Policy Regularization via Noisy Advantage Values for Cooperative Multi-agent Actor-Critic methods

Jian Hu∗∗
National Taiwan University
Taipei, Taiwan
janhu9527@gmail.com

Siyue Hu∗
National Taiwan University
Taipei, Taiwan
husiyuehusiyue@gmail.com

Shih-wei Liao
National Taiwan University
Taipei, Taiwan
liao24@gmail.com

ABSTRACT
Recent works have applied the Proximal Policy Optimization (PPO) to the multi-agent cooperative tasks, such as Independent PPO (IPPO); and vanilla Multi-agent PPO (MAPPO) which has a centralized value function. However, previous literature shows that MAPPO may not perform as well as Independent PPO (IPPO) and the Fine-tuned QMIX on Starcraft Multi-Agent Challenge (SMAC). MAPPO-Feature-Pruned (MAPPO-FP) improves the performance of MAPPO by the carefully designed agent-specific features, which may be not friendly to algorithmic utility. By contrast, we find that MAPPO may face the problem of The Policies Overfitting in Multi-agent Cooperation (POMAC), as they learn policies by the sampled advantage values. Then POMAC may lead to updating the multi-agent policies in a suboptimal direction and prevent the agents from exploring better trajectories. In this paper, to mitigate the multi-agent policies overfitting, we propose a novel policy regularization method, which disturbs the advantage values via random Gaussian noise. The experimental results show that our method outperforms the Fine-tuned QMIX, MAPPO-FP, and achieves SOTA on SMAC without agent-specific features. We open-source the code at https://github.com/hijkzzz/noisy-mappo.

KEYWORDS
Multi-agent, Reinforcement Learning, Noise, PPO

1 INTRODUCTION
Multi-Agent Reinforcement Learning (MARL) has seen revolutionary breakthroughs with its successful application to multi-agent cooperative tasks such as robot swarms control [6], autonomous vehicle coordination [1] and computer games [20]. As for scalability and communication security problems, decentralized execution of multi-agent policies that act only on their local observations is widely used. An intuitive approach for decentralized multi-agent policy learning is the Independent Q Learning (IQL) [25]. However, IQL does not address the non-stationarity introduced due to the changing policies of the learning agents. Thus, unlike single-agent Reinforcement Learning (RL) algorithms, there is no guarantee of convergence even at the limit of infinite exploration. Therefore, the Centralized Training and Decentralized Execution (CTDE) [10], which allows for agent to access global information during training stage, is widely used in MARL algorithms [13, 19].

Many CTDE algorithms, e.g. MADDPG [13], MAAC [7], QMIX [19], have been proposed for multi-agent cooperative tasks. Among these algorithms, the finetuned QMIX [5] achieves the SOTA performance in the popular MARL benchmark environment Starcraft Multi-Agent Challenge (SMAC) [20]. To enable effective CTDE for multi-agent Q-learning, the Individual-Global-Max (IGM) principle [23] of equivalence of joint greedy action and individual greedy actions is critical. QMIX ensures that the IGM condition holds by the mixing network with Monotonicity Constraint [19]. However, the mixing network leads to limitations in its scalability, and monotonicity constraints prevent it from learning correctly in non-monotonic environments [23]. We turn our attention to the efficient single-agent RL algorithms, such as Trust Region Policy Optimization (TRPO) [21] and Proximal Policy Optimization (PPO) [22], as their unlimited expressive power and high sample efficiency.

Recently, the literature [3] applies the PPO to the multi-agent tasks directly, called Independent PPO (IPPO); literature [3] also extend IPPO to the vanilla Multi-agent PPO (MAPPO) which using the centralized value function. However, the MAPPO may not perform as well as Independent PPO (IPPO) and the Fine-tuned QMIX [5]. Then literature [27] further improves the performance of MAPPO and IPPO by carefully designed agent-specific features, which may be not friendly to algorithmic utility, called MAPPO-Feature-Pruned (MAPPO-FP) (details in Sec. 3); literature [11] proposed a multi-agent TRPO algorithm, but only for the case where each agent has a private reward. By contrast, we find that MAPPO may face the problem of The Policies Overfitting in Multi-agent Cooperation (POMAC) as they learn policies by the sampled centralized advantage values [15]. Then POMAC may lead to updating the policies of some agents in a suboptimal direction and prevent the agents from exploring better trajectories.

In this paper, (1) To mitigate multi-agent policies overfitting, we propose two policy regularization methods, i.e. Noisy-Value MAPPO (NV-MAPPO) and Noisy-Advantage MAPPO (NA-MAPPO), which disturb the advantage values via random Gaussian noise. (2) Empirical results show that our approaches achieve better performance than the Fine-tuned QMIX [5]. MAPPO-FP [27], and is much better than MAPPO, achieving SOTA performance in SMAC.

Although our method is very simple, it shows that noise perturbation of the advantage function can significantly improve the performance of the multi-agent actor-critic algorithms. Interestingly, our findings demonstrate that replacing agent-specific features (in MAPPO-FP) with noise may make MAPPO works better.
2 BACKGROUND

Dec-POMDP We consider a cooperative task, which can be described as a decentralized partially observable Markov decision process (Dec-POMDP) [17]. The cooperative agent chooses sequential actions under partial observation and environment stochasticity. Dec-POMDP is a tuple \((S, A, O, R, P, \pi, y)\) where \(S\) is state space, \(A\) is joint action space, \(O_i = O(s, i)\) is partial observation for agent \(i\) at global state \(s\). \(P(s'|s, A)\) is the state transition probability in the environment given the joint action \(A = (a_1, \ldots, a_N)\). Every agent has the same shared reward function \(R(s, A)\). 

CTDE Centralized training with decentralized execution (CTDE) paradigm [10], in which agents can obtain additional information and centralized joint learning; while in the testing phase, agents make the decision based on their own private observation. Next, we introduce some CTDE algorithms for the multi-agent credit assignment [2].

Credit assignment Multi-agent credit assignment [2] is a critical challenge in cooperative settings, joint actions typically generate only global rewards, making it difficult for each agent to deduce its own contribution to the team’s success. Many CTDE algorithms have been proposed to solve this problem: COMA [4] trains decentralized agents by a centralized critic with counterfactual advantages. MADDPG [13] and MAAC [7] trains a joint critic to extend DDPG [12] to the multi-agent setting, which can be seen as implicit credit assignment [28]. VDN [24], QMIX [19] (details in Sec. 3) decompose the joint action-value function \(Q_{tot}\) to individual action-value functions \(Q_i\) by the value mixing networks. However, the monotonic constraints limit the expressive power of QMIX, which may learn error argmax action in nonmonotonic cases [23] [14]. Besides, we consider a task including millions of agents, but only several states, the mixing network faces the problem of explosion in the size of \(Q_i\).

Policy Gradient (PG) Then, we briefly introduce the Policy Gradient (PG) and Proximal Policy Optimization (PPO) in single-agent RL. In the on-policy case, the gradient of the objective value function \(V^\pi(s_0) = \mathbb{E}_{a^i} \left[ \sum_{t=0}^\infty \gamma^t r_t \right] \), where \(\gamma \in [0, 1)\) with respect to some parameter of the policy \(\pi\)

\[
\nabla V^\pi(s_0) = \mathbb{E}_a \left[ \sum_{t=0}^\infty \gamma^t \nabla \log \pi(a_t | s_t) A^\pi(s_t, a_t) \right]
\]

(2)

where \(A^\pi(s_t, a_t) : = Q^\pi(s_t, a_t) - V^\pi(s_t)\) is the advantage value function [15] of policy \(\pi\), where \(Q^\pi(s_t, a_t) : = r_t + \gamma V^\pi(s_{t+1})\) is the state action value function. Intuitively, PG makes the policy \(\pi\) closer to the actions with large advantage value by gradient ascent.

Proximal Policy Optimization (PPO) To improve the sample efficiency of PG, Trust Region Policy Optimization (TRPO) [21] aims to maximize the objective function \(V^\pi(s_0)\) subject to, trust region constraint which enforces the distance between old and new policies measured by KL-divergence to be small enough, within a parameter \(\delta\).

\[
J_{\text{TRPO}} = \mathbb{E}_{a_t, s_t - \pi_{old}} \left[ \frac{\pi(a_t | s_t)}{\pi_{old}(a_t | s_t)} A^\pi_{old}(s_t, a_t) \right]
\]

(3)

where \(\frac{\pi(a_t | s_t)}{\pi_{old}(a_t | s_t)}\) is the Importance sampling (IS) weight, with KL-divergence constraint,

\[
\mathbb{E}_{s - \rho_{old}} \left[ D_{KL}(\pi_{old}(\cdot | s) || \pi_{old}(\cdot | s)) \right] \leq \delta
\]

(4)

where \(\rho_{old}\) is the discounted state distribution [21] sampled by policy \(\pi_{old}\). [21] prove that \(J_{\text{TRPO}}\) is equivalent to the Natural Policy Gradient (NPG) [8], which enable the the objective function to be difficult to solve. Therefore, PPO-clip [22] proposes an approximate objective function (Eq. 6),

\[
r_i = \frac{\pi(a | s)}{\pi_{old}(a | s)}
\]

(5)

\[
J_{\text{PPO-clip}} = \mathbb{E}_{a_t, s_t - \pi_{old}} \left[ \min \left( r_{A_{old}}(s, a) \text{clip}(r, 1 - \epsilon, 1 + \epsilon) A_{old}(s, a) \right) \right]
\]

(6)

The function clip\((r, 1 - \epsilon, 1 + \epsilon)\) clips the ratio to be no more than \(1 - \epsilon\) and no less than \(1 + \epsilon\), which approximates the KL-divergence constraint.

3 RELATED WORKS

In this section we briefly introduce some related work, such as Independent PPO (IPPO), MAPPO-Feature-Pruned [27] and multi-agent TRPO [11].

Independent PPO (IPPO) & Non-stationarity IPPO train an independent PPO agent for each agent in the multi-agent system, and the literature [3] shows that it works effectively as well in some multi-agent tasks. However, applying the single-agent policy gradient algorithms to the multi-agent faces the problem of environmental non-stationarity. Specifically, for a certain agent \(i\) in a multi-agent system, we can treat other agents’ policies as part of the environment; then, the Bellman Equation is

\[
V^\pi_i(s) = \sum_a \pi_i(a | s) \sum_{s', r, d_i, \pi^-} p(s', r | s, d_i, \pi^-) \left( r + \gamma V^\pi_{a_i}(s') \right)
\]

(7)

where \(p(s', r | s, a, \pi^-)\) is the state transition function in the multi-agent setting, and \(\pi^-\) denotes the policies of other agents. Since the policy of each agent is updated synchronously, the state transition function \(p\) is non-stationary, and thus the convergence of the Bellman Equation cannot be guaranteed.

MAPPO extends IPPO’s independent critics to a centralized value function with global information \(s\). As for the global information, the centralized critic is more accurate than the independent critics. However, literature [3] demonstrates that it does not work well in some complex environments.

MAPPO-Feature-Pruned (MAPPO-FP) [27] finetunes the hyperparameters of MAPPO to enable it to perform well in complex multi-agent tasks such as SMAC. MAPPO-FP feeds well-designed artificial features (agent-specific features) to the
critic networks, which significantly improved MAPPO’s performance in SMAC. The agent-specific features (shown in Figure 1) concatenate the global state \( s \) with agent-specific information, such as agent actions mask and agent’s information.

**Figure 1:** MAPPO-FP concatenates artificially designed agent’s information with global state.

multi-agent TRPO Recently, [11] propose a multi-agent TRPO algorithm with a theoretical analysis. However, this algorithm can only optimize decentralized policies based on local observations and private rewards for each agent, which may not be suitable for complex cooperative tasks with shared rewards. Our method only needs the shared reward without credit assignment.

4 PRELIMINARIES

In this section, we first analyze the objective functions of MAPG and MAPPO and then pose the problem of POMAC.

**Multi-agent PG (MAPG)** MAPG trains the policies of \( N \) agents with a shared advantage function, as shown in Eq. 8,

\[
g = \sum_{t} E_{a_t,s_t} \left[ \nabla \log \pi^t \left( a^t \mid r^t \right) A(s_t, \tilde{a}_t) \right] \tag{8}
\]

where the shared advantage values are estimated with a centralized value function,

\[
A^\pi(s_t, a_t) = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) \tag{9}
\]

Fortunately, CTDE allows us to train a central value function using global information \( s \).

**Multi-agent PPO (MAPPO)** We consider a multi-agent TRPO objective function,

\[
j^{TRPO} = \frac{1}{N} \sum_{i} E_{a_t,s_t} - \pi^t \left[ \frac{\pi^t(a^t \mid r^t)}{\pi^t_{old}(a^t \mid r^t)} A^{old}(s_t, \tilde{a}_t) \right] \tag{10}
\]

\[

\text{argmin}_{\pi^t} \left[ D_{KL} \left( \pi^t \| \pi^t_{old} \right) \right] \leq \delta \tag{11}
\]

Then, we can use PPO-clip to optimize the independent policies of the agents, called vanilla Multi-agent PPO (MAPPO),

\[
r^t = \frac{\pi^t(a^t \mid r^t)}{\pi^t_{old}(a^t \mid r^t)} \tag{12}
\]

\[
j^{MAPPO-clip} = \frac{1}{N} \sum_{i} E_{a_t,s_t} - \pi^t \left[ \min \left( r^t A^{old}, \text{clip}(r^t, 1 - \epsilon, 1 + \epsilon) A^{old} \right) \right] \tag{13}
\]

Since there is no monotonicity constraint in the actor-critic methods, the expressiveness of MAPPO and MAPG is not limited.

**POMAC** Then we can obtain the expected policy gradient of MAPG and MAPPO for an agent \( i \in N \),

\[
\frac{\partial J}{\partial \pi^t(a^t \mid r^t)} \propto E_{a^t \sim \pi^t} \left[ A^\pi(s_t, a^t, \tilde{a}^{\pi|s_t}) \right] \tag{14}
\]

\[
= E_{a^t \sim \pi^t} \left[ r(s_t, a^t, \tilde{a}^{\pi|s_t}) + V(s_{t+1}) - V(s_t) \right]
\]

Interestingly, as this gradient expects the actions of agents other than agent \( i \), its value can represent the contribution of agent \( i \). Thus this marginal advantage function may be seen as a implicit multi-agent credit [4, 19, 24] for agent \( i \).

However, in practice, MAPG and MAPPO estimate this gradient via sampling. According to the large number law, we need a large number of samples with the state \( s \) to estimate this expected gradient accurately. When the number of agents \( N \) is large, it is almost impossible for us to traverse the action space of all the agents in a batch of samples to obtain the true gradient. In addition, the bias of the approximate centralized value function is large at the beginning of the training. Therefore, in practice, we can usually only obtain the sampled mean gradient with deviations. These deviations may cause the policy of agent \( i \) to be updated in a sub-optimal direction, preventing the exploration of trajectories with higher returns. We call this problem: The Policies Overfitting in Multi-agent Cooperation (POMAC).

**Figure 2:** These bar charts represents the action probability. The red area indicates the amount of improvement of the probability by the policy gradient.

To explain this problem more straightforwardly, we consider a multi-agent cooperative task with two agents. We assume that there is only one sample with reward \( r_t \) and state \( s_t \) in a batch, and the advantage value

\[
A^\pi(s_t, \tilde{a}_t) := r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) \tag{15}
\]

is obtained by agent two and is not related to agent one. The stochastic policy gradients with the this shared advantage value, i.e.

\[1\] For example, agent one and agent two are far apart and agent two gets a reward \( r_t \).
\[
\frac{\partial f}{\partial \sigma^2(\bar{a}_i | s_i)} \propto A^2(s_t, \bar{a}_t) \]
may improve probabilities of the policies of both agents, shown in Figure 2a. But intuitively, the agent one’s policy should not be updated as the advantage value is not related to agent one, shown in Figure 2b. By contrast, assuming we have an infinite number of samples to compute an unbiased advantage value: since agent one is independent of the advantage values under state \(s_t\), the value of marginal advantage function (Eq. 14) of agent one will be equal to zero.

5 METHOD

5.1 Noisy Advantage Values

Intuitively, POMAC may be solved by explicit credit assignment; that is we can decompose the centralized advantage value to each agent \(i\), i.e., \(A(s, \bar{a}) = \sum_N A^i(s, \bar{a}^i)\). In this way, the shared advantage value will do not affect unrelated agents. However, it is usually not easy to decompose the correct \(A^i\) in a multi-agent system. Since the sampled advantage values are usually biased, our core motivation is to smooth these advantage values with noise to prevent multi-agent policies overfitting caused by POMAC and environmental non-stationarity, likewise the label smoothing in image classification. In this section, we propose two noisy advantage values methods for policy regularization,

(I) Noisy-Advantage MAPPO (NA-MAPPO) demonstrated in Algo. 1 (Appendix A). We sample a gaussian noise for each agent \(i\),

\[
x^i \sim \mathcal{N}(0, 1), \forall i \in N
\]

where \(N\) is the number of agents. Next, we mix the advantage values \(A^b\) with the noises by a weight \(\alpha\) (Eq. 17), perturbing the advantage values. Then, we can use these noisy advantage values to train multi-agent policies.

\[
A^i_B = (1 - \alpha) \cdot A^b + \alpha \cdot x^i, \forall i \in N, b \in B
\]

(II) Noisy-Value MAPPO (NV-MAPPO) We randomly sample a gaussian noise vector \(\bar{x}^i \sim \mathcal{N}(0, \sigma^2)\) for each agent \(i\), where \(\sigma^2\) is the variance can be seen as the noise intensity (we fine-tune \(\sigma\) for each scenario, shown in Appendix B.2). Then we concatenate the noise \(\bar{x}^i\) with global state \(s\). As shown in Figure 3, we feed the concatenated features to the centralized value network to generate noise value \(\sigma^i\) for each agent \(i\),

\[
\sigma^i = V(\text{concat}(s, \bar{x}^i)), \forall i \in N
\]

The random noise \(\bar{x}^i\) disturbs the centralized value network and propagate to the advantage value \(A^i = r + \gamma V^i(s_t) - V^i(s_{t+1})\), perturbing the advantage values. These advantage noises bring the following benefits,

1. The advantage noises prevent the multi-agent policies overfitting caused by the sampled advantage values with deviations and environmental non-stationarity.
2. The policies trained by \(N\) noisy value networks are similar to policies ensemble.
3. The different noises \(x^i\) of each agent drive the multi-agent policies go in different directions, which encourage agents to explore diverse trajectories.

We then combine the noisy value function with MAPPO, MAPG, and IPPO to propose NV-MAPPO, Noisy-Value MAPG (NV-MAPG), and Noisy-Value IPPO (NV-IPPO), demonstrated in Algo. 2 (Appendix A). At last, we also show the difference between our method and MAPPO-FP in Table 1.

| Algo. | MAPPO-FP | Noisy-MAPPO |
|-------|-----------|-------------|
| Expert agent-specific features | Yes | No |
| Centralized Value-function | No | Yes |
| Noise | No | Yes |

Table 1: The difference between MAPPO-FP and Noisy-MAPPO.

6 EXPERINMENTS

In this section, we first evaluate the performance of NV-MAPPO, NA-MAPPO, and NV-IPPO in SMAC; and we analyze how these noises affect their performance and the entropy of the policies of MAPPO. We then evaluate the expressive power of NV-MAPPO on two non-monotonic matrix games.

6.1 Benchmark Environments

6.1.1 Starcraft Multi-agent Challenge (SMAC). [20] focuses on micromanagement challenges where each unit is controlled by an independent agent that must act based on local observations, which has become a common-used benchmark for evaluating state-of-the-art MARL approaches, such as [4, 14, 19, 23]. SMAC offers diverse sets of scenarios, which are classified as Easy, Hard, and Super Hard scenarios. We use the hardest scenarios in SMAC as our main benchmark environment.
6.1.2 Non-monotonic Matrix Game. [23] [14] show the non-monotonic matrix games that violates the monotonicity constraint. For the matrix game Table 3a (Sec. 6.3); in order to obtain the reward 8, both agents must select the first action 0 (actions are indexed from top to bottom, left to right); if only one agent selects action 0, they obtain reward -12. QMIX learns incorrect $Q_{\text{test}}$ in such non-monotonic matrix games [23] [14]. We use two payoff matrices (Sec. 6.3, Table 3a and 3b) to evaluate the expressive power of NV-MAPPO.

6.1.3 Evaluation Metric. Our primary evaluation metric is the function that maps the steps for the environment observed throughout the training to the median test-winning percentage/median test return of the evaluation. Just as in QMIX [19], we repeat each experiment with several independent training runs (five independent random experiments).

6.2 SMAC

In this section, we evaluate the performance of the algorithms on SMAC. We test our noisy value function on MAPPO and IPPO, i.e., NV-MAPPO nad NV-IPPO, respectively, in SMAC. We use the Fine-tuned QMIX [5] and MAPPO-FP as the baseline, as they achieve SOTA performance in SMAC among the previous works; we do not compare NV-MAPPO with MADDPG as the past experiments [18, 28] shows that it does not perform well under SMAC.

6.2.1 Performance Comparison. The experimental results in Table 2 demonstrate that (1) performance of NV-MAPPO significantly exceeds that of MAPPO on most hard scenarios, 2 such as 5m_vs_6m (+65%), corridor (+97%), 6h_vs_8z (+87%) and 3s5z_vs_3s6z (+31%). (2) NV-IPPO achieves extraordinarily high win rates in Super Hard scenarios 3s5z_vs_3s6z (96%) and 6h_vs_8z (94%); we speculate that this is because the noise also prevents IPPO from overfitting due to non-stationarity. (3) The average performance of NV-MAPPO on hard scenarios is better than that of Fine-tuned QMIX and MAPPO-FP. (4) We compare MAPPO and NV-MAPPO in the Appendix B.1 and show that NV-MAPPO also performs significantly better than MAPPG.

2[20] illustrates that some of these hard scenarios are more difficult to explore.

| Scenarios       | Difficulty | NV-MAPPO | NA-MAPPO | NV-IPPO | MAPPO | MAPPO-FP | IPPO | Fine-tuned QMIX |
|-----------------|------------|----------|----------|---------|-------|----------|------|----------------|
| 2s3z            | Easy       | 100%     | -        | -       | 100%  | 100%     | 100% | 100%           |
| 1c3s5z          | Easy       | 100%     | -        | -       | 100%  | 100%     | 100% | 100%           |
| 3s5z            | Easy       | 100%     | -        | -       | 100%  | 100%     | 100% | 100%           |
| 2s_vs_1sc       | Easy       | 100%     | -        | -       | 100%  | 100%     | 100% | 100%           |
| 3s_vs_5z        | Hard       | 100%     | 100%     | 100%    | 98%   | 100%     | 100% | 100%           |
| 2c_vs_64zg      | Hard       | 100%     | 100%     | 100%    | 100%  | 98%      | 100% | 100%           |
| 5m_vs_6m        | Hard       | 89%      | 85%      | 87%     | 25%   | 89%      | 87%  | 90%            |
| 8m_vs_9m        | Hard       | 96%      | 96%      | 96%     | 93%   | 96%      | 96%  | 100%           |
| MMM2            | Super Hard | 96%      | 96%      | 86%     | 96%   | 90%      | 86%  | 100%           |
| 3s5z_vs_3s6z    | Super Hard | 87%      | 72%      | 96%     | 56%   | 84%      | 82%  | 75% (env=8)    |
| 6h_vs_8z        | Super Hard | 91%      | 90%      | 94%     | 15%   | 88%      | 84%  | 91%            |
| corridor        | Super Hard | 100%     | 100%     | 98%     | 3%    | 100%     | 98%  | 100%           |
| 27m_vs_30m      | Super Hard | 100%     | 98%      | 72%     | 98%   | 94%      | 69%  | 100%           |
| Avg. Score      | Hard       | 95.5%    | 93.2%    | 91.9%   | 64.9% | 93.4%    | 88.8%| 95.1%          |

Table 2: Median test win percentage of MARL algorithms in all scenarios.

The test results for MAPPO-FP and IPPO are from [27].

All these results indicate that the noisy value function works well in practical tasks. Since we use Fine-tuned QMIX [5] as the baseline, the median test-winning rates of QMIX are significantly better than the experimental results in the past literature [14, 19, 20, 26]. So far, NV-MAPPO and NV-IPPO together achieve SOTA in SMAC. Specifically, NV-IPPO (for 3s5z_vs_3s6z and 6h_vs_8z) and NV-MAPPO (for other hard scenarios) have an average win rate of 97% for all hard scenarios.

Figure 4: Comparing Noisy-Advantage MAPPO (NA-MAPPO) with Noisy-Value MAPPO (NV-MAPPO).

The win rates of Noisy-Advantage MAPPO have a large variance.

6.2.2 Comparing NA-MAPPO with NV-MAPPO. In previous sections, We have proposed two noise-based methods, i.e., NA-MAPPO with NV-MAPPO, to resolve the POMAC. In this section, we compare their performance in SMAC. As shown in Figure 4, we find that the Noisy-Advantage method may harm the stability of the algorithm in some scenarios, i.e, the win rates of Noisy-Advantage methods have a large variance. We speculate that it may be the
explicit noises destroy the original direction of the policy gradient. However, the performance of the NA-MAPPO is still comparable to NV-MAPPO in some hard scenarios of SMAC; and we note that the NA-MAPPO is extremely easy to implement. All of these results indicate the noise advantage values do improve the performance of vanilla MAPPO.

6.2.3 Analysis of the Variance of $v^i$. Next, we perform further experimental analysis on how the noisy value function of NV-MAPPO affects the performance. We show the standard deviation of the value function $v^i$ in agent dimension for some Hard scenarios in Figure 5. We find that the large variance of $v^i$ in some scenarios implies that the performance improvement of NV-MAPPO over vanilla MAPPO in these scenarios is also large, such as 3s5z_vs_3s6z and 6h_vs_8z (see Figure 5 and Figure 4). This law reveals that the performance improvement of NV-MAPPO does come from noise perturbation of value function.

6.2.4 Analysis of Policy Entropy. Finally we analyze the effect of the noisy value function on the entropy of the policies on scenario 3s5z_vs_3s6z. As shown in Figure 6, the entropy of vanilla MAPPO’s policies drops rapidly and falls into the local optimal solution, thus the winning rate is always zero. As for NV-MAPPO, we smoothed the sampled advantage values and the noise prevents policies overfitting, thus the entropy of the policies decreases more cautiously.

6.3 Non-monotonic Matrix Game

In this section, we evaluate the expressiveness of NV-MAPPO using two non-monotonic matrix games; As shown in Figure 7a and 7b, since there are no constraints on the value function of MAPPO (e.g., monotonicity constraints), the test performance of NV-MAPPO in both of these non-monotonic games are significantly better than QMIX. Since we use the Fine-tuned QMIX, the test returns of QMIX in matrix 3b is better than that in the past literature [14].

Table 3: Non-monotonic matrix games from [23] (a) and [14](b)

| (a) Payoff matrix 1 | (b) Payoff matrix 2 |
|---------------------|---------------------|
| 8 -12 -12           | 12 0 10             |
| -12 0 0             | 10 10 10            |

Figure 6: The average entropy of the polices of vanilla MAPPO and NV-MAPPO.

7 CONCLUSION

In this paper, we propose two noisy advantage-value methods (NV-MAPPO and NA-MAPPO) to mitigate the overfitting of multi-agent policies. The experimental results show that NV-MAPPO and NV-IPPO together achieve extraordinarily high win rates in all scenarios and achieve SOTA in SMAC, without limitation of expressiveness and artificial agent-specific features. Our work demonstrates that the perturbation of policies with noisy advantage values effectively improves the performance of the multi-agent actor-critic algorithms in some scenarios.
A PSEUDOCODE

Algo. 2 and Algo. 1 demonstrate NV-MAPPO and NA-MAPPO, respectively. NV-MAPPO adds Gaussian noise to the input layer of the Value Network, and NA-MAPPO adds Gaussian noise directly to the normalized advantage values.

B EXPERIMENTAL DETAILS

B.1 Ommited Figures

Here, we echo the experiments in Sec. 6. Figure 8 shows the experimental results for Fine-tuned QMIX, MAPPO, MAPG, NV-MAPPO, and NV-MAPG, which demonstrates the performance gains from noise (MAPG vs NV-MAPG, MAPPO vs NV-MAPPO).

B.2 Hyperparameters

Our hyperparameters are hevily based on recent papers [27] and [5], who fine-tune PPO [3] and QMIX [4], respectively, to make them work well in complex multi-agent tasks, such as SMAC. Table 4 shows the common hyperparameters of QMIX and MAPPO. Table 5 shows the hyperparameters of NV-MAPPO and NA-MAPPO for each scenarios, where the values of $\sigma$ and $\alpha$ are depend on the scenarios.

For the noise shuffle interval of NV-MAPPO, we find that the performance of the fixed gaussian noise vectors $\chi^i$ is comparable to that of the noise of periodic shuffle (100 episodes), as shown in Figure 9. In addition, we do not recommend frequent noise updates because it may destroy the stability of the learning of algorithms. The idea of periodic noise updates comes from the periodic target network updates in Deep Q-networks (DQN) [16]. Note that the fixed noise vector cannot be seen as an identifier for an agent, as the value network cannot infer which agent it is just by an noise vector $\chi$ and state $s$ (unless you also feeds the observation $s'$ of the agent, likewise MAPPO-EP).

Other settings For the non-monotonic matrix games, we set the number of environments of all algorithms to 32, buffer length to 1, noise vector dim to 10, training epochs to 10, and $\sigma$ to 1. At last, we use StarCraft 2 (SC2.4.10) in the latest PyMARL in our experiments.

1PPO Code: https://github.com/marlbenchmark/on-policy
2QMIX code: https://github.com/hijkzzz/pymarl2
Algorithm 1: NA-MAPPO

**input**: Initialize parameters $\theta; \phi; \mathcal{D} \leftarrow \{\};$ batch size $B; N$ agents; noise weight $\alpha$; entropy loss weight $\eta; \lambda$ for GAE($\lambda$);

1. Sample Gaussian noise $x^i \sim N(0, 1), \forall i \in N$;
2. for each episodic iteration do
   3. for episodic step $t$ do
      4. $\tilde{a}_t = [\pi^0_b(o^i_t), \forall i \in N]$;
      5. Execute actions $\tilde{a}_t$, observe $r_t, s_{t+1}, o_{t+1}$;
      6. $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, \tilde{a}_t, \hat{A}_t, r_t, s_{t+1}, o_{t+1})\}$;
   end
3. Sample random batch $B$ from $\mathcal{D}$;
4. Compute advantage $\hat{A}_1, \ldots, \hat{A}_b$ and returns $\hat{R}_1, \ldots, \hat{R}_b$ via GAE($\lambda$);
5. then mixing the noise with the normalized advantage values:
6. $A^*_b = (1 - \alpha)\hat{A}_b + \alpha \cdot x^i, \forall i \in N, b \in B$
7. for each training epochs do
   8. Update critic by minimizing the loss $L(\phi)$;
      $L(\phi) = \frac{1}{B} \sum_{b=1}^{B} (v_b(\phi) - \hat{R}_b)^2$
9. Update policy by using loss $L(\theta)$;
   10. $r^i_b(\theta) = \frac{\pi^i_b(a^i_b|o^i_b)}{\pi^{old}_b(a^i_b|o^i_b)}, \forall i \in N, b \in B$
   11. $L(\theta) = \frac{1}{B \cdot N} \sum_{b=1}^{B} \sum_{i=1}^{N} \left[ \min(r^i_b(\theta)\hat{A}^*_b, \text{clip}(r^i_b(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}^*_b) \right.$
   12. $\left. - \eta H(\pi^i_b(o^i_b)) \right]$
   13. where $H$ is the Shannon Entropy.
14. end
end

Algorithm 2: NV-MAPPO

**input**: Initialize parameters $\theta; \phi; \mathcal{D} \leftarrow \{\};$ batch size $B; N$ agents; noise variance $\sigma^2$; entropy loss weight $\eta; \lambda$ for GAE($\lambda$);

1. Sample random noise vectors $\bar{x}^i$ for each agent, $\bar{x}^i \sim N(0, \sigma^2), \forall i \in N$;
2. for each episodic iteration do
   3. for episodic step $t$ do
      4. $\tilde{a}_t = [\pi^0_b(o^i_t), \forall i \in N]$;
      5. Execute actions $\tilde{a}_t$, observe $r_t, s_{t+1}, o_{t+1}$;
      6. $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, \tilde{a}_t, r_t, s_{t+1}, o_{t+1})\}$;
   end
3. if at noise vectors shuffle interval then
   4. Shuffle the noise vectors $\bar{x}^i$ in agent dimension.
end
4. Sample random batch $B$ from $\mathcal{D}$;
5. Noise value function forward for each agent, $v_b^i(\phi) = V(\phi, \text{concat}(s_b, \bar{x}^i)), \forall i \in N, b \in B$;
6. Compute advantage $\hat{A}^1, \ldots, \hat{A}^i$ and returns $\hat{R}^1, \ldots, \hat{R}^i$ via GAE($\lambda$) with $v_b^i(\phi), \forall i \in N, b \in B$;
7. for each training epochs do
   8. Update critic by minimizing the loss $L(\phi)$;
      $L(\phi) = \frac{1}{B \cdot N} \sum_{b=1}^{B} \sum_{i=1}^{N} (v_b^i(\phi) - \hat{R}_b^i)^2$
9. Update policy by using loss $L(\theta)$;
   10. $r^i_b(\theta) = \frac{\pi^i_b(a^i_b|o^i_b)}{\pi^{old}_b(a^i_b|o^i_b)}, \forall i \in N, b \in B$
   11. $L(\theta) = \frac{1}{B \cdot N} \sum_{b=1}^{B} \sum_{i=1}^{N} \left[ \min(r^i_b(\theta)\hat{A}^*_b, \text{clip}(r^i_b(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}^*_b) \right.$
   12. $\left. - \eta H(\pi^i(b|o^i_b)) \right]$
   13. where $H$ is the Shannon Entropy.
end
Table 4: Common hyperparameters used in the SMAC domain for all algorithms.

| hyperparameters | MAPPO and MAPG | QMIX |
|-----------------|----------------|------|
| num envs        | 8              | 8    |
| buffer length   | 400            | -    |
| batch size (episodes) | -     | 128  |
| num GRU layers  | 1              | 1    |
| RNN hidden state dim | 64       | 64   |
| fc layer dim    | 64             | 64   |
| num fc before RNN | 1       | 1    |
| num fc after RNN | 1             | 1    |
| num noise dim   | 10             | -    |
| Adam [9] lr     | 5e-4           | 1e-3 |
| Q(\lambda)     | -              | 0.6, (0.3 for 6h_vs_8z) |
| GAE(\lambda)   | 0.95           | -    |
| entropy coef    | 0.01           | -    |
| PPO clip        | 0.2            | -    |
| noise shuffle interval (episodes) | +∞     | -    |
| \epsilon anneal steps | -      | 100k, (500k for 6h_vs_8z) |

Table 5: Hyperparameters for NV-MAPPO, NA-MAPPO, NV-IPPO, NV-MAPG and vanilla MAPPO in SMAC.

| map             | PPO epochs | mini-batch | gain | network | stacked frames | NV-MAPPO | NV-MAPG | NV-IPPO | NA-MAPPO |
|-----------------|------------|------------|------|---------|----------------|----------|---------|---------|----------|
|                 |            |            |      |         |                | \sigma   | \sigma  | \sigma  | \sigma   |
| 2s3z            | 15         | 1          | 0.01 | rnn     | 1              | 1        | 1       | -       | -        |
| 1c3s5z          | 15         | 1          | 0.01 | rnn     | 1              | 1        | 1       | -       | -        |
| 3s5z            | 5          | 1          | 0.01 | rnn     | 1              | 1        | 1       | -       | -        |
| 2s_vs_1sc       | 15         | 1          | 0.01 | rnn     | 1              | 1        | 1       | -       | -        |
| 3s_vs_5z        | 15         | 1          | 0.01 | mlp     | 4              | 1        | 1       | 1       | 0.05     |
| 2c_vs_64zg      | 5          | 1          | 0.01 | rnn     | 1              | 1        | 1       | 1       | 1        |
| 5m_vs_6m        | 10         | 1          | 0.01 | rnn     | 1              | 8        | 3       | 0       | 0.05     |
| 8m_vs_9m        | 15         | 1          | 0.01 | rnn     | 1              | 1        | 0.05    | 1       | 0.05     |
| corridor        | 5          | 1          | 0.01 | mlp     | 1              | 3        | 1       | 1       | 0.06     |
| MMM2            | 5          | 2          | 1    | rnn     | 1              | 0        | 0.5     | 0       | 0        |
| 3s5z_vs_3s6z    | 5          | 1          | 0.01 | rnn     | 1              | 10       | 1       | 8       | 0.05     |
| 6h_vs_8z        | 5          | 1          | 0.01 | mlp     | 1              | 1        | 1       | 1       | 0.06     |
| 27m_vs_30m      | 5          | 1          | 0.01 | rnn     | 1              | 1        | 1       | 1       | 0        |

Table 4: Common hyperparameters used in the SMAC domain for all algorithms.