and 3) **actor-critic methods**.

Firstly, **value-based** methods are indirect ways of finding the policy via value estimate. What we want to optimize by training the neural network on the

experience samples is the value-based loss function to minimize:

$$L(\theta) = \mathop{\mathbb{E}}_{(s,a,r,s')}\left[\left(Q_{Target} - Q_{Predict}\right)^2\right], \text{ where} \tag{9}$$

$$Q_{Target} = r + \gamma \max_{a'} Q(s', a'; \theta) \tag{10}$$

$$Q_{Predict} = Q(s, a; \theta) \tag{11}$$

Corresponding loss gradient with the Target Network (with fixed parameters θ^-) nd Experience Replay Buffer ($\mathcal{D}$) as discussed in Section 2.4.2

$$\nabla_\theta L(\theta) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim\, \mathcal{U}(\mathcal{D})}\left[\left(Q_{Target(\theta^-)} - Q_{Predict}\right)\nabla_\theta Q(s, a; \theta)\right], \text{ where} \tag{12}$$

$$Q_{Target(\theta^-)} = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{13}$$

$$Q_{Predict} = Q(s, a; \theta) \tag{14}$$

Secondly, **policy-gradient-based** methods are directly way of obtaining the policy by finding the direction of increasing the values. In this method, the objective function to optimize is the **performance function to maximize**:

$$J(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[G(\tau)\right] \tag{15}$$

Then its policy-gradient is

$$\nabla_\theta J(\theta) = \nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[G(\tau)\right] \tag{16}$$

$$= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} G(\tau)\nabla_\theta \log \pi\left(A_t|S_t\right)\right] \tag{17}$$

Finally, **Actor-Critic** methods are combined both value-based and policy-gradient-based. The role of Actor is a policy improvement, whereas the role of Critic is a policy evaluation. Overall architecture is designed to reduce the variance, where actor is basically agent for selecting an action whereas critic is a value-estimate to guide the actor to select better action. **In this work, we used Actor-Critic method**.

2.6 State-of-Art DRL Algorithm: Proximal Policy Optimization (PPO)

PPO is one of state-of-art performance DRL algorithms, which can be used for both single-agent and multi-agent settings. It is composed of advantageous features of precursor DRL algorithms together with clipping methods. The PPO algorithm is mainly composed of the following three features, namely 1) Advantage Actor-Critic (A2C) with Multi-**Workers**; 2) Experience Replay Buffer with Minibatches; and 3) **Clipping Policy and Value**. We will explain each of these three features of PPO in detail.

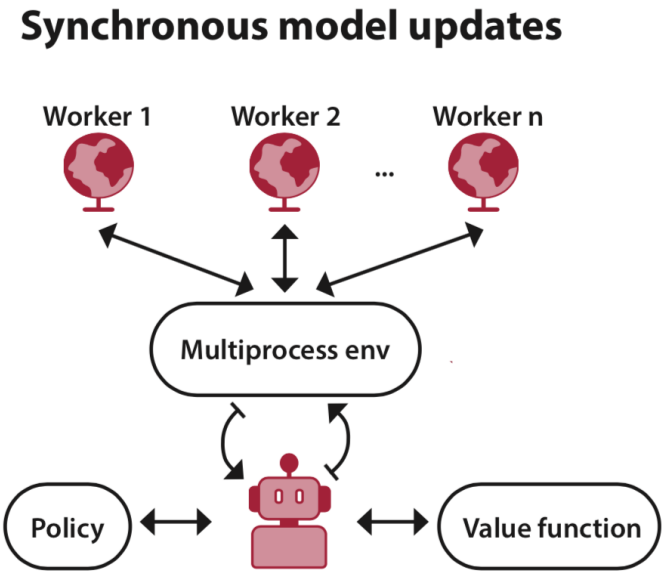


Figure 6: A2C: synchronous policy updates with multi-workers to generate experiences. Excerpt from [1]

Firstly, the PPO is similar to the A2C, which is another actor-critic DRL method with two key features: 1) for multi-core utilization, concurrent workers/actors generate a broad set of experience samples in parallel as shown in Figure 6; 2) single neural network shared by both policy and value function.

Secondly, Experience Replay Buffer is implemented to the Actor-Critic ar-

chitecture. Actor-critic-network training procedure with the experience replay is following: a) **Actors/Workers** with its policy π interact with the environment to gather experience samples and store them to the Experience Replay Buffer. b) Once all the actors/worker finish collecting the designated sampling size, uniformly select minibatch size of samples from the buffer is used to train the **critic network** (policy evaluation). The critic network is parameterized with θ value function to minimize the mean squared error between target Q-value and predicted Q-value. c) Then the **actor network** (policy improvement) update the policy, i.e. find the better policy, from the updated parameters by critic network.

Thirdly, clipping policy and value is a core feature of the PPO. Clipping the policy and value restricts drastic change of the gradient, thus not only preventing a divergence but also allowing the reuse of the experience replay buffer. **Clipped policy objective** function uses A^{GAE} for further reduction of the variance. It is a more advanced advantage function (A-function), estimated by Generalized Advantage Estimator (GAE), which balances between variance and bias. [21]

$$J(\phi, \phi^{-}) = \mathop{\mathbb{E}}_{(s,a,A^{GAE})} \Bigg[\min \Bigg(\frac{\pi(a \mid s; \phi^{-})}{\pi(a \mid s; \phi)} A^{GAE}, \quad \text{clamp}\Bigg(\frac{\pi(a \mid s; \phi^{-})}{\pi(a \mid s; \phi)} A^{GAE}; 1-\epsilon; 1+\epsilon \Bigg) \Bigg) \Bigg] \tag{18}$$

In the same manner, **clipped value loss** is also considered to further prevent divergence and reduce the variance.

$$L(\theta, \theta^{-}) = \mathop{\mathbb{E}}_{(s,a,G,V)\sim\mathcal{U}(\mathcal{D}(\theta^{-}))} \Bigg[\max \Bigg(G - V(s;\theta), \quad G - \Big(V + \text{clamp}\Big(V(s;\theta) - V, -\delta, \delta \Big) \Big) \Bigg) \Bigg] \tag{19}$$

All in all, Figure 7 shows the main DRL algorithm to be used throughout this project: Advantage Actor-Critic Proximal Policy Optimization.

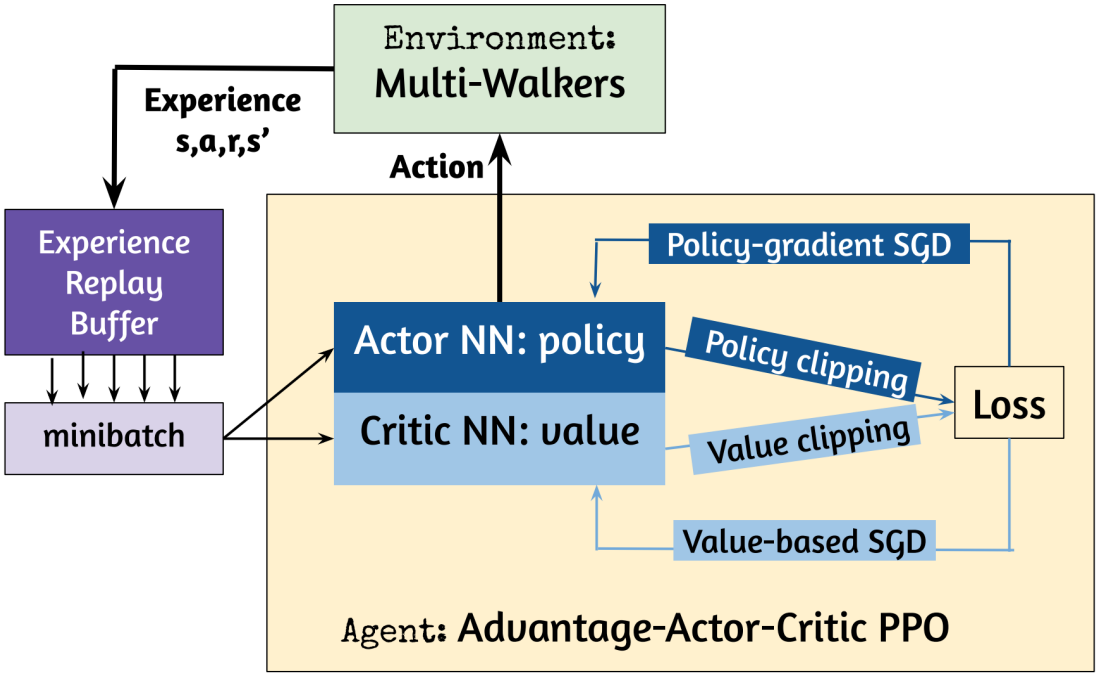


Figure 7: Advantage Actor-Critic Proximal Policy Optimization Architecture

CHAPTER 3

Background on Multi-Agent DRL & Related Previous Works

3.1 Centralized Learning and Decentralized Execution

In multi-agent DRL (MADRL) settings, multiple agents share a single environment with each other agents, thus the optimal policy of single-agent depends not only on the environment, but also on the policies of the other agents. Due to the shared environment, when one agent's learning changes the environment, simultaneously another agent's learning from the environment becomes obsolete. This causes non-stationary of the environment (causing difficulty of convergence) as well as high variance in the estimated values (causing noisy estimate).

The Actor-Critic architecture examined in Section 2.6 for the single-agent setting can be extended to the MADRL with the generally accepted multi-agent training framework: ''Centralized Learning with Decentralized Execution (CLDE)''. The CLDE is design to reduce the non-stationarity by centralized learning during training, but each of multi-agent executes an action based on its own observation (i.e. local information). In analogy, students (= multi-agents) study together by exchanging information with others in a study group (= centralized learning), but students should take an exam based on each student's own knowledge (= decentralized execution).

We can apply the general CLDE framework to the Actor-Critic architecture. The **Centralized Critic Network** is trained to minimize the value-function loss for better estimate of Q-value. Then the predicted Q-value is used in the **Decentralized Actor Network** to be trained via policy-gradient.

3.2 Independent Learning vs. Parameter Sharing

In MADRL, there are two extreme cases in terms of sharing information among agents: independent learning (also called concurrent learning) and parameter sharing.

Independent Learning (Figure 8a) is the most decentralized but naive MADRL. Multiple agents are trained independently based on its own local observation in which the existence of other agents are implicitly recognized as a part of environment. As to increase the number of agents, no scalability is expected because each agent should generate own experience samples without sharing with other actors/workers (as described in Section 2.6), thus memory capacity becomes an issue.

Parameter Sharing (Figure 8b) is the most centralized MADRL method without explicit communication among agents. Multiple agents share the network parameters of actor-network and critic-network. This is an appropriate learning strategy for the **homogeneous agents**. For those similar or identical agents, if one agent has less chance to learn from the environment, such agent is beneficial by following the shared policy that was properly updated by other agents. Parameter sharing strategy also shares the rewards, which encourages the multiple agents to participate cooperatively to accomplish a common goal. Unlike independent learning, parameter sharing is scalable because experiences gathered from multiple actors/workers are all are shared in the experience replay buffer.

3.3 Related Works on MADRL for Walkers Environments

There are two related previous studies using the same multi-walker environments. One study initially created the multi-walker environments using a precursor algorithm of PPO.[11] Another study used same PPO algorithm along with other state-of-algorithms. [12] Table 6 summarizes the performance of the common environment of 3-walkers. In the next chapter in Table 10, we will show that our study outperforms both studies by improving the performance in terms of average accumulated reward.

Table 6: Performance for the 3-Walkers Environment from Other Studies

Ref	Agent	Max Avg. Reward (Independent Learning)	Max Avg. Reward (Parameter Sharing)
Gupta et al. [11]	TRPO	51	54
Terry et al. [12]	PPO	38	41

3.3.1 Multi-Agent Walker Environment using TRPO

Gupta et al. [11] tried to extend re-usability of single-agent DRL algorithms to multi-agent settings for the three multi-agent environments including multi-walkers. They implemented a single-agent algorithm, Trust Region Policy Optimization (TRPO) in conjunction to the parameter sharing strategy for the multi-agent settings. With a fully independent learning (they used term 'concurrent learning') as a performance baseline, they applied parameter-sharing version of DRL algorithms to multi-walker environment with other experiments such as global vs. local reward. However, in this study we would like to employ parameter sharing concept to the most up-to-date algorithm of Proximal Policy Optimiza-

tion (PPO) for the same multi-walkers environment by systematic performance comparison with 2, 3, and 4-Walkers.

3.3.2 3-Walker Environment using PPO

Terry et al. [12] expanded the concept of parameter sharing to MADRL to wide span of DRL algorithms. Total 12 different DRL algorithms were compared in conjunction to parameter sharing as well as independent learning for the same three different multi-agent environments that Gupta et al. [11] used. According to their observation, parameter sharing with PPO outperformed for the 3-walkers environment. Based on their extensive testings, they provided a tuned hyper-parameter set for PPO as well as other DRL algorithms. However, as we will discussed in Sections 5.1 and 5.3, their tuned hyperparameter set has an issue (such as reward clipping). [17]

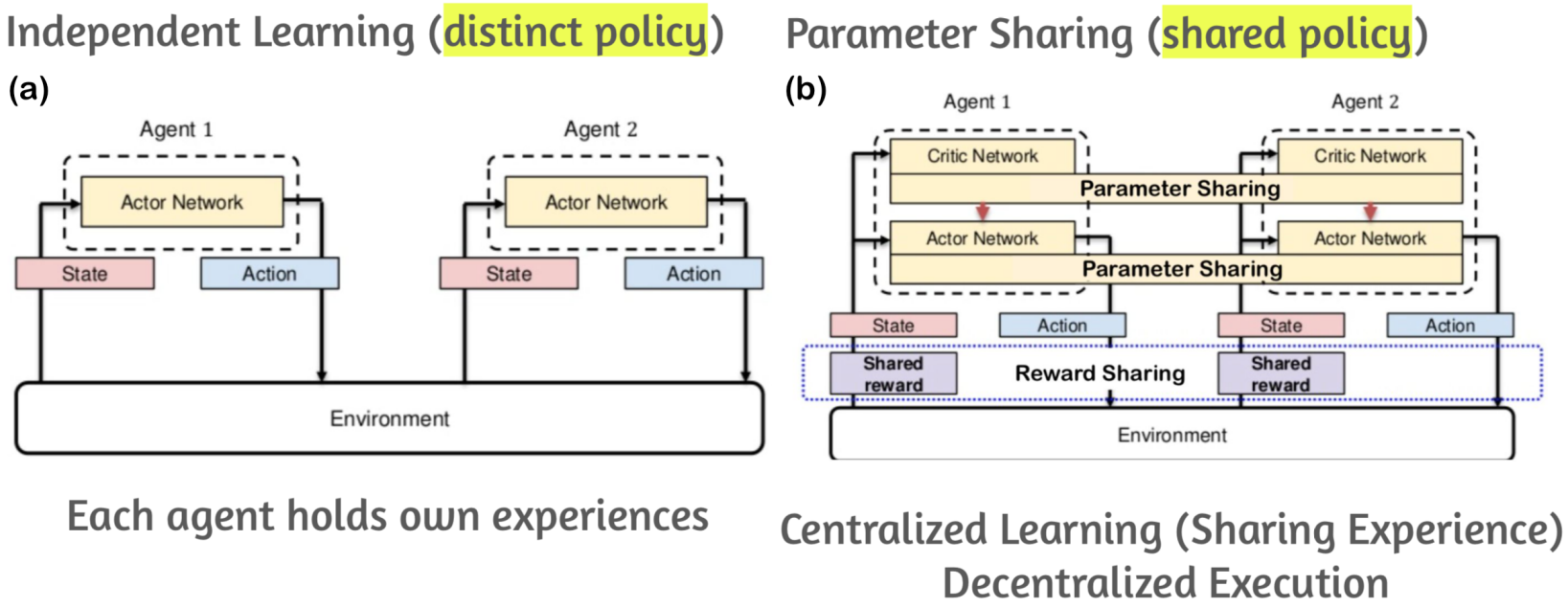


Figure 8: Two Extreme Cases of Multi-Agent Learning Strategies

CHAPTER 4

Experiments

A single walker environment is classified as a solved problem once the average score is achieved a certain threshold. There are benchmark studies with a single walker environment to compare which DRL algorithm achieves the maximum score in less number of trials. [19] One of state-of-art DRL algorithms, called Proximal Policy Optimization (PPO), outperforms especially for single walker-like environments. Thus, throughout this project, I will focus on the PPO algorithm. With the PPO, I will observe whether there exists a mutual transferability of the performance of PPO for single-agent vs. multi-agent settings depicted in Figure 9. All the DRL runs carried out in this project is tabulated in Table 7.

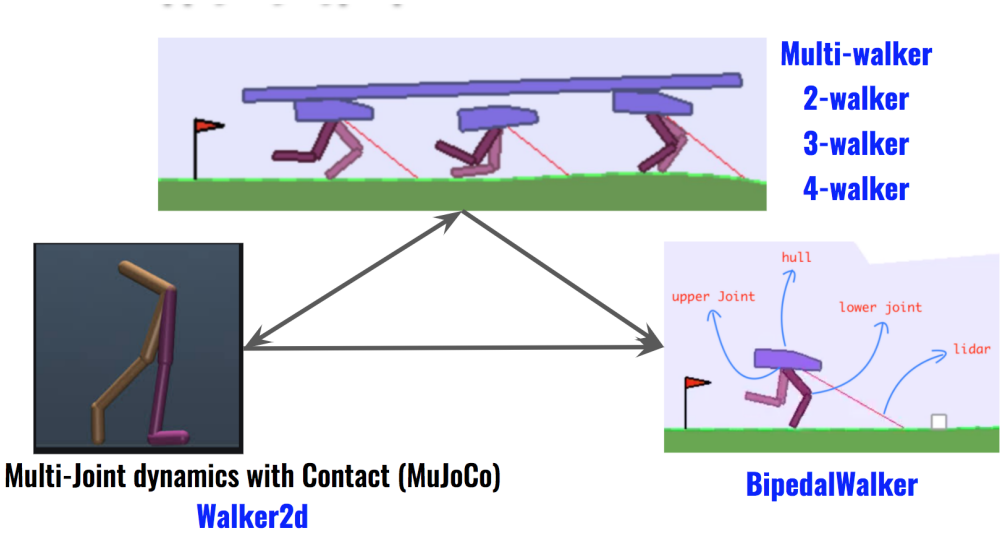


Figure 9: All Walkers Environment Considered in this Project

Table 7: Comprehensive List of DRL Experiments in This Work

Agent	• Single-PPO • Multi-PPOs ✓ Parameter Sharing ✓ Independent Learning
Environments	• Single-Walker ✓ Bipedal Walker ✓ A Package Carrying Bipedal Walker ✓ MoJoCo Walker2d • Multi-Walkers ✓ A Package Carrying 2-Walkers ✓ A Package Carrying 3-Walkers ✓ A Package Carrying 4-Walkers
Hyperparameters	• General DRL Training Related ✓ Reward Clipping ✓ Size of Hidden Layers in Neural Network • PPO Related ✓ Minibatch Size ✓ Experience Replay Buffer Size = Training Batch Size ✓ Reuse Ratio = $\left(\frac{\text{Training Batch Size}}{\text{Minbatch Size}}\right)$ ✓ Clipping Policy Coefficient ✓ Clipping Value-function Coefficient ✓ Kullback–Leibler (KL) divergence Initial Coefficient

4.1 Environments (Observation-space and Action-space)

Pre-defined components of single- and multi-agent walkers environments are tabulated in Table 8 and Table 9, respectively.

Table 8: Environment: Basis Single-Walker, BipedalWalker [14]

Reward function	- positive reward is given for moving forward to the right in slightly uneven terrain; - if the walker falls, reward of -100.
End goal	- walking forward to obtain total 300+ points in 1600 time steps
Walker configuraiton	- simple 4-joints walker robot with hulls, thighs and legs
State-space	- 24-vector space (see Figure 10a) - hull angle speed, angular velocity, horizontal speed, vertical speed, position of joints and joints angular speed, legs contact with ground, and 10 lidar range finder measurements
Action-space	- 4-vector space (see Figure 10b) - assigning $[-1, 1]$ value to torques and velocities
Episode termination	- when the walker's body touches ground; - the walker reaches far right side of the environment (reaching the goal)

Table 9: Environment: Multi-Walker Environment from PettingZoo [17]

Reward function	- positive reward is given to each walker proportional to the displacement of the package (that is 130 times the displacement of the walker's position); - if any walker or the package falls, reward of -100.
End goal	- moving a package on top of agents as far as possible to the right
Walker configuraiton	- replicate of BipedalWalker with a package on top of agent's hull
State-space	- each agent has 32-dimensional observation space - 24-dimensional vector from BipedalWalker - 4 additional vectors for replacements (dx, dy) of left; (dx, dy) of right of neighboring walkers; - if no neighbors, simply set it 0.0.
Action-space	- same as BipedalWalker
Episode termination	- if any walker or the package falls

(a)

Type: Box(24)

Num	Observation	Min	Max	Mean
0	hull_angle	0	2*pi	0.5
1	hull_angularVelocity	-inf	+inf	-
2	vel_x	-1	+1	-
3	vel_y	-1	+1	-
4	hip_joint_1_angle	-inf	+inf	-
5	hip_joint_1_speed	-inf	+inf	-
6	knee_joint_1_angle	-inf	+inf	-
7	knee_joint_1_speed	-inf	+inf	-
8	leg_1_ground_contact_flag	0	1	-
9	hip_joint_2_angle	-inf	+inf	-
10	hip_joint_2_speed	-inf	+inf	-
11	knee_joint_2_angle	-inf	+inf	-
12	knee_joint_2_speed	-inf	+inf	-
13	leg_2_ground_contact_flag	0	1	-
14-23	10 lidar readings	-inf	+inf	-

(b)

Num	Name	Min	Max
0	Hip_1 (Torque / Velocity)	-1	+1
1	Knee_1 (Torque / Velocity)	-1	+1
2	Hip_2 (Torque / Velocity)	-1	+1
3	Knee_2 (Torque / Velocity)	-1	+1

Figure 10: Continuous state-space (a) and action-space of for BipedalWalker environment (b), which serves as a basis unit for 2-, 3- and 4-walker environments

4.2 MADRL Framework and Agents/Policies

We use **RLlib** as a DRL framework [15]. It provides the Advantage Actor-Critic PPO algorithm that we discussed in Section 2.6 and shown in Figure 7. Moreover, many state-of-art DRL algorithms support for both single-agent and multi-agent settings in a form of python API.

For the single-agent PPO, RLlib has a module to register the OpenAI's gym environments for BipedalWalker as well as Walker2d environment.[14]

For the multi-agent PPO, RLlib provides an interface of PettingZoo, a collection of multi-agent environments [17]. We can thus use multi-walker environment by using the RLlib's `multiagent` module. However, we have to modify the PettingZoo's default 3-walker environment to create 1-, 2-, and 4-Walkers environments. Inside the `multiagent` module of RLlib, we can set configuration parameters either for parameter sharing or for independent learning. For the parameter sharing, we set the same shared policy ID (usually first agent's policy) to each of multi-agents. In contrast, for independent learning, we set distinct policy ID to each of multi-agents.

4.3 RLlib Installation and Resource Limit

RLlib is an open-source library for scalable DRL offering a unified API supports for TensorFlow and PyTorch. RLlib also offers high scalability with multi-agent settings. RLlib is a part of Ray. The Ray is an universal API for building distributed applications for machine learning. As of October 2020, the most up-to-date python3.7 wheel version (python package distribution), `Ray v1.1.0dev0` [16] was installed on the Conda (virtual environment management system) with Python 3.7 of the MacBook Pro. System specification are following: macOS High Sierra

version 10.13.6; dual core 2.9 GHz Intel Core i7; 8 GB 1600 MHz DDR3 memory.

Note that two cores with a hyper-thread enabled serve a quad core, thus 4-Walkers was the maximum environment we can run. Also, due to 8 GB memory limit, we could not complete the independent learning MADRL runs for the 4-Walkers. Full installation procedure is listed in Appendix A.

CHAPTER 5

Results & Discussion

5.1 Reward Shaping is Crucial for Proper Goal Setting in DRL

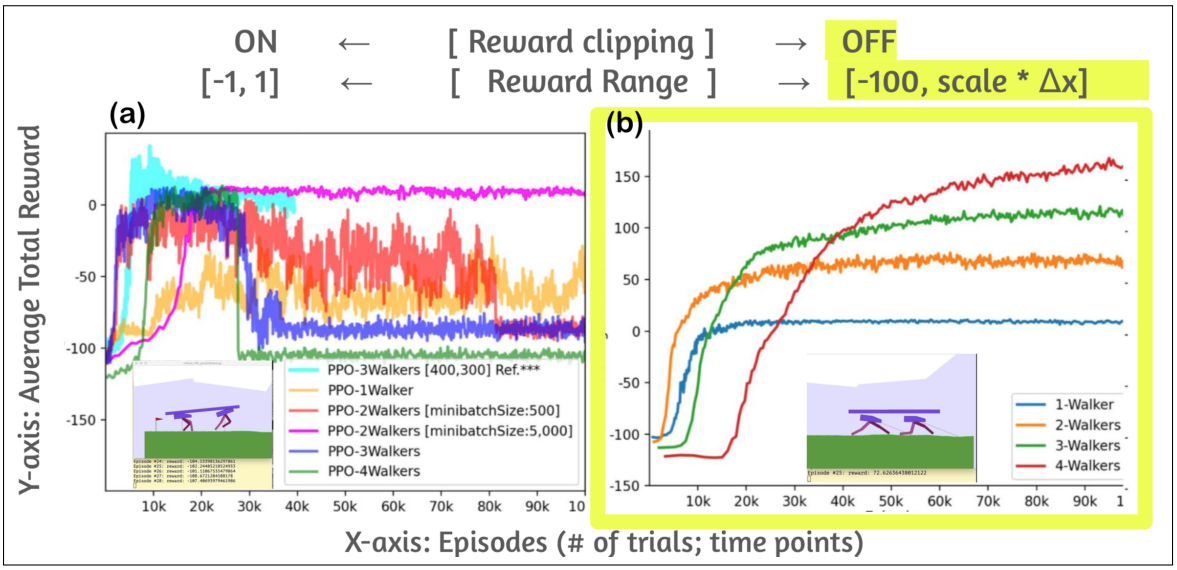


Figure 11: Effect of reward clipping on the performance of 2-, 3-, and 4-walkers environments with reward clipping (a) vs without reward clipping (b). Note that for (b) we changed other hyperparameters but turning off the reward clipping was the dominant parameter to scale up the values (y-axis). In (a) the ''PPO 3-walkers with $[400, 300]$ Ref***'' refers to the result from [12], they used reward clipping.

Observation.

Initially we used the tuned parameters set (see Figure 13a) for the 3-walkers environment from the work by Terry et al. [12] One of parameter was `clip_rewards=True`. In general, reward clipping is used only when comparing performances among many environments with various reward range. Reward clipping was designed to normalize a reward range $[-1, 1]$ by taking a sign of the value-function (e.g. `np.sign(values)` in Numpy).

As observed in Figure 11a), reward clipping suppresses the improvement of performance, its maximum mean reward is less than 10 for all 1-, 2-, 3-, and 4-walkers environments.

Even if a walker fell or a package was fallen, a minimum reward of -1 was assigned to the corresponding action, which doesn't motivate standing or walking. If we observed from the roll-out simulation, indeed walkers tend to jump and roll frequently instead of standing and moving forward.

As observed in Figure 11b), turning off the reward clipping restore the intended reward function, the mean rewards keep increased up to the maximum mean rewards of ~ 50, 100, and 150 for 2-, 3-, and 4-walkers, respectively.

Intended reward function was assigning a proper reward ranging from -100 to 1 proportional to the displacement of a package location. If walker fell or a package was fallen, the corresponding action should be assigned -100, which demotivates falling rather motivates walking and moving forward.

Although we should not use reward clipping for performance comparison of walkers environments, two interesting observations using reward clipping were 1) difficulty of convergence in loss function during training indicates that the reward signal from environment is not corrected reflected; 2) suppressed mean reward range revealed subtle effect of hyperparameters on the performance, otherwise its effect was diluted in the large mean reward values.

Interpretation & Supporting Information.

Reward-function is pre-defined as an environment specific manner, which can't be controlled by an agent. Agent can only received the emitted reward signal from environment and select an optimal sequence of action to maximize the accumulated reward.

Designing reward function (also called Credit Assignment) is one of research fields in DRL [18]. As observed in our cases with reward clipping the goal task is rather balancing a package while standing at the same location, wheres without reward clipping the goal task is restored to walking and moving forward. Therefore, depending on the reward shaping, a goal task of DRL can be altered.

5.2 Larger Capacity of Neural Network is Better for High-Dimensional, Continuous Observation Space

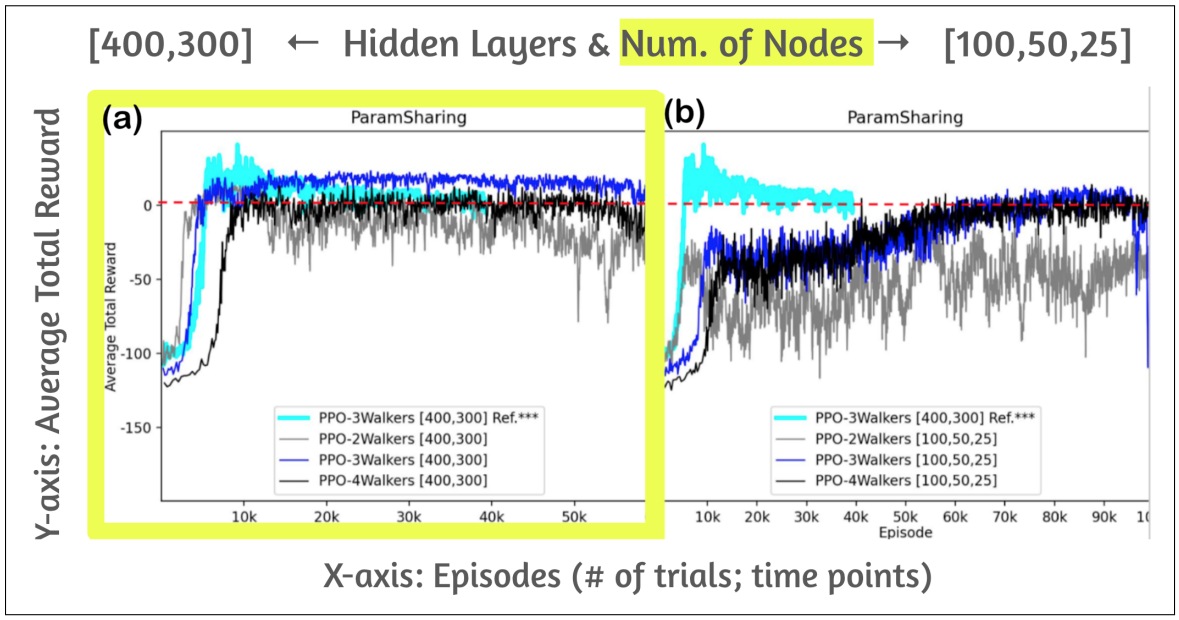


Figure 12: Effect of NN capacity on the performance of PPO. Note that the ''PPO 3-walkers with $[400, 300]$ Ref***'' refers to the result from [12].

Observation.

The mean reward from PPO with larger capacity of NN with larger number of nodes (e.g. $[400, 300]$) shows positive (Figure 12a), whereas that with smaller capacity of NN with more number of layers (e.g. $[100, 50, 25]$) shows smaller values moreover very noisy.

Interpretation & Supporting Information.

Due to high dimensional continuous state/action-space in the 2-, 3-, 4-walkers environments, the larger size of NN has better capability of differentiating from state to state.

Henderson et al. benchmark general NN training parameters (such as NN capacity, NN architecture, activation function, optimizer etc.) and specific hyper-parameters of many DRL algorithms (Deep Q-learning, A2C, PPO, etc.) for the frequently used DRL single-agent environments. [19]

NN capacity is one of hyperparameters and optimal capacity depends on the environment whether discrete state/action-space or continuous, high-dimensional actions-space. Unless our environment has small scale state/action-space, it would be safe to use relatively larger capacity such as 2-hidden layer with size of $[400, 300]$.

DRL suffers non-stationary due to lack of supervised target label. To mitigate that non-stationary problem, two strategies are suggested by using 1) a target network for temporary stationary and 2) a large enough network for telling a difference between similar states due to correlation among generated experience data. [1]

5.3 Optimal Minibatch Size and Sampling Reuse Ratio are Important Hyperparameters of PPO to Improve the Performance

(a)

RL method	Hyperparameter	Value for Pursuit / Waterworld / Multiwalker
PPO	`sample_batch_size`	100
	`train_batch_size`	5000
	`sgd_minibatch_size`	500
	`lambda`	0.95
	`kl_coeff`	0.5
	`entropy_coeff`	0.01
	`num_sgd_iter`	10
	`vf_clip_param`	10.0
	`clip_param`	0.1
	`vf_share_layers`	`True`
	`clip_rewards`	`True`
	`batch_mode`	`truncate_episodes`

(b)

Hyperparameter	**Value**
Timesteps per batch	320000
SGD minibatch size	32768
SGD epochs per iteration	20
Adam stepsize	1e-4
PPO clip param	0.2
GAE parameter (λ)	0.95
Discount (γ)	0.995

Figure 13: PPO hyperparameter set from other works: Ref. [12] for 3-Walker multi-agent environment (a) and from Ref. [15] for MuJoCo Walker2d single-agent environment (b)

Observation.

Figure 13a) and b) show the tuned hyperparameter set of PPO for 3-walkers and single-walker, respectively, from other researchers. There are several differences to be aware of: 1) The former is multi-agent environment, whereas the latter is single-agent; 2) The former uses multi-core CPUs, whereas the latter used multi-GPUs; 3) Besides a problematic hyperparameter of `clip_rewards=True` as already

discussed in Section 5.1, we notice rather drastic differences in their minibatch size as well as train batch size; 4) But the ratio between train batch size over minibatch size is approximately similar to 10.

Given their computing system difference (CPU vs GPU), they may find a different tuned use different sizes. Direct use of their tuned hyperparameters to our environment is not transferable. Therefore, we should tune minibatch and train batch size optimal to our computing system (4-core CPUs).

Figure 14 shows that the tuned hyperparameter set (5K minibatch; 50K train batch) outperform for all 2-, 3-, and 4-walkers environments. While fixing the

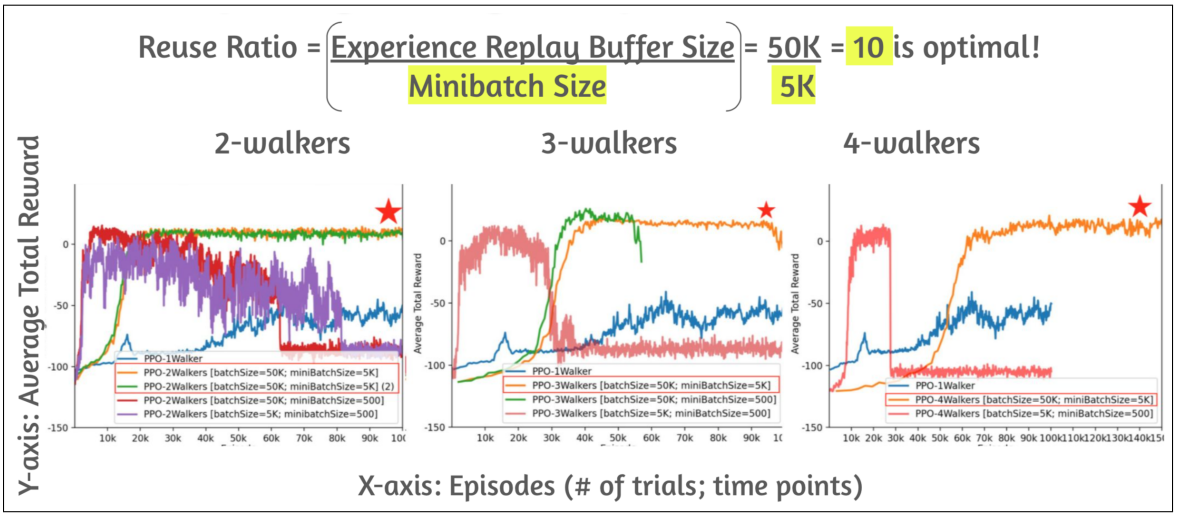


Figure 14: Larger minibatch size of 5K improves performance of PPO by providing assorted gradients for better value estimate

experience replay buffer size at 50K, we change minibatch size to experiment with different sampling reuse ratio. Figure 15 shows that more sampling reuse of 10 improve performance of PPO because value-function needs more number of optimization.

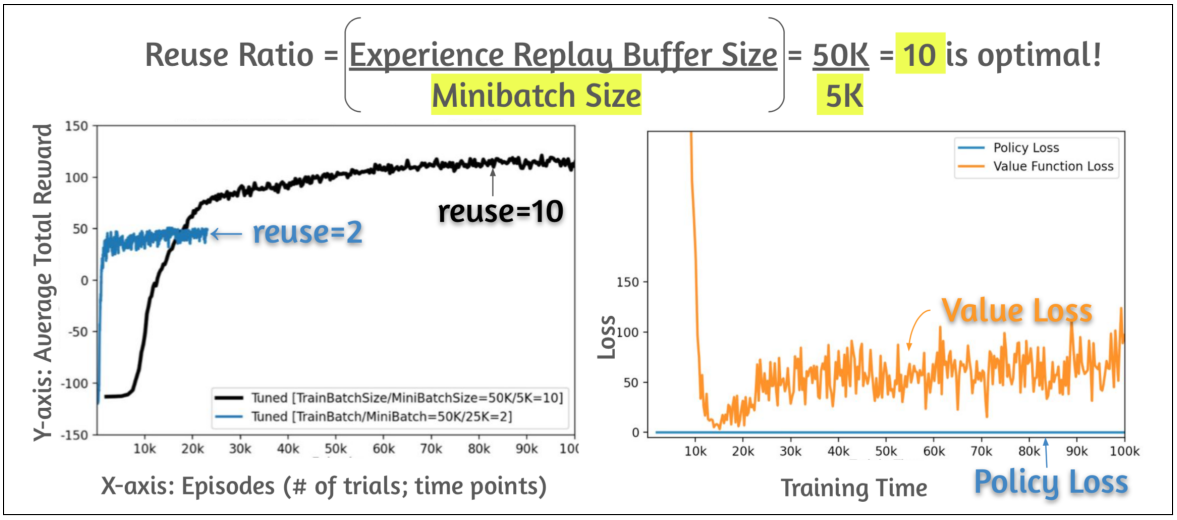


Figure 15: Effect of varying reuse ratio on the PPO performance for 3-walkers environment (left); corresponding training loss of policy-gradient vs. value-function (right)

Interpretation & Supporting Information.

In DL, there is some consensus that smaller batch size is better for generalization than larger batch size, perhaps owing to the supervised learning, thus free from non-stationary issue unlike DRL. DRL is usually suffers non-stationary, known as ''Moving Target'' problem. Hence, a large enough train batch is needed to compute more various gradient descent, thus more frequent update from minibatch size stochastic gradient descent (SGD) for better optimization. [22, 23]

Cobbe et al. [20] improved version of PPO, called 'Phasic Policy Gradient' (PPG): 1) separating the shared network of PPO into policy-network and value-network; 2) allowing the policy-network and value-network are trained with different reuse ratio from minibatch/train-batch; 3) improved performance than PPO from diverse single-agent environment using larger reuse ratio (9) for the value-network and smaller reuse ratio (1) for the policy-network.

This is consistent with our multi-agent walkers environment: 1) we found

that reuse ratio of 10 outperformed than 2; given that PPO is shared network, thus we can only use same reuse ratio for both policy-network and value-network. this implies that policy-part of network converges faster within 2 updates, the value-part of network converges slowly, thus need more update like 10. 2) as shown in the inset graph in Figure 15 we confirmed our conjecture from the almost negligible loss of policy-gradient, whereas yet noisy loss of value-function from our training log. In short, as depicted in Actor-Critic PPO algorithm in Figure 16, DRL

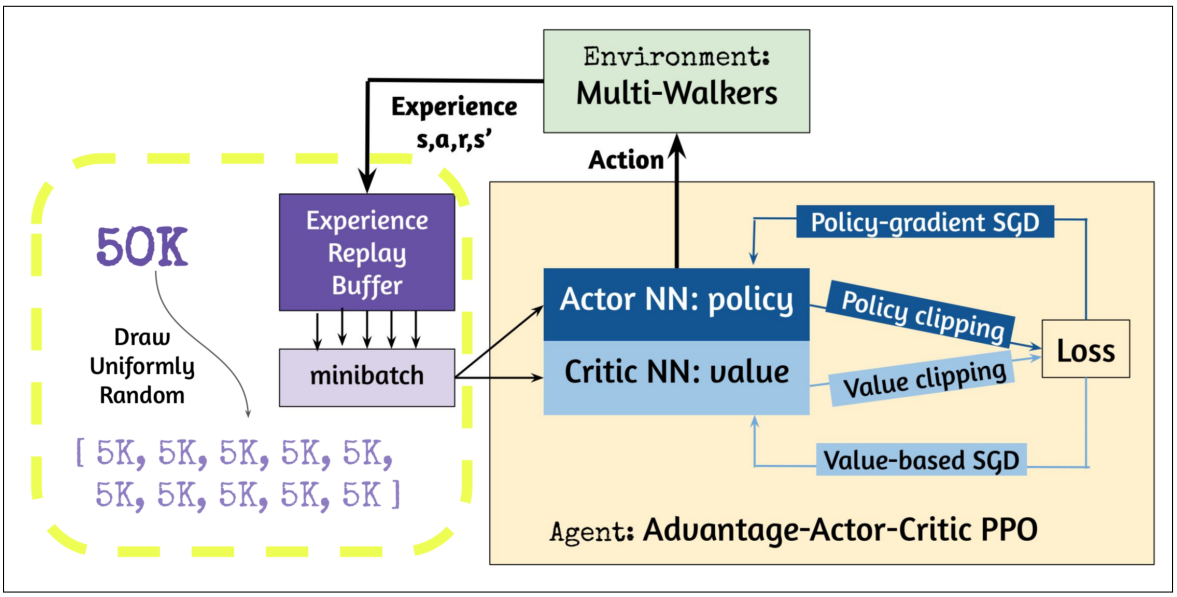


Figure 16: PPO Algorithm with Optimized Experience Replay Buffer Size and Minibatch Size

may need larger sizes of train batch and minibatch that SL approach of DL. Thus, minibatch size together with its reuse ratio is an important hyperparameter specific for the environment as well as computing system. Also, as already discussed in Section 2.6, using a shared single network for both policy and value functions (feature of A2C algorithm, which is one of components of the PPO) is beneficial for computational efficiency, however, at a potential risk of different scales of the policy and value function updates.

5.4 Synergic Effect of Combined Optimal Hyperparameters

Observation.

Together with the tuned optimal minibatch size and train batch size discussed in Section 5.3, we tried to use other tuned PPO hyperparameters from the single-agent environment (Walker2d) from benchmark work. [19]

Figure 17 shows 7 different runs of PPO using different hyperparameter set from less performed sub-optimal set (gray solid line) up to well performed optimal set (black solid line). We found that the contribution to the performance is significant in this order: 1) Turning off the reward clipping - this is rather general training control of DRL than policy algorithm (see detailed discussion in Section 5.1); 2) Minibatch size of 5K (see detailed discussion in Section 5.3); 3) Clipping policy coefficient of 0.3 than 0.1 (less strict constraint, thus allows more diverse gradient); and 4) KL divergence initial coefficient. However, the best performance of PPO for the multi-agent environments is achieved by combined contribution of those hyperparameters.

Interpretation & Supporting Information.

By increasing the clipping policy coefficient ϵ to 0.3 from 0.1 (denoted as red color), we can select new policy little bit more deviated from the old policy providing more diverse gradients. This can be observed by red and green curve in b) where the mean reward is increased earlier with ϵ to 0.3 than 0.1 (i.e curve is shifted left).

$$J(\phi, \phi^{-}) = \underset{(s,a,A^{GAE})}{\mathbb{E}}\Bigg[\min\Bigg(\frac{\pi(a \mid s;\phi^{-})}{\pi(a \mid s;\phi)}\, A^{GAE}, \quad \text{clamp}\Bigg(\frac{\pi(a \mid s;\phi^{-})}{\pi(a \mid s;\phi)}\, A^{GAE}; 1-\epsilon; 1+\epsilon\Bigg)\Bigg)\Bigg] \tag{20}$$

The increased ϵ together with the larger minibatch size of 5K provides well

(a)

Comparison of PPO Hyperparameters

PPO default	Walker2d	Hyperparameters	{2,3,4}-Walker
0.3	default	clip_param	0.1
0.2	1.0	kl_coeff	0.5
30	20	num_sgd_iter	10
0.0	default	entropy_coeff	0.01
128	33K	sgd_minibatch_size	5K
4K	320K	train_batch_size	50K

(b)

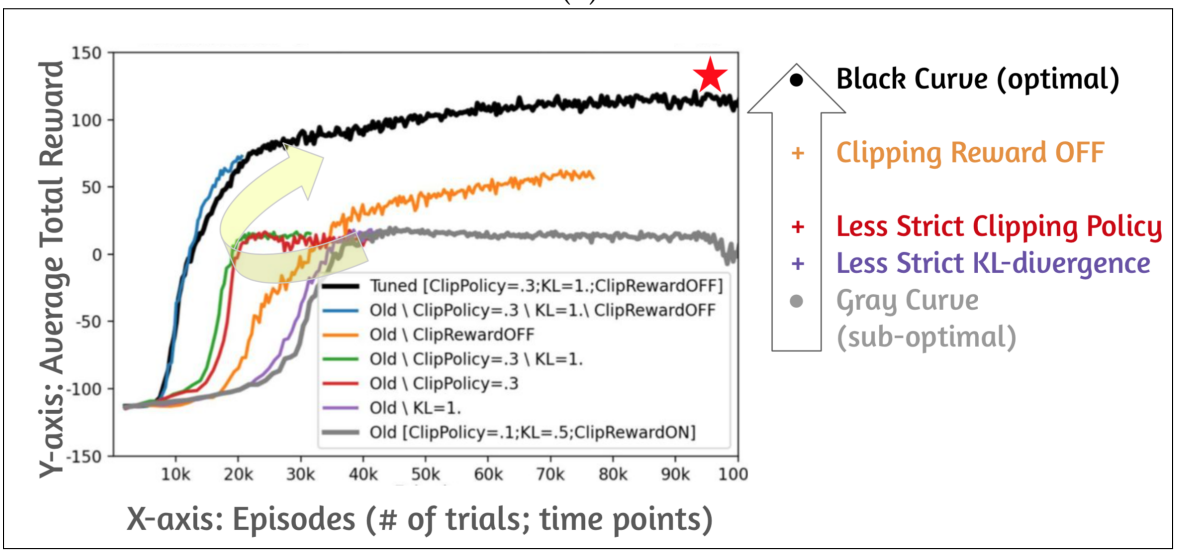


Figure 17: Comparison of PPO hyperparameter set between single-agent MuJoCo Walker2d vs. our sub-optimal hyperparameter set for multi-agent walkers (a); Finding the most contributed hyperparameter to the performance for the 3-walker environment by changing the parameter one at a time (b)

distributed samples the experience replay buffer, which in turn resulting in decent performance improvement. Moreover, A^{GAE}, the advantage function estimated by Generalized Advantage Estimator (GAE) from the experience buffer, reduces variance (i.e. noise in DRL) resulting in better performance of PPO.

5.5 Mutual Transfer of Tuned Hyperparameters between Single-Agent (SA) and Multi-Agent (MA) Environments

5.5.1 Transfer hyperparameters from MA to SA environment

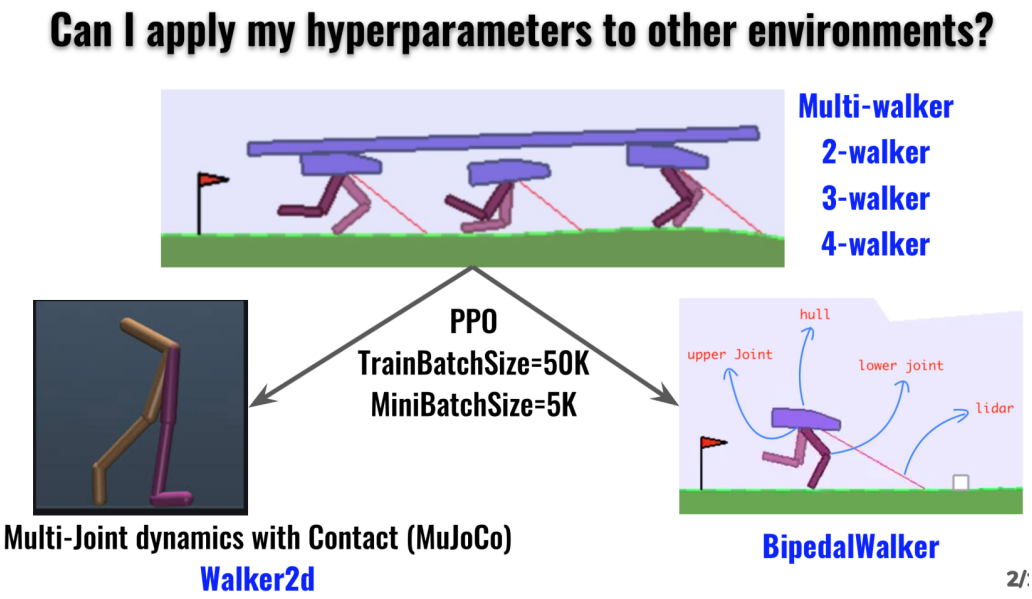
In Section 5.3, we found the optimal minibatch/train-bath size of 5K/50K from 2-, 3-, and 4-walkers environment. We try to transfer this hyperparameters to similar but single-agent walker environment. As demonstrated in Figure 18b) and c), hyperparameters from MA is fully transferable to SA for both Walker2d (little bit different legs configuration, where there are feet but no hull) and BipedalWalker (identical legs configuration to multi-walkers but without carrying a package on top).

5.5.2 Transfer hyperparameters from SA to MA environment

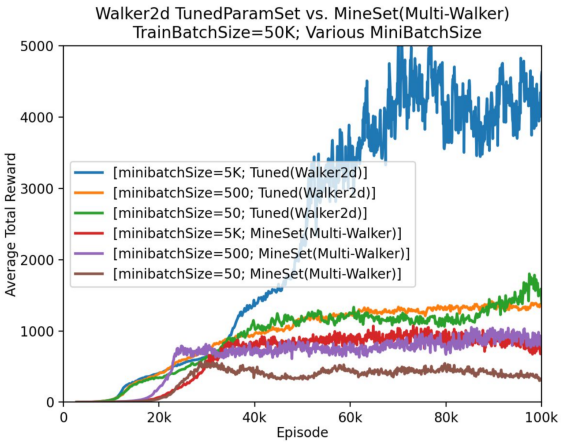
As was already discussed already in Section 5.4 hyperparameters are compared between single-agent Walker2d and multi-walkers in Figure 17a). Tuned hyperparameter set of Walker2d with minibatch/train-bath size of 5K/50K can be fully transferred to all 2-, 3-, and 4-walkers environments, i.e. dramatically improved performance of PPO (Figure 19b-d).

To conclude, there is a clear mutual transferability between MA and SA as long as their environment are similar. It seem that little difference in the configuration of the environments (such as with or without feet, or with a package or not) are tolerable and still transferable the tuned hyperparameters.

(a)



(b)



(c)

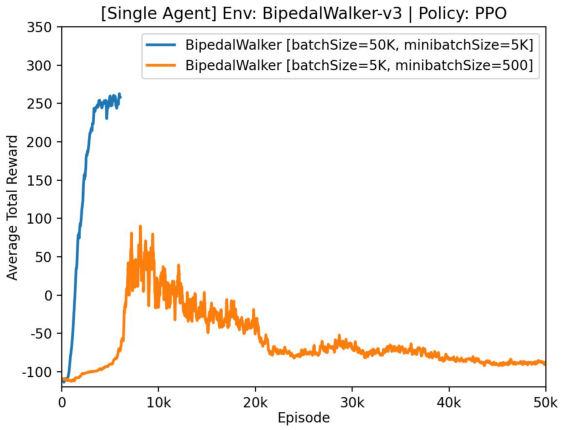
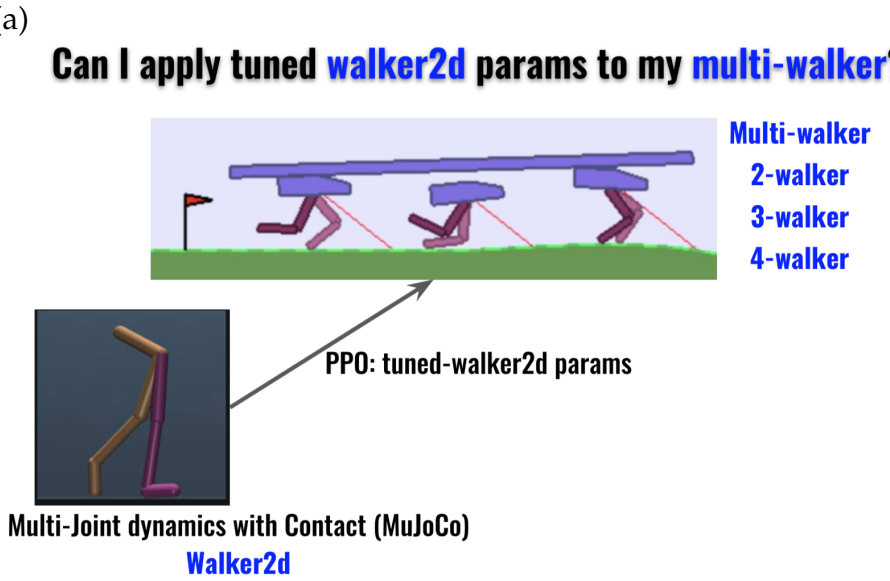
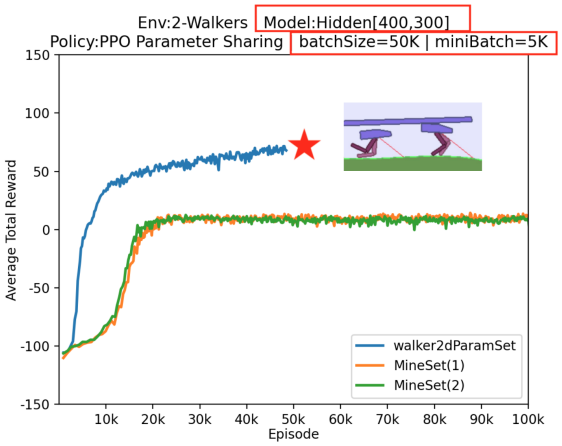


Figure 18: Hyperparameter transfer flow from MA to SA (a); Resulting improvement of performance in single-agent environment: Walker2d (b) and BipedalWalker (c)

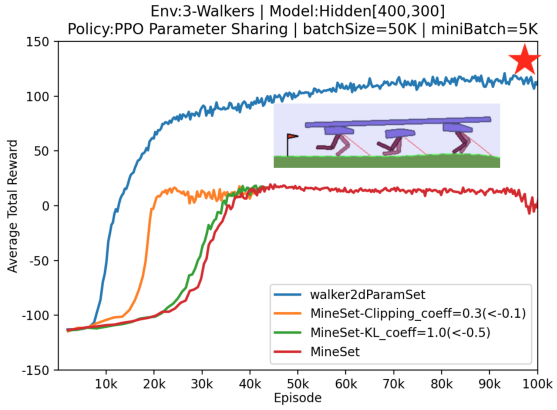
(a)



(b)



(c)



(d)

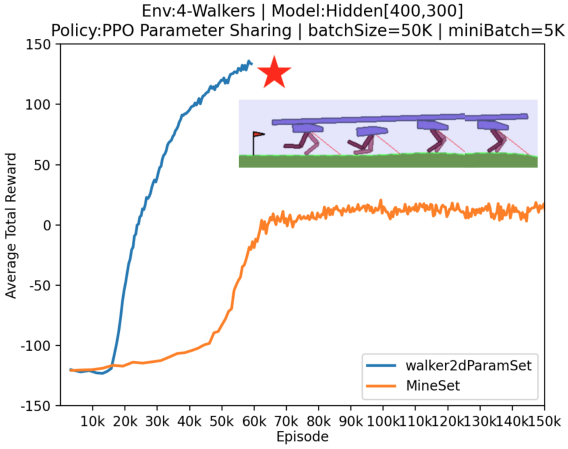


Figure 19: Hyperparameter transfer flow from SA to MA (a); Resulting improvement of performance for 2, 3, and 4-walkers environment in (b) through (d)

5.6 Parameter Sharing vs. Independent Learning in Multi-Agent DRL

Observation.

As observed in Figure 20, performance-wise (mean reward on y-axis) there is not much difference between parameter sharing and independent learning, albeit maximum average reward is little bit higher in independent learning (Table 11).

In parameter sharing multi-agents, all PPO runs for 2-, 3- and 4-walkers were all completed up to 10K episode. However in independent learning multi-agent, we could not complete the PPO run for the 4-walkers environment due to out of memory capacity issue. Thus, it is clear that parameter sharing is scalable and efficient strategy for the multi-agent DRL without any sacrifice of performance.

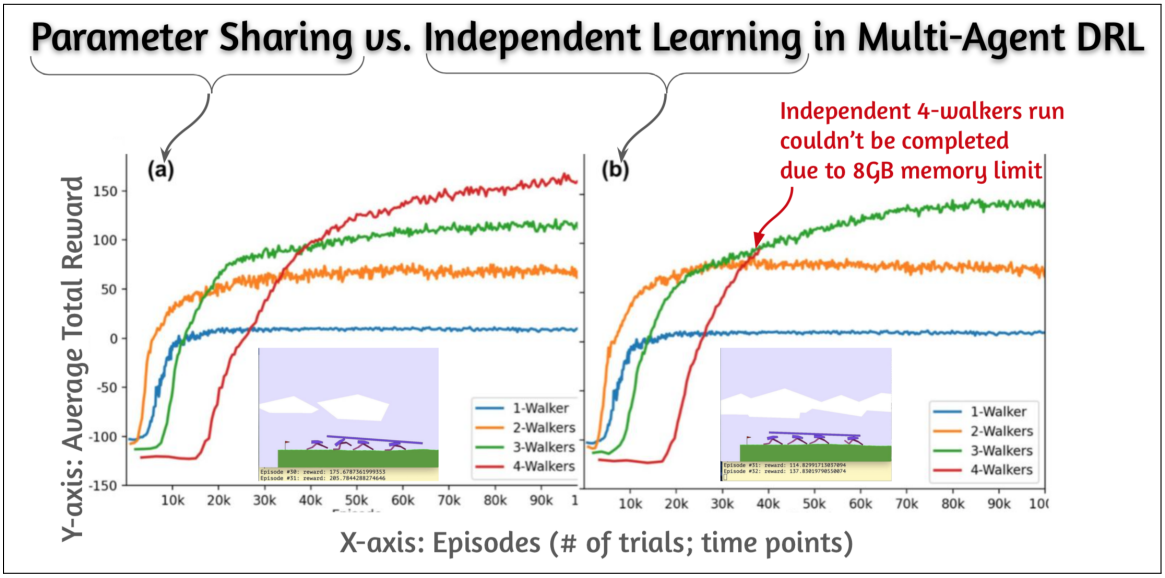


Figure 20: Comparison of Multi-Agent Learning Strategies

Interpretation & Supporting Information.

Before finding the optimal tuned hyperparameters, the PPO runs for the multi-walker environment showed quite different performance between parameter shar-

ing and independent leaning. However, such difference was vanished after applying to the optimal hyperparameters to the PPO runs. Probably such performance difference indicates difficulty of convergence of training (thus required tuning the hyperparameters) rather than behavioral difference of parameter sharing vs. independent learning.

To sum up, we found the optimal hyperparameter set for the multi-agent PPO algorithm, which improve performance much better than other studies as tabulated in Table 10.

Table 10: Comparison of Performance for the 3-Walkers Environment with Other Studies

Ref	Agent	Max Avg. Reward (Independent Learning)	Max Avg. Reward (Parameter Sharing)
Gupta et al. [11]	TRPO	51	54
Terry et al. [12]	PPO	38	41
This work	PPO	145	121

Finally, we tabulate maximum average reward achieved from our PPO-based MADRL runs for multi-walkers shown in Table 11. There is a clear scalability of performance for the parameter sharing strategy. Yet, we still need to further investigation on the scalability of performance in independent learning if more memory resource is available.

Table 11: Maximum Average Reward for Multi-Walkers Achieved in This Study in 100K Episodes (except independent 4-walkers, terminated at 38K)

Environment	Max Avg. Reward (Independent Learning)	Max Avg. Reward (Parameter Sharing)
1-Walker	11	--
2-Walkers	87	76
3-Walkers	145	121
4-Walkers	94	168

CHAPTER 6

Conclusion & Future Work

Based on the findings from our MADRL experiments for the multi-agent walkers using PPO algorithm, we can generalize three concepts.

First, Unlike DL with explicit target labels, DRL needs larger minibatch size better estimate of values from various gradients. Therefore, minibatch size together with experience replay buffer size are critical hyperparameters in PPO algorithm.

Second, for the homogeneous multi-agent environments, there is a mutual transferability between single-agent and multi-agent environments to be able to share the tuned hyperparameters.

Third, for the homogeneous multi-agent environments trained with a well tuned hyperparameters, the parameter sharing is a better strategy for the MADRL in terms of performance and efficiency with reduced parameters and less memory.

To reconfirm the findings with other cases as well as to refine the multi-agent algorithms, we can think of the following four tasks as a future study.

The first study as one immediate next task will be a scalability check for the parameter sharing strategy of the MADRL by increasing the number of walkers up to 8 to 10.

The second study as another immediate next task will be a new collection of DRL environments with a similar end-goal from both single-agent and multi-agent settings. Mutual hyperparameter transfer will be attempt to the newly collected environments to determine a new set of tuned hyperparameter set.

The third study as a short-term goal will be a dynamic/adaptive hyperparameter change to enhance performance (opposed to the fixed hyperparameters

throughout the DRL runs used in this work) by Curriculum Learning (CL). CL decomposes a task into several sub-tasks as pre-defined phases, monitors performance metrics regularly and moves on to the next phase if the pre-defined condition is met.

The fourth study as a long-term goal will be an *explicit communication* among multi-agents. In this MADRL study, we assumed that multi-agents cooperate via *implicit communication* by observing the existence of other agents as a part of the shared environment. Very recently, there are more advanced studies on the MADRL to combine Game Theory based Nash Equilibrium and explicit communication among agents. [26, 28, 27]) We will set this as a long-term future goal to better understand MADRL toward Artificial General Intelligence (AGI).

To conclude, reward-driven, sequential and evaluative learning, the DRL, would be closer to AGI if multiple DRL agents learn to collaborate to capture the true signal from the shared environment. This work provides one instance of implicit cooperative learning of MADRL.

LIST OF REFERENCES

[1] M. Morales, **Grokking Deep Reinforcement Learning**, Shelter Island, NY, USA: Manning Publications, 2020.

[2] W. Schultz, **Neuronal Reward and Decision Signals: From Theories to Data**, Physiological Reviews, vol. 95, 3, 853-951, 2015.

[3] A. Senior, R. Evans, J. Jumper et al. **Improved protein structure prediction using potentials from deep learning**, Nature 577, 706–710, 2020.

[4] C. Liu, C. Chang and C. Tseng, **Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems** in IEEE Access, vol. 8, 71752-71762, 2020.

[5] R. Li et al. **Deep Reinforcement Learning for Resource Management in Network Slicing** in IEEE Access, vol. 6, 74429-74441, 2018.

[6] **Bullet Real-Time Physics Simulation**, Accessed on: Nov, 2020. [Online]. Available: `https://pybullet.org/wordpress/`

[7] **Roboschool**, Accessed on: Nov, 2020. [Online]. Available: https://openai.com/blog/roboschool/

[8] R. Sutton and A. Barto, **Reinforcement Learning: An Introduction**, 2nd ed. Cambridge, MA, USA: The MIT Press, 2018.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, **Proximal Policy Optimization Algorithms**, CoRR abs/1707.06347, 2017.

[10] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel and I. Mordatch. **Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments** in Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17), Curran Associates Inc., Red Hook, NY, USA, 6382–6393, 2017

[11] J. Gupta, M. Egorov and M. Kochenderfer, **Cooperative Multi-agent Control Using Deep Reinforcement Learning** in Lecture Notes in Computer Science, vol. 10642, Autonomous Agents and Multiagent Systems. G. Sukthankar and J. Rodriguez-Aguilar Ed. Springer, Cham. 2017

[12] J. Terry, N. Grammel, A. Hari, L. Santos and B. Black, **Revisiting Parameter Sharing In Multi-Agent Deep Reinforcement Learning**, CoRR abs/2005.13625, 2020

[13] E. Liang, R. Liaw, **Scaling Multi-Agent Reinforcement Learning**, Accessed on: Nov, 2020. [Online]. Available: `https://bair.berkeley.edu/blog/2018/12/12/rllib/`

[14] **BipedalWalker-v2**, Accessed on: Nov, 2020. [Online]. Available: `https://gym.openai.com/envs/BipedalWalker-v2/`

[15] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan and I. Stoica, **RLlib: Abstractions for Distributed Reinforcement Learning** in Proceedings of the 35th International Conference on Machine Learning, vol. 80, 3053-3062, 2018.

[16] **Ray v1.1.0dev0 Documentation: RLlib: Scalable Reinforcement Learning**, Accessed on: Nov, 2020. [Online]. Available: `https://docs.ray.io/en/master/rllib.html`

[17] **PettingZoo: Multi-Agent Reinforcement Learning Environments**, Accessed on: Nov, 2020. [Online]. Available: `https://www.pettingzoo.ml`

[18] A. Ng, D. Harada and S. Russell, **Policy invariance under reward transformations: theory and application to reward shaping** in Proceedings of the Sixteenth International Conference on Machine Learning, Bled, Slovenia, 1999.

[19] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup and D. Meger, **Deep Reinforcement Learning that Matters**, CoRR abs/1709.06560, 2019.

[20] K. Cobbe, J. Hilton, O. Klimov and J. Schulman, **Phasic Policy Gradient**, CoRR abs/2009.04416, 2020.

[21] J. Schulman, P. Moritz, S. Levine, M. Jordan and P. Abbeel, **High-Dimensional Continuous Control Using Generalized Advantage Estimation**, in 4th International Conference on Learning Representations (ICLR 2016). San Juan, Puerto Rico, May 2-4, 2016.

[22] S. Zhang and R. Sutton, **A Deeper Look at Experience Replay**, CoRR abs/1712.01275, 2020.

[23] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland and W. Dabney, **Revisiting Fundamentals of Experience Replay**, CoRR abs/2007.06700, 2020.

[24] E. Bengio, J. Pineau and D. Precup, **Interference and Generalization in Temporal Difference Learning**, CoRR abs/2003.06350, 2020.

[25] T. Schaul, D. Borsa, J. Modayil and R. Pascanu, **Ray Interference: a Source of Plateaus in Deep Reinforcement Learning**, CoRR abs/1904.11455, 2019.

[26] Y. Lu and Y. Kai, **Algorithms in Multi-Agent Systems: A Holistic Perspective from Reinforcement Learning and Game Theory**, CoRR abs/2001.06487, 2020.

[27] A. Nowé, P. Vrancx and Y.M. De Hauwere, **Game Theory and Multi-agent Reinforcement Learning** in Reinforcement Learning. Adaptation, Learning, and Optimization, vol. 12, M. Wiering and M. van Otterlo Ed. Springer, Berlin, Heidelberg. 2012.

[28] J. Foerster, **Deep Multi-Agent Reinforcement Learning**, PhD Thesis, University of Oxford, Oxford, UK, 2018 [Online]. Available: `https://ora.ox.ac.uk/objects/uuid:a55621b3-53c0-4e1b-ad1c-92438b57ffa4`

APPENDIX

RLlib Installation on MacOS

Procedure

1. conda create -name LocalRay python=3.7
2. mkdir LocalRay
3. cd LocalRay/
4. git clone https://github.com/ray-project/ray.git
5. conda activate LocalRay
6. mv /Downloads/ray-1.1.0.dev0-cp37-cp37m-macosx_10_13_intel.whl .
7. pip install -U setuptools
8. pip3 install multidict==4.7.3
9. conda install -c anaconda psutil
10. conda install -c anaconda yarl
11. pip install ray-1.1.0.dev0-cp37-cp37m-macosx_10_13_intel.whl
12. pip install pandas
13. pip install ray[tune]
14. pip install ray[rllib]
15. pip install pettingzoo[sisl]
16. pip install tensorflow