Deep Multi-Agent Reinforcement Learning with Discrete-Continuous Hybrid Action Spaces

Haotian Fu[1] , **Hongyao Tang**[1] , **Jianye Hao**[1*] , **Zihan Lei**[2] , **Yingfeng Chen**[2] , **Changjie Fan**[2]
[1]College of Intelligence and Computing, Tianjin University
[2]Fuxi AI Lab in Netease
{haotianfu, bluecontra, jianye.hao}@tju.edu.cn,
{leizihan, chenyingfeng1, fanchangjie}@corp.netease.com

Abstract

Deep Reinforcement Learning (DRL) has been applied to address a variety of cooperative multi-agent problems with either discrete action spaces or continuous action spaces. However, to the best of our knowledge, no previous work has ever succeeded in applying DRL to multi-agent problems with discrete-continuous hybrid (or parameterized) action spaces which is very common in practice. Our work fills this gap by proposing two novel algorithms: Deep Multi-Agent Parameterized Q-Networks (Deep MAPQN) and Deep Multi-Agent Hierarchical Hybrid Q-Networks (Deep MAHHQN). We follow the centralized training but decentralized execution paradigm: different levels of communication between different agents are used to facilitate the training process, while each agent executes its policy independently based on local observations during execution. Our empirical results on several challenging tasks (simulated RoboCup Soccer and game Ghost Story) show that both Deep MAPQN and Deep MAHHQN are effective and significantly outperform existing independent deep parameterized Q-learning method.

1 Introduction

Reinforcement learning (RL) has recently shown a great success on a variety of cooperative multi-agent problems, such as multiplayer games [Peng *et al.*, 2017], autonomous cars [Cao *et al.*, 2013] and network packet delivery [Ye *et al.*, 2015].

In many such settings, it is necessary for agents to learn decentralized policies due to the partial observability and limited communication. Fortunately, we can learn such policies using the paradigm of centralized training and decentralized execution. Forester et al. [2016] developed a decentralized multi-agent policy gradient algorithm; Lowe et al. [2017] extended DDPG [Lillicrap *et al.*, 2016] to multi-agent setting with a centralized Q-function; Rashid et al. [2018] employs a Qmix network that estimates joint action-values as a complex non-linear combination of per-agent action values that condition on local observations only.

These popular multi-agent reinforcement learning methods all require the action space to be either discrete or continuous. However, very often the action space in real world is discrete-continuous hybrid, such as Robot soccer [Hausknecht, 2016; Masson *et al.*, 2016] and Real Time Strategic games [Xiong *et al.*, 2018]. In such settings, each agent usually needs to choose a discrete action and continuous parameters associated with it at each time step. An obvious approach to handling this is to simply approximate hybrid action spaces by a discrete set or relax it into a continuous set [Hausknecht and Stone, 2016]. However, such approaches suffer from a number of limitations. Discrete approximation needs to balance the trade-off between learned policy quality and discrete action space explosion. If the continuous action space is discretized coarsely, the learned policy quality would decrease significantly; otherwise, the discrete action space would increase exponentially. If using continuous relaxation, it would be difficult to map the learned optimal actions in the relaxed continuous space to the feasible and optimal discrete actions.

An alternative and better solution is to learn directly over hybrid action spaces. Following this direction, Xiong et al. [2018] proposed Parameterized Deep Q-Network (P-DQN) for single-agent learning in hybrid action spaces without approximation or relaxation by seamlessly integrating DQN [Mnih *et al.*, 2013] and DDPG [Lillicrap *et al.*, 2016]. However P-DQN cannot be directly applied to multi-agent settings due to the non-stationary property in multi-agent environments. In multi-agent settings, explicit coordination mechanism among agents' hybrid action spaces needs to be introduced.

In this work, we propose two approaches to address cooperative multi-agent problems in discrete-continuous hybrid action spaces based on centralized training and decentralized execution framework. The first approach, Deep Multi-Agent Parameterized Q-networks (Deep MAPQN), extends the architecture of P-DQN to multi-agent settings by leveraging the idea of Qmix [2018]. Our algorithm utilizes a joint action-value function to update policies of hybrid actions for all agents. However, Deep MAPQN requires to compute continuous parameters for all the discrete actions of all agents and thus may suffer from high computational complexity when the discrete part of hybrid action space has large dimensions. To alleviate this problem, we propose the second approach, Deep Multi-Agent Hierarchical Hybrid Q-networks (Deep

*Corresponding author

MAHHQN). In contrast to Deep MAPQN, Deep MAHHQN only needs to calculate continuous parameters of the optimal discrete action for each agent. In addition, it further alleviates the non-stationary issue of multi-agent environments by realizing centralized training fashion at both action levels and augmenting each centralized training framework with information about policies of other action levels. Empirical results on standard benchmark game Half Field Offense (HFO) and a large-scale video game *Ghost Story* show the superior performance of our approaches compared to independent P-DQN.

2 Background

2.1 Cooperative Stochastic Game

In this work, we consider a $Cooperative\ Stochastic\ Game$ [Wei *et al.*, 2018a] in partially observable settings modeled as a tuple $\{S, U, r, P, \gamma, H, N\}$. This game for N agents is defined by a set $\mathcal{S}$ of states describing the possible configurations of all agents, a set of observations $\mathcal{O}_1, ..., \mathcal{O}_N$ for each agent and a joint action space of N agents defined as $U = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$. At each time step, each agent i chooses an action $a_i \in \mathcal{A}_i$ using policy π_i and constitutes a joint action $\vec{a} \in U$, which produces the next state s' following the state transition function $P(s'|s, \vec{a})$: $S \times U \times S \to [0, 1]$. All agents share the same reward function $r(s, \vec{a})$: $S \times U \to \mathbb{R}$. γ is the discount factor and H is the time horizon.

2.2 Deep Multi-agent Reinforcement Learning

Multi-agent learning has been investigated comprehensively in both discrete action domains and continuous action domains under framework of centralized training and decentralized execution.

MADDPG [Lowe *et al.*, 2017] mainly focuses on multi-agent problems with continuous action spaces. The core idea is to learn a centralized critic $Q_I^\mu(x, a_1, \cdots, a_N)$ for each agent which conditions on global information. $Q_I^\mu(x, a_1, \cdots, a_N)$ takes as input the actions of all agents, $a_1, \cdots, a_N$, in addition to global state information x (i.e., $x = (o_1, \cdots, o_N)$) and outputs the centralized action-value for agent i.

Qmix [Rashid *et al.*, 2018] employs a mixing network that estimates joint action-values Q_{tot} as a complex non-linear combination of per-agent action value that conditions only on local observations. The weights of the mixing network are produced by separate hypernetworks which take the global state information as input. Importantly, Qmix ensures a global $argmax$ performed on Q_{tot} yields the same result as a set of individual $argmax$ operations performed on each agent's action-value Q_i by a monotonicity constraint:

$$\frac{\partial Q_{tot}}{\partial Q_i} \geq 0, \tag{1}$$

Unfortunately, such multi-agent algorithms cannot be applied to hybrid action spaces. Qmix belongs to the class of value-based reinforcement learning approaches that are commonly considered suitable for discrete action space problems. But they cannot be applied to continuous settings [Sutton and Barto, 1988; Silver *et al.*, 2014]. MADDPG belongs to the class of policy-based approaches which directly learn deterministic policies over either discrete action spaces or continuous action spaces. But such approaches cannot learn a policy over hybrid action spaces [Mnih *et al.*, 2016; Schulman *et al.*, 2017].

2.3 Parameterized Deep Q-Networks

To handle reinforcement learning problems with hybrid action space, Xiong et al. [2018] propose a new framework called Parameterized Deep Q-Networks (P-DQN). The core idea is to update the discrete-action and continuous-action policies separately combining the structure of DQN [Mnih *et al.*, 2013] and DDPG [Lillicrap *et al.*, 2016]. P-DQN first chooses low-level parameters associated with each high level discrete action, then figures out which hybrid action pair maximizes the action-value function.

More concretely, we can define the discrete-continuous hybrid action space $\mathcal{A}$ as:

$$\mathcal{A} = \{(k, x_k) | x_k \in \mathcal{X}_k \text{ for all } k \in [K]\}, \tag{2}$$

where $[K] = \{1, \cdots, K\}$ is the discrete action set, $\mathcal{X}_k$ is a continuous set for all $k \in [K]$. Then we can define a deterministic function which maps the state and each discrete action k to its corresponding continuous parameter x_k:

$$x_k = \mu_k(s; \theta), \tag{3}$$

where θ are weights of the deterministic policy network. We further define an action-value function $Q(s, k, x_k; \omega)$ which maps the state and hybrid actions to real values. Here ω are weights of the value network.

P-DQN updates the action-value function Q by minimizing the following loss:

$$l^Q(\omega) = \frac{1}{2}\left[Q(s, k, x_k; \omega) - y\right]^2, \tag{4}$$

where $y = r + \max_{k \in [K]} \gamma Q(s', k, \mu_k(s'; \theta); \omega)$, and s' denotes the next state after taking hybrid action (k, x_k). The policy μ_k for the continuous part is updated by minimizing the following loss with parameters ω fixed:

$$l^\Theta(\theta) = -\sum_{k=1}^{K} Q(s, k, \mu_k(s; \theta); \omega) \tag{5}$$

3 Methods

To handle hybrid action spaces in multi-agent settings, one natural approach is to adopt the independent learning paradigm and equip each agent with an independent P-DQN algorithm. However, one major issue is that each agent's policy changes during training, resulting in the non-stationarity of environments. As we will show in our experiments in Section 4, independent P-DQN does not perform well in practice.

In this paper, we propose two novel deep multi-agent learning methods for hybrid action spaces, Deep MAPQN and Deep MAHHQN. By leveraging the current state-of-the-art single-agent deep RL for hybrid action spaces and coordination techniques for multi-agent learning, both algorithms can support multiple agents to learn effective coordination policies directly over the hybrid action spaces.

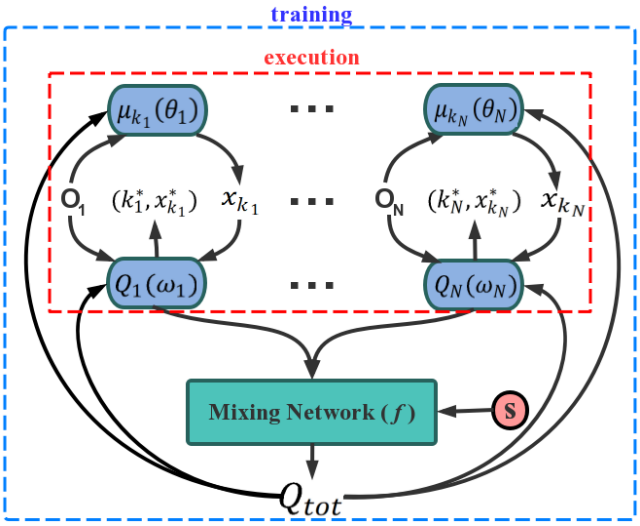

Figure 1: The overall Deep MAPQN structure

3.1 Deep Multi-Agent Parameterized Q-Networks (Deep MAPQN)

The first algorithm is a natural extention of single-agent P-DQN. We leverage Qmix [Rashid *et al.*, 2018] architecture to coordinate the policy update over hybrid action spaces among agents. The overall structure of Deep MAPQN is shown in Figure 1.

For each agent, we adopt the same settings in P-DQN. Concretely, considering a game with N agents, each agent i uses a deterministic policy network $\mu_{k_i}(\theta_i)$ and an action value network $Q_i(\omega_i)$ to output hybrid action $(k_i^*, x_{k_i}^*)$. The deterministic policy network $\mu_{k_i}(\theta_i)$ takes each agent's observation o_i as input and outputs the optimal continuous parameters x_{k_i} for all the discrete actions $k_i \in \{1, \cdots, K_i\}$. Then the action value network Q_i outputs the optimal hybrid action by:

$$(k_i^*, x_{k_i}^*) = argmax_{(k_i, x_{k_i})} Q_i\left(s_i, (k_i, x_{k_i}); \omega_i\right), \quad (6)$$

where ω_i are parameters for action value network of agent i.

To achieve coordinated update among agents' action value networks, we utilize a mixing network and produce a fully centralized state-action value function Q_{tot} which can facilitate the coordinated update of the decentralized policies in hybrid action spaces. The mixing network consists of a feedforward neural network and separate hypernetworks. The hypernetworks take the global state $\mathbf{s}$ as the input and output the weights of the feedforward network. The feedforward network takes each agent's output Q_i as input and mixing them monotonically, producing the joint action value denoted by Q_{tot}.[1] Here we define the mixing network as a non-linear complex function f and denote this process by:

$$Q_{tot} = f(\mathbf{s}, Q_1, \cdots, Q_N; \omega_{mix}) \quad (7)$$

We update the mixing network weights ω_{mix} along with each agent's action value network weights ω_i by minimizing:

$$\mathcal{L}(\omega) = \mathbb{E}_{s,\vec{k},\vec{x}_k,r,s' \sim \mathcal{D}}[y^{tot} - Q_{tot}(\mathbf{s}, \vec{k}, \vec{x}_k)]^2, \quad (8)$$

[1]The detailed structure for the mixing network can be found at: https://bit.ly/2Eaci2X.

where $y^{tot} = r + \gamma \max_{\vec{k}', \vec{x}_{k'}} Q_{tot}(\mathbf{s}', \vec{k}', \vec{x}_{k'}(\theta'))$, and $(\vec{k}, \vec{x}_k)$ is the joint action, θ' are parameters of target policy networks. Our framework for computing Q_{tot} ensures off-policy learning updates while each agent can still choose the greedy action with respect to its own Q_i in a decentralized fashion. This is because a global argmax on Q_{tot} is equivalent with argmax on each Q_i as explained in Section 2.2.

Finally we need to compute gradients for deterministic policy network. We first take the sum of Q-values of all the discrete actions for each agent i:

$$\widehat{Q}_i = \sum_{k_i=1}^{K_i} Q_i\left(s_i, k_i, x_{k_i}; \omega_i\right), \text{where } k_i \in \{1, 2, \cdots, K_i\}, \quad (9)$$

and then feed them into the mixing network, producing the value of Q_{tot}^s. This process can be denoted by:

$$Q_{tot}^s = f(\mathbf{s}, \widehat{Q}_1, \cdots, \widehat{Q}_N; \omega_{mix}) \quad (10)$$

We update all agents' continuous policies μ_i $(i \in N)$ by maximizing Q_{tot}^s with parameters ω_i and ω_{mix} fixed, the gradient can be written as:

$$\nabla_{\theta_i} l(\theta_i) = \mathbb{E}_{s,\vec{k} \sim \mathcal{D}}[\nabla_{\theta_i} \mu_{k_i}(o_i) \nabla_{x_{k_i}} Q_{tot}^s(\mathbf{s}, \vec{k}, \vec{x}_k; \omega) \mid_{x_{k_i} = \mu_{k_i}(o_i)}] \quad (11)$$

In this way we can update the policies of continuous parameters for all the different agents and different discrete actions in one single training step.

However, this algorithm may result in high computational complexity during both training and execution phases: every time we compute the joint action value Q_{tot}, we need to compute continuous parameters for all the K discrete actions of all the agents. This is particularly severe when the discrete part of hybrid action space has a large dimension. The same problem exists in original P-DQN as well.

3.2 Deep Multi-Agent Hierarchical Hybrid Q-Networks (Deep MAHHQN)

To address the problem we mentioned in previous section, we propose another novel algorithm Deep MAHHQN, as inspired by Hierarchical Learning [Kulkarni *et al.*, 2016; Tang *et al.*, 2018].

The overall structure of Deep MAHHQN is illustrated in Figure 2. Different from Deep MAPQN, when choosing hybrid actions, a Deep MAHHQN agent first chooses a discrete action through high-level network and then decides the corresponding continuous parameters conditioning on the given discrete action and individual observation through low-level network. This is consistent with human's decision-making process since in real world humans usually tend to decide what to do before deciding to what extent to do it. We train the high-level network and low-level network separately and both of them follow the centralized training but decentralized execution paradigm.

As for the high-level network, each agent utilizes the basic DQN [Mnih *et al.*, 2015] framework to select a discrete

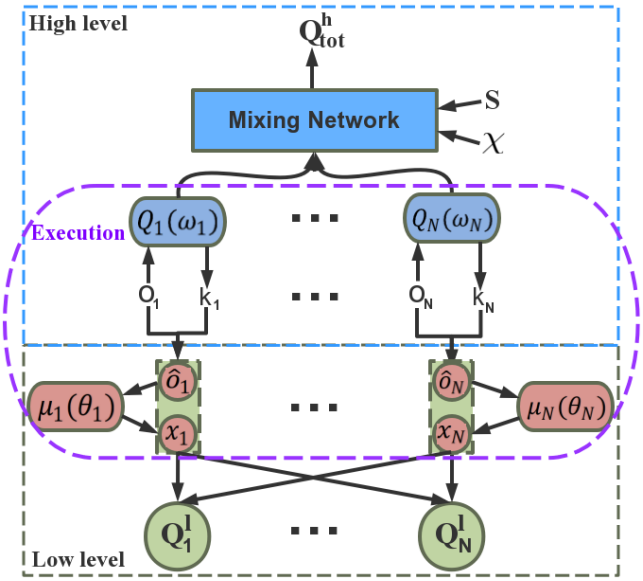

Figure 2: The overall Deep MAHHQN structure

action k_i given its own observation o_i. The high-level policies among agents over discrete actions are coordinated by using the mixing network structure [Rashid *et al.*, 2018]. Importantly, unlike Qmix which only takes global state $\mathbf{s}$ as inputs for the hypernetworks to produce weights of mixing networks, we further consider low-level network's current policies for each agent. This information is critical for coordinating agents' discrete actions since the hybrid action spaces are highly correlated in determining the global optimal policies. Concretely, at each training step, we first calculate the continuous parameters x_i related to each agent's high level action k_i using the current low-level network's policy (i.e., $x_i = \mu_i(\varphi(o_i, k_i))$). Then we combine them (i.e., $\chi = \{x_1, \cdots, x_N\}$) with the global state $\mathbf{s}$ and feed them into the hypernetworks to generate weights for the mixing network. The high-level networks' parameters are updated by minimizing the following loss:

$$\mathcal{L}(\omega^h) = \mathbb{E}_{s,\vec{k},r,s' \sim \mathcal{D}}(y^h_{tot} - Q^h_{tot}(\mathbf{s}, \vec{k}, \chi))^2, \tag{12}$$

where $\vec{k}$ denotes the joint discrete action sampled from the replay buffer, $y^{tot} = r + \gamma \max_{\vec{k}'} Q^h_{tot}(\mathbf{s}', \vec{k}', \chi')$, χ' is a set of continuous parameters from low-level target policies.

For the low-level part, each agent i chooses continuous parameters x_i according to its new observation:

$$\widehat{o}_i = \varphi(o_i, k_i), \tag{13}$$

where k_i is the discrete action obtained from the high-level part. In our experiments, we simply concatenate o_i and k_i as the new observation $\widehat{o}_i$. In order to learn a coordinated policy over the corresponding continuous parameters, we apply Multi-agent Actor-critic framework [Lowe *et al.*, 2017] and further modify it with a centralized Q function for each agent. Considering N agents with low-level policies parameterized by $\theta = \{\theta_1, \cdots, \theta_N\}$, and let $\mu = \{\mu_1, \cdots, \mu_N\}$ be the set of all agents' low-level policies. The gradient with expected return for agent i with low-level policy μ_i can be written as:

$$\nabla_{\theta_i} l(\theta_i) =$$
$$\mathbb{E}_{s,k,x \sim \mathcal{D}}[\nabla_{\theta_i} \mu_i(\widehat{o_i}) \nabla_{x_i} Q^l_i(\mathbf{s}, k_1, x_1, \cdots, k_N, x_N)|_{x_i = \mu_i(\widehat{o_i})}] \tag{14}$$

Here $Q^l_i(\mathbf{s}, k_1, x_1, \cdots, k_N, x_N)$ is a centralized action-value function that takes as input the hybrid actions of all agents $(k_1, x_1), \cdots, (k_N, x_N)$, in addition to the global state $\mathbf{s}$, and outputs the low-level Q-value for agent i. The experience replay buffer $\mathcal{D}$ contains the tuples $(\mathbf{s}, k_1, x_1, \cdots, k_N, x_N, r_1, \cdots, r_N, \mathbf{s}')$. The centralized Q function is updated as:

$$\mathcal{L}(\omega^l_i) = \mathbb{E}_{s,k,x,r,s' \sim \mathcal{D}}[y_i - Q^l_i(\mathbf{s}, k_1, x_1, \cdots, k_N, x_N)]^2,$$
$$y_i = r_i + \gamma Q^{l'}_i(\mathbf{s}', k'_1, x'_1, \cdots, k'_N, x'_N)|_{x'_j = \mu'_j(\widehat{o_i}')}, \tag{15}$$

where $Q^{l'}_i$ denotes the low-level target Q network for agent i. Note that k'_i are derived from high-level target policies. Combining (14) and (15) yields our proposed low-level network.

Compared with Deep MAPQN, Deep MAHHQN only needs to calculate one discrete action with the optimal continuous parameters for each agent. This would significantly reduce the algorithm's computational complexities, which will be validated in our experimental results. Moreover, both low-level and high-level training frameworks of Deep MAHHQN are augmented with extra information about policies of other agents and policies of other action levels. In hybrid action environments, we expect that such kind of communication would better alleviate the non-stationary issue of the environments.

When training Deep MAHHQN, we let the low-level network train alone for m steps and then start training high-level and low-level network together. The main reason is that the associated low-level continuous parameters play an important role when our high-level network computes the value of Q^h_{tot}. Otherwise, the gradients for high-level network can be very noisy and misleading since the low-level policies are still exploring at a high rate at the very beginning.

4 Experimental Results

In this section, we evaluate our algorithms in 1) the standard benchmark game HFO, 2) 3v3 mode in a large-scale online video game *Ghost Story*. We compare our algorithms, Deep MAPQN and Deep MAHHQN, with independent P-DQN in all our experiments.

4.1 Experiments with Half Field Offense (HFO)

Half field Offense (HFO) is an abstraction of full RoboCup 2D game. The environment features in a hybrid action space. Previous work [Hausknecht and Stone, 2016; Wang *et al.*, 2018; Wei *et al.*, 2018b] applied RL to the single-agent version of HFO, letting one agent try to goal without a goalkeeper (1v0). In this paper, we apply our proposed algorithms on the challenging multi-agent problems of HFO , which includes **1v2** (two agents with the shared objective of defending the goal against one opponent) and **2v1** (two agents with the

shared objective of scoring the goal against one goalie). The opponent (or goalie) we play against are all built-in hand-coded agents.

We use the high level feature set and each agent's observation is a 21-d vector consisting of its position and orientation; distance and angle to the ball and goal; an indicator if the agent can kick etc. A full list of state information can be found at the official website https://github.com/mhauskn/HFO/blob/master/doc/manual.pdf.

In our experiments, we use a discrete-continuous hybrid action space. The full set of hybrid actions is: **Kick To** $(target_x, target_y, speed)$**; Move To** $(target_x, target_y)$**; Dribble To** $(target_x, target_y)$**; Intercept()**. Valid values for $target_{x,y} \in [-1, 1]$ and $speed \in [0, 3]$. In this settings, acting randomly is almost unlikely to score or successfully defend the goal and the exploration task proves too difficult to gain traction on a reward that only consists of scoring goals. Thus, we do reward engineering to our two tasks respectively to alleviate the sparse reward problem, which will be described in details in following sections.

1v2 Defense

In the defending scenario, our models control two agents with the shared objective of defending the goal. Note that for defensive agents, only two actions **Move To** and **Intercept** are applicable since the defensive players do not control the ball. The reward for each time step is calculated as a weighted sum of the following three types of statistics:

- Move to ball reward: A reward proportional to the change in distance between the agent and the ball.
- Punishment for no agent in goal area. We add a punishment if there's no defensive agent in the goal area.
- Bonus points for game result. Agents will get extra positive points if they successfully defend the goal and vice versa.

We can see from Figure 3 that both Deep MAPQN and Deep MAHHQN achieve much higher performance than P-DQN. This demonstrates the benefits of explicitly coordinating the joint hybrid policies among agents. In addition, Deep MAHHQN is found to outperform Deep MAPQN after convergence. We attribute this to the improved communication between different agents when we do centralized training for Deep MAHHQN. We further examine the learned behaviours of policies in order to better understand the methods. The agents of Deep MAPQN and Deep MAHHQN tend to play different roles automatically when defending the ball: one agent moves directly to the goal area and act like a goalkeeper while the other one approaches the offensive player trying to capture the ball. In contrast, the agents of P-DQN tend to approach the offensive player or move to the goal area together without cooperating with an appropriate division of their roles. A video of our learned policies may be viewed at https://youtu.be/ndJYZFL5BxE.

2v1 Offense

We further evaluate our algorithms on 2v1 offense mode which have larger action space and the coordination task is expected to be more challenging. In this scenario, our models control two agents aiming to score a goal against one goalie. We add a discrete action **Shoot()** to our action list and there are totally five types of actions for this experiment. The reward for each time step is calculated as a weighted sum of the following types of statistics:

- Move to ball reward: Similar to last section.
- Kick to goal reward: A reward proportional to change in distance between the ball and the center of the goal.
- Additional punishment: To avoid long shot, we add an additional punishment if the ball is too far away from both agents when it is still outside the goal area.
- Bonus points for game result: Agents will get extra points if they successfully score a goal.

As shown in Figure 4, in this challenging offense problem, independent P-DQN fails to learn the coordination policy. In practice, we observe that independent P-DQN agents first try to approach the ball, then dribble it to somewhere and stop without shooting. A primary reason for this may be the lack of information of other teammates. In comparison, our proposed methods Deep MAPQN and Deep MAHHQN perform much better. Similar to the previous defense setting, the Deep MAPQN agents learn policies a little faster than Deep MAHHQN in terms of the number of time steps required. The underlying reason is that in Deep MAPQN we update policies of continuous parameters associated with all the discrete actions

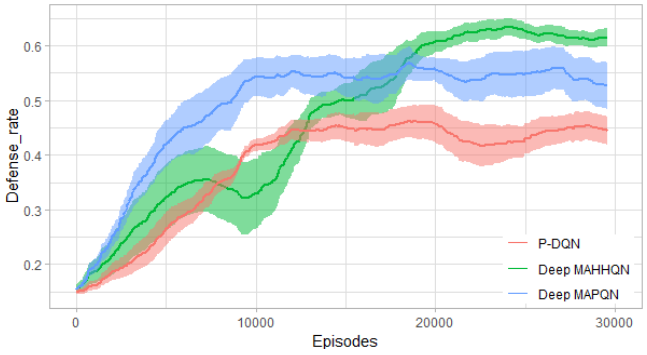

Figure 3: Successfully defense rates for Deep MAPQN, Deep MAHHQN and P-DQN in 1v2 defense mode of HFO

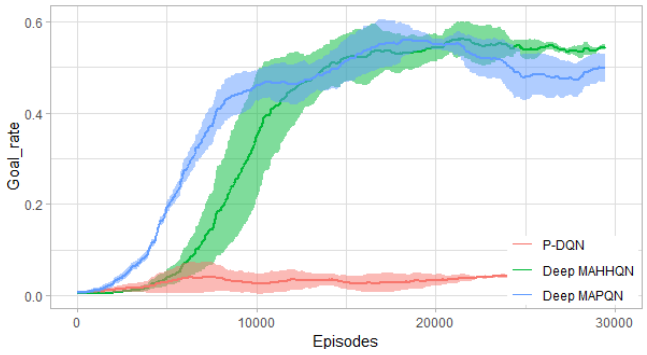

Figure 4: Goal rates for Deep MAPQN, Deep MAHHQN and P-DQN in 2v1 offense mode of HFO

Figure 5: A screen shot of a 3v3 game mode in *Ghost Story*

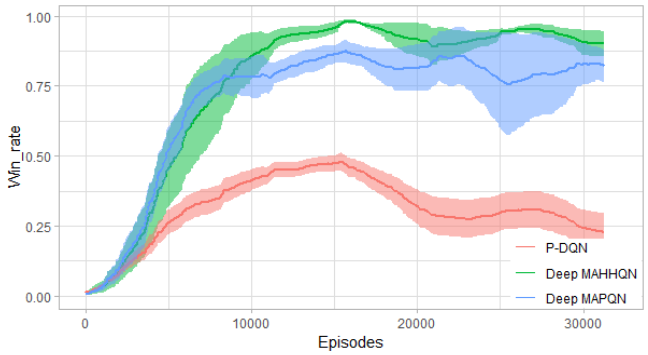

Figure 6: Win rates for Deep MAPQN, Deep MAHHQN and P-DQN in 3v3 mode of *Ghost Story*

at one single training step. However, such a framework also introduces huge computational complexity since each training step of Deep MAPQN occupies much more computing resources than Deep MAHHQN. This problem can be seen more clearly in our second game (Ghost Story) with a more complicated practical environment.

4.2 Experiments with Ghost Story

We evaluate the proposed algorithms with *Ghost Story* (or *QianNvYouHun*) – a fantasy massive multiplayer online role-playing game (MMORPG) from *NetEase*, in which each of the learning agents controls an individual hero. We performed our evaluation in the "3v3" game mode, where 3 hero agents cooperate with each other to fight against the other 3 built-in AI on the opposite side. At each time step, every hero can choose to move or use one of its own skills. The game ends when all 3 heroes on the same side are killed.

The observation for each agent is a 97-d feature vector which is manually constructed using output from the game engine. Concretely, these features consist of some basic properties of the agents: agent's Health Point (*HP*), value range of attack, value range of defense[2], carried skills and Cool Down (*CD*), carried buffs[3], relative positions etc.

We simplify the actions of each hero into five hybrid action types: $Move(x, y)$, move to a relative position(x,y); and four skills, $Tanlang(x, y)$, $Siguai(x, y)$, $Tianshou()$ and $Hegu(x, y)$. When a hero player chooses to use a skill, the enemies near the relative position (x, y) will be attacked or added some buffs depending on the specific types of selected skills. Skill $Tianshou$ has no parameters since it functions as adding a buff on the hero agent itself. More details of our experimental settings can be found at https://bit.ly/2Eaci2X.

At each time step, each agent receives a joint reward consists of four parts: (1) the change in *HP* for all heroes; (2) a punishment for the agents which did not do anything (e.g. No enemy is near the skill's target point; the selected skill's *CD* is not zero); (3) small bonus points for killing an enemy hero and huge bonus points for winning the game.

[2]The agent's true attack or defense value is uniformly distributed on the given value range.

[3]Buffs can be beneficial, such as increase agent's own defense value, or harmful, such as decrease its own attack value, but buffs can only exist for a short time period.

Figure 6 shows our experimental results. Clearly we can see independent P-DQN fails to learn a coordinated policy that can consistently defeat the enemies. Both Deep MAPQN and Deep MAHHQN outperform independent P-DQN learning method and get a win rate over 75%. Moreover, we find Deep MAHHQN method performs slightly better than Deep MAPQN, which further validates that the improved communication between agents using information about policies of other action levels can better stabilize the centralized training process. It indicates that Deep MAHHQN framework provides a better way to handle problems with large hybrid action space and number of agents. A final note is that the actual training time of Deep MAPQN is about three days while Deep MAHHQN takes less than one day to train on the same NVidia Geforce GTX 1080Ti GPU. This is quite reasonable since each MAPQN agent need to compute continuous parameters for all the discrete actions at each single training step while in Deep MAHHQN we only need to calculate low level continuous parameters associated with the selected optimal high-level discrete action.

5 Conclusion

This paper should be seen as the first attempt at applying deep reinforcement learning in cooperative multi-agent settings with discrete-continuous hybrid action spaces. Two novel approaches are proposed under the paradigm of centralized training and decentralized execution. The experimental results show their superiority to independent parameterized Q-learning method under both the standard testbed HFO and a large-scale MMORPG game. As future work, we wish to extend our algorithms to competitive multi-agent settings, and conduct additional experiments with more agents and larger hybrid action space to further investigate the difference of performance between our two proposed algorithms.

Acknowledgments

The work is supported by the National Natural Science Foundation of China (Grant Nos.: 61702362, U1836214), Special Program of Artificial Intelligence and Special Program of Artificial Intelligence of Tianjin Municipal Science and Technology Commission (No.: 569 17ZXRGGX00150).

References

[Cao *et al.*, 2013] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial Informatics*, 9:427–438, 2013.

[Foerster *et al.*, 2016] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Proceedings of NeurIPS*, pages 2137–2145, 2016.

[Hausknecht and Stone, 2016] Matthew J. Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *In Proceedings of ICLR*, pages 861–868, 2016.

[Hausknecht, 2016] Matthew J. Hausknecht. Half field offense : An environment for multiagent learning and ad hoc teamwork. In *Proceedings of ALA*, pages 1391–1398, 2016.

[Kulkarni *et al.*, 2016] Tejas D. Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Proceedings of NeurIPS*, pages 3675–3683, 2016.

[Lillicrap *et al.*, 2016] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *Proceedings of ICLR*, pages 1052–1059, 2016.

[Lowe *et al.*, 2017] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of NeurIPS*, pages 6382–6393, 2017.

[Masson *et al.*, 2016] Warwick Masson, Pravesh Ranchod, and George Konidaris. Reinforcement learning with parameterized actions. In *Proceedings of AAAI*, pages 1934–1940, 2016.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of ICML*, pages 1928–1937, 2016.

[Peng *et al.*, 2017] Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *CoRR*, abs/1703.10069, 2017.

[Rashid *et al.*, 2018] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of ICML*, pages 4292–4301, 2018.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[Silver *et al.*, 2014] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin A. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of ICML*, pages 387–395, 2014.

[Sutton and Barto, 1988] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 16:285–286, 1988.

[Tang *et al.*, 2018] Hongyao Tang, Jianye Hao, Tangjie Lv, Yingfeng Chen, Zongzhang Zhang, Hangtian Jia, Chunxu Ren, Yan Zheng, Changjie Fan, and Li Wang. Hierarchical deep multiagent reinforcement learning. *CoRR*, abs/1809.09332, 2018.

[Wang *et al.*, 2018] Qing Wang, Jiechao Xiong, Lei Han, Peng Sun, Han Liu, and Tong Zhang. Exponentially weighted imitation learning for batched historical data. In *Proceedings of NeurIPS*, pages 6291–6300, 2018.

[Wei *et al.*, 2018a] Ermo Wei, Drew Wicke, David Freelan, and Sean Luke. Multiagent soft q-learning. In *Proceedings of AAAI*, 2018.

[Wei *et al.*, 2018b] Ermo Wei, Drew Wicke, and Sean Luke. Hierarchical approaches for reinforcement learning in parameterized action space. In *Proceedings of AAAI*, 2018.

[Xiong *et al.*, 2018] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peter P Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Hao Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *CoRR*, abs/1810.06394, 2018.

[Ye *et al.*, 2015] Dayong Ye, Minjie Zhang, and Yun Yang. A multi-agent framework for packet routing in wireless sensor networks. *Sensors*, 15(5):10026–10047, 2015.