COMP390

2021/22

Extending QMIX with Simple Techniques from Single-Agent Reinforcement Learning

Student Name:	Peng Zheng
Student ID:	201522735
Supervisor Name:	Dr Bei Peng

DEPARTMENT OF
COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Dedicated to my parents

Acknowledgements

Firstly, I would like to thank my supervisor, Dr. Peng, my favorite teacher throughout my time at university, as well as the first and the best supervisor in my life ever. I cherish the time that we spent together when discussing some interesting issues and really appreciate your patience and encouragement. Hope we could keep in touch and wish you all the best.

Also, thanks to all my teachers who support my study, and friends who are always behind my back. I am so grateful for your company. I will miss the time as an undergraduate at University of Liverpool and XJTLU.

I dedicate this work to my parents, who are the best in the world. Thanks so much for giving birth to me and raising me so well. Thank you, mom, for cheering me up in my frustration; thank you, dad, you are the very person who always inspired me and led me to think thoroughly. Though we hardly meet during the pandemic, my best wishes are always with you.

COMP390

2021/22

Extending QMIX with Simple Techniques from Single-Agent Reinforcement Learning

DEPARTMENT OF
COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Abstract

Multi-Agent Reinforcement Learning (MARL) is a powerful paradigm to address some real-world intractable scenarios, where a team of agents cooperate with each other, intended to attain a common goal. QMIX is one of the state-of-art value-based deep MARL algorithms, which performs outstandingly on the challenging StarCraft Multi-Agent Challenge (SMAC) benchmark. However, it is a critical problem that QMIX performs unstably on some cooperative tasks. Therefore, to improve the stability and performance of QMIX, this project investigates some extensions of QMIX with two techniques, namely multi-step learning and dueling networks. Multi-step learning is an intermediate method to address the issue of bias-variance trade-off existing in its two extremes. Dueling network is a deep Q-network with the characteristic of representing state value and advantages separately. The performance of these extensions has been evaluated on a cooperative multi-agent task, predator-prey, in two different levels of difficulty. Experiments show that those extensions significantly outperform QMIX and improve the stability to some extent.

Table of Contents

1. Introduction and Background

1.1. Introduction

QMIX [1] is a value-based cooperative multi-agent reinforcement learning (MARL) algorithm in the paradigm of centralized training with decentralized execution (CTDE) [2], where the decentralized polices based on local histories are trained with global information in a centralized manner. A key concept in CTDE is to ensure that the global joint action yields the same result as a set of individual actions selected by each agent, which is referred to as Individual-Global-Max (IGM) [3, 4]. The IGM consistency has been realized in QMIX by enforcing the monotonicity between centralized action-value function (CAF) Qtot and individual action-value function (IAF) Qa. Specifically, it proposes an architecture consisting of agent networks that represents each Qa, and a mixing network combining them into Qtot in a complicated non-linear fashion. The weights of the mixing network are restricted to be non-negative to achieve the monotonicity constrain. Hence, more complex CAFs can be represented by QMIX with a factorization method that decentralized policies can be extracted in linear time via individual argmax operations. QMIX has been evaluated to achieve an outstanding performance on StarCraft II micromanagement challenges.

However, the training stability of QMIX still remains an issue, and as it shown in Figure 1, in a toy predator-prey task [5], the training variance represented by shading area around the curve is relatively high, after five seeds have been run. Also, in another predator-prey task in Figure 2, QMIX fails the task, so there should be space for improvement to the performance of QMIX. Thus, to address the issue of stability and improve the performance, this project explores some extensions of QMIX, namely n-QMIX with the method of multi-step learning [6], d-QMIX with the architecture of dueling networks [7], and their combination, dn-QMIX.

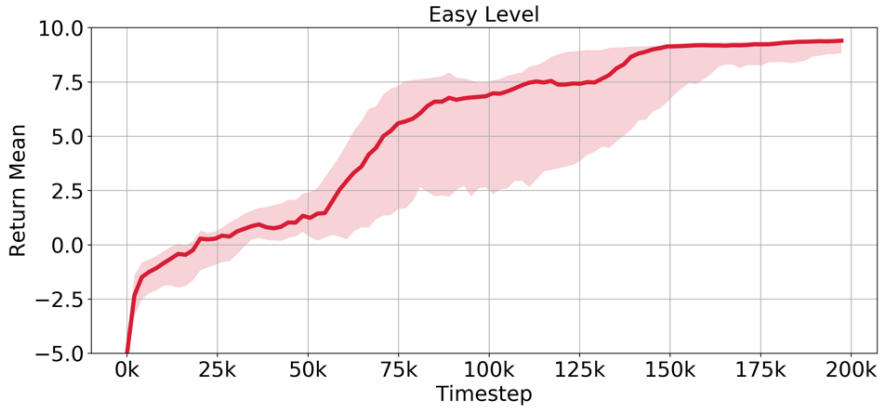


Figure 1

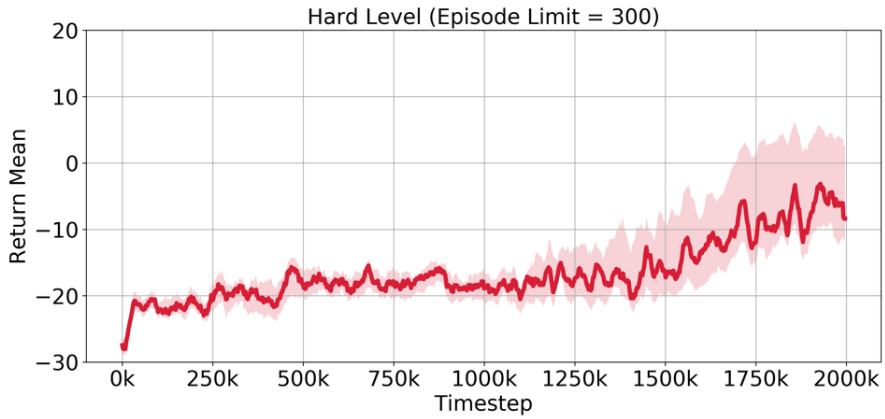


Figure 2

Multi-step learning also called n-step learning is a method lying between Monte Carlo and one-step Temporal Difference (TD) method [6]. Monte Carlo methods observe the full sequence of rewards to update from the current state to terminate state, so in theory, the return should be an unbiased estimation of the true state value. However, because all the actions taken from the current state are considered in Monte Carlo, for example, if there are two traces after taking a same action and having highly different returns that are far from the true state value, the variance of estimating is relatively high. In other words, Monte Carlo overfits the episode. In contrast, one-step TD method updates based upon reward at the very next timestep and takes the state value as a proxy for rewards left in whole sequence, so it has a low variance. However, representing the future rewards accumulated from current state by the estimated state value at one step later is biased, and actions may have delayed rewards in most cases, so one-step TD underfits the episode. Regarding the bias-variance trade-off, an intermediate method, multi-step learning has been proposed: consider more than one, but less than all before terminating. Theoretically, all multi-step returns are the approximations of the full-step return at termination but

truncated after n steps and corrected by the estimated state value at the n-th step. Experiments suggest that multi-step learning has a high potential to perform better than its two extreme methods.

Dueling network which has been first proposed in [7], is a Q-network architecture that can be combined with some existing model-free reinforcement learning algorithms. An important idea of the dueling architecture is to represent state values V and state-dependent action advantages A separately. To be specific, comparing with the Deep Q-networks (DQN) [8], rather than followed with a single stream of fully connected (FC) layers, the convolutional layers are followed by two sequences of FC layers to estimate the state value and advantages respectively. The two streams are then combined by an aggregating layer to generate the estimated state-action value function Q. In addition, regarding the identifiable issue, that is to uniquely recover V and A given a specific Q, the aggregator uses V plus A minus the average of advantages to output Q. The dueling network can directly evaluate the state value without the information of how each action effects each state. According to experiments, the dueling architecture is especially powerful at making good decisions as the number of actions that do not impact the environment increases.

The performance of those aforementioned extended algorithms has been tested and compared with QMIX on two level of predator-prey tasks. The hypotheses of this project are: 1) n-QMIX and d-QMIX will outperform QMIX at least in the easy level of predator-prey task with smaller variances. 2) For n-QMIX, as the value of n increases, its performance will drop. Experiments showed that n-QMIX, d-QMIX and dn-QMIX outperform QMIX with smaller variances at easy level. In addition, when n is tuned to two, the performance of n-QMIX in easy-level task has been improved, and then gradually drops as n becomes larger. At hard level, only d-QMIX performs better than QMIX in terms of different maximum timesteps per episode. n-QMIX and dn-QMIX outperform QMIX only when episode limit equals to three hundred.

1.2. Aims and objectives

1.2.1. The aim

Investigate some extensions of QMIX, a value-based cooperative MARL algorithm by using two simple techniques, namely dueling networks and multi-step learning, intended to achieve a better performance in some predator-prey tasks.

1.2.2. Objectives

i. Review some previous research literature to understand the underlying principles of reinforcement learning, QMIX, dueling networks and multi-step learning.
ii. Learn some significant frameworks and libraries that will be utilized in the implementation stage, such as PyMARL [9] and PyTorch.
iii. Implement the code of extensions on PyMARL.
iv. Set cooperative scenarios of predator-prey tasks and test the performance of different extensions in comparison with QMIX.
v. Analyze the testing results and understand how algorithms work differently in different tasks.

1.3. Background

1.3.1. Reinforcement Learning

Reinforcement Learning [6] is one of the three fundamental machine learning patterns, along with Supervised Learning (SL) and unsupervised learning (UL). RL is superior in solving sequential decision-making tasks. In addition, unlike SL, RL can be applied to unknown realms since it allows an agent to enhance the performance through interacting with the environment via *trial-and-error*. Compared to UL, RL emphasizes the goal of maximizing the future total expected reward. Any RL problem can be formalized as a 5-tuple Markov Decision Process (MDP) [10], $(\mathrm{S}, \mathrm{A}, \mathrm{P}, \mathrm{R}, \gamma)$, where S is a set of states of the environment; A is a collection of actions an agent can select; P is the transition function denoting the probability of transitioning to state s' and receiving reward r at time t, given the preceding state s and action a, which is specified as followed:

$$\mathrm{P}(\mathrm{s}', \mathrm{r} \mid \mathrm{s}, \mathrm{a}) = \Pr\{\mathrm{S}_\mathrm{t} = \mathrm{s}', \mathrm{R}_\mathrm{t} = \mathrm{r} \mid \mathrm{S}_{\mathrm{t}-1} = \mathrm{s}, \mathrm{A}_{\mathrm{t}-1} = \mathrm{a}\},$$

(Equation 1)

where $\forall \mathrm{s}, \mathrm{s}' \in \mathrm{S}, \forall \mathrm{a} \in \mathrm{A}, \forall \mathrm{r} \in \mathrm{R}$; R is the reward function: $\mathrm{S} \times \mathrm{A} \to \mathbb{R}$. The expected reward for each pair of state and action is defined as below:

$$\mathrm{r}(\mathrm{s}, \mathrm{a}) = \mathbb{E}[\mathrm{R}_\mathrm{t} \mid \mathrm{S}_{\mathrm{t}-1} = \mathrm{s}, \mathrm{A}_{\mathrm{t}-1} = \mathrm{a}] = \sum_{\mathrm{r} \in \mathrm{R}} \mathrm{r} \sum_{\mathrm{s}' \in \mathrm{S}} \mathrm{P}(\mathrm{s}', \mathrm{r} \mid \mathrm{s}, \mathrm{a}),$$

(Equation 2)

where $\forall \mathrm{s}, \mathrm{s}' \in \mathrm{S}, \forall \mathrm{a} \in \mathrm{A}, \forall \mathrm{r} \in \mathrm{R}$; $\gamma \in [0,1)$ is the discount rate that determines how much immediate rewards are preferred to the future rewards.

In RL, the goal of the agent is to maximum the total reward that it can expect to cumulate in the long term, which is denoted as return Gt. The discounted return is formalized as:

$$\mathrm{G}_\mathrm{t} = \mathrm{R}_{\mathrm{t}+1} + \gamma \mathrm{R}_{\mathrm{t}+2} + \gamma^2 \mathrm{R}_{\mathrm{t}+3} + \ldots = \sum_{\mathrm{k}=0}^{\infty} \gamma^\mathrm{k} \mathrm{R}_{\mathrm{t}+\mathrm{k}+1},$$

(Equation 3)

where $\gamma \in [0,1)$. If $\gamma = 0$, the agent is said to be myopic for only concerning the immediate reward at next step. A policy $\pi(\mathrm{a}|\mathrm{s})$, maps from states to the probabilities of possible actions to be selected. It defines the distribution of

probability that $A_t = a$ if $S_t = s$. The agent's policy will change along with its experience in RL methods.

The state-value function $v_\pi(s)$ is the expected return, starting from a state s and following a policy π afterwards, which is formally defined by

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right], \forall s \in S.$$

(Equation 4)

The action-value function $q_\pi(s, a)$ is similarly defined as the expected return beginning from state s, taking action a, under policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right],$$

(Equation 5)

where $\forall s \in S$, $\forall a \in A$.

1.3.2. Multi-step Learning

Multi-step learning or n-step learning is an intermediate method that lies between two extremes, Monte Carlo and one-step TD method [6]. Monte Carlo methods update the estimation of $v_\pi(s)$ or $q_\pi(s, a)$ at the end of the episode, based on the fully observed sequence of rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-t-1} R_T,$$

(Equation 6)

where T is the terminate step. The full-step return G_t in Monte Carlo is also the target of the update. In contrast, the updating of one-step TD method is based on bootstrapping: it takes the reward at one next step and uses the estimated discounted value of next state as a proxy for the remaining rewards, which is called the one-step return:

$$G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1}),$$

(Equation 7)

where V_t is an estimation of the true state-value v_π at time t. Comparing to Monte Carlo, the discounted estimation $\gamma V_t(S_{t+1})$ substitutes for $\gamma R_{t+2} +$

$\gamma^2 R_{t+3} + \ldots + \gamma^{T-t-1} R_T$ in (Equation 6) of full-step return. An intermediate method, multi-step learning performs an update based upon more than one, but less than all the rewards until the episode terminates. For example, the two-step target is the first two rewards plus the discounted estimated value of the state after the next state:

$$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}),$$

(Equation 8)

For more general cases, the n-step return is defined as:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}),$$

(Equation 9)

for $n \in [1, \infty)$ and $t \in [0, T - n)$. Similarly, all n-step returns are approximations of the full-step return that is truncated after n steps and corrected by the discounted $V_{t+n-1}(S_{t+n})$. For those steps at $t \geq T - n$, the return $G_{t:t+n}$ is defined to be equal to the full-step return G_t.

Figure 3 shows the backup diagrams of multi-step learning, which forms a spectrum from one-step TD to Monte Carlo.

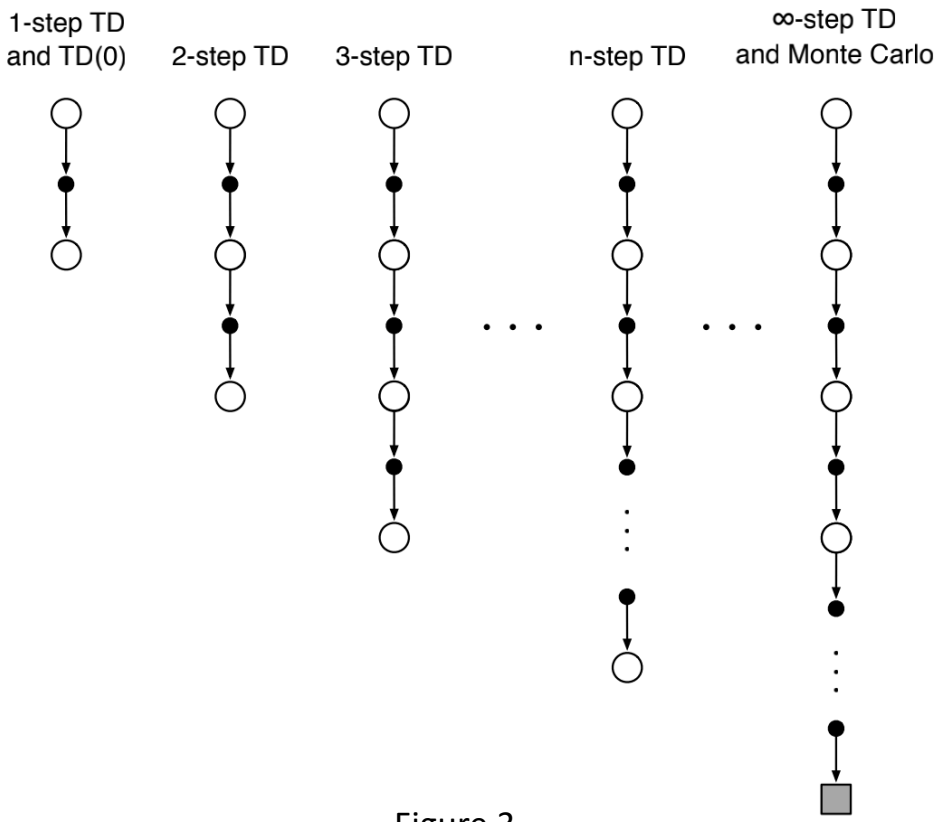


Figure 3

In theory, the Monte Carlo is unbiased because it performs the update based on a sequence of true rewards that are observed at the end of the episode. Formally, a bias of an estimator $\hat{\theta}$ is equal to $\mathbb{E}[\hat{\theta}] - \theta$. Thus, for all the methods above, Bias = $\mathbb{E}[\text{target}] - v_\pi(S_t)$. Specifically, in Monte Carlo, Bias = $\mathbb{E}[G_t] - v_\pi(S_t)$ where $\mathbb{E}[G_t] = v_\pi(S_t)$ by definition, so the full-return G_t should be unbiased. In contrast, for one-step TD method, the term R_{t+1} is the true reward gained at time t+1, so it is unbiased. However, $V_t(S_{t+1})$ is an estimation of the true value function $v_\pi(S_t)$, so the bias exists in one-step TD which underfits the episode. The variance indicates how noisy an estimator is and is also a criterion to evaluate the stability of an algorithm. In Monte Carlo, all the actions taken from current state S_t will affect the return G_t. For example, if two trajectories after taking the action A_t have very different returns, the estimated value G_t will deviate from the true value function $v_\pi(S_t)$. Thus, Monte Carlo method has high variance. In other words, it overfits the episode. By comparison, one-step TD methods updates based upon reward at the very next timestep and take its state value as a proxy for rewards left in whole sequence, so it has low variance. Multi-step learning, as an intermediate method, deals with the issue of bias-variance trade-off, which is referred to as a major advantage of this method. It has great potential to perform better than other two methods with a well-tuned n especially for complicated environments.

1.3.3. Dueling Networks

Dueling networks is a modified Deep Q-Networks architecture designed for model-free RL algorithms [7]. The Deep Q-Networks (DQN) was first proposed in [8], which combines RL with deep convolutional neural networks for the approximation of optimal action-value function,

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a, \pi\right],$$

(Equation 10)

where the maximum target Q value is achieved at policy $\pi = P(a|\, s)$, after performing action (a) and making an observation (s). Formally, the value function is redefined as $Q(s, a; \theta_i)$, where θ_i is a parameter representing the weights of DQN at iteration i. To improve the stability of Q-network, firstly, the experience reply, a mechanism inspired form biology has been used in DQN to randomize over the data, which smooths over the unstable changing in data distribution as well as removes correlations in the observation. Secondly, to reduce the target correlations, the adjusted action values Q have been iteratively updated in a

periodical manner. Specifically, the experiences of an agent have been denoted as $e_t = (s_t, a_t, r_t, s_{t+1})$ at timestep t and stored in a dataset $D_t = \{e_1, \ldots, e_t\}$. During the learning process, updates have been applied on experience samples $(s, a, r, s') \sim U(D)$, which are randomly and uniformly selected from the pool where samples are stored. At iteration i, the update follows the loss function as below:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2],$$

(Equation 11)

where θ_i^- is the parameter to calculate target for iteration i and is only updated with the parameter θ_i in a fixed timestep period.

Based on the similar architecture of convolutional neuro network in DQN, in Dueling networks, two streams rather than a single stream of FC layers follow the convolutional layers to represent the state value V and advantages A separately, and are then combined via an aggregator to compute the action-value function Q. In Figure 4 shown as below, on the top presents the normal single-stream DQN while on the bottom is the dueling network.

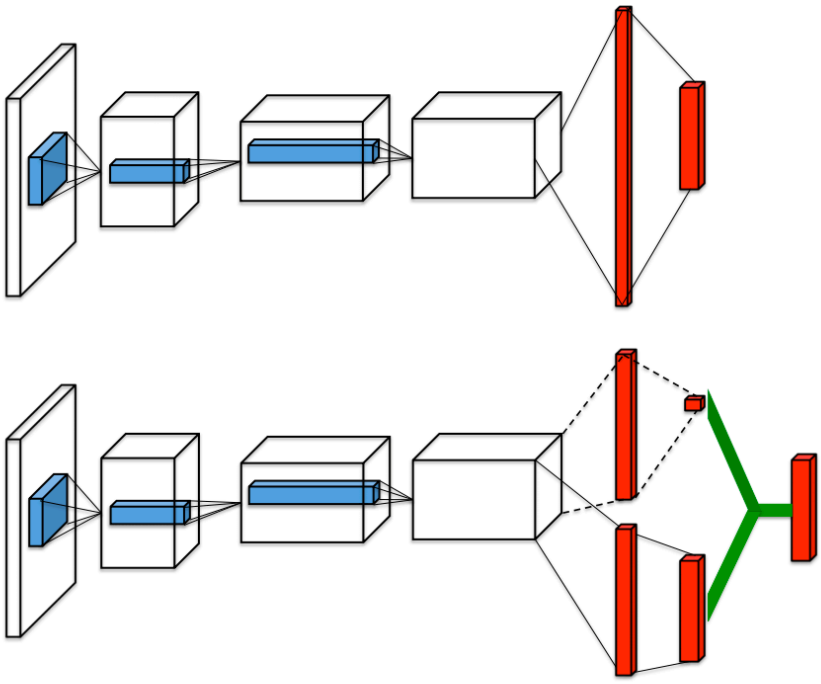


Figure 4

The advantage value intuitively shows that, given a certain state, how advantageous taking an action is, comparing to other actions. Thus, by definition, the advantage is equal to Q minus V under policy π:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s, a).$$

(Equation 12)

Considering the dueling architecture, the (Equation 12) is modified as followed:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha),$$

(Equation 13)

where θ is the similar parameter of convolutional layers in DQN; α, β denote the parameters of two sequences of FC layers, A and V respectively. However, (Equation 13) cannot be applied to aggregate V and A since it is not identifiable and unable to recover unique values of V and A. For instance, adding a constant to $V(s; \theta, \beta)$ and subtracting the same amount from $A(s, a; \theta, \alpha)$ will result the same $Q(s, a; \theta, \alpha, \beta)$. In this case, to handle the identifiability issue, an alternative equation involved with an average operator has been proposed:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|A|}\sum_{a'} A(s, a'; \theta, \alpha)\right).$$

(Equation 14)

Comparing with another aggregator combined with a max operator in [7], though (Equation 14) is off target by the constant of mean, it improves the stability, because rather than compensating the difference between optimal advantage, it is only supposed to change as fast as the average advantage of actions. In addition, using (Equation 14) will not modify the rank of A and Q and protect the ε-greedy policy based on Q values in (Equation 13). As a result, the state value V can be evaluated by the dueling networks, under the condition with a lack of the knowledge on how actions effect the state. By the experiment results, the dueling architecture performs outstandingly on some tasks where there are a large number of actions with the similar values. Furthermore, in theory, dueling networks can be applied to some existing RL algorithms that are combined with Q-networks such as Double DQN [11] and QMIX [1].

1.3.4. Multi-Agent Reinforcement Learning and QMIX

Multi-Agent Reinforcement learning (MARL) has extended the RL problems to some cooperative or competitive tasks involved with multiple agents, where a group of agents learn polices from the interaction with environment. This project mainly focuses on the settings where a group of agents perform cooperatively. An agent could get access to its own information of observation but cannot observe the entire state or environment, which is called the problem of Decentralized Partially Observable Markov Decision Process (Dec-POMDP) [4]. Specifically, the term decentralization indicates that agents will individually execute the policies without any communication to each other. Similar to 5-tuple MDP mentioned in Section 1.3.1, Dec-POMDP is formalized as $(S, U, P, r, Z, O, n, \gamma)$, where $s \in S$ represent the state of environment; an agent $a \in A \equiv 1, \dots, n$ takes an action $u_a \in U$ to form a joint action $u \in U \equiv U^n$; $P(s'|s,u): S \times U \times S \to [0,1]$ is the state transition function; $r(s,u): S \times U \to \mathbb{R}$ is the reward function shared by a team of agents; $z \in Z$ is an individual observation based on the observation function $O(s,a): S \times A \to Z$; $\gamma \in [0,1)$ is the discount factor. In addition, an individual stochastic policy $\pi^a(u^a|\tau^a): T \times U \to [0,1]$ is conditioned on an action observation history of each agent $\tau^a \in T \equiv (Z \times U)^*$.

The learning of decentralized policies according to local observation histories can often be done in a centralized manner. QMIX [1] is one of those cooperative MARL algorithms implemented the paradigm of CTDE with a critical concept of IGM consistency in which the CAF achieves the same result as a set of IAFs, following the equation:

$$\underset{u}{\operatorname{argmax}}\, Q_{tot}(\tau, u) = \begin{pmatrix} \underset{u^1}{\operatorname{argmax}}\, Q_1(\tau^1, u^1) \\ \cdots \\ \underset{u^n}{\operatorname{argmax}}\, Q_n(\tau^n, u^n) \end{pmatrix}.$$

(Equation 15)

Another one is the value decomposition networks (VDN) proposed in [12]. The CAF $Q_{tot}(\tau, u)$ in VDN has been represented by the sum of IAFs, $Q_a(\tau^a, u^a; \theta^a)$ for each agent a:

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \sum_{i=1}^{n} Q_i(\tau^i, u^i; \theta^i),$$

(Equation 16)

where $\boldsymbol{\tau} \in \mathbf{T} \equiv \mathcal{T}^{\mathrm{n}}$ is the joint action-observation history and $\mathbf{u}$ is the joint action.

QMIX is based on the idea that fully factorizing the value function in VDN is not mandatory to ensure the IGM, and instead it enforces a monotonic relationship between $\mathrm{Q_{tot}}$ and each $\mathrm{Q_a}$, which is denoted as:

$$\frac{\partial \mathrm{Q_{tot}}}{\partial \mathrm{Q_a}} \geq 0, \forall \mathrm{a} \in \mathrm{A}.$$

(Equation 17)

(Equation 17) is a sufficient but not necessary condition to satisfy the IGM consistency in (Equation 15), which extends the representation of CAF to a larger number of monotonic functions comparing to VDN. To realize the condition in (Equation 17), QMIX uses an architecture shown in Figure 5 to produce $\mathrm{Q_{tot}}$, which is structured by a mixing network, a set of agent networks and hypernetworks.

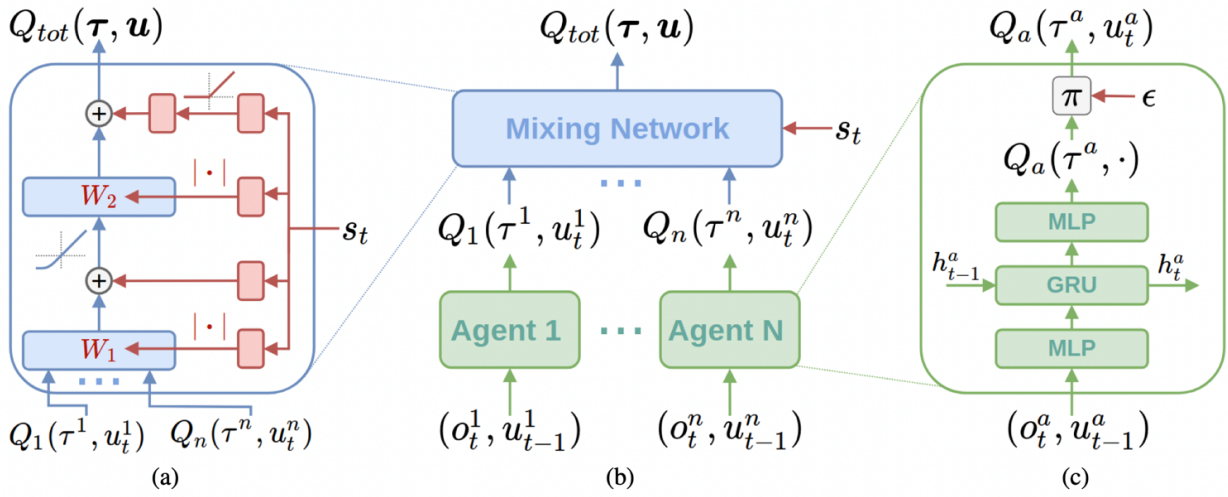


Figure 5

Shown by Figure 5(a), each agent network represents its IAF $\mathrm{Q_a}(\tau^{\mathrm{a}}, \mathrm{u^a})$ and uses the architecture of deep recurrent Q-network (DRQN) where a gated architecture GRU is constructed between two FC layers. The outputs of agent networks serve as inputs of the mixing network and are mixed monotonically to compute $\mathrm{Q_{tot}}$. According to Figure 5(b), the mixing network is a feed-forward neural network and to ensure the monotonicity in (Equation 17), its weights are enforced to be non-negative. The weights are produced via hypernetworks. The input of each hypernetwork is state $\mathrm{s_t}$ and the output is a vector containing the weights for one layer in mixing network. Besides, there is a single linear layer in

each hypernetwork which is followed with an absolute activation function to generate non-negative weights. Similar to the loss formula is (Equation 11), the update of QMIX follows the loss as below:

$$\mathrm{L}(\theta) = \sum_{\mathrm{i}=1}^{\mathrm{b}}[(\mathrm{r} + \gamma \max_{\mathrm{u}'} \mathrm{Q}_{\mathrm{tot}}(\boldsymbol{\tau}', \mathbf{u}', \mathbf{s}; \theta^{-}) - \mathrm{Q}_{\mathrm{tot}}(\boldsymbol{\tau}, \mathbf{u}, \mathbf{s}; \theta))^2].$$

(Equation 18)

According to experiments, QMIX performs competitively on StarCraftII micromanagement challenge benchmark.

2. Design

2.1. Algorithm

2.1.1. Multi-step Learning

QMIX updates the target by one-step TD method (Equation 7), so the extension, n-QMIX will utilize the multi-step learning algorithm (Equation 9), which has been designed as a function using the backwards induction. Formally, the derivation process is presented as follows:

First, compute the $G_{t+1:t+n}$ (Equation 19) via the formula of multi-step learning in (Equation 9):

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^{n} V_{t+n-1}(S_{t+n}),$$

(Equation 9)

$$G_{t+1:t+n} = R_{t+2} + \gamma R_{t+3} + \ldots + \gamma^{n-2} R_{t+n} + \gamma^{n-1} R_{t+n+1} + \gamma^{n} V_{t+n}(S_{t+n+1}).$$

(Equation 19)

Then, multiply both sides of equation (Equation 19) by γ and get:

$$\gamma G_{t+1:t+n} = \gamma R_{t+2} + \gamma^{2} R_{t+3} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^{n} R_{t+n+1} + \gamma^{n+1} V_{t+n}(S_{t+n+1}).$$

(Equation 20)

Subsequently, by (Equation 9) minus (Equation 20), there is,

$$\begin{aligned} G_{t:t+n} - \gamma G_{t+1:t+n} \\ = \gamma G_{t+1:t+n} + R_{t+1} + \gamma^{n} V_{t+n-1}(S_{t+n}) - \gamma^{n} R_{t+n+1} \\ - \gamma^{n+1} V_{t+n}(S_{t+n+1}). \end{aligned}$$

(Equation 21)

The final backwards induction formula that has been implemented in the coding is summarized as:

$$G_{t:t+n} = \gamma G_{t+1:t+n} + \gamma G_{t+1:t+n} + R_{t+1} + \gamma^n V_{t+n-1}(S_{t+n}) - \gamma^n R_{t+n+1} - \gamma^{n+1} V_{t+n}(S_{t+n+1}).$$

(Equation 22)

The detailed coding implementation will be fully discussed in Chapter 3 afterwards. Here is a simple pseudo code of the n-step learning function:

Algorithm: Build targets via multi-step learning

1: **Input:** rewards and estimated action-value Qs at each timestep
2: **Base case:** Initialize the last n returns
3: **For** each timestep $t \in [0, T - n - 1]$:
4: Compute the n-step return by
5: $G_{t:t+n} = \gamma G_{t+1:t+n} + \gamma G_{t+1:t+n} + R_{t+1} + \gamma^n V_{t+n-1}(S_{t+n}) - \gamma^n R_{t+n+1} - \gamma^{n+1} V_{t+n}(S_{t+n+1})$
6: **Output:** n-step returns for each timestep

Figure 6

2.1.2. Dueling Networks

Section 1.3.3 mentioned that dueling networks can be applied to other RL algorithms based on Q-networks. In addition, according to Section 1.3.4, the agent networks in QMIX are RDQNs. Thus, this project extends QMIX with dueling networks by applying the dueling architecture to the agent networks. Specifically, the last single steam of FC layer in an agent network is divided into two streams of layers to generate the state value V and advantages A respectively, which are subsequently aggregated to compute the final action-value Q.

In Figure 7, on the left is an agent network in QMIX, which has been modified according to the dueling architecture shown on the right. All the other components are as same as the non-modified version, except the last FC layer, which is separated into two steams that will be combined via the aggregating layer to produce Q_a. The code has been implemented via the Pytorch library and will be discussed in the Chapter 3. In addition, another QMIX extension, QPLEX also uses the underlying principles of dueling networks but is implemented in a different way. In testing stage, the performance of d-QMIX will be also compared with QPLEX in the easy-level task.

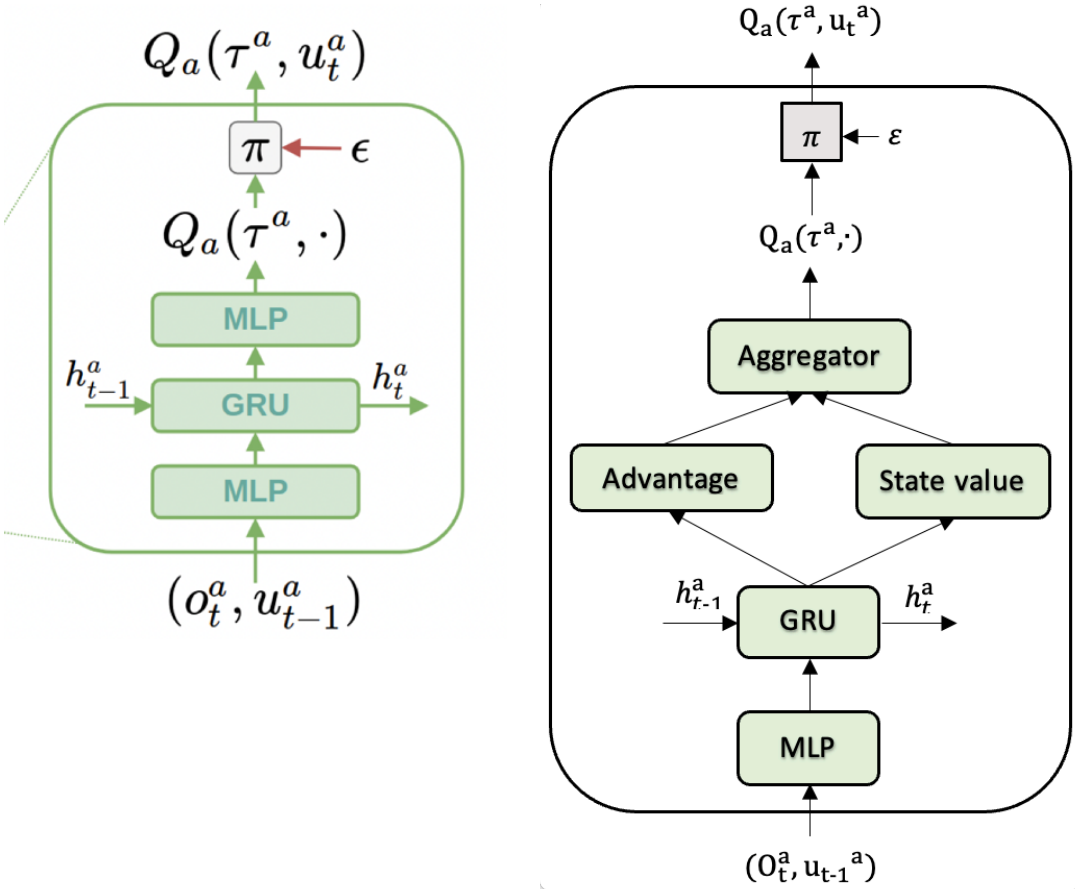
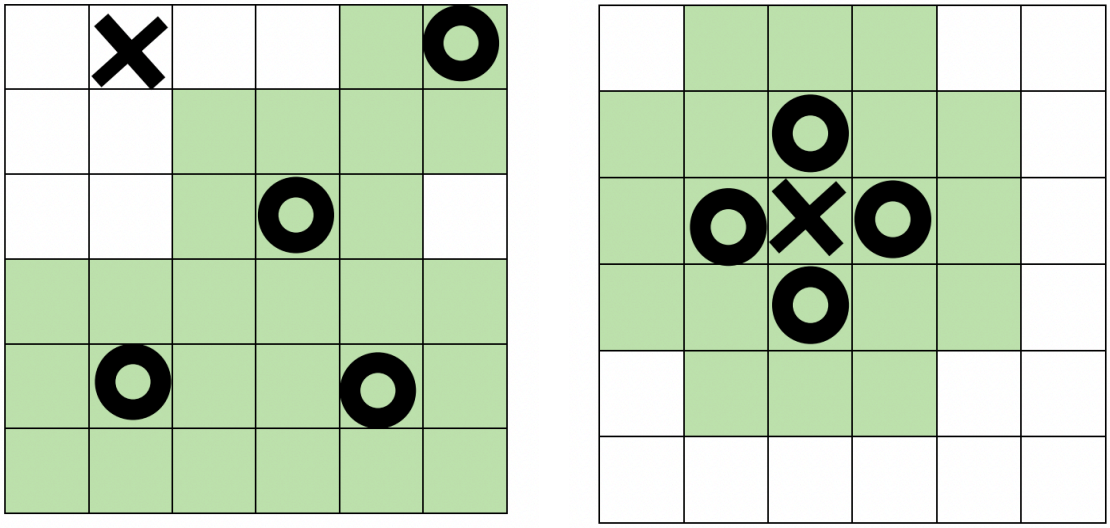

$Q_a(\tau^a, u_t^a)$
π
ϵ
$Q_a(\tau^a, \cdot)$
MLP
h_{t-1}^a
GRU
h_t^a
MLP
(o_t^a, u_{t-1}^a)
$Q_a(\tau^a, u_t^a)$
π
ε
$Q_a(\tau^a, \cdot)$
Aggregator
Advantage
State value
h_{t-1}^a
GRU
h_t^a
MLP
(O_t^a, u_{t-1}^a)

Figure 7

2.2. Testing environment

The performance of all the algorithms, QMIX, n-QMIX, d-QMIX and dn-QMIX will be tested based on the scenarios of cooperative predator-prey tasks [5] in two levels of difficulty. The detailed settings of each task will be introduced later in this section. Before testing, two hypotheses have been proposed in advance: first, the extensions n-QMIX and d-QMIX will perform better than QMIX, at least in the easy level task with smaller variances; second, as the n value in n-QMIX increases, the performance will be improved when n reaches the best tuned value, and then drop with a higher n value. Additionally, considering the issue of statistical significance and the sensitivity to the initial weights of Q-networks, at least three seeds have been run for each algorithm.

2.2.1. Easy level



(a) Initial State (b) Final State

Figure 8

The states of easy level predator-prey task, where circles represent predators, and the cross represents the prey. The observation area for each predator is painted as green.

The easy-level task involves four predators trying to capture one prey in a 6 by 6 grid-world (Figure 8(a)). Each agent can move right and left, go up and down, or stay at the same position. Each predator can observe a 3 by 3 square centered

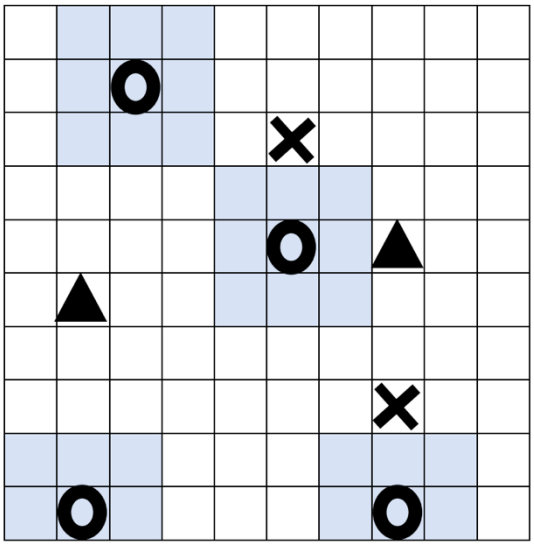
on itself. There are two situations where a prey is captured, one of which is that a prey is surrounded by four predators and cannot move, shown in Figure 8 (b). Another is that two predators execute the catch action simultaneously to capture the target prey. Both situations need the cooperation between predators who are trained via different algorithms to learn the cooperative task. The team of predators will be rewarded by 10 points if the prey has been captured. Furthermore, the team of predators will receive a punishment -0.1 at each timestep, which motives predators to capture the prey faster under the goal of maximizing the total reward. Also, the episode limit, or the maximum time steps per episode is 50, which indicates if predators cannot capture the prey within 50 steps, the game will be forced to terminate. The total timestep for one seed is 200,000. Here is a table of hyperparameters, where all the other parameters not mentioned below, will follow the default setting.

Predator	4
Prey	1
Reward	10
Capture conditions	Prey = 2 (The number of predators that must execute "catch" action at the same time)
World shape	[6,6]
Agent observation	[2,2] (A 3x3 grid world centered on an agent)
Total timestep	200,000
Episode limit	50
Punishment per timestep	-0.1

Table 1

2.2.2. Hard level

The hard-level predator-prey task is analogous to the easy-level but has some different parameter settings. It involves a 10 by 10 gird-world, where 4 predators trying to capture 4 preys. The preys are divided into two categories: stags and hares. A stag or hare is defined to be captured if it is surrounded by four predators. Also, similarly, another capture situation for the stag is that two predators execute the catch action at the same time, while capturing a hare only involves one predator to execute the catch action.



The graph of hard level predator-prey task, where circles represent predators; crosses represent stags and triangles represent hares. The observation area for each predator is marked as blue.

Figure 9

Furthermore, the rest possibilities which represent the chances that a stag or hare will not move at each step are 0.1 and 0.5 respectively. Considering the difficulty of cooperation, 10 points will be given to the predators if a stag has been captured and 1 point rewarded for a hare. Besides, different values of the episode limit have been tested for each algorithm. The total timestep for one run is 2,000,000. Here is a table of hyper-parameters for the hard-level task:

Predator	4
Prey	2 stags and 2 hares
Reward	reward_stag = 10, reward_hare = 1
Rest possibility	p_stag = 0.1, p_hare = 0.5 (Possibility that a stag or hare will not move at each step)
Capture conditions	stag = 2, hare = 1 (Number of predators that must simultaneously execute “catch” action)
World shape	[10,10]
Agent observation	[2,2] (A 3x3 grid world centered on an agent)
Total timestep	2,000,000
Episode limit	300/200/100/50
Punishment per timestep	-0.1

Table 2

3. Implementation

3.1. Pymarl framework

The extensions of QMIX are coded based on Pymarl [9], an open-source deep MARL framework written in Pytorch, where the original QMIX algorithm has already been implemented. Under the directory **./src/config/algs**, there is a file called **qmix.yaml**, which describe some specific parameters of QMIX. This project uses the default setting: use epsilon greedy policy to select actions; update the target network every 200 episodes; train agents by Q_learner. The **q_learner.py** is in **./src/learners**, which is the key part to be extended with the multi-step learning method. In the directory of **./src/modules/agents**, there is a file of the agent network, **rnn_agent.py**, which is the place to be combined with dueling architecture. In addition, the predator-prey environment file, **stag_hunt.py** is in **./src/config/envs** borrowed from [13], where some hyperparameters have been adjusted to create different levels of games. To run the code, first build the Docker file following the instructions in terminal:

```
cd docker
bash build.sh
```

Then running an experiment follows:

```
python3 src/main.py –config=qmix –env-config=stag_hunt
```

The results are be stored in in the folder of Results, where the **info.json** file includes the critical data to analyze the performance intuitively by plotting graphs.

3.2. Algorithm implementation

3.2.1. Multi-step Learning

According to Section 2.1.1, multi-step learning method has been designed to substitute the one-step TD method in QMIX to calculate targets, following the principles of backwards induction. Figure 10 is a screenshot of n-step learning function implemented in the q_learner file:

```
def build_n_targets(self, rewards, terminated, mask, target_qs, gamma, n):

    # Base cases: Initialise the last n multi-step returns
    ret = target_qs.new_zeros(*target_qs.shape)
    for t in range(n,0,-1):
        ret[:, -t] = target_qs[:, -t] * (1 - th.sum(terminated, dim=1))

    # Backwards update  of the multi-step learning target in each step
    for t in range(ret.shape[1] - n - 1, -1, -1):
        ret[:, t] = gamma * ret[:, t + 1] + mask[:, t] * \
                    (rewards[:, t] + (gamma ** n) *
                     (target_qs[:, t + n] - rewards[:, t + n] - gamma * target_qs[:, t + n]) * (1 - terminated[:, t]))

    # Returns n step return from t=0 to t=T-1
    return ret
```

Figure 10

where the parameter **rewards** denote a set of rewards gained at each timestep; the two parameters, **terminated** and **mask** are passed to handle the last n returns mentioned in Section 1.3.2, which are defined to be equivalent to the full-step return in Monte Carlo; **target_qs** stores the estimated state-value at each step; **gamma** is the discount factor γ and **n** is the n value set for n-step learning function. The whole structure is the same as the pseudo code in *Figure 6*, where the last n multi-step returns are initialized as base cases firstly, and then other n-step targets are updated via backwards induction.

```
# Calculate 1-step Q-Learning targets
#targets = rewards + self.args.gamma * (1 - terminated) * target_max_qvals

# Calculate n-step Q-Learning targets
n = 2
targets = self.build_n_targets(rewards, terminated, mask, target_max_qvals, self.args.gamma, n)
```

Figure 11

As shown in Figure 11, QMIX calculates targets by one-step TD method, while n-QMIX utilizes the multi-step learning function constructed in Figure 10 to produce targets, where the n value can be tunned manually.

3.2.2. Dueling Networks

The dueling architecture is combined in the agent network that has been implemented in **rnn_agent.py** shown in Figure 12 following the structure in Figure 7.

```
class RNNAgent(nn.Module):
    def __init__(self, input_shape, args):
        super(RNNAgent, self).__init__()
        self.args = args

        self.fc1 = nn.Linear(input_shape, args.rnn_hidden_dim)
        self.rnn = nn.GRUCell(args.rnn_hidden_dim, args.rnn_hidden_dim)
        #self.fc2 = nn.Linear(args.rnn_hidden_dim, args.n_actions)
        self.fc2 = nn.Linear(args.rnn_hidden_dim, args.n_actions)
        self.fc3 = nn.Linear(args.rnn_hidden_dim, 1)

    def init_hidden(self):
        # make hidden states on same device as model
        return self.fc1.weight.new(1, self.args.rnn_hidden_dim).zero_()

    def forward(self, inputs, hidden_state):

        x = F.relu(self.fc1(inputs))
        h_in = hidden_state.reshape(-1, self.args.rnn_hidden_dim)
        h = self.rnn(x, h_in)
        #q = self.fc2(h)

        a = self.fc2(h)
        v = self.fc3(h)

        n_actions = a.shape[1]
        average = th.mean(a, dim=1, keepdim=True, out=None)
        average = average.repeat(1, n_actions)
        v = v.repeat(1, n_actions)
        q = v + a - average

        return q, h
```

Figure 12

The former last FC layer is separated into two streams of FC layers: **self.fc2 = nn.Linear(args.rnn_hidden_dim, args.n_actions)** represents advantages A and **self.fc3 = nn.Linear(args.rnn_hidden_dim, 1)** stands for state value V. Then the outputs of those layers, **a = self.fc2(h)** and **v = self.fc3(h)** are aggregated via **q = v + a - average**, instead of computing Q value directly as **q = self.fc2(h)** commented in function **def forward(self, inputs, hidden_state)**.

4. Testing and evaluation

The performance of those extensions has been tested in two levels of predator-prey tasks using the performance of QMIX as the baseline. To ensure the statistical significance, five seeds of easy-level task and three seeds of hard-level task were counted. Besides, three criteria are taken into consideration, namely absolute performance, stability, and learning rate, among which the absolute performance uses the final converged value of return as the indicator; stability is evaluated via variance shown by the shaded area; learning rate is relevant to the convergent speed which denotes how fast the agents are learning the policies. Furthermore, the data was gathered following the procedure, where pause the training every 2000 timesteps to run 20 tests and take the average of returns. The results are analyzed by PyCharm, and figures are generated via matplotlib, where x-axis and y-axis represent timestep and return mean respectively.

4.1. Easy level

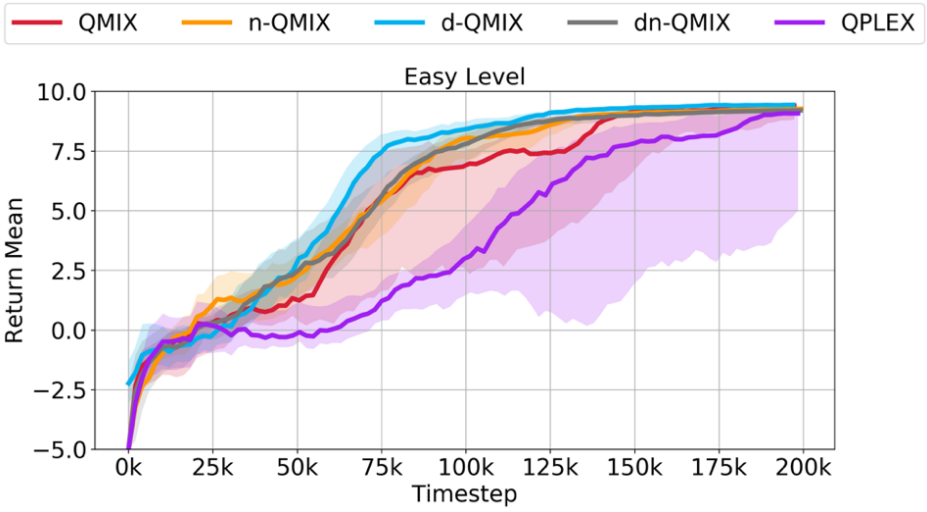


Figure 13

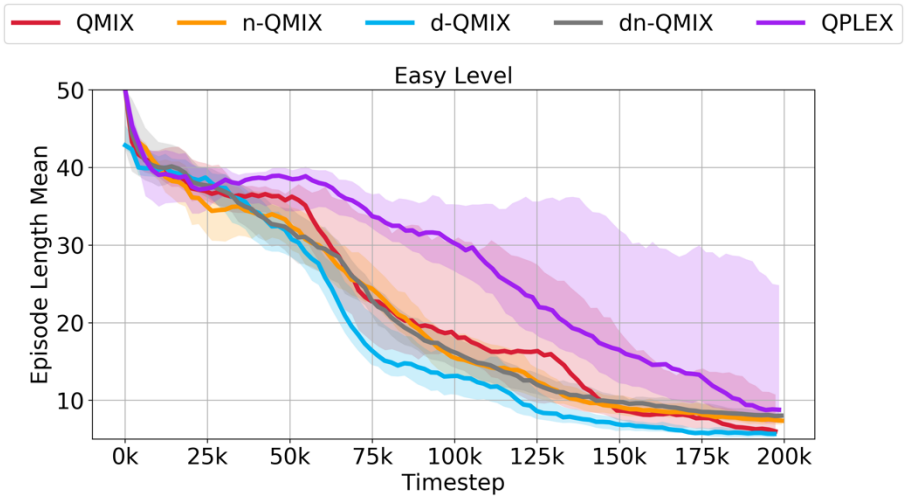


Figure 14

Three extensions of QMIX, n-QMIX, d-QMIX and dn-QMIX have been tested in easy-level task, using QMIX and another MARL algorithm also combined with dueling architecture, QPLEX [3] as the baselines. In general, d-QMIX achieves the best performance among all the algorithms, as is shown in Figure 13. Besides, though compared with QMIX, n-QMIX and dn-QMIX have very close final values, it can still be concluded that these extensions perform slightly better than QMIX with faster convergence speeds. Regarding the variance, QMIX and QPLEX have extremely high variances, while all the other extensions achieve relatively low variances, which is consistent with the first hypothesis in Section 2.2. Thus, we can generally conclude, in the easy-level, three extensions outperform QMIX and QPLEX. In addition, in Figure 14, the final optimal episode length converges to 5 steps, contributed by d-QMIX. In theory, considering that the prey-captured reward given to the predators is 10, the final optimal value should be 10 – 0.1x5 = 9.5, which consists with the result in Figure 13.

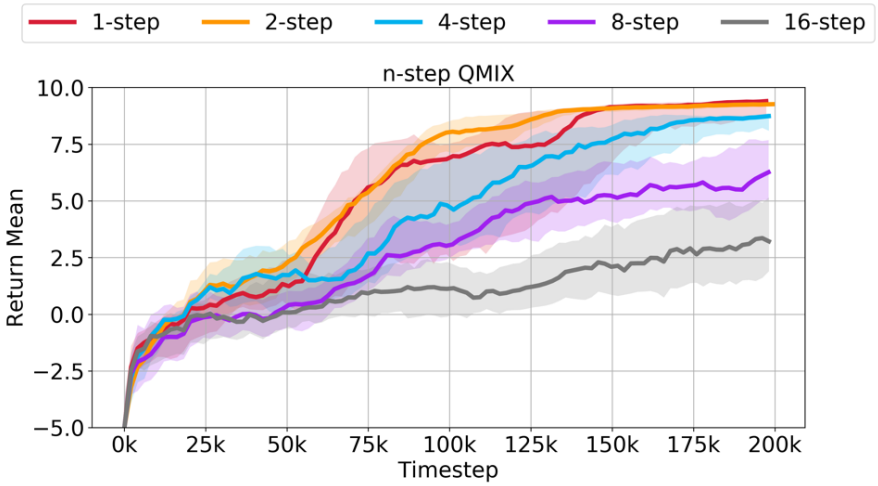


Figure 15

The hyper-parameter n in n-QMIX is a crucial value to address the issue of bias-variance trade-off, so different values of n have been tested, intended to find the most suitable n. In Figure 15, the red curve is the original QMIX using one-step learning, comparing to which only 2-step QMIX outperform it with the faster learning rate and higher stability. As the n value increases from 2, the performance drops gradually, and the variance suggested by the shading area also becomes higher, which is consistent with the second hypothesis. Therefore, the best tuned n is equal to 2. Based on this knowledge, only the performance of 2-QMIX in the hard-level task has been tested, since the time for this project is very limited.

4.2. Hard level

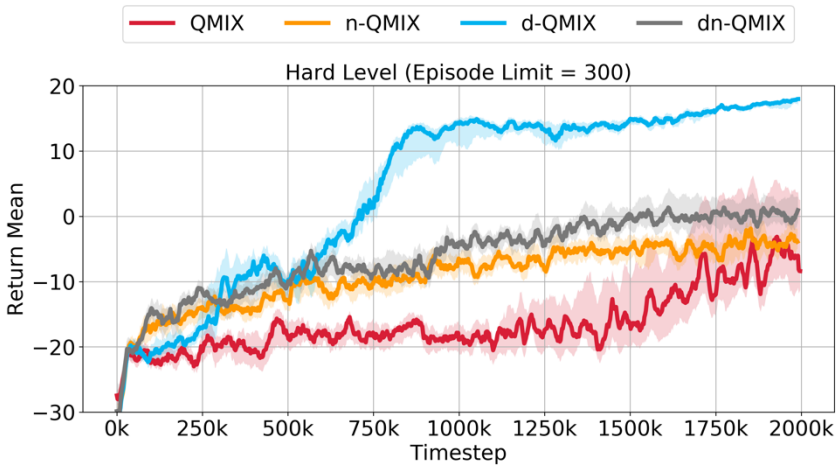


Figure 16

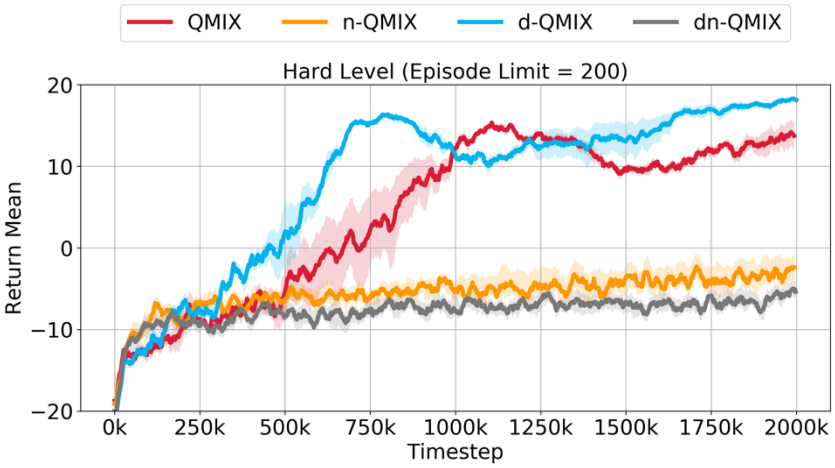


Figure 17

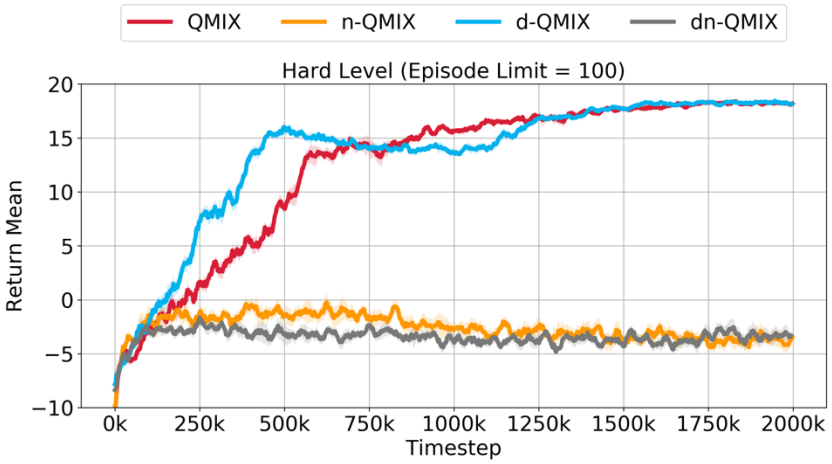


Figure 18

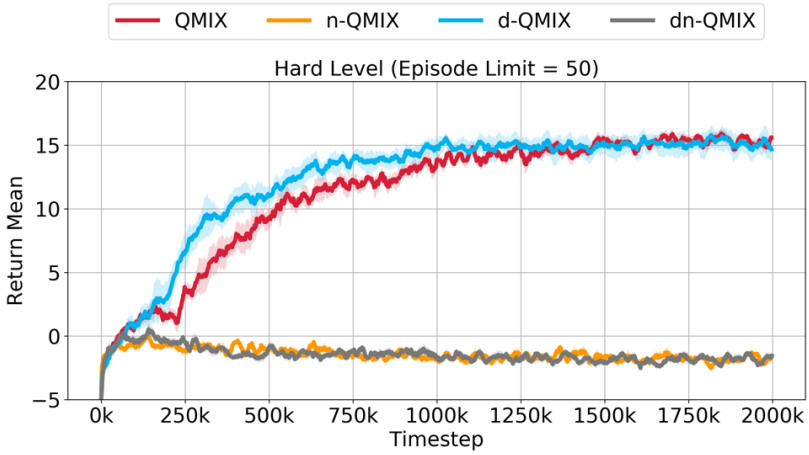


Figure 19

For hard-level predator-prey task, the performance of three extensions, n-QMIX (n=2), d-QMIX and dn-QMIX have been tested and compared with QMIX in terms of different training strategies, that is, using different episode limits. The episode limit is the maximum timestep per episode, which indicates if the agents cannot finish the cooperative task within certain timesteps, the game will be forced to end. It is an important hyper-parameter to be adjusted for the training setting, which may influence the learning rate and sample efficiency.

As the graphs shown above, when episode limit = 300, the d-QMIX outstandingly performs better than any other algorithms in the hard-level task. When episode limit = 200, 100 and 50, the d-QMIX still achieves the best performance, but it is not that obvious comparing to QMIX. n-QMIX and dn-QMIX nearly fail the task in all training strategies. Though they slightly outperform QMIX when episode limit = 300, their better performance does not count, due to failing the task as well. Also, for other episode limits, n-QMIX and dn-QMIX perform much worse than QMIX.

In summary, using d-QMIX with episode limit = 100 is the best training strategy to finish the hard-level task, based on which the agents can learn to reach the theoretical global value in a faster manner. In theory, the idea maximum total reward should be 10x2 + 1x2 = 22 if 4 preys have been captured at timestep 0. However, it is obviously impossible, and according to Figure 20, the episode length converges to around 40 steps. Thus, the theoretical final value should be 10x2 + 1x2 – 0.1x40 = 18. As shown in Figure 16, Figure 17, Figure 18 and Figure 19, for every episode limit except 50, trained by d-QMIX, the total reward converges to 18.

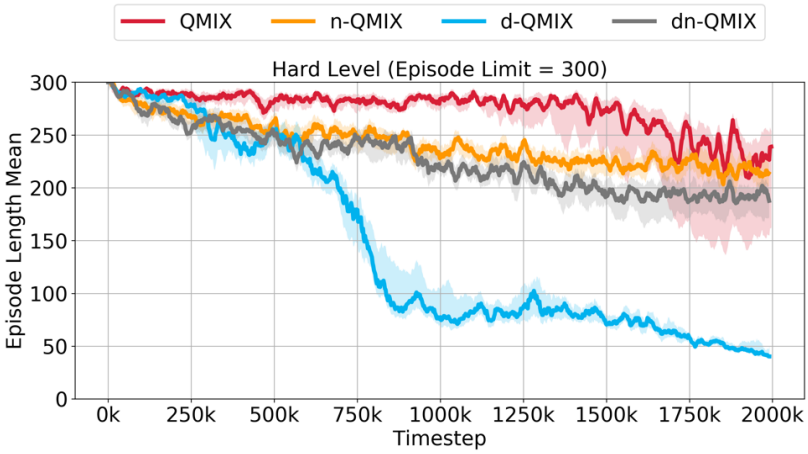


Figure 20

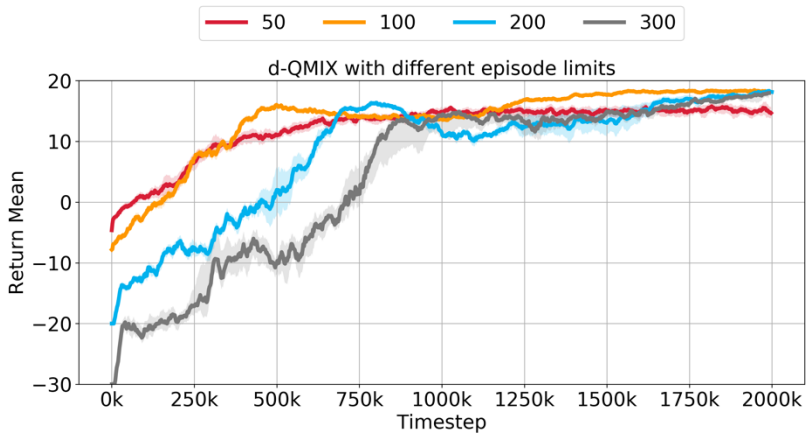


Figure 21

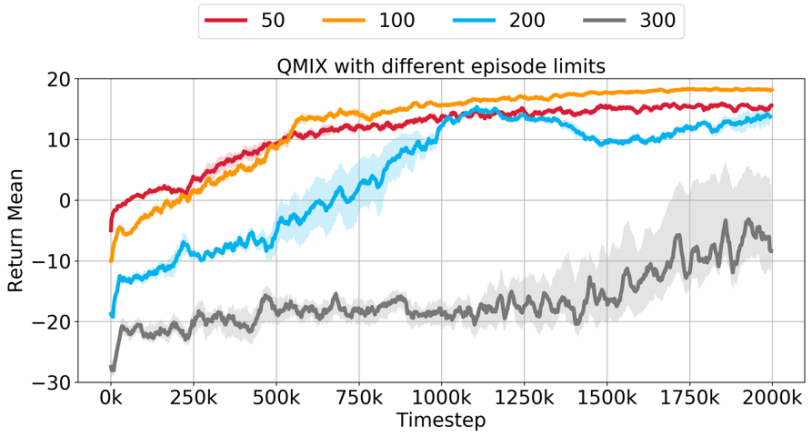


Figure 22

Apart from general result analysis, there are two interesting issues worthy of noting. The first is that the performance of the original QMIX drops dramatically when episode limit = 300 (Figure 22). In theory, given a larger training episode limit, the agents can explore and observe more in the grid world. Also, the punishment received per each step will stimulate agents to capture the prey. In other words, though a larger maximum timestep per episode does not necessarily indicate a better final converged value or a faster learning speed, it should not damage the performance severely. In contrast, the performance gap between each episode limit of d-QMIX in Figure 21 is a reasonable result of tuning hyperparameter.

An assumption has been made to explain this phenomenon, that is, the episode limit might be a part of the environment that an agent can observe and would influence the action selecting strategy. Based on the design of QMIX implemented in Pymarl, it does not set the episode limit as a factor of the environment. The assumption does not make sense. However, recent research on time limits in RL suggests that if taking the timesteps left in an episode as a parameter of input to calculate the state value, an agent will thus be aware of the remaining time and would perform better in some time-limited tasks. In paper [14], it discusses a simple task called The Last Moment problem, in which there is a Markov Decision Process model consisting of two states as shown in Figure 23. When an agent is at the initial state A, it has two actions to choose: staying at state A with no rewards or jumping to state B with 1 point. However, if an agent jumps to state B before the last step, it will fall into a trap and keep getting punishment. Given a fixed timestep T, if an agent is time-unaware, it is hard to master the task when T>1, and obviously, the best strategy in this case should be staying at A. However, if an agent could notice how many steps remain in current episode, it will directly learn the optimal strategy, that is, staying at A until step T-1 and jump to B at step T. Regarding this promising idea of taking the remaining time as a part of the environment that an agent can observe in single agent settings, it could be extended to some multi-agent tasks and combined with QMIX.

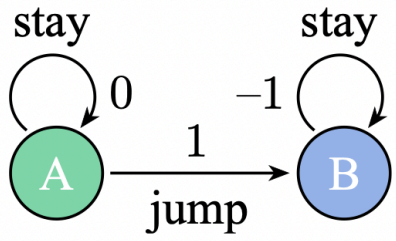


Figure 23

Imagine a modified predator-prey task, where instead of using a general punishment rule at each timestep, an agent will only be punished, if it moves rather than stay. This setting is reasonable because in real-life, normally, staying still consumes less resource than moving around. Thus, if an agent can realize that the remaining time has become limited in the episode, the best strategy is staying at current place and waiting for a prey to come, rather than moving around and being punished.

Another reasonable assumption is that there might be a maximum learning timestep in an episode for QMIX, which could be defined as *timestep threshold* for simplicity. If the current timestep exceeds the timestep threshold, an agent will not learn to capture the prey, but move around and get punishment. For example, if the timestep threshold is 100 for QMIX, and the episode limit set in advance is 300, at the first 100 steps, the agent will learn to capture preys with the goal of maximizing the total reward. After that, they reach the state of saturation and would not learn to finish the task but keep being punished at each step. Therefore, this could be a possible reason why QMIX fails the task if episode limit = 300. In addition, comparing to the setting with episode limit = 100, the average episode length of episode limit = 300 is longer, so given a fixed total number of timesteps in a run, there will be less episodes for training. According to the experiment results, within 2,000,000 total steps, the total episode number for episode limit = 300 is 8,572 averaged by three seeds, and 35,355 for episode limit = 100. Therefore, in this case, we hypothesize that if more total timesteps have been given to ensure enough amount of training episodes, the total return of episode limit = 300 will finally converge to the same value as episode limit = 100. To prove the hypothesis in future work, when running experiments of hard-level tasks, we can cancel the punishment per timestep, and fix the total training episode number, instead of the total timestep. Furthermore, since based on current experiments, the d-QMIX is not sensitive to training episode limit at least in the interval of [50,300], we could raise another hypothesis, that is, the timestep threshold for d-QMIX is higher, and if there is a task involved with more theoretical learning steps, the performance of d-QMIX will be much better than QMIX. For example, if the grid-world size has been adjusted from [10,10] to [15,15], there will be more room for preys to move around, which cause a longer total timestep for predators to finish the task.

The second issue is why d-QMIX can achieve a general better performance than QMIX, and especially when episode limit = 300, why its performance will not sharply drop as QMIX. Mentioned in the Section 1.3.3, the dueling architecture contributes to more accurate state values in those states where whatever action

an agent takes, the environment will not be influenced, because it computes the state value and advantages separately. Additionally, if there are a relatively large number of similar actions in a task, the dueling network can evaluate the policy faster and recognize the best action to take. The above conclusions drawn in [7] can be applied to explain the superior performance of d-QMIX in the hard-level predator-prey task. At the beginning of training, it is difficult for predators to capture a prey. Thus, taking any action will not affect the environment because no prey has been captured. We refer all these actions as similar actions since the state value will remain the same. In this case, d-QMIX can find better policies especially at the start of training. According to the hypothesis of timestep threshold in previous paragraph, because of its superiority of handling states where actions do not affect the environment, the theoretical number of maximum learning timesteps per episode of d-QMIX is larger than QMIX. Therefore, it performs stably in terms of different episode limits.

4.3. Further analysis and summary

In summary, the three extensions of QMIX have successfully contribute to the improvement of performance to varying degrees, among which d-QMIX is the most remarkable one. In easy-level predator-prey task, d-QMIX outperforms QMIX particularly in stability and learning rate. The capabilities of identifying the best strategy and reaching the optimal value are critical when evaluating the performance. However, if an agent can be trained to finish a task in a faster speed with low variance, it suggests less learning resource to consume and lower risk to afford, which is also of vital importance. Those two advantages especially the stability is powerful when addressing some real-world problems. For example, there is a sea rescue task, where four lifeguards receive a message that there is a drowning person in a bounded area. The person is drifting in the sea randomly with a life ring. The goal of the rescue team is to find the person within a fixed timestep, because as the time goes by, the drowning man will suffer from the lack of physical strength and then die. Also, at least two lifeguards should execute the rescue action simultaneously to save the person’s life.

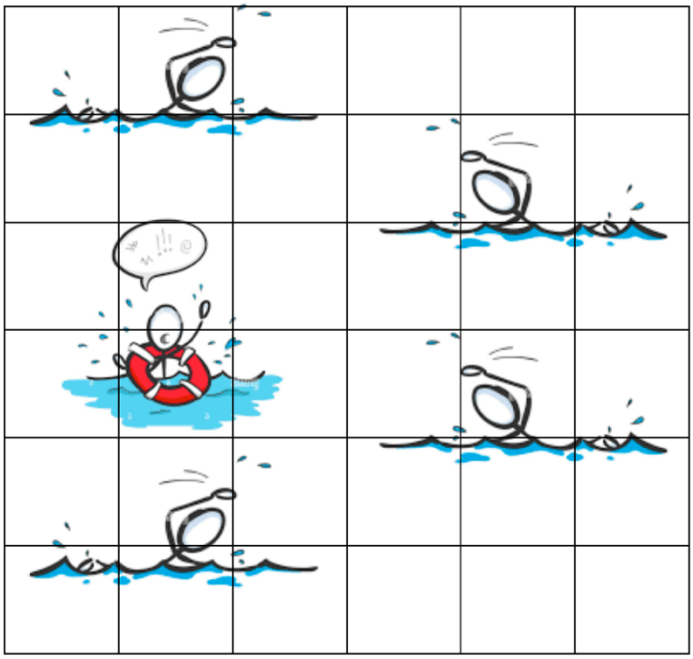


Figure 24

Because of some similar environment settings, this task can be roughly mapped to our easy-level predator-prey task shown in Figure 24. Based on the results in Section 4.1, if the rescue team is trained by QMIX, the performance is highly unstable compared with d-QMIX. Thus, there is a higher risk of not finishing the task. However, unlike losing points if agents fail to complete the predator-prey task, in this sea rescue setting, the consequence of not finishing the task is losing a human's life, which is unaffordable. In other words, failing the rescue task is not only an issue of missing the reward, but causes a significant punishment. In addition, comparing to another algorithm also using the dueling architecture, QPLEX, statistically, there is a 20% probability that the person has not been rescued and a 40% chance that the person has not been rescued within a given timestep, which is 20. 40% of losing a human's life is an extremely high risk. Thus, the superior stability of d-QMIX plays an irreplaceable role in real-world settings where the failure of task indicates serious consequences.

5. Conclusion

In summary, this project proposed three extensions of QMIX based on two techniques, namely multi-step learning and dueling networks. The performance was evaluated in two levels of predator-prey tasks, using QMIX and/or QPLEX as the baseline. Results showed that those extensions can improve both absolute performance and stability in some degree. Specifically, at easy level, n-QMIX, d-QMIX and dn-QMIX outperformed QMIX and QPLEX with lower variances, among which d-QMIX achieves the best performance. In addition, the best tuned n for n-QMIX was experimented to be two, and as the value became larger, the performance dropped gradually. At hard level, different maximum training timesteps per episode have been tested and analyzed, which showed that d-QMIX achieved the best performance in all settings, while n-QMIX and dn-QMIX only outperformed QMIX when episode limit was three hundred. Furthermore, projecting a real-world setting, the sea rescue to a predator-prey task, we strongly argued that the advantageous stability of d-QMIX is powerful and nonnegligible, which would contribute to solving more complicated cooperative tasks in a stable and efficient manner. In future work, 1) some other techniques used to extend single-agent RL algorithm in [15] could be explored in different settings; 2) the performance of the most promising extension in this project, d-QMIX could be tested in some other cooperative tasks such as StarCraft II; 3) the idea of taking the remaining time as the input to calculate the state value to solve some multi-agent time-limited tasks could also be explored.

6. BCS Project Criteria and Self-Reflection

6.1. BCS Project Criteria

An ability to apply practical and analytical skills gained during the degree programme.	In this project, the practical skills have been enhanced when implementing the code of those extended algorithms, which is shown in Section 3. The analytical skills have been improved in literature reviewing (Section 1.3) and result analysis (Section 4).
Innovation and/or creativity.	The innovation of this project reflects on combining some techniques to a deep MARL algorithm, intended to improve the performance and stability.
Synthesis of information, ideas, and practices to provide a quality solution together with an evaluation of that solution.	This criterion has been satisfied during the whole process of the project. The preparation stage including proposal writing and background reading, gathered information, ideas and practices. The implementation stage reflected how to solve the proposed issue. The evaluation stage provided a testing process of the solution.
That your project meets a real need in a wider context.	Firstly, it contributes to the research on deep MARL algorithms and provides innovative ways to extend QMIX. Secondly, it projects the sea rescue to

	the predator-prey task, which reflects a real-world issue that is worthy of noting.
An ability to self-manage a significant piece of work.	This ability has been improved in conducting and carrying out a plan, as well as the time management during this project.

Table 3

6.2. Self-Reflection

I have finished a research project on deep multi-agent reinforcement learning as my final year project. Generally speaking, I am really satisfied with what I have done and achieved so far. Firstly, I read a large number of literatures related to the research field of my project, such as RL, DQN and QMIX. Secondly, I familiarized myself with the Pymarl framework and mastered Pytorch library. In addition, I implemented the code of newly designed algorithms based on Pymarl and tested their performance on different levels of predator-prey tasks. In the end, I analyzed the results and discussed some interesting issues, as well as draw some conclusions.

The testing and evaluation stage is the part that I enjoyed most. Though it is a really time-consuming process, in which a testing seed might take me 8 hours to run, I appreciate the sense of achievement when I finally obtained some promising results. Also, I did enjoy designing and exploring different levels of predator-prey tasks, in which I could observe the performance of algorithms in different settings. In addition, I am so happy that I could fully engage in the process of result analysis, where I searched and read some latest published papers to analyze some phenomena and obtain some interesting ideas.

There were two aspects which troubled me in this project. First is that initially, I spent quite a long time to learn the Pytorch and get familiarized with Pymarl, because these complicated frameworks are hard to get started at the beginning. Second is when running some experiments, I will be highly intense and cannot stop thinking whether this extension would successfully outperform QMIX in this new setting. Sometimes, I would wake up in the mid-night and get up to check

the results. However, I believe I have successfully overcome those difficulties and really appreciate the progress that I have made in the end.

To sum up, I have greatly enhanced both my research skills and self-management skills during the final year project, and my contributions are as followings: 1) Proposed extensions of a deep MARL algorithm, QMIX. 2) Designed two levels of predator-prey tasks to test the performance of those extensions. 3) Analyzed the results and explored some real-world issues that could be reflected via predator-prey tasks. In the future, I will explore some other extensions of QMIX and investigate more interesting cooperative multi-agent tasks.

7. References

[1] R. Tabish, S. Mikayel, S. d. W. Christian, F. Gregory, F. Jakob and W. Shimon, "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning," in *International Conference on Machine Learning*, Stockholm, 2018.

[2] F. A. Oliehoek, M. T. Spaan and N. Vlassis, "Optimal and Approximate Q-value Functions for Decentralized POMDPs," *Journal of Artificial Intelligence Research,* vol. 32, pp. 289-353, 2008.

[3] Wang, Jianhao; Zhizhou, Ren; Terry, Liu; Yang, Yu; Chongjie, Zhang, "QPLEX: Duplex Dueling Multi-Agent Q-Learning," in *International Conference On Learning Representations*, 2021.

[4] F. A. Oliehoek and C. Amato, "A Concise Introduction to Decentralized POMDPs," *Springer,* vol. 1, 2015.

[5] P. Stone and M. Veloso, "Multiagent Systems: A Survey from a Machine Learning Perspective," *Autonomous Robots,* vol. 8, pp. 345-383, 2000.

[6] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Cambridge: The MIT Press, 2020.

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

[8] Mnih et al., "Human-level control through deep reinforcement learning," *Nature,* vol. 518, 2015.

[9] M. Samvelyan, T. Rashid and C. Schroeder, "The StarCraft Multi-Agent Challenge," *CoRR,* vol. abs/1902.04043, 2019.

[10] C. J. Watkins and P. Dayan, *Machine Learning,* vol. 8, no. 3-4, pp. 279-292, 1992.

[11] C. J. Maddison, A. Huang, I. Sutskever and D. Silver, "Move Evaluation in Go Using Deep Convolutional Neural Networks," in *International Conference on Machine Learning*, 2015.

[12] Sunehag. et al., "Value-Decomposition Networks For Cooperative Multi-Agent Learning," in *International Conference on Autonomous Agents and Multi-agent Systems*, 2017.

[13] T. Rashid, G. Farquhar, B. Peng and S. Whiteson, "Weighted QMIX: Expanding Monotonic Value Function Factorisation," in *Advances in Neural Information Processing Systems*, 2020.

[14] P. Fabio, T. Arash, L. Vitaly and K. Petar, "Time Limits in Reinforcement Learning," in *International Conference on Machine Learning*, Stockholm, 2018.

[15] Hessel. et al., "Rainbow: Combining Improvements in Deep Reinforcement Learning," *CoRR,* vol. abs/1710.02298, 2017.

8. Appendices

8.1. DQN

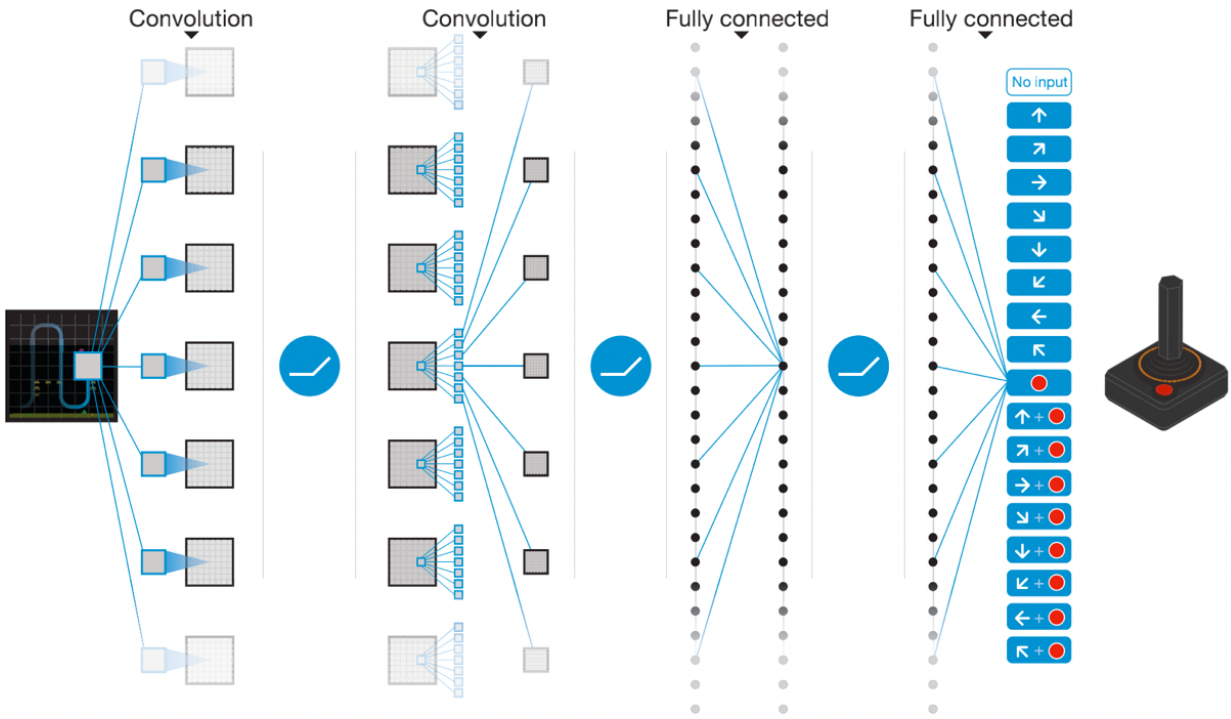


Figure 25

The deep convolutional neural network used in DQN

The input of the architecture in Figure 25 is an image with the size of 84 x 84 x 4, preprocessed by a map $\emptyset$. Three convolutional layers and two FC layers subsequently follow the input layer. For each valid action, there is only a single output. Each hidden layer is followed by a 'rectifier nonlinearity', which in other words, is the function of max (0, x).

8.2. QPLEX

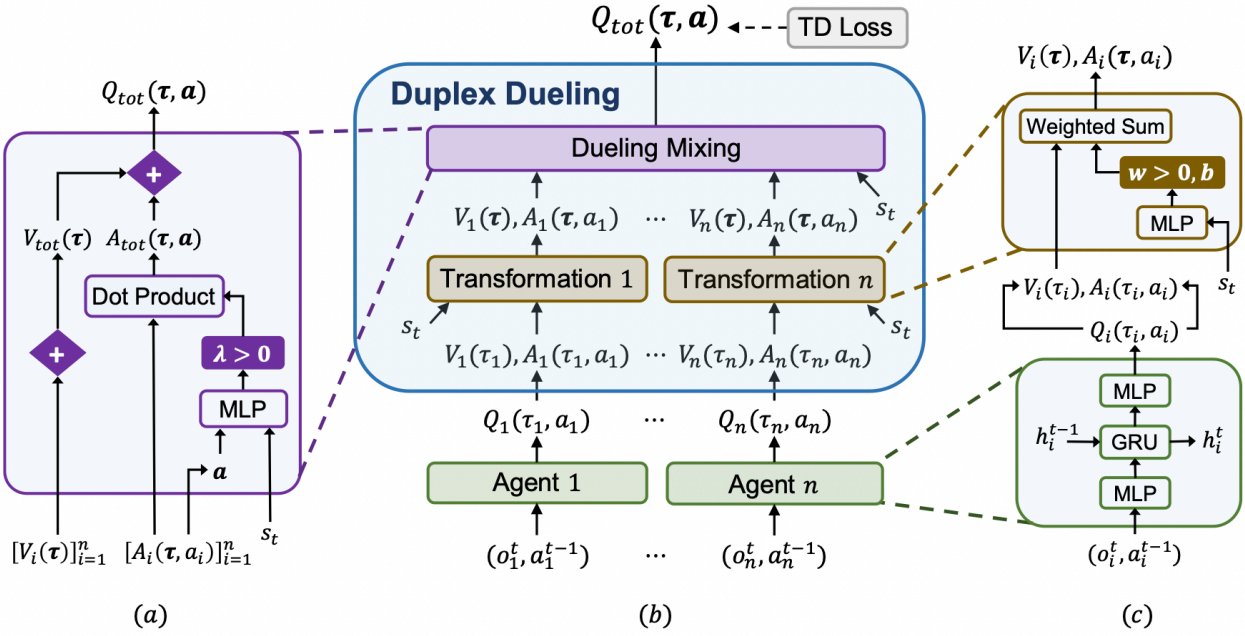


Figure 26

Duplex dueling multi-agent Q-learning (QPLEX) is a MARL method proposed in [3], which constructs a structure using duplex dueling networks to factorize the CAF into IAFs. Specifically, it follows the equation: Q = V + A [7] to represent both CAF and IAFs and enhances the IGM consistency to an advantage-based IGM, which ensures:

$$\underset{a \in A}{\operatorname{argmax}} A_{tot}(\boldsymbol{\tau}, \boldsymbol{a}) = \begin{pmatrix} \underset{a^1 \in A}{\operatorname{argmax}} A_1(\tau^1, a^1) \\ \cdots \\ \underset{a^n \in A}{\operatorname{argmax}} A_n(\tau^n, a^n) \end{pmatrix},$$

(Equation 23)

where the action-dependent advantage a ∈ A and the joint-action observation history $\boldsymbol{\tau} \in \mathbf{T} \equiv \mathcal{T}^n$. In Figure 26, (a) presents the structure of dueling mixing network; (b) is the high-level architecture of QPLEX; (c) illustrates the agent network as well as the transformation network.

8.3. Additional testing results

Figures below present some additional results for the hard-level predator-prey tasks, where Figure 27, Figure 28 and Figure 29 respectively show the curves of episode length mean for each algorithm in terms of different episode limits within 2,000,000 timesteps.

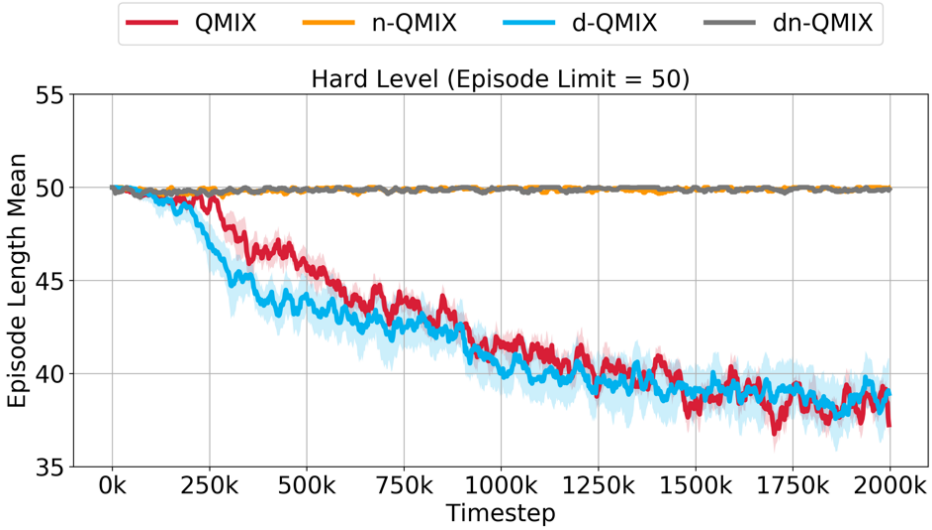


Figure 27

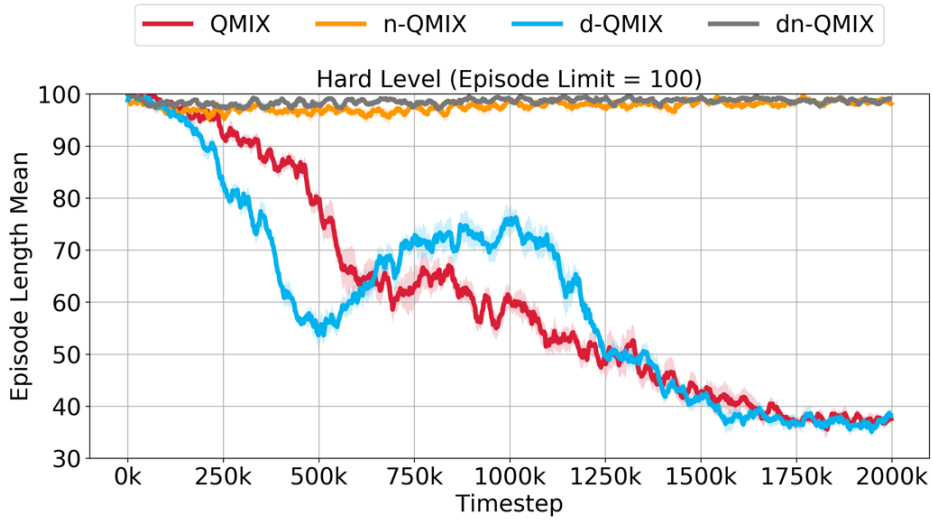


Figure 28

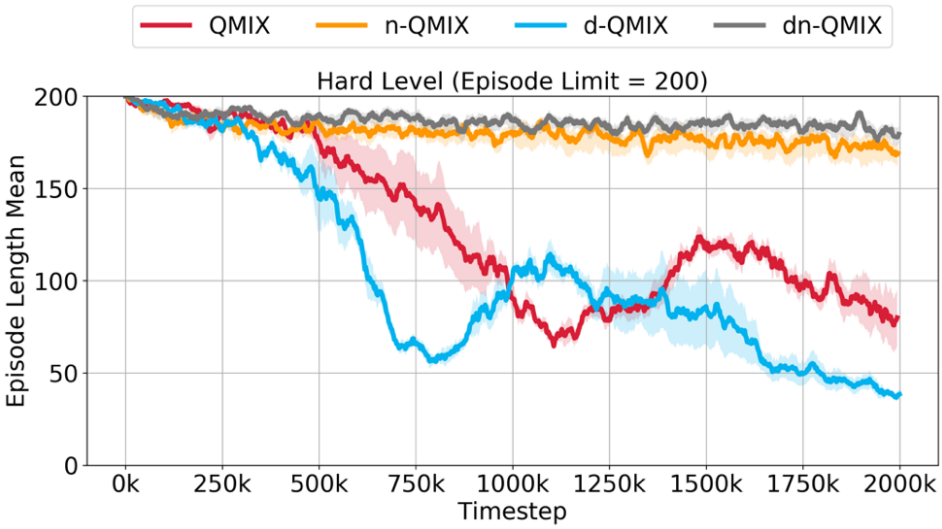

QMIX
n-QMIX
d-QMIX
dn-QMIX
Hard Level (Episode Limit = 200)
Episode Length Mean
200
150
100
50
0
0k
250k
500k
750k
1000k
1250k
1500k
1750k
2000k
Timestep

Figure 29