Value Decomposition based on the Actor-Critic framework for Cooperative Multi-Agent Reinforcement Learning

Yuyang Zhou

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2022

Abstract

With the development of deep learning, Multi-Agent Reinforcement Learning (MARL) has become a popular method in multi-agent decision-making tasks. In recent years, algorithms under the Multi-Agent Actor-Critic (MAAC) framework achieved competitive results in cooperative MARL by comparing achieved returns, such as MAA2C, MADDPG, and MAPPO. However, the credit assignment problem is still challenging in MAAC algorithms. Although some MAAC methods try to mitigate it, such as COMA, they cannot achieve competitive returns. Therefore, Value Decomposition (VD) methods are proposed to mitigate the credit assignment problem. However, sometimes they cannot achieve as high returns as MAAC methods. Most recently, the Value Decomposition Actor-Critic (VDAC) framework, which implements VD methods for MAAC methods, was proposed. However, all these algorithms are implemented with different implementation details and the advantages of VDAC methods are vague. Therefore, we reimplement VD methods for MAA2C and MADDPG with the same implementation details and compare four VDAC algorithms to their original VD methods and MAAC methods in a total of eight different tasks. The results of our experiments show that VDAC methods are suitable to be used in environments containing numerous agents. Finally, we empirically find that MAA2C+VD methods mitigate the credit assignment problem in the MAAC framework.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Yuyang Zhou*)

Acknowledgements

Firstly, I need to appreciate Giorgos for his detailed co-supervision during the summer. Besides that, I also want to thank Stefano for his supervision and academic resources. Finally, I learned a lot at the Autonomous Agents Research Group. Thanks to all group members for sharing their brilliant ideas.

Table of Contents

Chapter 1

Introduction

Many multi-agent decision-making tasks can be modeled as multi-agent reinforcement learning (MARL). Tasks also can be classified as cooperative, competitive, and mixed. In this dissertation, we will mainly focus on cooperative tasks, which means all agents will observe a shared reward r_t at each time step. For example, multiple robots' cooperative delivery in the warehouse and multiple agents aim to load food [25, 7].

In recent years, with the incorporation of deep learning, MARL algorithms have achieved some great progress, such as Value Decomposition (VD) methods [32, 44, 40], and centralised policy gradient methods [18, 11, 45]. Both of these two kinds of MARL algorithms are under the Centralised Training Decentralised Execution (CTDE) training scheme. Therefore, the CTDE training scheme plays an indispensable role in the achieved progress. For the CTDE training scheme, agents have extra information, such as other agents' actions, and observations during training, but agents only have their own sight during execution. The algorithms based on the Multi-Agent Actor-Aritic (MAAC) framework are a paradigm of the CTDE training scheme. For example, MAA2C [5], MADDPG [18], Actor-Attention-Critic [13], LIIR [8], and MAPPO [45] are all based on the MAAC framework. Algorithms based on the MAAC framework combine the advantages of value-based and policy-based algorithms, which can achieve a bias and variance trade-off.

The credit assignment problem is proposed in the article [21], and discussed in the article [1]. For MARL algorithms, the credit assignment problem still exists. In cooperative MARL, all agents share the same global reward at each time step. In detail, for example, in a football game, all agents will get a score when an agent shoots and scores. This score should be only related to agents who shot, passed the ball, attracted defence, etc. However, some agents even did nothing, they also got the reward. As the training progresses, these agents will learn to do nothing for achieving the rewards. Such agents are called *lazy agents* [34] in reinforcement learning.

Although there are some methods designed for dealing with the credit assignment problem in MAAC algorithms, such as COMA [11], they cannot show a competitive performance by comparing achieved returns in most cooperative tasks done by [25]. Currently, it is still challenging for MARL algorithms based on the MAAC framework, especially with the increasing number of agents. As for value decomposition (VD) methods, such as VDN [34], and QMIX [28], they have been the paradigm in MARL algorithms for dealing with the credit assignment problem. However, most VD methods are based on the Q learning framework, a kind of value-based algorithm, which has a higher bias than policy-based algorithms.

Therefore, it is worthwhile to implement VD methods in the MAAC framework for dealing with the credit assignment problem. This kind of algorithm is named VDAC (Value Decomposition Actor-Critic) [33]. Indeed, there are several algorithms [41, 33, 26] implementing VD methods in the MAAC framework. However, some implementation details in these VDAC algorithms, such as $TD(\lambda)$, and the choice for the optimizer are different in their code. Some researchers [9, 12, 4] assert that implementation details sometimes highly influence the achieved returns for MARL algorithms. Besides that, the advantages of VDAC methods are vague in previous research. So, We would like to reimplement VDAC methods with the same implementation details, and find out what is main advantage of VDAC methods.

In this dissertation, we reimplement VDN and QMIX for MAA2C, and MADDPG with the same implementation details. Furthermore, We compare four reimplemented VDAC methods to their original VD methods and MAAC methods. There are a total of eight algorithms here. For the experiments, we evaluate these algorithms in two discrete action-space environments, which are MPEs [18] and LBF [25, 7], totally including eight tasks. According to our experiments, we empirically find that VDAC methods are pretty suitable in environments containing numerous agents and dense rewards. Besides that, we also find that the credit assignment problem is mitigated by MAA2C+VD methods.

Chapter 2

Background

This chapter mainly introduces the required knowledge of reinforcement learning (RL) for understanding the dissertation.

2.1 Markov Decision Processes

It is crucial to firstly introduce Markov Decision Processes (MDPs) because most reinforcement learning algorithms are designed for dealing with problems that can be modeled as MDPs. A MDP [17] can be defined as a tuple $< A, S, P, r >$, where:

- A, the set of actions,
- S, the set of states,
- P, the transition probability: $P(s_{t+1}|(s_t, a_t))$ denotes the probability that the environment transits to state s_{t+1} given the state s_t and action a_t,
- r, the reward value: $r(s_{t+1}|(s_t, a_t))$ denotes the immediate reward received after transiting from state s_t to s_{t+1} due to action a_t.

Within a MDP, the agent can interact with the environment during discrete time steps t. At each time, an agent does an action, a_t, and the environment transits to a new state s_{t+1} with a certain probability $P(s_{t+1}|(s_t, a_t))$. Then, the agent achieves its reward r_{t+1}. While the interaction between agents and the environment keeps going on, the transition trajectory generates, described as $< s_0, a_0, r_1, ..., s_t, a_t, r_{t+1} >$.

It is important that MDPs satisfy the Markov property (memorylessness) [20], which means the possibility of future events only depends on the current state. In other words, past states and future states are independent.

With the information above, it is possible to calculate the expected cumulative returns for any given state or action-state pair. Therefore, we can define the state value (V-value) function, and the action-state value (Q-value) function, given a policy π, which can be described as follows:

$$V_\pi(s_t) = \mathbb{E}[G_t|s_t] = \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i r_{t+i+1}|s_t], \tag{2.1}$$

$$Q_\pi(s_t, a_t) = \mathbb{E}[G_t|s_t, a_t] = \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i r_{t+i+1}|s_t, a_t], \tag{2.2}$$

where γ is the discount factor $\in [0,1]$. Because of the Markov property, we can further rewrite the V-value function and Q-value function with Bellman equation [6]:

$$V_\pi(s_t) = \sum_A \pi(a_t|s_t) \sum_{s_{t+1}, r_{t+1}} p(s_{t+1}|(s_t, a_t))[r_{t+1} + \gamma V_\pi(s_{t+1})]. \tag{2.3}$$

$$Q_\pi(s_t, a_t) = \sum_{s_{t+1}, r_{t+1}} p(s_{t+1}|(s_t, a_t))[r_{t+1} + \gamma V_\pi(s_{t+1})]. \tag{2.4}$$

2.2 Temporal Difference Learning

Temporal difference (TD) learning [37] is a paradigm of valued-based RL algorithms. For TD learning, the agent can learn from the previous trajectory and update per time step. The updating rule of TD is described as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)], \tag{2.5}$$

where α is the learning rate, γ is the discount factor.

Q learning [42] is a classic off-policy example of TD learning, which updates the Q-value function per time step. The updating rule is shown as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \underset{a}{max}(Q(s_{t+1}, a)) - Q(s_t, a_t)]. \tag{2.6}$$

With the updating Q-value function, actions that can achieve higher returns should have a higher Q-value and vice versa. In the execution time, $\varepsilon - greedy$ [39] is typically used to select the action.

Although TD learning is widely used in RL, it is still a bootstrapping learning method, especially with the incorporation of deep learning. Therefore, TD learning is a biased estimation of Q-value. In the next section, a more direct learning method will be introduced.

2.3 Policy-based Learning

Compared to value-based algorithms, policy-based algorithms directly learn the policy $\pi_\theta(a|s) = p(a|s,\theta)$, where s and a are the current state and action, and θ is the parameters for the policy. When agents need to execute, the choice of actions will be according to the probability distribution of $\pi_\theta(a|s)$. The theorem used for policy-based algorithms is named the *policy gradient theorem* [35], which can be described as follows:

$$\nabla_\theta J(\theta) = \int_s d^\pi(s) \int_a \nabla_\theta log\pi_\theta(a|s) Q^\pi(s,a) dads, \tag{2.7}$$

where $d^\pi(s) = lim_{t\to\infty} Pr(s_t = s|s_0, \pi)$ is the state distribution within the policy π, and $\nabla_\theta J(\theta)$ is the gradient of the expected cumulative return. After we have the gradient of the expected cumulative return, we can use gradient ascent to update the policy π for maximizing the expected cumulative return. The updating rule is shown below:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta), \tag{2.8}$$

where α is the learning rate for the gradient ascent.

Policy-based algorithms directly learn the policy, however, they suffer from the high variance problem. Finally, a popular framework named the actor-critic framework combines the advantages of both value-based and policy-based algorithms. It will be discussed specifically in the next section.

2.4 The Actor-Critic framework

There are numerous algorithms based on the Actor-Critic (AC) framework, such as AC [16], advantage AC (A2C), and asynchronous advantage AC (A3C) [5], PPO [30], DDPG [31]. The idea of AC algorithms is still based on gradient policy algorithms. Within the AC framework, the policy gradient can be described as follows:

$$g = \mathbb{E}[\sum_{t=0}^{\infty} \Phi_t \nabla_\theta log\pi_\theta(a_t|s_t)], \tag{2.9}$$

where Φ_t can be any item shown below:

- $\sum_{t=0}^{\infty} r_t$, the total return,
- $\sum_{t'=t}^{\infty} r_{t'}$, the return after the current action,
- $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$, the return incorporated with baseline,

- $Q^{\pi}(s_t, a_t)$, the action-state value function
- $A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$, the advantage function.

If Φ_t equals any of the first three items above, it will be a kind of classic policy gradient algorithm. These algorithms are unbiased estimations for the policy gradient, but they will have high variance. If $\Phi_t = Q^{\pi}(s_t, a_t)$, it is the classic AC. If $\Phi_t = A^{\pi}(s_t, a_t)$, it is the A2C algorithm.

As we can see in equation 2.9, the term $\pi_{\theta}(a_t|s_t)$ is the actor, who determines the agent's actions. The term Φ_t is the critic, who learns the expected cumulative return with a value-based method. That is why this framework is called actor-critic.

2.5 Deep Reinforcement Learning

The success of RL is highly related to the incorporation of deep learning. For valued-based algorithms, we use Q-learning as an illustration. Deep neural networks are used to approximate the Q-value, which becomes Deep Q-Learning (DQN) [22], perhaps the most classic and famous deep RL algorithm. DQN minimises the TD loss to approximate the Q-value, which can be represented as follows:

$$\mathbb{L}(\theta) = \sum_{i=1}^{n} [(y_i - Q(s_t, a_t; \theta_t))^2], \tag{2.10}$$

where θ is the parameters in deep neural networks, and $y_i = \gamma \underset{a}{max} Q(s_{t+1}, a; \theta_{t+1})$.

As for policy-based algorithms, neural networks are used to output the policy. They also use the policy gradient to optimize the policy. As we discussed before, the AC framework combines value-based and policy-based algorithms. Therefore, it is intuitive that we use neural networks to represent the actor π_{θ} and the critic Φ_t in equation 2.9. Gradient ascent and minimising the TD loss are used to optimize the actor and the critic respectively.

2.6 Multi-Agent Reinforcement Learning

Single-agent RL has been introduced in detail above. However, single-agent RL cannot be applied to solve multiple agents' decision-making problems. Therefore, multi-agent reinforcement learning (MARL) becomes necessary. Firstly, the derivative of MDPs used in this dissertation will be introduced.

2.6.1 Dec-POMDPs

Dec-POMDPs is the abbreviation of Decentralised Partially Observable Markov Decision Processes. In this dissertation, all experiments are in fully cooperative environments. A fully cooperative environment can be modeled as a Dec-POMDP [24], defined as a tuple $< U, S, \{A_j\}, P, r, \{O_j\}, \Omega, h >$, where:

- U, the set of agents,
- S, the set of states with initial state s_0,
- A_j, the set of actions for agent j, with $A = \times j A_j$,
- P, the state transition probability function: $P(s_{t+1}|(s_t, \boldsymbol{a}))$ means the probability that the environment transits to state s_{t+1} given the state s_t and the joint action $\boldsymbol{a}$ taken by agents,
- r, the global reward function: $r(s_{t+1}|(s_t, \boldsymbol{a})$, the immediate reward for all agents after agents take the joint action, $\boldsymbol{a}$ in the state s_t,
- O_j, the set of observations for agent j, with $O = \times j O_j$,
- Ω, the observation probability function,
- h, the horizon of the problem, determining the number of time steps is infinite or finite.

As shown above, $A = \times j A_j$ is the set of joint actions. $O = \times j O_j$ is the set of joint observations. At each time step t, a joint action $\boldsymbol{a} =< a_1, ..., a_j >$, where $a_j \in A_j$, is selected. The next state of the environment is influenced by the joint action $\boldsymbol{a}$. In a Dec-POMDP, each agent only knows its own action and observation. The joint observation $\boldsymbol{o} =< o_1, ..., o_j >$, where $o_j \in O_j$, is defined by the observation probability function $\Omega = Pr(\boldsymbol{o}|(s_{t+1}, \boldsymbol{a}))$. Besides that, each agent has an action-observation trajectory τ_j, on which it conditions a policy $\pi_j(a_j|\tau_j)$ for agent j. The goal for MARL is to find a policy $\Pi = \{\pi_1, ..., \pi_j, ..., \pi_n\}$ that can maximize agents' expected cumulative reward over an infinite or finite number of time steps.

2.6.2 MARL algorithms

Most MARL algorithms are extensions of single-agent RL, introduced in Section 2.2, Section 2.3, and Section 2.4. For extending single-agent RL to MARL, there are mainly two methods, independent learning, and centralised training decentralised execution (CTDE). As for independent learning [36], each agent only has their own observation during both training and execution. In other words, each agent is independent of other agents. This is an intuitive method that extends single-agent RL to MARL. However, if we simply consider each agent independently, independent learning cannot achieve competitive returns compared to centralised learning in some tasks [25] due to the required complicated cooperation. Therefore, independent learning is more inclined to be a baseline in MARL research.

CTDE, the mainstream in the current MARL, allows agents to share observations during the training time. Each agent only has their own observation during the execution time. The specific methods of CTDE will be introduced in the *related work chapter* .

Chapter 3

Related Work

This chapter will mainly introduce previous achievements highly related to this dissertation. Centralised training decentralised execution [10] (CTDE) is a paradigm of current MARL. It allows each agent to share information, such as observation, and actions with each other during training, but each agent's policy is only determined by its own observation. At present, many competitive MARL algorithms by comparing achieved returns are based on CTDE. In this dissertation, all used algorithms are also based on CTDE. MARL algorithms based on CTDE can be classified into two classes. The first one is the centralised policy gradient learning, and the second one is value decomposition (VD) methods. These two kinds of MARL will be discussed below.

3.1 Centralised Policy Gradient Learning

This kind of algorithm based on CTDE is also under the AC framework. However, compared to the single-agent AC, centralised policy gradient learning has decentralised actors and centralised critics. In other words, the actor only has its own observation, whereas the critic has global information.

3.1.1 MADDPG

MADDPG is the abbreviation of Multi-Agent Deep Deterministic Policy Gradient [18], which is the extension of DDPG [31] to MARL. Actors in MADDPG take the Q-values as inputs. A Q-value can be described as $Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, a_1, ..., a_N)$, where $\boldsymbol{\mu}$ is the set of all agents' deterministic policies, $\boldsymbol{x}$ is the extra information, such as other agents' observations, and additional state information [18]. The outputs of actors are the deterministic policies.

The actions sampling from the deterministic policies will be the inputs of the critics. Then each critic will output the $Q_i^{\pi}(\boldsymbol{x}, a_1, ..., a_N)$ to judge the actors' actions. Actions that can achieve higher returns will receive higher Q-values and vice versa. Here is a loop for MADDPG.

It is similar to policy gradient learning, the gradient of the expected cumulative return for the agent i also can be represented as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{x}, a \sim \mathcal{D}}[\nabla_{\theta_i} \boldsymbol{\mu}_i(a_i|o_i) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, a_1, ..., a_N)|a_i = \boldsymbol{\mu}_i(o_i)], \tag{3.1}$$

where θ_i is the parameter for the agent i, and $\mathcal{D}$ is the reply buffer containing experiences of all agents. With this formula, we can use gradient ascent $\theta_i \leftarrow .\theta_i + \alpha \nabla_{\theta_i} J(\theta_i)$ to optimize actors. We minimise the TD loss to optimize the critics. the formula for the TD loss can be shown as follows:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\boldsymbol{x}, a}[(Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, a_1, ..., a_N) - y)^2], \tag{3.2}$$

where $y = r_i + r Q_i^{\boldsymbol{\mu}'}(\boldsymbol{x}', a_1', ..., a_N')|a_j' = \boldsymbol{\mu}_j'(o_j)'$, and $\boldsymbol{\mu}'$ means the set of target policies with parameters θ'. A brief framework of MADDPG used in its original paper [18] is shown in Figure 3.1.

As the discussion above, the actors in MADDPG directly output actions to the critics. Therefore, according to the back propagation (BP) rule [43], when we optimize the critic loss, it requires the gradient of $Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, a_1, ..., a_N)$ with respect to θ_i. The BP formula in this circumstance can be described as follows:

$$\frac{\partial Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, a_1, ..., a_N)}{\partial \theta_i} = \frac{\partial Q_i^{\boldsymbol{\mu}}(\boldsymbol{x}, a_1, ..., a_N)}{\partial a_i} \frac{\partial a_i}{\partial \theta_i}. \tag{3.3}$$

From the above formula, it is obvious that the action space for MADDPG should be differentiable, so the action space is required to be continuous. In the original paper, they used Gumbel-Softmax [14, 19] to differentiate the action space for learning in discrete action-space environments.

3.1.2 MAA2C

Multi-Agent A2C (MAA2C) is the extension of A2C to MARL, which learns a joint V-value function instead of the Q-value function. The gradient of the expected cumulative return for MAA2C can be given by:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{a_i \sim \pi_i}[\nabla_{\theta_i} log \pi_i(a_i|\tau_i) A_{\lambda}(\boldsymbol{x}, a_i)], \tag{3.4}$$

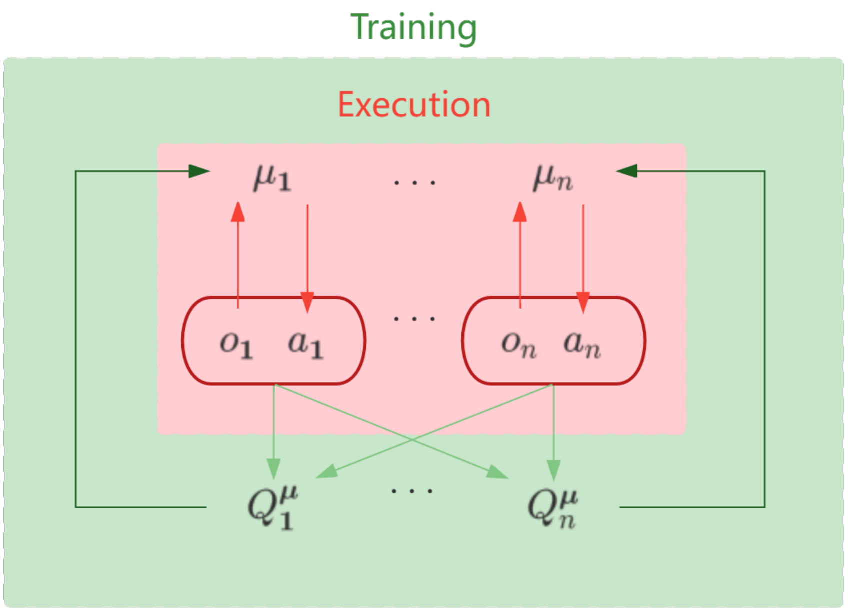

Figure 3.1: The framework of MADDPG [18]

where $A_\lambda(\boldsymbol{x}, a_i) = R_i + \gamma V_\lambda(\boldsymbol{x}') - V_\lambda(\boldsymbol{x})$, λ is the parameters in the centralised critic, and τ_i is the local action-observation history for agent i. The updating rule for MAA2C is just like other centralised policy gradient algorithms, using gradient ascent to update actors, minimising the TD loss to update the critic. Although it is a pretty simple method to extend A2C to MARL, it still can show a competitive performance in discrete action-space environments by comparing the achieved cumulative rewards [25].

3.2 Value Decomposition

Centralised policy gradient learning is more inclined toward policy-based algorithms, whereas Value Decomposition (VD) methods are value-based algorithms. Based on the fact that VD methods learn a joint Q-value, they only can be used in cooperative environments. The two most common VD methods will be introduced below.

3.2.1 VDN

Value Decomposition Networks [34] (VDN) is the first VD method. It learns a linear decomposition of the joint Q-value. VDN utilises networks to approximate the individual Q-value. Then individual Q-values are summed to a joint Q-value for training. The updating rule for the joint Q-value is the same as Q-learning 2.6, which is minimising the TD loss. Because of the linear decomposition of VDN, some complex non-linear relationships between the joint Q-value and individual Q-values are hard for VDN to learn. Therefore, a more complex VD method QMIX is designed.

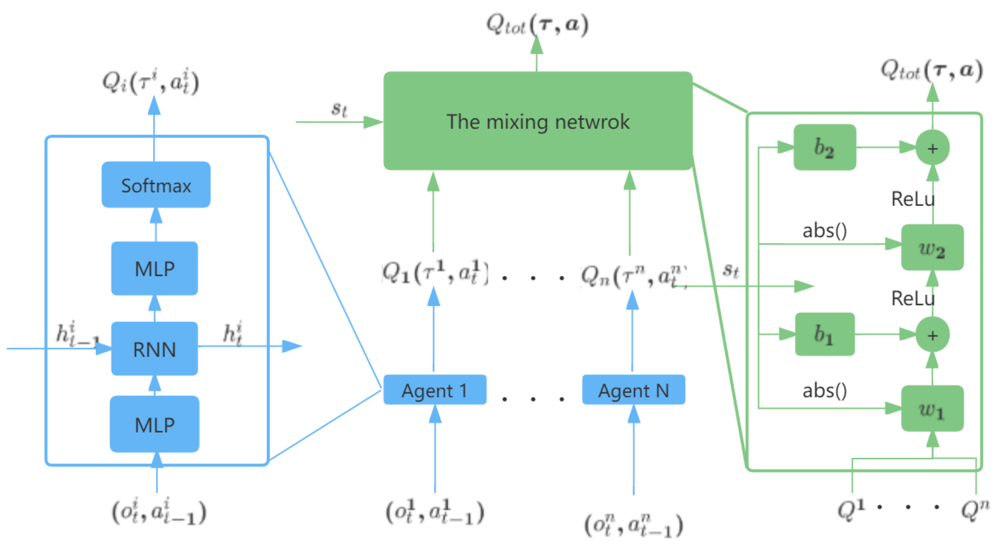

Figure 3.2: The green part represents the mixing network. The blue part is the structure for each agent [28]

3.2.2 QMIX

QMIX [28] utilises a mixing network to decompose the joint Q-value instead of a simple sum used in VDN. The mixing network enables nonlinear decomposition for QMIX, which can suit more complex scenarios. The restraint caused by the mixing network is monotonicity, which means the optimal joint action should equal the combination of individual actions. The framework of QMIX is shown in Figure 3.2, where τ^i used in the figure represents the local action-observation history for agent i.

Chapter 4

Methodology

This dissertation reimplements VD for MADDPG and MAA2C with the same implementation details and finds what kind of tasks are more suitable to use VDAC methods. In this chapter, we will introduce baselines and our reimplemented algorithms in our experiments. Besides that, the implementation details also will be introduced.

4.1 VDAC Methods

VDAC methods are the main algorithms that this dissertation would like to investigate. As we mentioned in chapter 1, the credit assignment problem in the MAAC framework is still challenging. In pure VD methods, the gradient of the joint Q-value will backpropagate to an individual Q-value, which can give each agent a judgement about its action. The action able to achieve a higher expected return will get a higher Q-value. As training keeps going, each agent will learn which action can help the whole team get higher expected returns. Therefore, VD methods can partially mitigate the credit assignment problem. From our perspective, if we implement VD methods within the MAAC framework, the bias will be lower than value-based methods because of the centralised policy gradient, and the credit problem can be mitigated because of VD methods. These two points will be verified in chapter 5 and chapter 6. There are four VDAC methods we investigate, and they will be introduced in the following subsections.

4.1.1 VMIX

VMIX means implementing the mixing network for MAA2C. However, it is notable that MAA2C learns a centralised V-value. Therefore, VMIX uses a mixing network to decompose the centralised V-value, instead of decomposing the joint Q-value. The monotonicity for the joint Q-value is also extended to the centralised V-value, which can be described as follow:

$$\frac{\partial V_{tot}}{\partial V_i} \geq 0, i \in \{1, ..., n\}. \tag{4.1}$$

VMIX is called VDAC-mix (value-decomposition actor-critic) in the paper [33]. The gradient of the expected cumulative return can be described as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{a_i \sim \pi}[\nabla_\theta log\pi(a_i|\tau_i)(Q(s,\boldsymbol{a}) - V_{tot}(s))], \tag{4.2}$$

where θ denotes the shared parameters for all actors, and $Q(s,\boldsymbol{a}) = r + \gamma V_{tot}(s')$, s' means the previous state. The loss of the centralised and decomposed critic can be represented as follows:

$$L_t(\lambda) = (y_t - V_{tot}(s_t))^2 = (y_t - f_{mix}(V_\lambda(o_t^1), ..., V_\lambda(o_t^n)))^2, \tag{4.3}$$

where f_{mix} denotes the mixing network, and λ denotes the parameters for the centralised critic. $y_t = r + \gamma V_{tot}(s_{t-1})$. The structure for VMIX is shown in Figure 4.1.

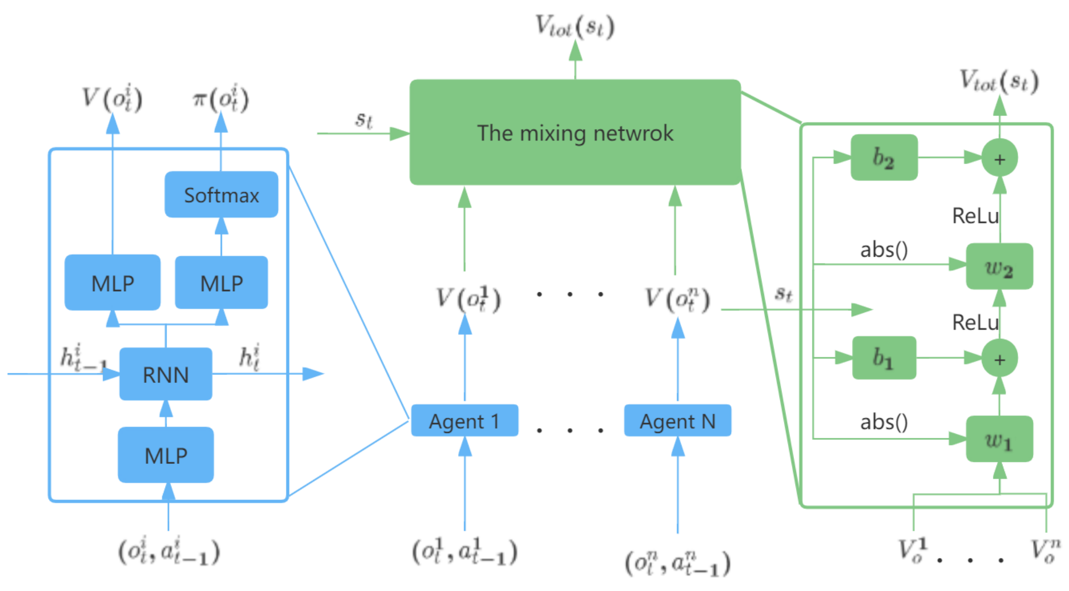

Figure 4.1: The structure for VMIX. The blue part represents the actor networks. The green part represents the decomposed centralised critic network.

4.1.2 VSUM

The structure for VSUM is pretty similar to VMIX. The only difference is replacing the mixing network with the sum operation. The sum operation can be described as follows:

$$V_{tot}(s_t) = V_\lambda(o_t^1) + V_\lambda(o_t^2) + ... + V_\lambda(o_t^n). \tag{4.4}$$

Both the gradient of the expected cumulative return and the loss function for VSUM are the same as equation 4.2, and equation 4.3.

4.1.3 FACMAC

FACMAC combines QMIX and MADDPG. It learns a centralised joint Q-value and uses QMIX to decompose the centralised joint Q-value. The gradient of the expected cumulative return for FACMAC [26] can be shown as follows:

$$\nabla_\theta J(\boldsymbol{\mu}) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{\mu}\sim\mathcal{D}}[\nabla_\theta \boldsymbol{\mu} \nabla_{\boldsymbol{\mu}} Q_{tot}^{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{\mu}, s)], \tag{4.5}$$

where $\boldsymbol{\mu}$ is the deterministic policies with the shared parameters θ for all agents, and $\mathcal{D}$ is the replay buffer. The centralised and decomposed critic can be updated by minimising the following loss:

$$L_t(\lambda) = (y_t - Q_{tot}^{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\tau}))^2 = (y_t - f_{mix}(Q_\lambda(\tau^1, a_t^1), ..., Q_\lambda(\tau^n, a_t^n)))^2, \tag{4.6}$$

where $y_t = r + \gamma Q_{tot}(\boldsymbol{x}', \boldsymbol{\mu}', \boldsymbol{\tau}')$. Here $\boldsymbol{\mu}'$ means the target joint deterministic policy. And f_{mix} means the mixing network, the same as QMIX.

4.1.4 FACMAC-sum

FACMAC-sum simply replaces the mixing network in FACMAC with the sum operation, just like the difference between VMIX and VSUM. The sum operation can be described as follows:

$$Q_{tot}^{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\tau}) = Q_\lambda(\tau^1, a_t^1) + Q_\lambda(\tau^2, a_t^2) + ... + Q_\lambda(\tau^n, a_t^n). \tag{4.7}$$

The rest parts for FACMAC-sum are the same as FACMAC.

4.2 Implementation Details

According to previous research [9, 12, 4], implementation details in MARL algorithms sometimes are crucial for the expected cumulative return. Therefore, for ensuring all algorithms are in a fair comparison, our implementations are based on the open-source EPyMARL [25] codebase, the extension of PyMARL [29]. Other implementation details we mainly considered include *the type of optimizer*, *the hidden dimension for agents*, and *the number of steps for Q learning*. In the original papers about VDAC [41, 33, 26], they use $TD(\lambda)$ for their designed algorithm but use one-step Q learning ($TD(0)$) for other algorithms. Some researchers [12] think comparing algorithms with different updating steps is unfair. Therefore, we compare all algorithms with the same implementation details shown below:

- optimizer: Adam [15],
- hidden dimension: 128,
- n-step: 5.

Chapter 5

Results

This chapter will firstly introduce baselines, the experimental environments, and each task we used. Furthermore, the evaluation protocol, performance metrics, and computing resources will be introduced. Finally, we will represent the results of our experiments and gives some explanations for our results.

5.1 Baselines and Experimental Environments

The baselines and environments used in our experiments will be briefly introduced. Notably, all environments for our experiments are cooperative, and all action spaces are discrete.

5.1.1 Baselines

Both MAAC methods and VD methods are our baselines. MAAC methods we used are MADDPG and MAA2C. VD methods we used include QMIX and VDN. The methods for these four algorithms have been introduced specifically in Section 3.1, and Section 3.2.

5.1.2 Level-based Foraging

In the Level-Based Foraging (LBF) [2, 3] environment, agents need to collect food items scattered randomly in a grid-based world. Each agent has six actions, moving in four directions (up, down, left, right), waiting, and collecting food items. Besides that, food items and agents are assigned levels. When agents would like to successfully collect a food item, the sum of levels for agents around the food must be higher than or equal to the level of the collected food. If agents successfully collect a food item, all of them will receive a shared reward equal to the level of the collected food item.

The range of the grid-based world, the number of agents, and the number of food can be specified. In our experiments, we only tune the number of agents and the number of food. The observation space in all our LBF tasks is 15*15. An episode will end after 50 steps. For the reward function, we set a shared reward equal to the level of the collected food item for all agents when a collecting operation is successful. Finally, in all our experiments, agents are not required to collect a food item simultaneously.

This environment allows different kinds of tasks by tuning specific parameters. More details about the LBF environment can be found on GitHub, under the MIT licence: `https://github.com/uoe-agents/lb-foraging`.

The naming convention for LBF tasks is that $s*s-Np-Mf$. For example, the task $15*15-3p-5f$ means three agents aim to load five food items in a 15x15 grid world.

5.1.3 Multi-Agent Particle Environments

Multi-agent Particle Environments (MPEs) [23] contains several two-dimensional tasks. We used three tasks in MPEs, which are Speaker-Listener, Spread, and Predator-Prey. In MPEs, agents observe other agents' locations and locations of landmarks. The normal action space for each agent in MPEs is two-dimensional navigation, containing staying, moving up, moving down, moving left and moving right. The reward in our experiments is also shared by all agents. Then each task we used will be introduced in detail.

In the Speaker-Listener task, there are two agents, a speaker, and a listener, and three landmarks. Speaker cannot move but can observe the listener's target landmark, and tell the listener's relative location from the target landmark, and the recommended velocity to the listener. The listener cannot observe its own target landmark but can receive the message from the speaker. The goal of the two agents is to navigate the listener to its target landmark. The reward shared by two agents is the negative Euclidean distance of the listener towards its target landmark.

In the Spread task, three agents have their own landmark. All three agents aim to move to their own target landmark. During the moving time, they also need to avoid collisions with each other. The reward for agents is the sum of the negative distance of each agent towards their target landmarks and punishment will be applied if a collision happens.

It is worthwhile to mention that the Predator-Prey task is originally a mixed task.

There are three predators, one prey, and two obstacles. Predators aim to cooperatively catch the prey. The prey with a higher velocity aims to circumvent predators. All agents can observe their own relative location to other agents and obstacles. In the paper[25], they trained the prey agent by MADDPG for 25,000 episodes to make the task fully cooperative. In this dissertation, we use the same method. Therefore, the modified Predator-Prey task contains three agents controlling three predators. Predators need to cooperate with each other to catch the prey pretrained by MADDPG. All predators will be rewarded if they catch the prey.

5.2 Evaluation Protocol

For a fair comparison between sample-efficient off-policy algorithms and on-policy algorithms, we refer to the protocol used in the benchmarking paper by Papoudakis et al. [25]. We train off-policy algorithms for 20 million time steps and on-policy algorithms for 2 million time steps for all experiments. We perform in total 400 evaluations of each algorithm at constant intervals during training and each evaluation for 100 episodes.

5.3 Performance Metrics

We mainly consider two metrics for comparing the performance of each algorithm. Two performance metrics are shown below:

- Maximum returns: the highest return value from 400 evaluations during training. The highest value will be the average across five random seeds with the 95% confidence interval.
- Average returns: the average returns, also across five random seeds, achieved in all evaluation steps during training. The average returns and their confidence intervals will be plotted in a curve.

5.4 Computing Resources

All experiments presented in this dissertation were performed on CPUs. We used MLP clusters, and Eddie clusters to train our algorithms. The main types of CPU models are

Intel(R) Xeon(R) CPU E7- 4830 @ 2.13GHz, Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, and Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz.

5.5 The results for LBF

The average returns and maximum returns for LBF tasks are shown in Figure 5.1, and Table 5.1 respectively. Generally, it is easy to see that MAAC algorithms can achieve both higher maximum returns and average returns in 3p-5f and 5p-5f tasks than both VD methods and VDAC methods. However, with the increasing number of agents and food items, VSUM and VMIX start to achieve both higher returns and average returns than other algorithms. In the following subsections, each kind of algorithm will be analysed specifically.

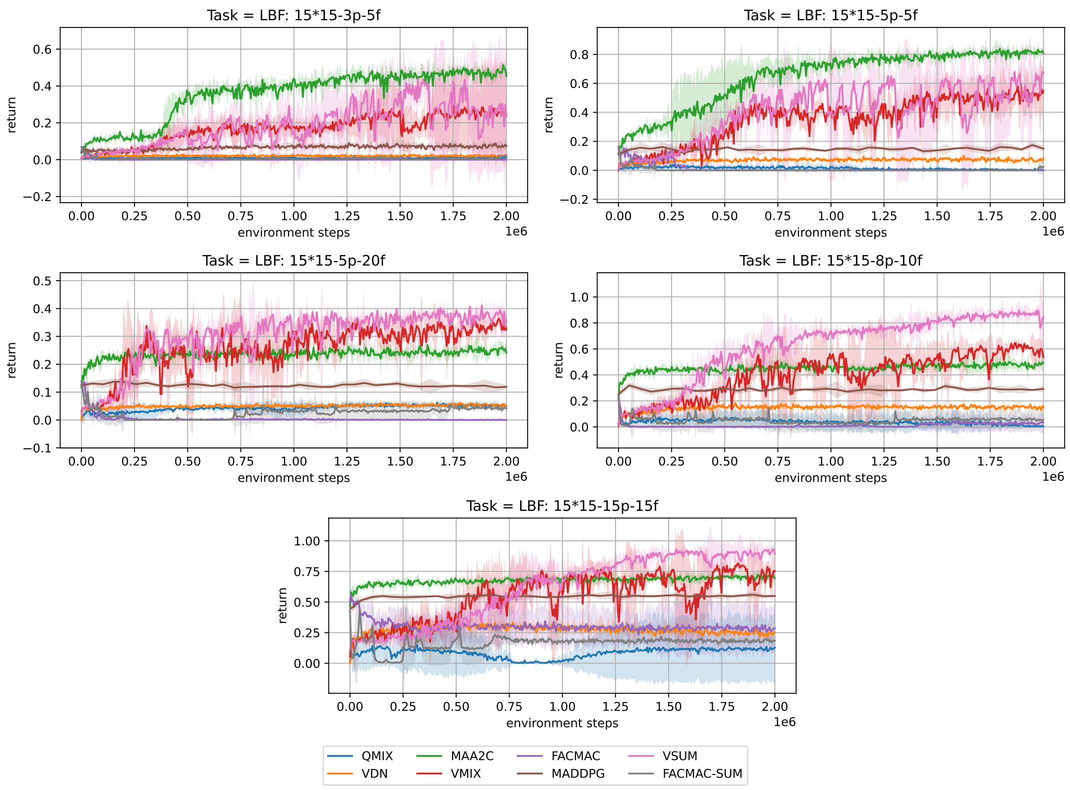

Figure 5.1: The average returns of 5 LBF tasks. All curves are plotted across 5 random seeds with the 95% confidence interval.

Table 5.1: Normalised maximum returns for LBF tasks across five random seeds with the 95% confidence interval. The highest return for each task has been bold.

ALGS \ TASKS	15*15-3P-5F	15*15-5P-5F	15*15-5P-20F	15*15-8P-10F	15*15-15P-15F
QMIX	0.029 ± 0.001	0.055 ± 0.042	0.066 ± 0.10	0.104 ± 0.055	0.276 ± 0.193
VDN	0.040 ± 0.004	0.115 ± 0.004	0.071 ± 0.003	0.204 ± 0.026	0.366 ± 0.056
MAA2C	$\mathbf{0.520 \pm 0.016}$	$\mathbf{0.866 \pm 0.022}$	0.289 ± 0.010	0.541 ± 0.029	0.747 ± 0.018
VMIX	0.347 ± 0.230	0.602 ± 0.141	0.391 ± 0.002	0.671 ± 0.002	0.852 ± 0.030
FACMAC	0.075 ± 0.029	0.182 ± 0.017	0.144 ± 0.017	0.245 ± 0.097	0.541 ± 0.089
MADDPG	0.109 ± 0.004	0.183 ± 0.010	0.145 ± 0.002	0.332 ± 0.010	0.515 ± 0.038
VSUM	0.481 ± 0.106	0.726 ± 0.010	$\mathbf{0.426 \pm 0.011}$	$\mathbf{0.926 \pm 0.057}$	$\mathbf{0.953 \pm 0.024}$
FACMAC-SUM	0.067 ± 0.022	0.180 ± 0.033	0.161 ± 0.035	0.319 ± 0.016	0.515 ± 0.038

5.5.1 VD methods

Generally, VD methods achieved the lowest return in all LBF tasks. Even in $3p-5f$, $5p-5f$, and $5p-20f$ tasks, VD methods almost learned nothing during training. We believe those results are caused by sparse rewards in these tasks because sufficient rewards are required for decomposing the global Q-value into individual Q-values. Although rewards are dense in task $5p-20f$, agents are too few to collect 20 food items. Under this circumstance, the time for collecting all food items is far from enough. Therefore, the final return is still low. In the rest two tasks, $8p-10f$, and $15p-15f$ with more agents, VD methods achieve slightly higher returns than previous tasks. This also proves our thought that VD methods require dense rewards to decompose the joint value.

5.5.2 MAAC methods

MAA2C and MADDPG are two baselines in this dissertation. MMA2C can achieve the highest maximum and average returns in $3p-5f$, $5p-5f$ tasks. However, as the number of agents and food items grows, they cannot achieve returns as high as they achieved in the first two tasks. We believe the requirement for cooperation between agents is more strict because of the higher number of food items. Besides that, with more agents in the environment, perhaps there will be more lazy agents who did nothing because of the centralised V-value in MAA2C. That is the reason we think why MAA2C cannot achieve higher returns than MAA2C+VDs algorithms. As for MADDPG, although each actor can get their own Q-value to consider their policies, the Gumbel-Softmax function used for discretising the action spaces seriously hinders the achieved returns of MADDPG in discrete action-space environments. Therefore, it always cannot achieve higher returns than MAA2C in all LBF tasks.

5.5.3 VDAC methods

VDAC methods are mainly algorithms we research. The results for them are perfectly under our expectations. They will be analysed as follows.

VMIX achieved slightly lower returns than VSUM. We believe that is because the relationship between the joint V-value and individual V-values is relatively simple in LBF tasks instead of some complex non-linear combinations. As we can see in LBF tasks, agents around a food item simply collect the food together. From our perspective, the degree of contributions to the achieved reward depends only on the level of the participating agents. Therefore, a simple sum operation is sufficient to represent the relationship between the joint V-value and individual V-values. In tasks with a few agents and sparse food items, VSUM and VMIX cannot achieve higher returns than MAA2C in $3p-5f$, $5p-5f$ tasks. The reason for this we believe is similar to the reason for low returns achieved by pure VD methods. The mixer requires sufficiently dense rewards to learn how to decompose the joint V-value. If rewards are so sparse in the environment, the joint V-value is more inclined to be randomly decomposed into agents' individual V-values. With more agents and food items, VSUM and VMIX achieved the first two high returns among all eight algorithms, and their achieved returns are significantly higher than other algorithms. We think the mixer here decreases the number of lazy agents because each actor will have a decomposed V-value to consider their actions rather than a centralised V-value for all agents like MAA2C.

MADDPG+VD methods cannot achieve a competitive return almost in all tasks, similar to MADDPG. Papoudakis et al. [25] mentioned that using Gumbel-Softmax is a biased categorical reparametarisation. Besides that, With the incorporation of the mixer in MADDPG, the mixer also requires sufficiently dense rewards to learn to decompose the global Q-value. Besides that, the idea for decomposing the centralised critic in MAA2C is to give each actor a critic to supervise its actions. However, each actor in MADDPG already has an individual critic to supervise their actions. In this case, the mixing network seems to be a little bit redundant. Perhaps MADDPG+VD methods can achieve much higher returns than MADDPG in some continuous action-space environments, such as MAMuJoCo [26], an environment for continuous multi-agent robotic control. However, at least in LBF tasks, MADDPG+VD methods cannot show a competitive return.

5.6 The results for MPEs

The average returns across five random seeds and maximum returns with the 95% confidence interval are respectively shown in Figure 5.2, and Table 5.2. It is obvious that MAA2C achieves the highest returns in all MPE tasks. However, in the last two tasks, the difference between the achieved returns of MAA2C and VSUM becomes smaller. VD methods perform much more competitively than they performed in LBF tasks. All algorithms will be discussed below in detail.

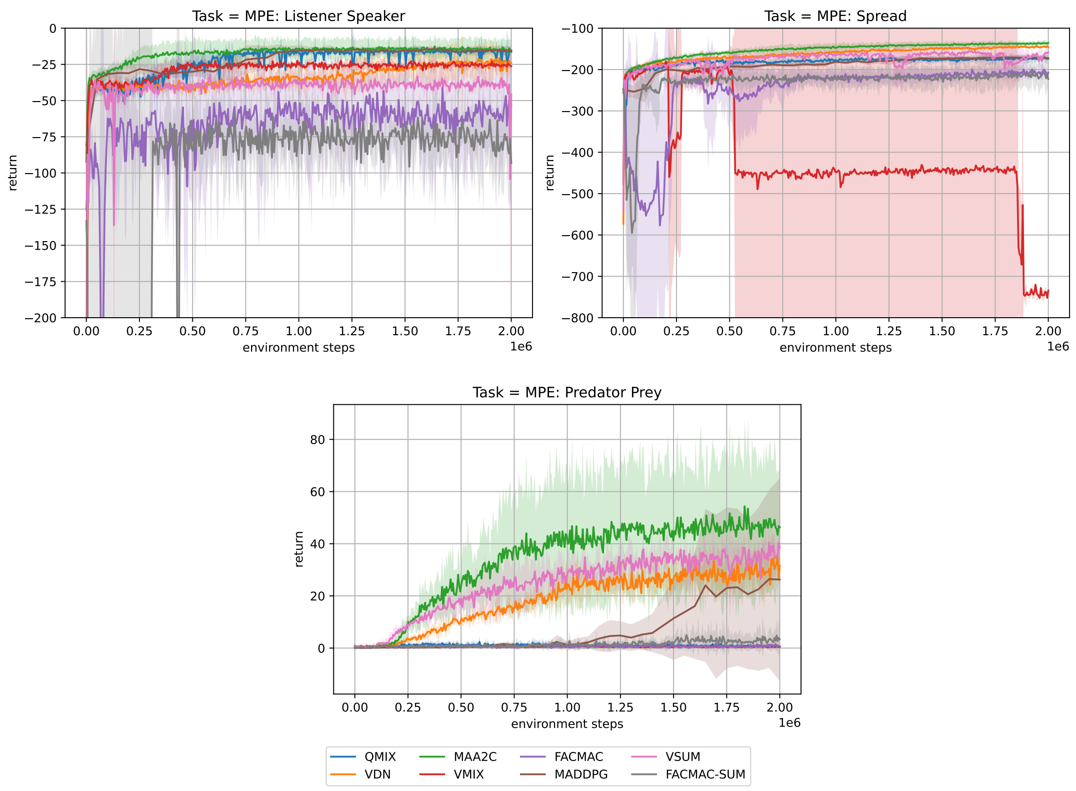

Figure 5.2: Average returns of 3 MPE tasks. All curves are plotted across 5 random seeds with the 95% confidence interval.

5.6.1 VD methods

VD methods show a much better performance in MPE tasks than they performed in LBF tasks by comparing their achieved returns. We believe this is because of the dense

Table 5.2: Maximum returns for MPE tasks across five random seeds with the 95% confidence interval. The highest return for each task has been bold.

ALGS \ TASKS	LISTENER SPEAKER	SPREAD	PREDATOR-PREY
QMIX	-12.23± 0.64	-167.51± 7.33	3.18±1.45
VDN	-17.86± 7.55	-141.02± 1.19	38.88±3.08
MAA2C	**−10.66 ± 5.943**	**−132.21 ± 5.43**	**58.78 ± 28.63**
VMIX	-18.53± 2.83	-160.50± 34.20	1.93±0.22
FACMAC	-40.81± 10.09	-192.09±4.54	3.13±0.37
MADDPG	-12.87± 1.16	-168.00± 5.54	28.60±37.61
VSUM	-30.34± 1.03	-153.03± 0.23	45.31±1.95
FACMAC-SUM	-44.85± 20.21	-194.14± 22.78	6.72±5.74

rewards in MPE tasks. When the rewards are sufficient, the global Q-value can be better decomposed into individual Q-values. Therefore, each agent can make better actions and the final achieved returns will be better. We believe that the reason why VD methods cannot achieve higher returns is the low number of agents. There are up to two agents in MPE tasks. Under these circumstances, VD methods cannot fully show their decomposing capability because of the few agents.

5.6.2 MAAC methods

MAA2C shows the highest return in all three MPE tasks. MADDPG also performs much better than in LBF tasks by comparing achieved returns. But in the Spread task, the achieved returns by MADDPG are lower than both VMIX and VSUM. In the Predator-Prey task, the achieved returns by MADDPG are much lower than VSUM. We believe with the increasing number of agents, the achieved return of MAAC methods will be much lower than VDAC methods, especially for MADDPG.

5.6.3 VDAC methods

In the Listener-Speaker task, all returns achieved by VDAC methods are lower than other methods. We believe this is because the number of agents here is low. As we mentioned before, we think implementing VD methods in the MAAC framework is for mitigating the credit assignment problem (lazy agents). There are only two agents in the Listener-Speaker task, so lazy agents are unlikely to exist here. However, in the last two tasks containing three agents, VMIX and VSUM perform much better than they performed in the Listener-Speaker task. This phenomenon also corresponds to our thought that VDAC methods can perform better as the number of agents increases. However, MADDPG+VD methods also cannot show competitive returns in all tasks. The reason for this we think is the same as the reason for the low returns they achieved in LBF tasks, which is explained in Subsection 5.5.3.

Chapter 6

Analysis

In this chapter, the two goals of this dissertation will be discussed. Firstly, we will analyse whether the credit assignment is mitigated. Then, what type of environments suitable for VDAC methods will be given. Furthermore, the limitations of this dissertation will be represented. Ultimately, suggestions for future research will be given. It is important to mention that FACMAC, and FACMAC-sum cannot perform well in discrete action-space environments because of the Gumbel-Softmax function, and we did not evaluate them in continuous action-space environments because of the limited time. So, they will not be analysed anymore, and VDAC methods mentioned below do not include FACMAC and FACMAC-sum.

6.1 Credit Assignment

As we can find in Table 5.1, MAA2C achieved the highest return in 15*15-3p-5f and 15*15-5p-5f tasks. However, MAA2C is outperformed by VMIX and VSUM in the last three LBF tasks, with more agents and food items, by comparing their achieved returns. It is reasonable to consider that in the last three tasks, only a few agents in MAA2C are working on collecting food items because of the shared joint V-value. There could be several lazy agents who did nothing but still received the shared rewards. However, in MAA2C+VD methods, any agents doing nothing will not receive high V-values because of the decomposed V-value for each agent. Therefore, as the training keeps on, the number of lazy agents will decrease. The problem of lazy agents is a huge part of the credit assignment problem. Consequently, the credit assignment problem in the MAAC framework is mitigated by the incorporation of VD methods because lazy agents no longer exist.

6.2 Suitable Environments for VDAC methods

To begin with, we believe that environments with numerous agents and dense rewards will be pretty suitable for using VDAC methods. VDAC methods are still based on the MAAC framework, and they also use the AC estimator. The difference between MAAC methods and VDAC methods is that the critic in VDAC methods is decomposed. If VDAC methods are used in an environment with sparse rewards, the mixer cannot learn how to decompose the centralised critic to each individual critic. Therefore, an environment with dense rewards is important for VDAC methods. Besides that, as the number of agents increases, a centralised critic in MAA2C is no longer sufficient to supervise all agents. Therefore, lazy agents could exist. As we discussed in Section 6.1, the lazy agents have been hugely mitigated by decomposing the centralised critic. Therefore, we reasonably believe VDAC methods are pretty suitable to be used in environments with numerous agents and dense rewards.

Our experiments in LBF tasks, shown in Table 5.1 can prove our view. In the first two tasks with fewer agents and sparse rewards, VDAC methods achieved lower returns than MAA2C. In the last three tasks with much more agents and denser rewards, VDAC methods achieved the highest two returns.

6.3 Limitations

All algorithms are evaluated in two environments. The LBF environment can contain many agents. However, the relationship between the joint V-value and individual V-values is relatively simple because the degree of the contribution to the achieved reward depends only on the level of the participating agents. In this case, the advantage of the mixing network cannot be fully exploited. As for MPEs, although the relationship between the joint V-value and individual V-values is more complex than the LBF tasks, there are only two or three agents, which is not very suitable for VDAC methods as we discussed before. If we have more computing resources and time, we may try to use the SMAC environment [29], which contains a series of StarCraft challenges with different difficulties.

6.4 Future Research

In this dissertation, we only consider two VD methods, VDN and QMIX. However, the monotonicity constraint in VDN and QMIX is strong. There have been several VD methods trying to release it, such as WQMIX [27], Qtran [32], Qatten [44], and Qplex [40]. These VD methods are worthwhile to implement in MAAC methods if the relationship between the global Q-value and individual Q-values is complex.

Moreover, MAAC methods we used are MADDPG and MAA2C, which both are relatively classic algorithms. Recently, there are many new MAAC methods, such as MAPPO [45], which also has a joint V-value just like MAA2C. Furthermore, MAPPO is more sample efficient because of the importance sampling [38]. It is worthwhile to try to decompose the centralised critic in MAPPO.

Chapter 7

Conclusions

In conclusion, we reimplement four VDAC methods, VMIX, VSUM, FACMAC, FACMAC-SUM with the same implementation details and evaluate eight algorithms in two discrete action-space environments including eight tasks. Firstly, we empirically find that VD methods are not suitable in environments where rewards are sparse because sufficiently dense rewards are required for decomposing the centralised critic. Furthermore, MADDPG is dramatically influenced by the Gumbel-Softmax function. From our perspective, we do not recommend using MADDPG in discrete action-space environments. More importantly, we empirically find that VDAC methods are able to achieve competitive returns in environments containing numerous agents and dense rewards. Besides that, the main goal set in chapter 1, the mitigation of the credit assignment has been achieved by MAA2C+VD methods.

In the end, we point out the limitations of this dissertation and give some suggestions about future research.

Bibliography

[1] Adrian K Agogino and Kagan Tumer. Unifying temporal and structural credit assignment problems. In *Autonomous Agents and Multi-Agent Systems Conference*, 2004.

[2] Stefano V Albrecht and Subramanian Ramamoorthy. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. *arXiv preprint arXiv:1506.01170*, 2015.

[3] Stefano V Albrecht and Peter Stone. Reasoning about hypothetical agent behaviours and their parameters. *arXiv preprint arXiv:1906.11064*, 2019.

[4] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters for on-policy deep actor-critic methods? a large-scale study. In *International conference on learning representations*, 2020.

[5] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256*, 2016.

[6] EN Barron and H Ishii. The bellman equation for minimizing the maximum cost. *Nonlinear Analysis: Theory, Methods & Applications*, 13(9):1067–1090, 1989.

[7] Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[8] Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[9] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International conference on learning representations*, 2019.

[10] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[11] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[12] Jian Hu, Siyang Jiang, Seth Austin Harding, Haibin Wu, and Shih-wei Liao. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning. *arXiv e-prints*, pages arXiv–2102, 2021.

[13] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International conference on machine learning*, pages 2961–2970. PMLR, 2019.

[14] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[17] Harold Joseph Kushner Kushner, Harold J Kushner, Paul G Dupuis, and Paul Dupuis. *Numerical methods for stochastic control problems in continuous time*, volume 24. Springer Science & Business Media, 2001.

[18] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[19] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[20] Andrei Andreevich Markov et al. Theory of algorithms. Springer, 1954.

[21] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[23] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[24] Frans A Oliehoek and Christopher Amato. *A concise introduction to decentralized POMDPs*. Springer, 2016.

[25] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS)*, 2021.

[26] Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, and Shimon Whiteson. Facmac: Factored multi-agent centralised policy gradients. *Advances in Neural Information Processing Systems*, 34:12208–12221, 2021.

[27] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 33:10199–10210, 2020.

[28] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International conference on machine learning*, pages 4295–4304. PMLR, 2018.

[29] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob

Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[31] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[32] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*, pages 5887–5896. PMLR, 2019.

[33] Jianyu Su, Stephen Adams, and Peter Beling. Value-decomposition multi-agent actor-critics. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11352–11360, 2021.

[34] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2085–2087, 2018.

[35] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[36] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[37] Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[38] Surya T Tokdar and Robert E Kass. Importance sampling: a review. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(1):54–60, 2010.

[39] Michel Tokic. Adaptive ε-greedy exploration in reinforcement learning based on value differences. In *Annual Conference on Artificial Intelligence*, pages 203–210. Springer, 2010.

[40] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. In *ICLR*, 2021.

[41] Yihan Wang, Beining Han, Tonghan Wang, Heng Dong, and Chongjie Zhang. Dop: Off-policy multi-agent decomposed policy gradients. In *International Conference on Learning Representations*, 2020.

[42] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[43] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[44] Yaodong Yang, Jianye Hao, Ben Liao, Kun Shao, Guangyong Chen, Wulong Liu, and Hongyao Tang. Qatten: A general framework for cooperative multiagent reinforcement learning. *arXiv preprint arXiv:2002.03939*, 2020.

[45] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.