GradPS: Resolving Futile Neurons in Parameter Sharing Network for Multi-Agent Reinforcement Learning

Anonymous Authors[1]

Abstract

Parameter-sharing (PS) techniques have been widely adopted in cooperative Multi-Agent Reinforcement Learning (MARL). In PS, all the agents share a policy network with identical parameters, which enjoys good sample efficiency. However, PS could lead to homogeneous policies that limit MARL performance. We tackle this problem from the angle of gradient conflict among agents. We find that the existence of futile neurons whose update is canceled out by gradient conflicts among agents leads to poor learning efficiency and diversity. To address this deficiency, we propose GradPS, a gradient-based PS method. It dynamically creates multiple clones for each futile neuron. For each clone, a group of agents with low gradient-conflict shares the neuron's parameters, which are updated according to the gradients of each agent group. Our method can enjoy good sample efficiency by sharing the gradients among agents of the same clone neuron. Moreover, it can encourage diverse behaviors through independently updating an exclusive clone neuron, without gradient conflict. Through extensive experiments, we show that GradPS can learn diverse policies with promising performance.

1. Introduction

Many real-world tasks can be modeled as cooperative Multi-Agent Reinforcement Learning (MARL) problems (Hernandez-Leal et al., 2019), where multiple agents must collaborate to achieve a common goal (Rashid et al., 2020). MARL is challenging due to issues such as non-stationary (Qiu et al., 2021), partial observability (Oliehoek & Amato, 2016), scalability, credit assignment (Foerster et al., 2018; Rashid et al., 2018), low sample efficiency, etc. Researchers have proposed various methods to address these issues. Many of them adopted the Centralized Training with Decentralized Execution (CTDE) paradigm (Oliehoek et al., 2008), where agents are trained jointly with central information, but the agents execute their policies decentrally with the agent's local observation. In the CTDE paradigm, Parameter Sharing (PS) (Rashid et al., 2018) is a widely adopted technique in MARL to alleviate the scalability and low sample efficiency issues.

In PS, all agents share a neural network policy determined by the same set of parameters. PS is effective for MARL tasks where the agent behaviors are identical or similar. However, PS can lead to homogeneous policies, limiting the diversity of agent behaviors and the overall capabilities of joint policies.

Existing studies on PS can be classified into four categories: (1) full PS approaches where all agents share the same parameters, the identity of an agent is used as input to the policy to increase policy diversity (Rashid et al., 2018; Qiu et al., 2021; Wang et al., 2020), (2) group-based PS where agents are group (either dynamically or statically) into multiple groups based on the behaviors (e.g., trajectories), each group shares the same set of parameters (Christianos et al., 2021; Li et al., 2024a). (3) partial PS where the neural network of the policy is divided into a shared part and an individual part (Li et al., 2021), (4) mask-based PS where the agent policy is determined by the combination of a shared policy network and an agent-specific mask that can be static (Kim & Sung, 2023) or dynamic (Li et al., 2024c). Our work is a neuron-based PS approach that addresses the problems of PS from the angle of neuron gradient conflict among agents.

Neuron gradient refers to the gradient of a loss with respect to a neuron. The gradient of the neuron represents the change in the knowledge learned by the agent. In a PS network, agents could lead different knowledge which lead to gradient conflict. The neuron gradient conflict indicates gradient directions for a neuron among some agents are different. Regarding learning efficiency, gradient conflict is harmful, as it cancels out the learning progress of agents. Regarding policy diversity, gradient conflict could be beneficial, suggesting that agents have learned different skills

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

and knowledge.

In this work, we study the gradient conflict in PS networks. We find the existence of futile neurons whose updates are canceling out significantly by gradient conflict. We find that these futile neurons can affect the expressiveness of the MARL policy network.

We propose a method, Gradient-based Parameter Sharing (GradPS), to resolve futile neurons in the PS network. GradPS records the gradient conflicts between agents for the last T interval during training. For each futile neuron, GradPS dynamically creates K clones for the neuron. For each clone, a group of agents with low gradient conflict shares the parameters of the clone. Its parameters are updated according to the gradient of the agents in the group. A group can consist of multiple agents, and their gradients (knowledge) with respect to the neuron can be shared among the group. A group can consist only of a few agents, which helps the agents to develop their specialized skills. As agent policies evolve, cloned neurons assigned to groups may once again become futile, while previously conflicting agents may no longer exhibit conflicts. To adapt to these changes, we re-evaluate inter-agent conflicts and restore a subset of cloned neurons with minimal conflicts to their original state, enabling a dynamic regrouping process.

Through extensive experiments on multiple benchmarks, we show that GradPS performs better than state-of-the-art PS methods. GradPS can learn diverse policies through utilizing the gradient conflict information.

2. Background

2.1. Dec-POMDPs

We consider Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs) in modeling cooperative multi-agent reinforcement learning (MARL) scenarios. A Dec-POMDP can be defined by a tuple $G = \langle \mathcal{S}, \{\mathcal{U}_i\}_{i=1}^N, P, r, \{\mathcal{O}_i\}_{i=1}^N, \{\sigma_i\}_{i=1}^N, N, \gamma \rangle$, please refer to Appendix A.1 for details.

2.2. Gradient Conflict

Gradient conflict is a common phenomenon in multi-task learning (MTL), where multiple different but related tasks are jointly trained by sharing a model (Caruana, 1997). One implementation of MTL is to jointly train a network for all tasks, with the goal of discovering and leveraging relationships between tasks.

Learning multiple tasks may result in worse overall performance and data efficiency. One reason for this is gradient conflict where the gradients of multiple tasks diverge, such that the update for one task negatively affects the other. Researchers (Yu et al., 2020) consider the gradient conflict for a vector, which is defined as follows.

Definition 1 (Vector Conflicting Gradients (Yu et al., 2020))**.** *For a vector, the gradients g_i and g_j $(i \neq j)$ for tasks i and j are said to be conflicting with each other if* $\cos \phi_{ij} < 0$*, where ϕ_{ij} is the angle between g_i and g_j.*

Conflicting gradients pose a challenge to optimizing the multi-task objective, as different gradients of individual tasks may impede the learning of knowledge.

Here, vector gradient conflict refers to the conflict between gradient vectors composed of neuron gradients across different tasks in a neural network. In contrast, the neuron gradient conflict proposed in this paper specifically pertains to conflicts in gradient values for a particular neuron among different agents within a PS network.

3. Related Work

3.1. Parameter Sharing (PS)

Multi-Agent PS Parameter sharing has been widely used in MARL algorithms due to its simplicity and high sample efficiency. However, it could lead to homogeneous agent behaviors. ROMA (Wang et al., 2020) adopts a full PS approach, where agent behaviors are encoded into roles represented by a hidden vector. Agent policies are sampled from the hidden vector. SePS (Christianos et al., 2021) adopts a group-based PS approach by creates multiple sets of parameters each shared by a group of agents. Agents are clustered into groups based on their behaviors (trajectories) in the beginning of training. SNP (Kim & Sung, 2023) employs a partial PS approach. It creates a big neural network, and then each agent builds its policy based on part of the network through network pruning. AdaPS (Li et al., 2024a) combines SNP and SePS by proposing a cluster-based partial PS approach. Kaleidoscope (Li et al., 2024c) adopts a mask-based PS approach, an agent policy is determined on both a full PS network and a mask, which is learned dynamically for each agent.

Multi-Task PS Parameter sharing is also widely used in multi-task learning (MTL), which learns a single model for multiple different tasks. By PS among different tasks, MTL methods can learn more efficiently with an overall smaller model size compared to learning with separate models. PCGrad (Yu et al., 2020) found that parameter sharing of different tasks in MTL produces gradient conflicts and hurt performance. To avoid gradient conflicts, CAGrad (Yu et al., 2020) seeks an update vector that maximizes the worst local improvement of any target within the average gradient neighborhood, optimizing the minimum reduction rate of any specific task loss. Gradient Vaccine (Wang et al., 2021c) uses task relevance to set gradient similarity goals and adaptively aligns task gradients to achieve these goals.

GradNorm (Chen et al., 2018) balances training in deep multi-task models by dynamically adjusting the gradient magnitude among tasks. Recon (Guangyuan et al., 2023) selects the layers with high conflict scores and turns them into task-specific layers.

GradPS is a dynamic MARL PS method that learns diverse policies with good learning efficiency. GradPS dynamically groups agents sharing the same neuron based on gradient conflicts, it can improve learning efficiency through sharing gradients without conflicts within a group, and increase diversity through effectively partitioning conflicting gradients among groups.

3.2. Neuron Learning Efficiency

Redo (Sokar et al., 2023) identifies the existence of dormant neurons, whose activation score is low during the training procedure. The dormant neurons lead to poor learning efficiency. Reset (Igl et al., 2021; Nikishin et al., 2022) improves the plasticity of neural networks through periodic resets of the weights of neurons in the last layer of a neural network. (Dohare et al., 2024) proposes continue backpropagation algorithm which periodically re-initiates some less-used neurons. Reborn (Qin et al., 2024) perturbs parameters of dormant neurons in MARL mixing network. Our work analyzes the learning efficiency of MARL neurons based on gradient among different agents. Moreover, we identify the existence of futile neurons, a special type of neurons whose update are canceled mostly by gradient conflicts.

4. The Futile Neuron Phenomenon in MARL

In this section, we study the gradient conflict in the Parameter Sharing (PS) network for MARL. We find the existence of gradient conflicts in PS, identify futile neurons whose updates are affected significantly by the gradient neurons, and find that futile neurons hurt MARL network expressiveness.

We study the most popular PS (Sunehag et al., 2018; Wang et al., 2021a; Rashid et al., 2018; Shen et al., 2022), where the parameters of the agent policy are shared among all agents. The observation of each agent is concatenated with its agent identity to increase policy diversity. The neural network of the PS agent network consists of two fully-connected (FC) layers and a GRU layer. We analyze the gradient in these FC layers.

4.1. Neuron Gradient Conflict Exists in MARL

In this section, we study the gradient of the temporal difference loss with respect to neurons for different agents with the VDN and QMIX methods.

Definition 2 (Neuron Gradient)**.** *During a target network update period, the accumulate sum of the gradients generated by the observation data of agent i on neuron n after backpropagation through loss is the neuron gradient ϕ_i.*

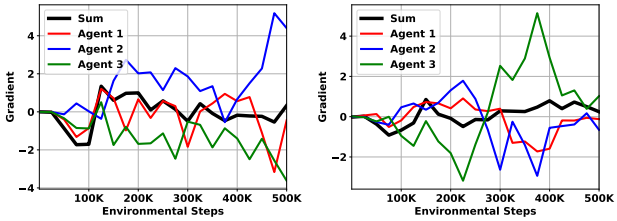

Figure 1: Gradients of a neuron for different agents: VDN (Left) and QMIX (Right) in the SMAC 3m environment.

In Figure 1, the gradient of two neurons for the SMAC 3m environment with the VDN and the QMIX method are plotted in the left and the right part of the graph, respectively. For VDN, different agents could have different gradients, which could be negative or positive. The aggregated gradient for the neuron (depicted as the sum) could become zero due to gradient conflict. For QMIX, depicted in Figure 1 (Right), the gradients for different agents fluctuate more frequently than the VDN methods. For Figure 1, the RMSProp optimizer is used, we show in Appendix Figure 1 that gradient conflicts exist for the SGD and Adam optimizer too.

These findings suggest that although the knowledge learned by agents may be diverse, their differences may cause different gradients for neurons. The differences of neuron gradient cancel out each other, which slows down the learning process of neurons. We define Neuron Gradient Conflict as follows.

Definition 3 (Neuron Gradient Conflict (NGC))**.** *During a target network update interval T, for neuron n, the accumulated gradient of agent i at the activation layer is ϕ_i^T, and the gradient of agent j is ϕ_j^T. The gradient conflict between agents i and j on this neuron within the T interval is defined as:*

$$C_{ij}^T = \sum_T (|\phi_i^T| + |\phi_j^T| - |\phi_i^T + \phi_j^T|) \tag{1}$$

We study the NGC C_{ij}^T of all the neurons in the first FC layer for interval T with QMIX. Figure 2 (left) shows the average value of the NGC for the 5m_vs_6m environment. The value of cell (i, j) of the table represents the average NGC among agent i and j. The darker the color of the cell in the table, the greater the conflict between the agents. It can be seen that there are gradient conflicts in such an environment where agents belong to the same agent type. Moreover, we find that the aggregate NGC behaviors among different agents vary. The conflict among agent 0 and 1 is low as their behavior are similar. The average NGC between

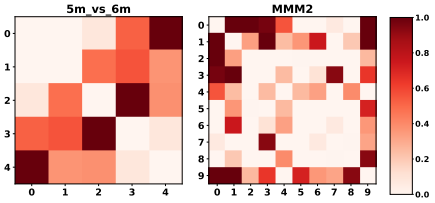

Figure 2: Average Neuron Gradient Conflict of neurons for each agent: 5m_vs_6m (Left) and MMM2 (Right).

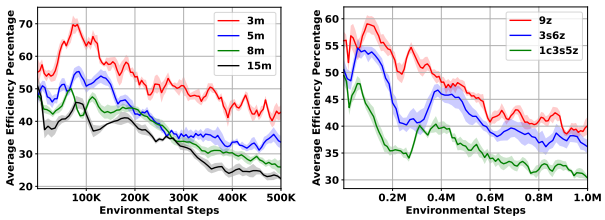

Figure 3: Neuron Gradient Efficiency in homogeneous (Left) and heterogeneous (Right) SMAC environments.

agent 0 and agent 4 is high as the learned behaviors of them are different.

Figure 2 (right) shows the average NGC for the SMAC MMM2 environment, where agents have heterogeneous agent types. Agents 0 and 1 belong to marauder agents, agents 2 to 8 are marine agents, and agent 9 is a medivac agents. The agent policies should be diverse to finish such task. It can be seen that the mean NGC among all the marine agents (agents 2 to 8) are small compared to the conflicts among other agent types. Agent 9 has a large gradient conflict with most of the agents, as it is a healing agent rather than a combat agent (e.g., marine). *The NGC among agents can be used as a measure to distinguish the role/task of agents during MARL task training.*

4.2. Futile Neurons in MARL

We study the learning efficiency of neurons in terms of the NGC among agents. It is defined as follows.

Definition 4 (Neuron Gradient Efficiency (NGE))**.** *During the last T periods of the target network update interval, for neuron x, the accumulated gradient for agent i of the update interval T is ϕ_i^T. The Neuron Efficiency for neuron x is defined as follows:*

$$e = \frac{1}{T}\sum_{t=1}^{T}(|\sum_i \phi_i^T|/\sum_i |\phi_i^T|) \tag{2}$$

NGE is a value among 0 and 1. To evaluate NGE in MARL, we conduct experiments in homogeneous SMAC environments with only one agent type. As is depicted in Figure 3 (left), *with the increase of training time, the neuron efficiency decrease*. This finding suggests that with the increase of the training time, the gradient conflicts among different agents increase, which may lead to inefficient learning. In Figure 3 (left), different environments with an increasing number of agents are considered. There are 3 and 5 agents in the 3m and the 5m environments, respectively. We find that *with increasing number of agents, the neuron efficiency decreases*. This is due to the fact that more agents could lead to more gradient conflicts.

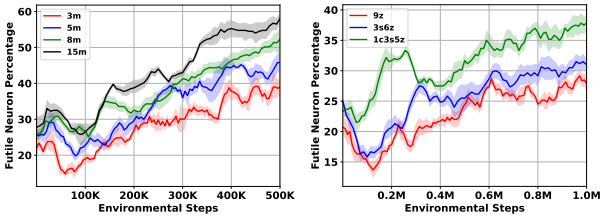

Figure 4: Percentage of Futile Neurons in homogeneous (Left) and heterogeneous (Right) SMAC environments.

In Figure 3 (Right), three environments with the same number (i.e., 9) of agents are shown. There are 1, 2, and 3 types of agents in the 9z, 3s6z, and 1c3s5z environments, respectively. We find that with the increase of agent type, the NGE decrease. For some neurons, their NGE can be very low. Such neurons are refer to Futile Neurons defined as follows.

Definition 5 (Futile Neuron)**.** *A neuron x is a futile neuron if it satisfies the following condition.*

$$e^n < \alpha \tag{3}$$

where $0 < \alpha < 1$ (i.e., $\alpha = 0.2$).

Figure 4 depicts the percentage of Futile Neurons for different SMAC environments. It shows that the percentage increases with time. And such percentage is non-neglectable. Figure 4 (left) shows that with an increasing agent count, the futile neuron percentage increases. Such a percentage increases with the number of agent types in Figure 4 (right).

4.3. Futile Neurons could hurts Network Expressiveness

The expressiveness of value factorization functions (Rashid et al., 2018; Son et al., 2019) is shown to significantly impact MARL performance. In this section, we conduct experiments to study whether futile neurons hurt agent network expressiveness. We conduct experiments on multiple simple one-step matrix games. In these games, there are only two agents with only one time step, all agents shared a team reward without receiving separated rewards.

We build the payoff matrix as a sum of two utility functions, and use VDN as the value factorization function, which can model the summation relationship. The payoff matrix is

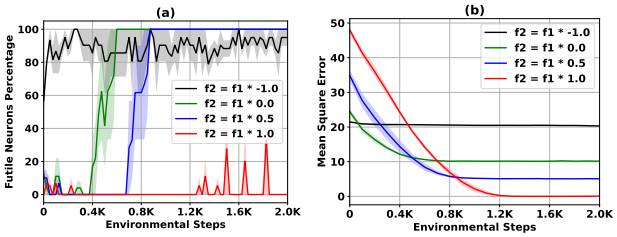

Figure 5: (a) Percentage of futile neurons, (b) MSE of reconstructed Q value

constructed as $Q(u_1, u_2) = f_1(u_1) + f_2(u_2)$, where u_1 and u_2 are the actions taken by agent 1 and 2, respectively, and f_1 and f_2 are two utility functions. $f_1(A) = 0$, $f_1(B) = 1$, $f_1(C) = 2$, $f_1(D) = 3$. $f_2 = w \cdot f_1$, where $w \in \mathbb{R}$. Each matrix games differ by the value w. Please refer to all matrices in Appendix D.2.3.

Figure 5 (left) shows the percentage of futile neurons during the training process. When $f_1 = f_2$ $(w = 1)$, there is basically no futile neuron, while in other scenarios, futile neurons gradually appear. The larger the discrepancy w from 1, the earlier the appearance of futile neurons. Moreover, we find that *the time of the appearance of futile neurons correlates with the time that the MSE of the Q values is close to convergence*, as shown in Figure 5 (right). For example, at step around $0.65k$, the futile neurons for the curve $f_2 = f_1 \times 0.5$ suddenly increase. At the same time, the MSE for $f_2 = f_1 \times 0.5$ converges, and the loss does not drop thereafter.

From Figure 5, we find that the higher percentage of futile neurons, the higher the MSE loss for the matrix game. It is possible that the gradient conflicts for these futile neurons cancel out most of the learning process, which hurts the expressiveness of the MARL networks.

5. Method

In this section, we propose the GradPS method to address the futile neuron phenomenon in MARL. The overview of the GradPS algorithm is described in Algorithm 1, and a schematic plot of GradPS is shown in Figure 6. The GradPS algorithm groups agents into K distinct groups for each futile neuron to reduce gradient conflict and increase diversity. Each futile neuron is duplicated K times to make K clone neurons. For a clone neuron j, the parameters are shared within group j, and the update of clone j is independent of other clones. Thus, the gradient conflict for futile neurons is alleviated. We describe the key operation of the algorithm in the following sections.

5.1. Conflict-based Grouping

The appearance of futile neurons indicates significant conflicts in the agents during recent periods. Gradient conflicts arise from the diverse observations and actions of individual agents, which gradient descent methods struggle to resolve effectively.

Algorithm 1 GradPS Algorithm

```
Get the accumulated gradients for all the neurons.
Identify the Futile Neurons
for Each futile neuron n do
    Grouping agents into K groups according to gradient conflicts (Sec. 5.1)
    Makes K clones of the futile neuron n, and assign each clone to a distinct group of agents.(Sec. 5.2)
end for
if Restore operation required then
    Calculate the variance of each neuron
    Restoring the clones to a normal neuron (Sec. 5.3)
end if
```

For each target update interval, the neuron gradient efficiency of each neurons is recorded, and futile neurons are identified. To reduce gradient conflicts among agents, we cluster agents into K groups based on their gradient conflict values accumulated over T periods, aiming to minimize the total gradient conflict within each group. The clustering is performed based on the average gradient conflict among agents. Please refer to Appendix C.2 and C.3 for details about the recording of gradients and the clustering.

5.2. Group-based Neuron Clone

After the grouping operation, for each futile neuron n, its input weights W_{in} and bias b_{in} are cloned K time. Each clone j is shared among the group j of agents. Each agent belongs to a distinct group.

The output of clone j for the futile neuron n is defined as $W_{in}^j x + b_{in}^j$. Each clone does not individually connect to the neurons in the next layer, their outputs are aggregated into $F(W_{in}^j x + b_{in}^j)$ as the input for the next layer, where F is the activation function.

During training, the input weights W_{in}^j for the clone neuron j are updated independently from other clones. As the grouping operation has made the gradient conflict in each group small, the update of W_{in}^j for the clone neuron j enjoys a better learning efficiency than without sharing. Moreover, agents can learn diverse policies without gradient conflict within a clone neuron. Although the efficiency and diversity improvement comes at the cost of more neurons, we will show in the experiment section that the cost is low.

5.3. Group Parameter Sharing Restoring

As agent policies evolve, cloned neurons assigned to groups may once again become futile, while previously conflicting

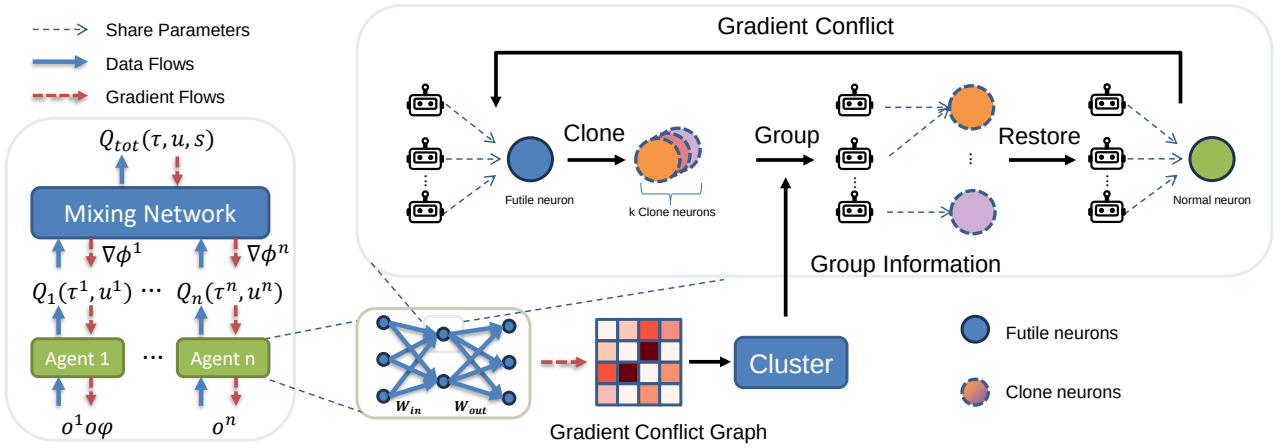

Figure 6: The schematic plot of GradPS. Each futile neurons is duplicated into K clones, each is shared by distinct group of agents. If the difference among clones are small, they will be restored into a normal neuron.

agents may no longer exhibit conflicts. To adapt to these changes, we re-evaluate inter-agent conflicts and restore a subset of cloned neurons with minimal conflicts to their original state, enabling a dynamic regrouping process.

In each restoration period, with a probability of ρ times the percentage of futile neurons, we select the group of clones, whose parameter variance are smallest, for restoration. The restoration process changes the weights of each clone to the same values.

It is possible to simply set the weights of the clones as the average weights of all the groups. However, such an operation could lead to forgetting learned knowledge among different groups. To this end, we add a regularization loss for parameter restoration and use a separate SGD optimizer for gradual parameter restoration. The regularization loss is defined as follows.

$$L_{reg} = \sum_{j=1}^{K} \left(|W_{in}^j - \overline{W_{in}}| + |b_{in}^j - \overline{b_{in}}| \right) \tag{4}$$

where $\overline{W_{in}}$ and $\overline{b_{in}}$ are average values among groups. After M updates by the SGD optimizer, when the group's parameters converge, we merge the clones and restore them into a normal neuron that is shared among all agents.

6. Empirical Evaluations

We show that GradPS performs better than the other parameters sharing (PS) methods for the SMAC and the Predator-Prey benchmarks. GradPS can improve the performance of MARL algorithms by reducing the percentage of futile neurons in PS networks. GradPS is able to learn diverse policies thanks to identifying hidden properties of agents based on gradient conflicts, and GradPS works for multiple MARL agent networks. Please refer to Appendix D.3 for a detailed experimental setup and more experimental results.

6.1. Environmental Setup

Environments. The StarCraft Multi-Agent Challenge (SMAC) (Samvelyan et al., 2019) is a popular benchmark used extensively in MARL, where multiple ally units controlled by MARL algorithms aim to defeat enemy units controlled by the game's built-in AI. **Predator-Prey** simulates a grid world where multiple predators collaborate to capture preys, which consists of stag and hare. Capturing a stag can lead to a higher reward than capturing a hare, but it requires close collaboration between two predators. In this environment, some agents hold hidden property that they will receive the same reward for capturing a hare as capturing a stag. Such hidden property cannot be observed by any agents, including themselves. We evaluate three environments of Predator-Prey: small, medium, and large. Each of them consists of different numbers of agents with varying grid sizes.

Baselines and training. We compare GradPS with various PS methods, including (1) Full Parameter Sharing (FuPS), (2) Full Parameter Sharing with index (FuPS+id), (3) Selective Parameter Sharing (SePS) (Christianos et al., 2021), (4) Structured Network Pruning with parameter Sharing (SNP) (Kim & Sung, 2023), (5) Kaleidoscope (Li et al., 2024c), (6) No Parameter Sharing (NoPS). We evaluate the performance of GradPS and all these methods with the QMIX (Rashid et al., 2018) value factorization function.

6.2. GradPS is superior to other PS methods

The experimental results for various PS methods and GradPS are shown in Figure 7. For the 2c_vs_64zg, the 27m_vs_30m, the 3s5z_vs_3s6z environments, as is shown in Figure 7 (a-c), GradPS achieves the best performance in terms of win rate thanks to its ability to reduce the futile neurons better than other PS methods, as is shown in Figure 7 (d-f). For the 5m_vs_6m, MMM2, 1c3s8z_vs_1c3s9z envi-

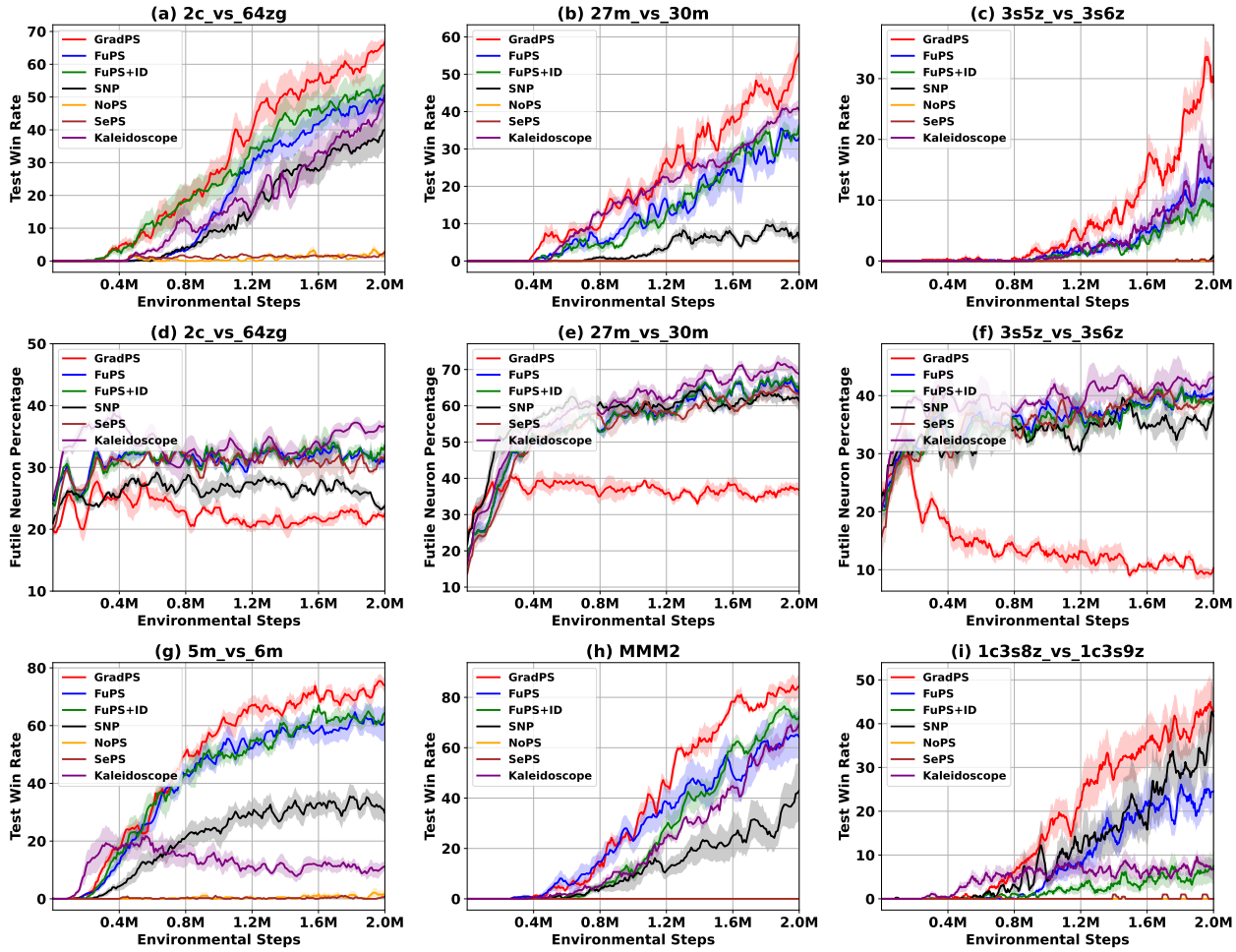

Figure 7: GradPS performs better than other PS methods: (a-c) The test win rate and the futile neuron percentage (d-f) of the 2c_vs_64zg, 27m_vs_30m, 3s5z_vs_3s6z environment. (g-i) The test win rate of the 5m_vs_6m, MMM2, and 1c3s8z_vs_1c3s9z environments.

ronments, GradPS performs the best among all the methods as well. Figure 8 depicts the results for the Predator-Prey environments. As shown in the graph, GradPS performs better than other PS methods for the Predator-Prey environments.

6.3. GradPS can learn diverse policies

For some agents (with hidden property) in predator-prey environments, capturing a hare can lead to the same reward as capturing a stag. After carefully inspecting the learned policies, we find that agents with such hidden property learn to prefer to capture hares.

Figure 8 shows the comparison with other PS methods. Although FuPS+ID can also learn slightly different behaviors, this becomes difficult as the number of agents increases.

To analyze such a phenomenon, we depict the average gradient conflict among agents for the last T update periods in Figure 10. We find that GradPS can distinguish agents with the hidden property through gradient conflicts. Agents with the hidden property have a large gradient conflict with agents without the property. GradPS learns different behaviors through the use of the grouping of cloned neurons. Further, we show that GradPS leads to less futile neurons in Appendix Figure 5 (e-f).

6.4. GradPS works for multiple PS agent networks

In this section, we investigate the applicability of GradPS by validating GradPS's ability to enhance performance across various value factorization algorithms (QPLEX, DMIX, RMIX) in different experimental scenarios (5m_vs_6m, MMM2, Predator-Prey Medium).

The agent architectures of QPLEX, DMIX, and RMIX feature a separate value head, distributional function, and risk-sensitive function, respectively. They are different from

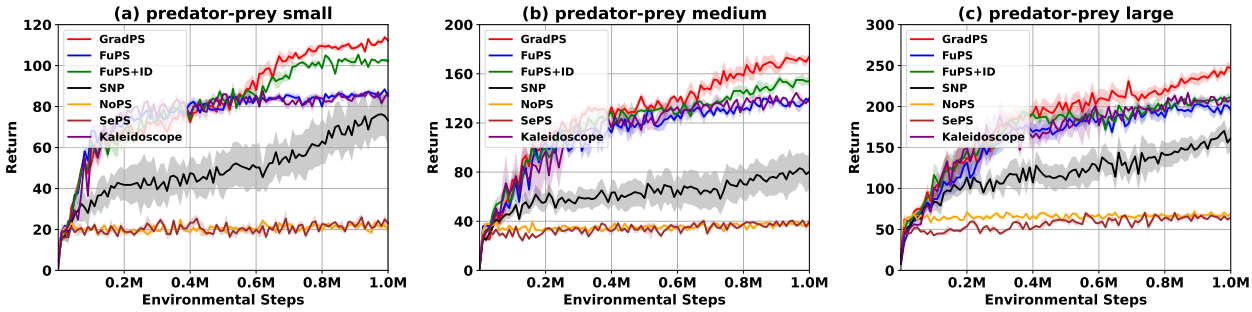

Figure 8: GradPS performs better than other PS methods on Predator-Prey: (a) Small, (b) Medium, (c) Large

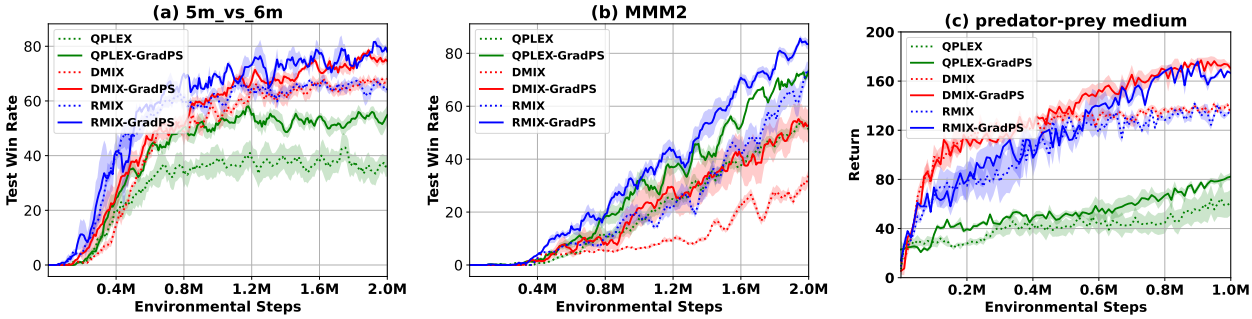

Figure 9: GradPS works for different agent networks (i.e., QPLEX, DMIX, RMIX): (a) 5m_vs_6m, (b) MMM2, (c) Predator-Prey Medium environment.

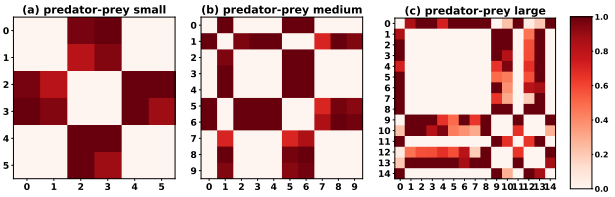

Figure 10: The average gradient conflicts among agents for the (a) small, (b) medium, and (c) large maps of the predator-prey benchmark. The darker the cell, the larger the gradient conflict. The agents with the hidden property in the small map are 2 and 3, in the medium map are 1, 5, and 6, and in the large map are 0, 9, 10, 12, and 13. Agents can be divided into two groups, with smaller conflicts within each group. GradPS successfully distinguishes agents with the hidden properties through gradient conflicts.

the architecture of agents used in QMIX. According to the experimental results presented in Figures 9, GradPS can improve the performance of multiple PS agent architectures and effectively reduce gradient conflicts between agents.

6.5. Parameter Sensitivity Analysis

We evaluate different hyperparameter settings of GradPS in Appendix Figure 10. The results show that the period of accumulating gradients T and the number of group K for sharing could impact GradPS performance slightly. The restoration probability ρ does not impact the performance.

7. Conclusion

In this work, we study the gradient conflict among agents for the Parameter Sharing (PS) networks of Multi-Agent Reinforcement Learning (MARL). We identify the existence of futile neurons whose update is canceled out by gradient conflicts. We show that such neurons hurt the policy expressiveness. We propose GradPS, a simple yet effective gradient-based PS method that resolves futile neuron issues. It dynamically creates multiple independent clones for each futile neuron. Each clone is shared among a group of agents with low gradient conflicts. GradPS can learn diverse behaviors through multiple clones to avoid gradient conflict, and enjoys good sample efficiency by sharing the gradients among agents of the same clone neuron. Through extensive experiments, we show that GradPS is promising.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. Our work has many potential societal consequences, none of which must be specifically highlighted here.

References

Caruana, R. Multitask learning. *Machine learning*, 28: 41–75, 1997.

Chen, Z., Badrinarayanan, V., Lee, C.-Y., and Rabinovich, A. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *ICML*, pp. 794–803. PMLR, 2018.

Christianos, F., Papoudakis, G., Rahman, M. A., and Albrecht, S. V. Scaling multi-agent reinforcement learning with selective parameter sharing. In *ICML*, pp. 1989–1998. PMLR, 2021.

Das, A., Gervet, T., Romoff, J., Batra, D., Parikh, D., Rabbat, M., and Pineau, J. Tarmac: Targeted multi-agent communication. In *ICML*, volume 97, pp. 1538–1546, 2019.

Dohare, S., Hernandez-Garcia, J. F., Lan, Q., Rahman, P., Mahmood, A. R., and Sutton, R. S. Loss of plasticity in deep continual learning. *Nature*, 632(8026):768–774, 2024.

Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *AAAI*, pp. 2974–2982, 2018.

Guangyuan, S., Li, Q., Zhang, W., Chen, J., and Wu, X.-M. Recon: Reducing conflicting gradients from the root for multi-task learning. In *ICLR*, 2023.

Hernandez-Leal, P., Kartal, B., and Taylor, M. E. Is multiagent deep reinforcement learning the answer or the question? A brief survey. In *AAMAS*, pp. 750–797, 2019. URL http://arxiv.org/abs/1810.05587.

Hu, S., Shen, L., Zhang, Y., and Tao, D. Learning multi-agent communication from graph modeling perspective. *arXiv preprint arXiv:2405.08550*, 2024.

Igl, M., Farquhar, G., Luketina, J., Boehmer, W., and Whiteson, S. Transient non-stationarity and generalisation in deep reinforcement learning. In *ICLR*, 2021. URL https://openreview.net/forum?id=Qun8fv4qSby.

Kim, W. and Sung, Y. Parameter sharing with network pruning for scalable multi-agent deep reinforcement learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pp. 1942–1950, 2023.

Li, C., Wu, C., Wang, T., Yang, J., Zhao, Q., and Zhang, C. Celebrating diversity in shared multi-agent reinforcement learning. In *NeurIPS*, 2021.

Li, D., Lou, N., Zhang, B., Xu, Z., and Fan, G. Adaptive parameter sharing for multi-agent reinforcement learning. In *ICASSP*, pp. 6035–6039. IEEE, 2024a.

Li, X., Liu, Z., Chen, S., and Zhang, J. Individual contributions as intrinsic exploration scaffolds for multi-agent reinforcement learning. *arXiv preprint arXiv:2405.18110*, 2024b.

Li, X., Pan, L., and Zhang, J. Kaleidoscope: Learnable masks for heterogeneous multi-agent reinforcement learning. In *NeurIPS*, 2024c.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NeurIPS*, pp. 6379–6390, 2017.

Mahajan, A., Rashid, T., Samvelyan, M., and Whiteson, S. MAVEN: multi-agent variational exploration. In *NeurIPS*, pp. 7611–7622, 2019.

Ng, A., Jordan, M., and Weiss, Y. On spectral clustering: Analysis and an algorithm. *NeurIPS*, 14, 2001.

Nikishin, E., Schwarzer, M., D'Oro, P., Bacon, P., and Courville, A. C. The primacy bias in deep reinforcement learning. In *ICML*, pp. 16828–16847, 2022. URL https://proceedings.mlr.press/v162/nikishin22a.html.

Oliehoek, F. A. and Amato, C. *A Concise Introduction to Decentralized POMDPs*. Springer Briefs in Intelligent Systems. Springer, 2016.

Oliehoek, F. A., Spaan, M. T., and Vlassis, N. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

Qin, H., Ma, C., Deng, M., Liu, Z., Mei, S., Liu, X., Wang, C., and Shen, S. The dormant neuron phenomenon in multi-agent reinforcement learning value factorization. In *NeurIPS*, 2024.

Qiu, W., Wang, X., Yu, R., Wang, R., He, X., An, B., Obraztsova, S., and Rabinovich, Z. RMIX: learning risk-sensitive policies for cooperative reinforcement learning agents. In *NeurIPS*, pp. 23049–23062, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/c2626d850c80ea07e7511bbae4c76f4b-Abstract.html.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N., and Whiteson, S. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In *ICML*, pp. 4292–4301, 2018.

Rashid, T., Farquhar, G., Peng, B., and Whiteson, S. Weighted QMIX: expanding monotonic value function factorisation for deep multi-agent reinforcement learning. In *NeurIPS*, 2020.

Samvelyan, M., Rashid, T., de Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G. J., Hung, C., Torr, P. H. S., Foerster, J. N., and Whiteson, S. The starcraft multi-agent challenge. In *AAMAS*, pp. 2186–2188, 2019.

Shen, S., Qiu, M., Liu, J., Liu, W., Fu, Y., Liu, X., and Wang, C. Resq: A residual q function-based approach for multi-agent reinforcement learning value factorization. In *NeurIPS*, 2022.

Shen, S., Ma, C., Li, C., Liu, W., Fu, Y., Mei, S., Liu, X., and Wang, C. Riskq: Risk-sensitive multi-agent reinforcement learning value factorization. In *NeurIPS*, 2023. URL https://openreview.net/forum?id=FskZtRvMJI.

Sokar, G., Agarwal, R., Castro, P. S., and Evci, U. The dormant neuron phenomenon in deep reinforcement learning. In *ICML*, pp. 32145–32168, 2023.

Son, K., Kim, D., Kang, W. J., Hostallero, D., and Yi, Y. QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *ICML*, pp. 5887–5896, 2019.

Sukhbaatar, S., Szlam, A., and Fergus, R. Learning multiagent communication with backpropagation. In *NeurIPS*, pp. 2244–2252, 2016.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V. F., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *AAMAS*, pp. 2085–2087, 2018.

Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing*, 17:395–416, 2007.

Wang, J., Ren, Z., Liu, T., Yu, Y., and Zhang, C. Qplex: Duplex dueling multi-agent q-learning. In *ICLR*, 2021a.

Wang, T., Dong, H., Lesser, V. R., and Zhang, C. ROMA: multi-agent reinforcement learning with emergent roles. *ICML*, 2020.

Wang, Y., Zhong, F., Xu, J., and Wang, Y. Tom2c: Target-oriented multi-agent communication and cooperation with theory of mind. *NeurIPS*, 2021b.

Wang, Z., Tsvetkov, Y., Firat, O., and Cao, Y. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. In *ICLR*, 2021c.

Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A., and Wu, Y. The surprising effectiveness of ppo in cooperative multi-agent games. *NeurIPS*, 35:24611–24624, 2022.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. Gradient surgery for multi-task learning. *NeurIPS*, 33:5824–5836, 2020.

A. Background

A.1. Dec-POMDPs

We consider Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs) (Oliehoek & Amato, 2016) in modeling cooperative multi-agent reinforcement learning (MARL) scenarios. A Dec-POMDP can be formally described by the tuple $G = \langle \mathcal{S}, \{\mathcal{U}_i\}_{i=1}^N, P, r, \{\mathcal{O}_i\}_{i=1}^N, \{\sigma_i\}_{i=1}^N, N, \gamma \rangle$, where $\mathcal{N}$ represents the set of agents, $\mathcal{S}$ is a finite set of states, and $\mathcal{U}_i$ is the set of actions available to agent i. At time step t, each agent i chooses an action $u_i^t \in \mathcal{U}_i$, forming a joint action $\boldsymbol{u}^t \in \mathcal{U}^N = \mathcal{U}_1 \times \ldots \times \mathcal{U}_N$. This leads to a transition to a new state $s^{t+1} \sim P(\cdot|s^t, \boldsymbol{u}^t)$ and a joint reward r^t. In consideration of partial observability, each agent can only access an individual observation $o_i^t \in O_i$, which is drawn from $o_i^t \sim \sigma^i(\cdot|s^t)$. γ denotes the discounting factor. Each agent acts base on individual policy $\pi_i(u_i|\tau_i)$, $\tau_i = (O_i \times U_i)^*$ represents agent's local action-observation history. the global action-observation history is denoted as $\tau \in \mathcal{T}^N := \tau_1 \times \ldots \times \tau_N$, on which it conditions the joint policy $\pi =< \pi_1, \ldots, \pi_N >$. The joint policy π has a joint action-value function: $Q^\pi(s_t, \mathbf{u}_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}}[R_t \mid s_t, \mathbf{u}_t]$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ is the discounted return.

A.2. Spectral clustering

Spectral clustering (Von Luxburg, 2007; Ng et al., 2001) is a graph-based clustering method that first treats data points as nodes in a graph and computes an affinity matrix based on similarity measures such as Euclidean distance or Gaussian kernel functions. The affinity matrix is then transformed into the Laplacian matrix. The Laplacian matrix L is constructed using the graph's affinity matrix A and its degree matrix D, as represented by the formula $L = D - A$. The Laplacian matrix captures the connectivity between nodes in the graph and reflects the structural information of the graph, playing a key role in spectral clustering. By performing spectral decomposition of the graph Laplacian, a low-dimensional embedding of the data is obtained. In this low-dimensional space, data points within the same cluster are more tightly grouped, while the distance between different clusters is larger, enabling the clustering algorithm to perform better on complex and nonlinear datasets.

Finally, a conventional clustering algorithm, such as k-means, is applied to this low-dimensional representation. Compared to traditional k-means, spectral clustering often outperforms in handling nonlinear structures and complex data, as it can better capture implicit relationships between data points.

B. Related work

B.1. Multi-agent reinforcement learning

Multi-agent reinforcement learning (MARL) provides a framework for modeling complex interactions among agents, with methods typically classified into policy-based, value decomposition, communication-based, and exploration-based approaches.

Policy-based methods, such as COMA (Foerster et al., 2018), MADDPG (Lowe et al., 2017), and MAPPO (Yu et al., 2022), optimize agent policies using gradients. In contrast, value decomposition methods, include VDN (Sunehag et al., 2018), QMIX (Rashid et al., 2018), QPLEX (Wang et al., 2021a), QTRAN (Son et al., 2019), ResQ (**?**), RiskQ (Shen et al., 2023), and RMIX (Qiu et al., 2021), which are particularly effective in cooperative tasks.

Communication-based approaches, such as CommNet (Sukhbaatar et al., 2016), TarMAC (Das et al., 2019), ToM2C (Wang et al., 2021b), and CommFormer (Hu et al., 2024), focus on enhancing multi-agent cooperation through improved communication. Exploration-based methods like MAVEN (Mahajan et al., 2019) and ICES (Li et al., 2024b) aim to strengthen agents' exploration capabilities to better adapt to dynamic and complex environments.

Our method, GradPS, is orthogonal to these techniques, focusing on improving learning efficiency by mitigating gradient conflicts among agents.

C. Method

C.1. Algorithm

The detailed GradPS algorithm is described in Algorithm 2.

Algorithm 2 GradPS

Require: cumulative gradient interval T, number of groups K, group restore probability ρ

Initialize parameters of the network ϕ
Initialize parameters of the target network ϕ'
Initialize replay buffer $\mathcal{D}$
Initialize Neural Network Optimizer
Initialize the SGD optimizer for restoring
for $e \in \{1, \ldots, m \text{ episodes}\}$ **do**
  Start a new episode;
  while Episode is not end **do**
    Get the Agent action a_i
    Execute a_i, obtain global reward r and the next state s'
    Update replay buffer $\mathcal{D}$
    Sample a batch $\mathcal{D}'$ from replay buffer $\mathcal{D}$
    $Loss(\theta, \phi) = (Q(s, a; \theta, \phi) - y_{s,a})^2$
    Accumulated the gradient of each agent
    Update ϕ by $Loss$
  end while
  if Target network update **then**
    Get the set of neurons that have accumulated T period gradients in the first activation layer
    Select the futile neurons
    for Each futile neuron **do**
      Grouping agents into K groups according to gradient conflicts
      Make K copies of the parameters
      Assign parameters of the corresponding group to each agent
      Clear the accumulated gradient of this neuron
    end for
    if Restore operation required **then**
      Calculate the variance of each neuron
      Set the SGD optimizer to start resetting the weights of this neuron
    end if
    if Some neurons have already been reset **then**
      Restore these neurons to fully parameter sharing
    end if
    $\phi' = \phi$
  end if
end for

C.2. Gradient Monitoring

The gradient of neurons refers to the gradient on the activation layer in a neural network. To calculate the gradient for each agent, we designed a gradient detection layer capable of computing the partial derivative of the loss with respect to the observation of a specific agent. Specifically, the input data to the detection layer has dimensions of $(batch \times agents, dim)$, and the layer's weights are a fixed all-ones matrix with dimensions $(agents, dim)$. The input dimensions are reshaped to $(batch, agents, dim)$, then multiplied element-wise by the all-ones matrix. The result is subsequently reshaped back to the original dimensions of the input data to produce the output.

After the backpropagation of loss, each detection layer obtains the calculated partial derivative, which is the gradient of each neuron. We obtain the gradient of loss in the detection layer and clear these gradients to prevent the optimizer from updating

the parameters of the detection layer. Afterward, the optimizer will continue to update the network parameters using the gradients of each parameter as usual.

This allows the gradient information to be retained in the gradient detection layer without altering the original data values. $Batch$ refers to the batch size, $agents$ represents the number of agents, and dim denotes the number of neurons.

C.3. Conflict-based Grouping

The gradients for the most recent T target network update periods are accumulated and recorded. Please refer to Appendix C.2 for a detailed procedure of gradient recording. At the end of each target network update period, we evaluate the neuron efficiency of neurons with T period data and identify futile neurons if their neuron efficiency (defined in (4)) is low.

The appearance of futile neurons indicates significant conflicts in the agents during recent periods. Gradient conflicts arise from the diverse observations and actions of individual agents, which gradient descent methods struggle to resolve effectively. To reduce gradient conflicts, We divide the agents into K groups based on their gradient conflict values accumulated over T periods, aiming to minimize the total gradient conflict within each group.

Based on the gradient conflict values between different agents, we apply the spectral clustering algorithm to clustering the agents. The clustering information is implicitly represented by the gradient conflict values between the agents. Therefore, we utilize the Gaussian kernel function to calculate the distance between agents based on gradient conflicts. First, the conflict matrix is transformed into an affinity matrix A using the Gaussian kernel function. The affinity matrix represents the similarity between agents. Each cell A_{ij} is defined as follows.

$$A_{ij} = e^{\left(-\frac{C_{ij}^2}{2\sigma^2}\right)} \tag{C.1}$$

where C_{ij} is the Neuron Gradient Conflict between agent i and j defined in (3), and σ is the scale parameter of the Gaussian kernel, which controls the influence of distance (conflict value) on similarity. Typically, σ is chosen to be close to the mean or median of the sample distances, ensuring that the affinity matrix is neither too sparse nor too dense.

The graph Laplacian matrix L is computed as $L = D - A$, where D is the degree matrix (the degree of each node is the sum of the weights of its connected edges) and A is the affinity matrix.

$$D_{i,j} = \begin{cases} 0, & if \quad i \neq j \\ \sum_j A_{i,j}, & if \quad i = j \end{cases} \tag{C.2}$$

The Laplacian matrix is subjected to spectral decomposition, and the top k eigenvectors corresponding to the smallest eigenvalues are selected as the embedding vectors. In the embedding space, K-means clustering is applied to these eigenvectors to partition the agents into k groups.

The spectral clustering algorithm clusters the agents according to D. After the clustering for each futile neuron, the neuron is shared by K group of agents. The gradient conflict inside each group is much lower than that among different groups.

C.4. Gradient Conflict

We apply VDN and QMIX to run 3m scenario. As shown in Figure 1, gradient phenomenon still exists when using different optimizers. The gradient conflict in multi-agent reinforcement learning is shown in Figure 2.

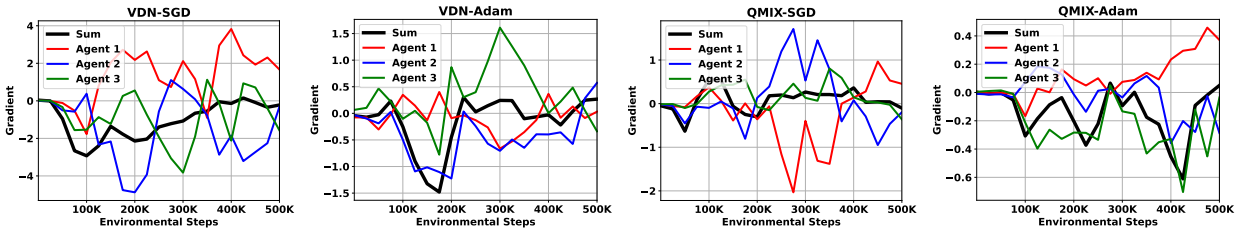

Figure 1: Gradient conflicts in VDN and QMIX when using different optimizers.

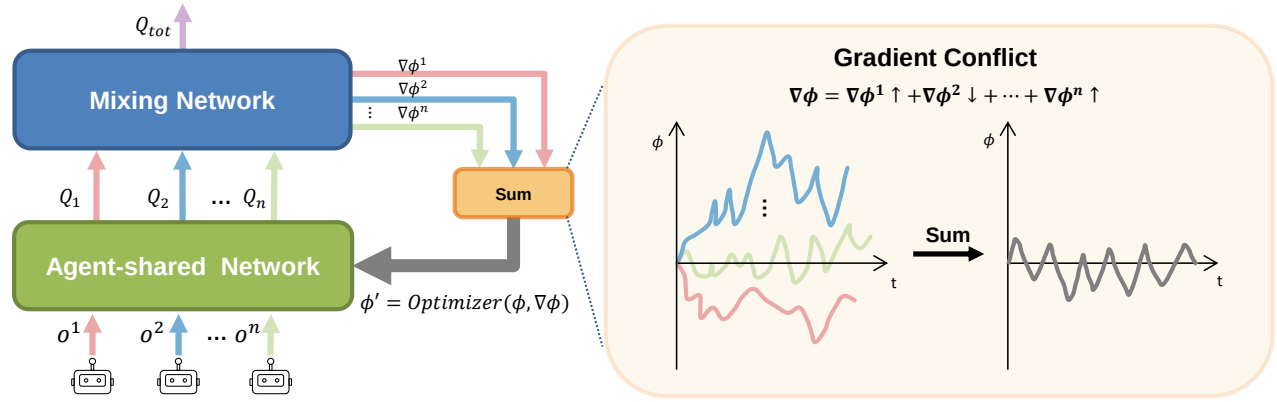

Figure 2: Gradient conflicts in multi-agent reinforcement learning (MARL). Different colors represent the observations and gradients of different agents. The overall gradient, computed by summing the gradients of all agents, is used to optimize the parameters of the agent network. However, conflicts may arise when positive and negative gradients counteract each other.

D. Experimental Details

D.1. Experimental Setup

We compare GradPS with six parameter sharing methods, including Kaleidoscope (Li et al., 2024c), Selective Parameter Sharing (SePS) (Christianos et al., 2021), Structured Network Pruning with parameter Sharing(SNP) (Kim & Sung, 2023), Parameter Sharing (FuPS), Full Parameter Sharing with index (FuPS+id), No Parameter Sharing (NoPS). For the sake of robustness, each experiment was carried out using different random seeds at least 5 times.

Table 1: Baseline parameter sharing algorithms

Algorithms	Description
FuPS[1] [2]	Agents share all the parameters.
FuPS+id	Agents share all the parameters with agent ID in input.
SNP (Kim & Sung, 2023)	The network is randomly pruned subnetworks.
SePS[3] (Christianos et al., 2021)	Agents are clustered to share parameters within each cluster.
NoPS	Agents do not share any parameters.
Kaleidoscope[4] (Li et al., 2024c)	Agents share parameters based on learnable masks.

We implement these algorithms based on their open-source repositories to carry out performance analyses with hyperparameters consistent with those in PyMARL. Our methods are implemented within the PyMARL framework, and each is evaluated using five random seeds with 95% confidence intervals. Specific hyperparameters of different environments are listed in Table 2. We conduct experiments on a cluster equipped with multiple NVIDIA GeForce RTX 3090 GPUs.

D.2. Environment

D.2.1. Predator-Prey

The predator-prey scenario simulates a grid world where multiple agents cooperate to capture prey scattered across the map. The prey consists of two types: stags and hares. Capturing a hare requires only one agent while capturing a stag necessitates at least two agents working together. Successfully capturing a hare provides a team reward of $+2$, while capturing a deer yields a reward of $+10$. Some predators possess a special, unobservable trait that enables them to receive the same team reward for capturing a hare as they would for capturing a stag. The goal is to maximize the total team reward within a limited timeframe. We have developed three different environmental configurations—small, medium, and large—each with

[1] https://github.com/oxwhirl/pymarl
[2] https://github.com/hijkzzz/pymarl2
[3] https://github.com/uoe-agents/seps
[4] https://github.com/LXXXXR/Kaleidoscope

Table 2: Hyperparameter of different environments

Hyperparameter	SMAC	Predator-Prey
Action Selector	epsilon greedy	epsilon greedy
Batch Size	32	32
Buffer Size	5000	5000
Learning Rate	0.0005	0.0005
Hypernet Embed Dimension	64	64
Target Update Interval	200	200
Futile Threshold α	0.2	0.2
Accumulated Period T	20	10
Group Number K	3	2
Restore Probability ρ	0.05	0.025
Reg Learning Rate	0.005	0.005

varying numbers of agents and prey, as well as different map sizes.

Game Rules

- **Agent Movement**: Agents can move in four directions or stay in place. Movement is restricted by the presence of other agents or preys.
- **Observation and Decision Making**: Each agent observes a 7x7 grid centered around itself, receiving information about nearby agents and preys. Decisions are based on this local observation.
- **Capture Mechanism**: To capture a stag, at least two agents must be adjacent to it and must choose the *capture* action at the same time. In contrast, capturing a hare requires only one agent. Successful capture relies on strategic positioning and synchronized actions among agents.
- **Rewards and Penalties**: Agents receive a positive reward for each prey captured through cooperative action.
- **Episode Termination**: An episode terminates if all preys are captured or after 150 steps, providing a fixed time frame for agents to maximize their collective reward.

Configuration	Number of All Predators	Number of Special Predators	Number of Stags	Number of Hares	Map Size	Reward for Stags	Reward for Hares
Small	6	2	6	6	20 x 20	+10	+2
Medium	10	3	10	10	25 x 25	+10	+2
Large	15	5	15	15	30 x 30	+10	+2

Table 3: Comparison of Predator-Prey Configurations

D.2.2. StarCraft II Multi-Agent Challenges (SMAC)

The StarCraft Multi-Agent Challenge (SMAC) (Samvelyan et al., 2019) is a popular benchmark used extensively in the domain of multi-agent reinforcement learning. Built on the StarCraft II game engine, SMAC specializes in micromanagement scenarios where each agent is controlled by an independent agent who must make decisions based on local observations. MARL algorithms coordinate a team of agents to engage in combat against an opposing team managed by the game's built-in AI. The performance of these algorithms is quantitatively evaluated by the test win rate or the test return of the gameplay.

Table 4 depicts the overview of SMAC scenarios used in the experiment.

Name	Difficulty	Ally Units	Enemy Units
3m	Easy	3 Marines	3 Marines
5m	Easy	5 Marines	5 Marines
15m	Easy	15 Marines	15 Marines
9z	Easy	9 Zealots	9 Zealots
3s6z	Easy	3 Stalkers & 6 Zealots	3 Stalkers & 6 Zealots
1c3s5z	Easy	1 Colossi & 3 Stalkers & 5 Zealots	1 Colossi & 3 Stalkers & 5 Zealots
5m_vs_6m	Hard	5 Marines	6 Marines
2c_vs_64zg	Hard	2 Colossi	64 Zerglings
MMM2	Super Hard	1 Medivac, 2 Marauders & 7 Marines	1 Medivac, 3 Marauders & 8 Marines
27m_vs_30m	Super Hard	27 Marines	30 Marines
1c3s8z_vs_1c3s9z	Super Hard	1 Colossi & 3 Stalkers & 8 Zealots	1 Colossi & 3 Stalkers & 9 Zealots
3s5z_vs_3s6z	Super Hard	3 Stalkers & 5 Zealots	3 Stalkers & 6 Zealots

Table 4: Overview of SMAC scenarios used in the experiment.

D.2.3. One step matrix game

The One-Step Matrix Game is a classic experimental environment used to evaluate the performance of multi-agent reinforcement learning (MARL) algorithms. By designing a simple reward structure in the form of a matrix, it simulates cooperative or competitive interactions among multiple agents, enabling the assessment of an algorithm's ability to handle nonlinear relationships and coordination challenges. In this study, the One-Step Matrix Game involves two agents, each selecting an action, where the combination of all agents' actions constitutes a joint action. The rows and columns of the matrix represent the action choices of the respective agents, and each element within the matrix denotes the reward value corresponding to a specific joint action.

u_1 \ u_2	A	B	C	D
A	0	-1	-2	-3
B	1	0	-1	-2
C	2	1	0	-1
D	3	2	1	0

Table 5: $f_2 = -1 \times f_1$

u_1 \ u_2	A	B	C	D
A	0	0	0	0
B	1	1	1	1
C	2	2	2	2
D	3	3	3	3

Table 6: $f_2 = 0 \times f_1$

u_1 \ u_2	A	B	C	D
A	0	0.5	1	1.5
B	1	1.5	2	2.5
C	2	2.5	3	3.5
D	3	3.5	4	4.5

Table 7: $f_2 = 0.5 \times f_1$

u_1 \ u_2	A	B	C	D
A	0	1	2	3
B	1	2	3	4
C	2	3	4	5
D	3	4	5	6

Table 8: $f_2 = 1 \times f_1$

Table 9: Simple one-step payoff matrix game for understanding the impact of futile neurons on network expressiveness.

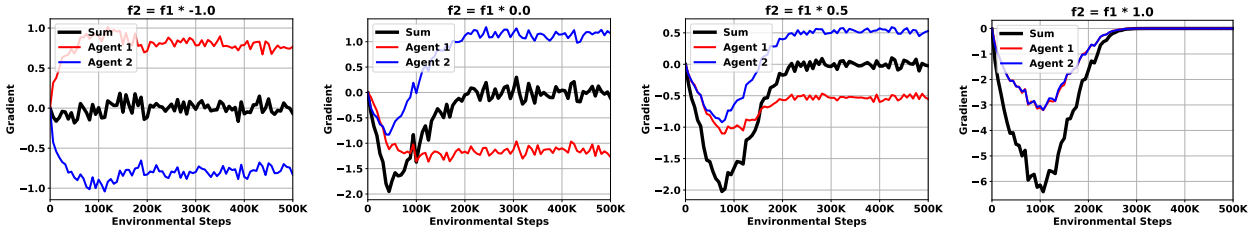

Figure 3: Conflicting neuron gradients in different one step payoff matrix games.

Figure 3 depicts the gradients for different agents in two payoff matrix game using the VDN methods. As it is depicted in the graph, the gradients for different agents can be significantly different. The conflict of gradients may lead to canceling out each other, which makes the sum of the gradient for one neuron close to zero. GradPS can further reduce bias by resolving gradient conflicts in neurons, as shown in Figure 4.

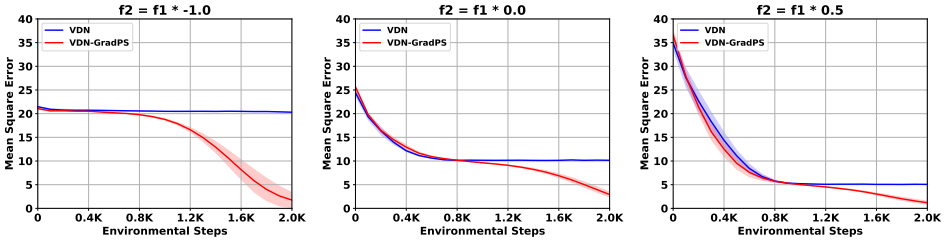

Figure 4: Applying GradPS to different one step payoff matrix games.

D.3. Experimental Results

Predator-Prey

Figure 5 presents the return and the percentage of futile neurons in GradPS within the predator-prey environment, in comparison to other parameter-sharing algorithms.

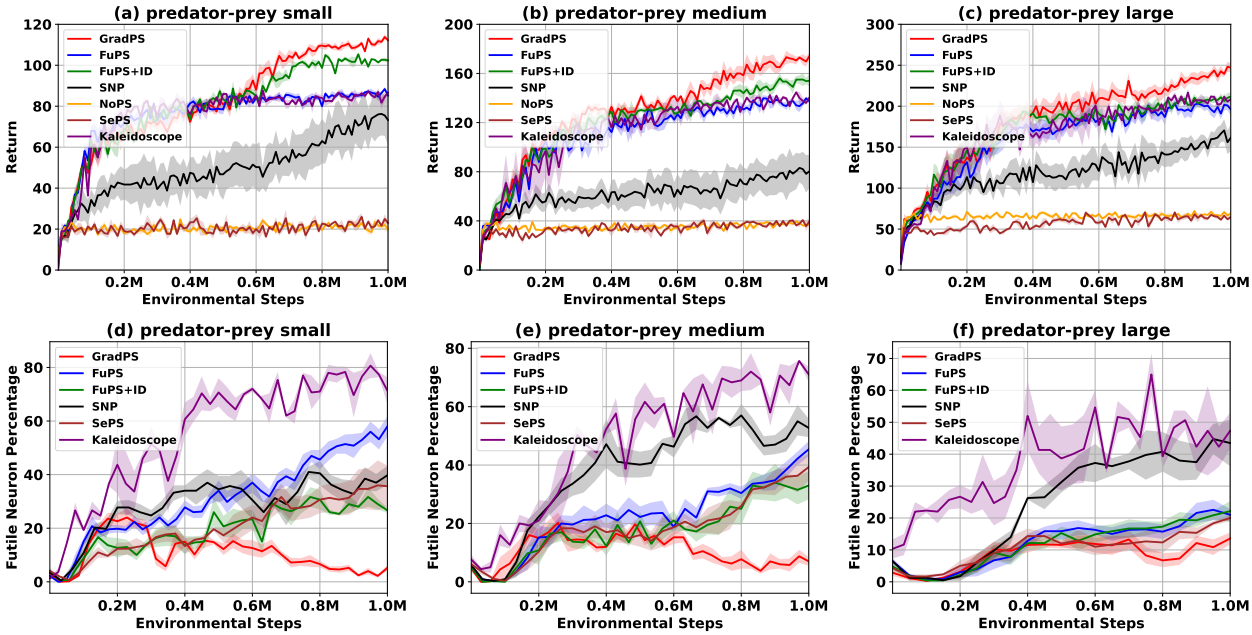

Figure 5: GradPS performs better than other PS methods on Predator-Prey: (a-c) the return for predator-prey small/medium/large environment. (d-f) the percentage of futile neurons for predator-prey small/medium/large environment.

SMAC

Figure 6 presents the win rate and the percentage of futile neurons in GradPS within the SMAC environment, in comparison to other parameter-sharing algorithms.

D.4. GradPS Works for Different Agent Architectures

The results are shown in figure 7.

D.5. Ablation Study and Discussion

D.5.1. The impact of network width

In this section, we study whether gradient conflicts are different in networks of different widths. As shown in Figure 8, the percentage of futile neurons remains approximately equal across networks of different widths.

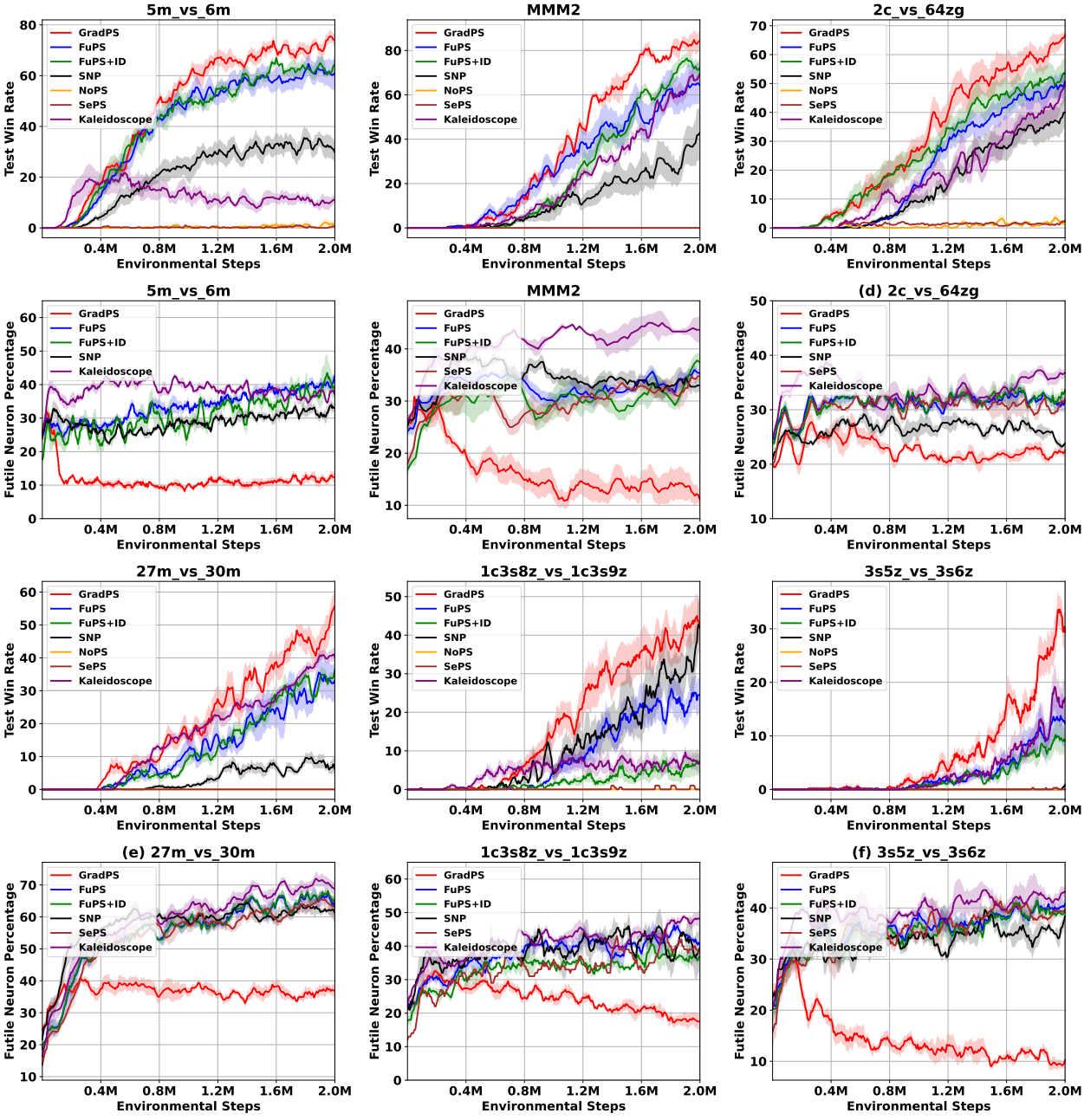

Figure 6: Comparison with other Parameter Sharing methods: the test win rate and the futile neuron percentage in SMAC

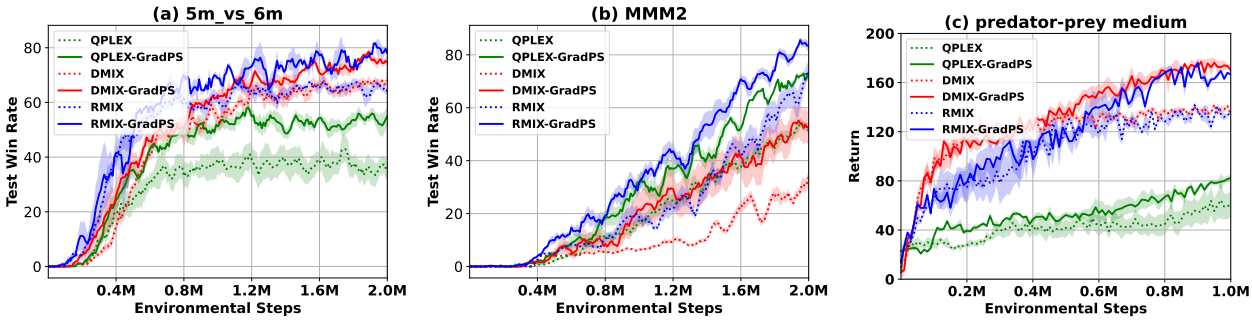

Figure 7: GradPS works for agent networks with different architectures: (a) 5m_vs_6m, (b) MMM2, (c) Predator-Prey medium environment.

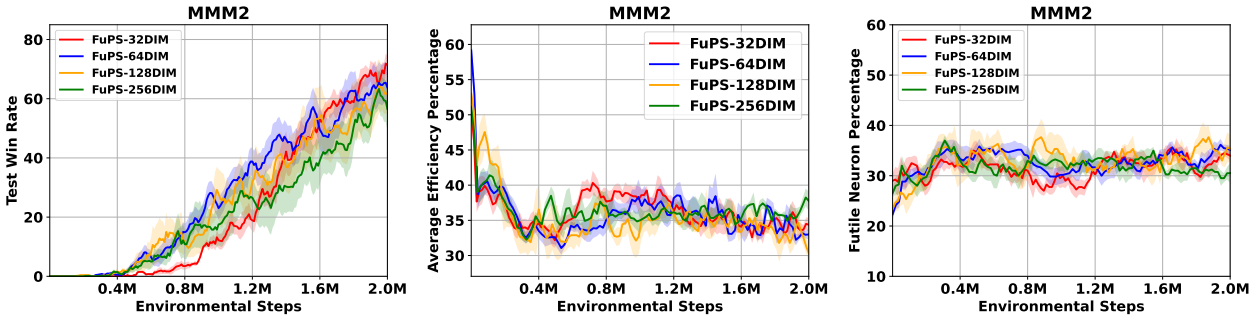

Figure 8: Average Efficiency Percentage and Futile Neurons Percentage in networks with different widths.

D.5.2. Sharing at different depths of network layers

In this section, we investigate whether gradient conflicts differ at different depths in the network. We deepen the QMIX agent network to have four linear-activation layers. We apply GradPS to these activation layers and detect gradient conflicts separately. The results are shown in Figure 9, the number of futile neurons in the first layer is greater than in the other layers.

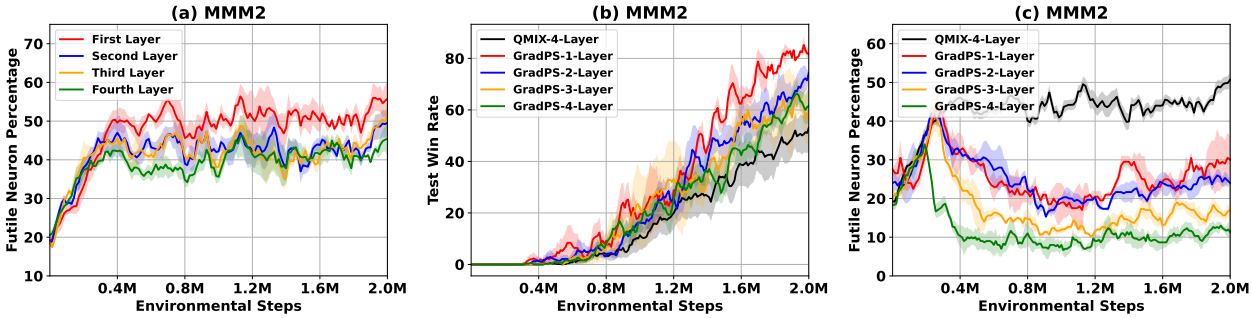

Figure 9: (a) The Futile Neurons Percentage at different depths of a four layer QMIX network. (b) The performance of GradPS in different network layers. (c) The Futile Neurons Percentage after using GradPS on different layers.

D.5.3. Sensitivity analyses of hyper-parameters

The ablation study in Figure 10 illustrates the impact of different hyperparameter settings in QMIX-GradPS, focusing on the ablation of the futile threshold α, the probability of the restoration ρ, the number of group k, and the group interval T. The default configuration of QMIX-GradPS is $T = 20$, $K = 3$, and $\rho = 0.05$. We modify each hyperparameter individually, and the experimental results indicate that appropriate hyperparameters can help better distinguish and utilize futile neurons, thereby improving learning efficiency.The results are shown in Figure 10.

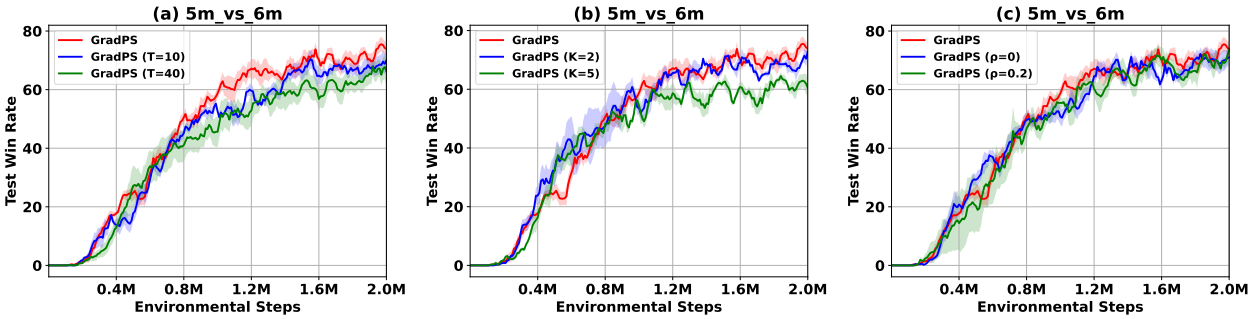

Figure 10: Ablation of different hyperparameters in GradPS. (a) the period of accumulating gradients T. (b) the number of groups K. (c) the restoration probability ρ.

D.6. Execution Time Table and Memory Usage Table.

Table 10: Execution time of different parameter sharing algorithms

Execution time	FuPS	GradPS	FuPS+id	SNP	SePS	NoPS	Kaleidoscope
Predator-Prey small	100%	112%	104%	111%	213%	222%	178%
Predator-Prey medium	100%	111%	102%	105%	134%	141%	124%
Predator-Prey large	100%	107%	99%	112%	168%	198%	154%
5m_vs_6m	100%	106%	101%	116%	136%	154%	129%
MMM2	100%	107%	101%	116%	142%	149%	120%
27m_vs_30m	100%	107%	101%	111%	158%	183%	136%

Table 11: Parameter usage of different parameter sharing algorithms

Parameter usage	FuPS	GradPS	FuPS+id	SNP	SePS	NoPS	Kaleidoscope
Predator-Prey small	100%	105%	100%	103%	112%	159%	114%
Predator-Prey medium	100%	104%	100%	102%	108%	171%	109%
Predator-Prey large	100%	103%	100%	101%	105%	178%	107%
5m_vs_6m	100%	116%	101%	106%	180%	261%	151%
MMM2	100%	122%	101%	108%	156%	354%	134%
27m_vs_30m	100%	114%	101%	105%	128%	466%	117%

E. Discussion

E.1. Societal impact

Our research primarily concentrates on the technical and theoretical aspects of multi-agent reinforcement learning, aiming to enhance the performance of these agents across a variety of tasks. While we do not foresee any direct negative consequences arising from our research, we are committed to maintaining an open dialogue. We highly appreciate and value constructive feedback from the community to ensure our work's contributions are beneficial and ethically sound.

E.2. Limitations and future work

Although the gradient-based grouped parameter sharing method we proposed has achieved promising results compared to other approaches, there is still room for improvement. Our hyperparameters, such as the number of groups and the threshold for defining futile neurons, require further tuning. Additionally, increasing the number of groups may lead to higher parameter overhead. Defining futile neurons more effectively and exploring adaptive grouping will be key directions for our future research.