Policy Adaptive Multi-agent Deep Deterministic Policy Gradient

Yixiang Wang and Feng Wu(✉)

School of Computer Science and Technology,
University of Science and Technology of China, Hefei, China
yixiangw@mail.ustc.edu.cn, wufeng02@ustc.edu.cn

Abstract. We propose a novel approach to address one aspect of the non-stationarity problem in multi-agent reinforcement learning (RL), where the other agents may alter their policies due to environment changes during execution. This violates the Markov assumption that governs most single-agent RL methods and is one of the key challenges in multi-agent RL. To tackle this, we propose to train multiple policies for each agent and postpone the selection of the best policy at execution time. Specifically, we model the environment non-stationarity with a finite set of scenarios and train policies fitting each scenario. In addition to multiple policies, each agent also learns a policy predictor to determine which policy is the best with its local information. By doing so, each agent is able to adapt its policy when the environment changes and consequentially the other agents alter their policies during execution. We empirically evaluated our method on a variety of common benchmark problems proposed for multi-agent deep RL in the literature. Our experimental results show that the agents trained by our algorithm have better adaptiveness in changing environments and outperform the state-of-the-art methods in all the tested environments.

Keywords: Reinforcement learning · Multi-agent reinforcement learning · Multi-agent deep deterministic policy gradient

1 Introduction

The development of modern deep learning has made reinforcement learning (RL) more powerful to solve complex decision problems. This leads to success in many real-world applications, such as Atari games [19], playing Go [22] and robotics control [12]. Recently, there is growing focus on applying deep RL techniques to multi-agent systems. Many promising approaches for multi-agent deep RL have been proposed to solve a variety of multi-agent problems, such as traffic control

This work was supported in part by the National Key R&D Program of China (Grant No. 2017YFB1002204), the National Natural Science Foundation of China (Grant No. U1613216, Grant No. 61603368), and the Guangdong Province Science and Technology Plan (Grant No. 2017B010110011).

© Springer Nature Switzerland AG 2021
T. Uchiya et al. (Eds.): PRIMA 2020, LNAI 12568, pp. 165–181, 2021.
https://doi.org/10.1007/978-3-030-69322-0_11

[18,27], multi-player games (e.g., StarCraft, Dota 2), and multi-robot systems [16].

Despite the recent success of deep RL in single-agent domains, there are additional challenges in multi-agent RL. One major challenge is the *non-stationarity* of multi-agent environment caused by agents that change their policies during the training and testing procedures. Specifically, at the training time, each agent's policy is changing simultaneously and therefore the environment becomes non-stationary from the perspective of any individual agent. To handle this issue, *multi-agent deep deterministic policy gradient* (MADDPG) [17] proposed to utilized a *centralized critic* with *decentralized actors* in the actor-critic learning framework. Since the centralized Q-function of each agent is conditioned on the actions of all the other agents, each agent can perceive the learning environment as stationary even when the other agents' policies change.

Although using a centralized critic stabilizes training, the learned policy of each agent can still be brittle and sensitive to its training environment and partners. It has been observed that the performance of the learned policies can be drastically worse when some agents alter their policies during execution [11]. To improve the robustness of the learned policies, *minimax multi-agent deep deterministic policy gradient* (M3DDPG) [13]—a *minimax* extension of MADDPG—proposed to update policies considering the worst-case situation by assuming that all the other agents acts adversarially. This minimax optimization is useful to learn robust policies in very competitive domains but can be too *pessimistic* in mixed competitive and cooperative or fully cooperative problems as shown later in our experiments.

In this paper, we consider one aspect of the non-stationarity issue in multi-agent RL, where the other agents may alter their policies as a result of changes in some environmental factors. This frequently happens in real-world activities. For example, in a soccer game, a heavy rain or high temperature usually causes the teams to change their strategies against each other. Take disaster response as another example. First responders often need to constantly adjust their plan in order to complete their tasks in the highly dynamic and danger environment. Therefore, it is often desirable for the agents to learn policies that can adapt with changes of the environment and others' policies.

Against this background, we propose *policy adaptive multi-agent deep deterministic policy gradient* (PAMADDPG)—a novel approach based on MADDPG—to learn adaptive policies for non-stationary environments. Specifically, it learns multiple policies for each agent and postpone the selection of the best policy at execution time. By doing so, each agent is able to adapt its policy when the environment changes. Specifically, we model the non-stationary environment by a finite set of known scenarios, where each scenario captures possible changing factors of the environment (e.g., weather, temperature, wind, etc. in soccer). For each scenario, a policy is learned by each agent to perform well in that specific scenario. Together with multiple policies for each agent, we also train a policy predictor to predict the best policy using the agent's local information. At execution time, each agent first selects a policy based on the policy

predictor and then choose an action according to the selected policy. We evaluated our PAMADDPG algorithm on three common benchmark environments and compared it with MADDPG and M3DDPG. Our experimental results show that PAMADDPG outperforms both MADDPG and M3DDPG in all the tested environments.

The rest of the paper is organized as follows. We first briefly review the related work about handling non-stationary in multi-agent deep RL. Then, we describe the background on the Markov game and the MADDPG method, which are building blocks of our algorithm. Next, we propose our PAMADDPG algorithm to learn multiple policies and policy predictors. After that, we present the experiments with environments, setup, and results. Finally, we conclude the paper with possible future work.

2 Related Work

In recent years, various approaches [20] have been proposed to tackle different aspects of non-stationarity in multi-agent deep RL. We sample a few related work about multi-agent deep RL as listed below.

2.1 Centralized Critic

Using the centralized critic techniques, [17] proposed MADDPG for multi-agent RL using a centralized critic and a decentralized actor, where the training of each agent is conditioned on the observation and action of all the other agents so the agent can perceive the environment as stationary. [13] extended MADDPG and proposed M3DDPG using minimax Q-learning in the critic to exhibit robustness against different adversaries with altered policies. [8] proposed COMA using also a centralized critic with the counterfactual advantage estimation to address the credit assignment problem—another key challenge in multi-agent RL.

2.2 Decentralized Learning

A useful decentralized learning technique to handle non-stationarity is self-play. Recent self-play approaches store the neural network parameters at different points during learning. By doing so, self-play managed to train policies that can generalize well in environments like Go [23] and complex locomotion tasks [2]. Another technique [6] is by stabilizing experience replay using importance sampling corrections to adjust the weight of previous experience to the current environment dynamics.

2.3 Opponent Modeling

By modeling the opponent, [9] developed a second separate network to encode the opponent's behaviour. The combination of the two networks is done either by concatenating their hidden states or by the use of a mixture of experts.

In contrast, [21] proposed an actor-critic method using the same policy network for estimating the goals of the other agents. [5] proposed a modification of the optimization function to incorporate the learning procedure of the opponents in the training of agents.

2.4 Meta-learning

By extending meta-learning approaches for single-agent RL such as model agnostic meta-learning [3] to handle non-stationarity in multi-agent domains, [1] proposed an optimization method to search for initial neural network parameters that can quickly adapt to non-stationarity, by explicitly optimizing the initial model parameters based on their expected performance after learning. This was tested in *iterated adaptation games*, where an agent repeatedly play against the same opponent while only allowed to learn in between each game.

2.5 Communication

In this direction, [7] proposed the deep distributed recurrent Q-networks, where all the agents share the same hidden layers and learn to communicate to solve riddles. [26] proposed the CommNet architecture, where the input to each hidden layer is the previous layer and a communication message. [25] proposed the individualized controlled continuous communication model, which is an extension of CommNet in competitive setting. [4] proposed reinforced inter-agent learning with two Q-networks for each agents where the first network outputs an action and the second a communication message.

As briefly reviewed above, most of the existing work focus on handling non-stationarity mainly during training procedure. Although meta-learning approaches can learn to adapt agents' policies between different game, it requires to *repeatedly* play iterated adaptation games. In contrast, we build our algorithm on top of MADDPG to address the non-stationarity problem in general multi-agent RL at execution time. Besides, we do not assume explicit communication among the agents during execution as in MADDPG.

A complete survey about recent efforts of dealing non-stationarity in multi-agent RL can be found in [10,20].

3 Background

In this section, we introduce our problem settings and some basic algorithms on which our approach is based.

3.1 Partially Observable Markov Games

In this work, we consider a *partially observable Markov games* [15] with N agents, defined by: a set of states $\mathcal{S}$ describing the possible configurations of all agents, a set of actions $\mathcal{A}_1, \ldots, \mathcal{A}_N$ and a set of observations $\mathcal{O}_1, \ldots, \mathcal{O}_N$ for each agent.

To choose actions, each agent i uses a stochastic policy $\mu_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \mapsto [0, 1]$, which produces the next state according to the state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \mapsto \mathcal{S}$.

At each time step, each agent i obtains rewards as a function of the state and agent's action $r_i : \mathcal{S} \times \mathcal{A}_i \mapsto \mathbb{R}$, and receives a local observation correlated with the state $\mathbf{o}_i : \mathcal{S} \mapsto \mathcal{O}_i$. The initial states are determined by a state distribution $\rho : \mathcal{S} \mapsto [0, 1]$. Each agent i aims to maximize its own total expected return: $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$, where $\gamma \in (0, 1]$ is a discount factor and T is the time horizon.

Here, we assume that the state transition function $\mathcal{T}$ is unknown and therefore consider to learn the policies μ_{θ_i} for each agent i using multi-agent *reinforcement learning* (RL) methods. Note that each agent must choose an action based on its own policy and local observation during execution.

3.2 Multi-agent Deep Deterministic Policy Gradient

Policy gradient methods are a popular choice for a variety of RL tasks. The main idea is to directly adjust the parameters θ of the policy in order to maximize the objective $J(\theta) = \mathbb{E}_{s\sim p^\mu, a\sim\mu_\theta}[R(s, a)]$ by taking steps in the direction of $\nabla_\theta J(\theta)$, i.e., the gradient of the policy written as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^\mu, a\sim\mu_\theta} \left[\nabla_\theta \log \mu_\theta(a|s) Q^\mu(s, a)\right] \tag{1}$$

where p^μ is the state distribution and Q^μ is the Q-function.

The policy gradient framework has been extended to deterministic policies $\mu_\theta : \mathcal{S} \mapsto \mathcal{A}$. In particular, under certain conditions the gradient of the objective $J(\theta) = \mathbb{E}_{s\sim p^\mu}[R(s, a)]$ can be written as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mathcal{D}} \left[\nabla_\theta \mu_\theta(a|s) \nabla_a Q^\mu(s, a)\big|_{a=\mu_\theta(s)}\right] \tag{2}$$

Since the *deterministic policy gradient* (DPG) [24] relies on $\nabla_a Q^\mu(s, a)$, it requires that the action space $\mathcal{A}$ (and thus the policy μ) be continuous. *Deep deterministic policy gradient* (DDPG) [14] is a variant of DPG where the policy μ and critic Q^μ are approximated with deep neural networks. DDPG is an off-policy algorithm, and samples trajectories from a replay buffer of experiences that are stored throughout training. It also makes use of a target network, as in DQN [19].

Multi-agent DDPG (MADDPG) [17] extends the DDPG method to multi-agent domains. The main idea behind MADDPG is to consider action policies of other agents. The environment is stationary even as the policies change, since $P(s'|s, a_1, \ldots, a_N, \pi_1, \ldots, \pi_N) = P(s'|s, a_1, \ldots, a_N) = P(s'|s, a_1, \ldots, a_N, \pi'_1, \ldots, \pi'_N)$ for any $\pi_i \neq \pi'_i$. The gradient can be written as:

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{\mathbf{x}, a\sim\mathcal{D}} \left[\nabla_{\theta_i} \mu_i(a_i|o_i) \nabla_{a_i} Q_i^\mu(\mathbf{x}, a_1, \ldots, a_N)\big|_{a_i=\mu_i(o_i)}\right] \tag{3}$$

where $Q_i^\mu(\mathbf{x}, a_1, ..., a_N)$ is a *centralized action-value function* that takes as input the actions of all agents, $a_1, \ldots, a_N$, in addition to the state information $\mathbf{x}$, and

outputs the Q-value for agent i. Here, Q_i^{μ} can be updated as:

$$\begin{aligned} \mathcal{L}(\theta_i) &= \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'}\left[(Q_i^{\mu}(\mathbf{x},a_1,\ldots,a_N)-y)^2\right], \\ y &= r_i + \gamma Q_i^{\mu'}(\mathbf{x}',a_1',\ldots,a_N')\big|_{a_j'=\mu_j'(o_j)} \end{aligned} \tag{4}$$

where $(\mathbf{x},a,r,\mathbf{x}')$ is sampled from the experience replay buffer $\mathcal{D}$, recoding experiences of all agents.

3.3 Dealing Non-stationarity in MADDPG

As aforementioned, one of the key challenges in multi-agent RL is the environment non-stationarity. This non-stationarity stems from breaking the Markov assumption that governs most single-agent RL algorithms. Since the transitions and rewards depend on actions of all agents, whose decision policies keep changing in the learning process, each agent can enter an endless cycle of adapting to other agents. Although using a centralized critic stabilizes training in MADDPG, the learned policies can still be brittle and sensitive to changes of the other agents's policies.

To obtain policies that are more robust to changes in the policy of other agents, MADDPG proposes to first train a collection of K different sub-policies and then maximizing the ensemble objective $\max_{\theta_i} J(\theta_i)$ as:

$$\begin{aligned} J(\theta_i) &= \mathbb{E}_{k\sim\text{uniform}(1,K),s\sim p^{\mu},a\sim\mu^{(k)}}[R_i(s,a)] \\ &= \mathbb{E}_{k,s}\left[\sum_{t=0}^{T}\gamma^t r_i(s^t,a_1^t,\ldots,a_N^t)\Big|_{a_i^t=\mu_i^{(k)}(o_i^t)}\right] \\ &= \mathbb{E}_s\left[\frac{1}{K}\sum_{k=1}^{K}Q_i^{\mu}(s,a_1,\ldots,a_N)\Big|_{a_i=\mu_i^{(k)}(o_i)}\right] \end{aligned} \tag{5}$$

where $\mu_i^{(k)}$ is the k-th sub-policies of agent i. By training agents with an ensemble of policies, the agents require interaction with a variety of the other agents' policies. Intuitively, this is useful to avoid converging to local optima of the agents' policies. However, the ensemble objective only considers the *average* performance of agents' policies training by *uniformly* sampling the policies of the other agents.

Alternatively, M3DDPG [13]—a variation of MADDPG—proposes to update policies considering the worst situation for the purpose of learning robust policies. During training, it optimizes the policy of each agent i under the

Algorithm 1: Training and execution for PAMADDPG

```
# At training time:
∀i : Π_i ← ∅, φ_i ← initialize the predictor parameters
foreach scenario c ∈ C do
    ∀i : Π_i ← learn and add a set of policies for agent i
    ∀i : φ_i ← learn and update the predictor for agent i
# At execution time:
∀i : h_i^0 ← ∅
for time step t = 1 to T do
    for agent i = 1 to N do
        o_i^t ← receive a local observation for agent i
        μ_i ← select a policy from Π_i by φ_i(o_i^t, h_i^{t-1})
        a_i^t ← select an action by μ_{θ_i}(o_i^t)
        h_i^t ← append o_i^t to h_i^{t-1}
    Execute actions ⟨a_1^t, ..., a_N^t⟩ to the environment
    Collect rewards ⟨r_1^t, ..., r_N^t⟩ from the environment
return ∀i : R_i = Σ_{t=0}^{T} γ^t r_i^t
```

assumption that all other agents acts adversarially, which yields the minimax objective $\max_{\theta_i} J(\theta_i)$ as:

$$\begin{aligned} J(\theta_i) &= \min_{a_{j\neq i}} \mathbb{E}_{s\sim p^{\mu}, a_i\sim\mu_i}[R_i(s,a)] \\ &= \min_{a_{j\neq i}^t} \mathbb{E}_s \left[\sum_{t=0}^{T} \gamma^t r_i(s^t, a_1^t, \ldots, a_N^t)\Big|_{a_i^t=\mu_i(o_i^t)} \right] \\ &= \mathbb{E}_s \left[\min_{a_{j\neq i}} Q_{M,i}^{\mu}(s, a_1, \ldots, a_N)\Big|_{a_i=\mu_i(o_i)} \right] \end{aligned} \tag{6}$$

where $Q_{M,i}^{\mu}(s, a_1, \ldots, a_N)$ is the modified Q function representing the current reward of executing $a_1, \ldots, a_N$ in s plus the discounted worst case future return starting from s. With the minimax objective, the training environment of each agent becomes stationary because the behavior of all the other agents only depends on $-r_i$, i.e., the negative reward of agent i itself. However, this adversarial assumption could be too *pessimistic* if the game among the agents is not zero-sum or even is cooperative.

Ideally, the well trained agents should be able to *adapt* their policies with the changes in the environment. This motivated the development of our algorithm that will be introduced in details next.

4 Policy Adaptive MADDPG

In this section, we propose *policy adaptive multi-agent deep deterministic policy gradient* (PAMADDPG), which is based on MADDPG, to deal with environment

non-stationarity in multi-agent RL. As in MADDPG, our algorithm operate under the framework of *centralized training* with *decentralized execution*. Thus, we allow the agents to share extra information for training, as long as this information is not used at execution time. We assume that the learned policies can only use local information and there is no explicit communication among agents during execution. Specifically, our algorithm is an extension of actor-critic policy gradient methods with multiple decentralized actors and one centralized critic, where the critic is augmented with extra information on the policies of the other agents.

In this work, we consider a setting where agents are trained and executed in an environment that can categorized into a finite set of scenarios. These scenarios are known during training. However, at execution time, agents have no prior knowledge about which scenario they will locate in. Therefore, the agents must act adaptively during execution. Note that the scenarios cannot be modeled as state variables because we make no assumption about the initial distribution and transition probabilities of scenarios, which can be any probabilities in our setting. Intuitively, a scenario in our setting models a collection of environmental factors that can cause the agents to alter their policies.

Let $\mathcal{C}$ denote a finite set of scenarios for the agents. Here, each scenario $c \in \mathcal{C}$ can be modeled by a partially observable Markov game as aforementioned. We assume that all the scenarios in $\mathcal{C}$ have identical state space and the same action and observation space for all the agents. Particularly, each scenario $c \in \mathcal{C}$ may have different state transition function $\mathcal{T}^c$ and different reward function r_i^c for each agent i, so that agents in different scenarios may require different policies. Formally, we define a scenario $c \in \mathcal{C}$ as a tuple: $\langle \mathcal{S}, \{\mathcal{A}_i\}, \{\mathcal{O}_i\}, \mathcal{T}^c, \{r_i^c\} \rangle$ with notations in Markov games.

As aforementioned, to be able to adapt in different scenarios, we propose to train multiple policies for each agent and postpone the selection of its policy at execution time. In addition to multiple policies for each agent, we also train a policy predictor that can be used by the agent to determine the best policy during execution. Given this, the agent is able to adapt its policy when the environment changes. As summarized in Algorithm 1, PAMADDPG consists of two main procedures: 1) learning multiple policies and 2) learning policy predictors, which will be described in details next.

4.1 Learning Multiple Policies

We can extend the actor-critic policy gradient method as described in MADDPG to work with each scenario. Specifically, given a scenario $c \in \mathcal{C}$, the gradient for policy μ_i^c with respect to parameters θ_i^c can be written as:

$$\nabla_{\theta_i^c} J(\mu_i^c) = \mathbb{E}_{\mathbf{x}, a \sim \mathcal{D}^c} \left[\nabla_{\theta_i^c} \mu_i^c(a_i|o_i) \nabla_{a_i} Q_i^{\mu,c}(\mathbf{x}, a_1, \ldots, a_N) \big|_{a_i = \mu_i^c(o_i)} \right] \quad (7)$$

where $\mathcal{D}^c$ is the experience replay buffer recording experiences with tuples $(\mathbf{x}, a_1, \ldots, a_N, r_1^c, \ldots, r_N^c, \mathbf{x}')$ of all agents at the scenario c and $\mathbf{x} = (o_1, \ldots, o_N)$.

Here, the centralized action-value function $Q_i^{\mu,c}$ is updated as:

$$\begin{aligned} \mathcal{L}(\theta_i^c) &= \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'}[(Q_i^{\mu,c}(\mathbf{x}, a_1, \ldots, a_N) - y)^2] \\ y &= r_i + \gamma\, Q_i^{\mu',c}(\mathbf{x}', a'_1, \ldots, a'_N)\big|_{a'_j=\mu_j'^c(o_j)} \end{aligned} \tag{8}$$

where $\mu'^c = \{\mu_{\theta'^c_1}, \ldots, \mu_{\theta'^c_N}\}$ is the set of target policies with delayed parameters θ'^c_i.

Here, the key challenge is that policies trained by MADDPG may converge to different local optima. Therefore, the other agents may choose policies that are different from the ones learned by MADDPG. To address this, we propose to train a collection of K different policies for each agent in a single scenario. Each policy can have different initial parameters and selection of the partners' policies. This will grow the populations in the policy set of each agent and further improve the robustness during testing. Unlike MADDPG, we do not ensemble the K policies to a single policy but keep all the individual policies as candidates for execution.

4.2 Learning Policy Predictors

We denote $\phi_i : \mathcal{H}_i \rightarrow \Delta(\Pi_i)$ the policy predictor that uses agent i's local observation history $h_i^t = (o_i^1, \ldots, o_i^t)$ to compute the distribution over agent i's policy set Π_i. Our goal is to determine at execution time which policy should be used by agent i in order to achieve the best performance. Here, we use a recurrent neural network to train a policy predictor ϕ_i, containing a layer of LSTM and some other layers. This structure allows the agent to reason about the current scenario using its observation sequence.

Here, $\phi_i(o_i^t, h_i^{t-1})$ is a function that takes the input of the current observation o_i^t and the last-step history h_i^{t-1} at the time step t, and outputs the policy distribution $p_i^t(\cdot) \in [0, 1]$ over agent i's policy set Π_i. Now, the action selection process of agent i at time step t can be written as:

$$\begin{aligned} p_i^t &= \phi_i(o_i^t, h_i^{t-1}) \\ \mu_i &= \arg\max_{\mu'_i \in \Pi_i} p_i^t(\mu'_i) \\ a_i^t &= \mu_{\theta_i}(o_i^t) \end{aligned} \tag{9}$$

Together with training the policy, we use replay buffer to train ϕ_i in order to avoid the early instability and adverse effects during training process. Specifically, we create a dedicated replay buffer $\mathcal{B}_i$ for ϕ_i during training. It stores (h_i, μ_i) at the end of each episode, where $h_i = (o_i^1, \ldots, o_i^T)$ is agent i's observation sequence at this episode and μ_i is the currently trained policy. The main training procedure of ϕ_i is to sample a random minibatch of samples (h_i, μ_i) from $\mathcal{B}_i$ and update the parameters of ϕ_i by minimizing the cross-entropy loss

function as follow:

$$\begin{aligned}\nabla_{p_i} J(\phi_i) &= \mathbb{E}_{(h_i,\mu_i)\sim\mathcal{B}_i}\left[\sum_{t=1}^{T} \mathrm{CE}\left(\phi_i(o_i^t, h_i^{t-1}), t\right)\right] \\ &= \mathbb{E}_{(h_i,\mu_i)}\left[\sum_{t=1}^{T}\sum_{\mu_i'\in\Pi_i} -y^{\mu_i'} \log\left(p_i^t(\mu_i')\right)\right] \qquad (10)\end{aligned}$$

$$\text{where } y^{\mu_i'} = \begin{cases} 1, & \mu_i' = \mu_i \\ 0, & \mu_i' \neq \mu_i \end{cases} \text{ and } p_i^t = \phi_i(o_i^t, h_i^{t-1}).$$

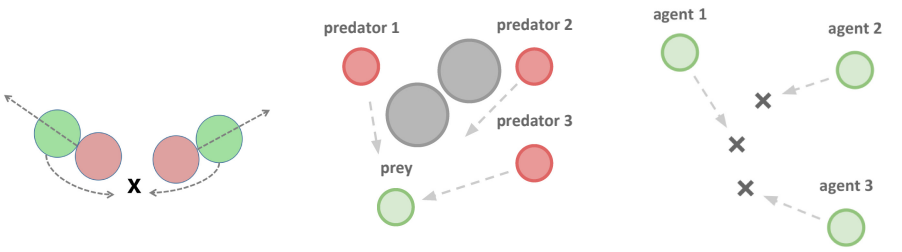
The overall learning procedures of PAMADDPG are outlined in Algorithm 2.

5 Experiments

We empirically evaluate our algorithm on three domains built on top of the particle-world environments[1] originally used by the MADDPG paper [17]. To create various scenarios, we modify some of the physical properties of the environments so that the agents must alter their policies in order to success in different scenarios. By doing so, we expect to examine the adaptiveness of our PAMADDPG algorithm when testing in different scenarios.

5.1 Environments

The particle world environment consists of N cooperative agents, M adversarial agents and L landmarks in a two-dimensional world with continuous space. In the experiments, we consider two mixed cooperative and competitive domains (i.e., Keep-away and Predator-prey) and one fully cooperative domain (i.e., Cooperative navigation), as shown in Fig. 1, and modify these domains to generate different scenarios as below.



(a) Keep-away (b) Predator-prey (c) Cooperative navigation

Fig. 1. Illustrations of the three environments.

[1] Code from: https://github.com/openai/multiagent-particle-envs.

Algorithm 2: Learning agents' policies and predictors

foreach *episode* **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observations $\mathbf{x} = (o_1, \ldots, o_N)$
  for *time step* $t = 1$ **to** T **do**
    For each agent i, select $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t the current policy and exploration noise
    Execute action $a = (a_1, \ldots, a_N)$ and observe reward $r = (r_1, \ldots, r_N)$ and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in $\mathcal{D}$ and set $\mathbf{x} \leftarrow \mathbf{x}'$
    for *agent* $i = 1$ **to** N **do**
      Sample a random minibatch of M samples $(\mathbf{x}^m, a^m, r^m, \mathbf{x}'^m)$ from replay buffer $\mathcal{D}$
      Set $y^m = r_i^m + \gamma\, Q_i^{\mu'}(\mathbf{x}', a')\big|_{a'_j = \mu'_j(o_j^m)}$
      Update critic by minimizing the loss:

$$\mathcal{L}(\theta_i) = \frac{1}{M} \sum_{m=1}^{M} (y^m - Q_i^{\mu}(\mathbf{x}^m, a^m))^2$$

      Update actor using the sampled gradient:

$$\nabla_{\theta_i} J(\mu_i) \approx \frac{1}{M} \sum_{m=1}^{M} \nabla_{\theta_i} \mu_i(o_i^m) \nabla_{a_i} Q_i^{\mu}(\mathbf{x}^m, a^m)\big|_{a_i = \mu_i(o_i^m)}$$

      Sample a random minibatch of K samples (h_i^k, μ_i^k) from replay buffer $\mathcal{B}_i$
      Update predictor ϕ_i by minimizing the loss:

$$\nabla_{p_i} J(\phi_i) \approx \frac{1}{K} \sum_{k=1}^{K} \sum_{t=1}^{T} \sum_{\mu'_i} -y^{\mu'_i} \log(p_i^t(\mu'_i))$$

    Update target network parameters θ_i for each agent i as: $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
  Collect history $h_i = (o_i^1, \ldots, o_i^T)$ and store (h_i, μ_i) in replay buffer $\mathcal{B}_i$ for each agent i

Keep-Away. This environment consists of L landmarks including a target landmark, $N = 2$ cooperating agents who know the target landmark and are rewarded based on their distance to the target, and $M = 2$ agents who must prevent the cooperating agents from reaching the target. Adversaries accomplish this by physically pushing the agents away from the landmark, temporarily occupying it. While the adversaries are also rewarded based on their distance to the target landmark, they do not know the correct target.

We create $K = 3$ scenarios that require agents to learn to adapt with. In each scenario, we simulate different "wind" conditions in the environment. The wind will affect the moving speed of the agents in a certain direction computed as: $v_i' = v_i + w * \beta_i$, where v_i is the original speed, $w = [w_N, w_W, w_S, w_E]$ is the wind force for four directions, and $\beta_i = 5$ is the acceleration rate. In the experiments, we consider no wind (i.e., $w = 0$) in Scenario 1, southwest wind (i.e., $w_S = w_W = 0.5$ and 0 otherwise) in Scenario 2, and northeast wind (i.e., $w_N = w_E = 0.5$ and 0 otherwise) in Scenario 3 respectively.

Predator-Prey. In this environment, $N = 4$ slower cooperating agents must chase $M = 2$ faster adversary around a randomly generated environment with $L = 2$ large landmarks impeding the way. Each time the cooperative agents collide with an adversary, the agents are rewarded while the adversary is penalized. Agents observe the relative positions and velocities of the agents, and the landmark positions.

We create $K = 3$ scenarios to simulate different body conditions for the good and bad agents. This is done by using different maximum speeds $\bar{v}$ and accelerations β for the agents in the environment, i.e., $(\bar{v}_{good}, \beta_{good}, \bar{v}_{bad}, \beta_{bad})$. We set the parameters so that the agents will compete in different levels, i.e., weak, medium, and strong. Specifically, we set $(3.0, 3.0, 3.9, 4.0)$ in Scenario 1, $(2.0, 4.0, 2.6, 5.0)$ in Scenario 2, and $(3.0, 5.0, 3.9, 6.0)$ in Scenario 3.

Cooperative Navigation. In this environment, agents must cooperate through physical actions to reach a set of L landmarks. Agents observe the relative positions of other agents and landmarks, and are collectively rewarded based on the proximity of any agent to each landmark. In other words, the agents have to "cover" all of the landmarks. Furthermore, the agents occupy significant physical space and are penalized when colliding with each other.

Similar to the Keep-away environment described above, we created $K = 3$ scenarios in this environment also with three wind conditions, i.e., no wind for Scenario 1, southeast wind for Scenario 2, and northwest wind for Scenario 3.

5.2 Setup

We compared our PAMADDPG algorithm with MADDPG[2] and M3DDPG[3], which are currently the leading algorithms for multi-agent deep RL, on the environments as described above. In our implementation, the agents' policies are represented by a two-layer ReLU MLP with 64 units per layer, which is the same as MADDPG and M3DDPG, and the policy predictors are represented by a two-layer ReLU MLP and a layer of LSTM on top of them.

We used the same training configurations as MADDPG and M3DDPG, and ran all the algorithms until convergence. Then, we tested the policies computed

[2] Code from: https://github.com/openai/maddpg.
[3] Code from: https://github.com/dadadidodi/m3ddpg.

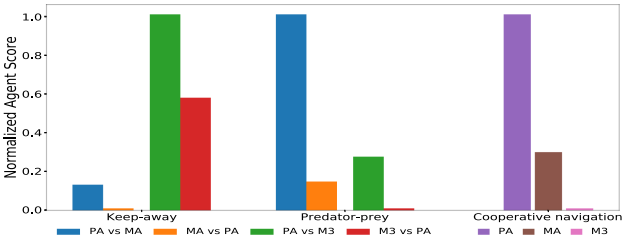


Fig. 2. Overall performance of PAMADDPG (PA), MADDPG (MA), and M3DDPG (M3) on the environments.

by the algorithms on each environment with 10,000 further episodes and report the averaged results. For fair comparison, all algorithms were tested on a fixed set of environment configurations. Each testing environment is generated by randomizing the basic configurations and randomly selecting a scenario. As aforementioned, the agents do not know which scenario is selected for the environment during testing procedure.

Note that MADDPG and M3DDPG do not consider different scenarios in their original implementations. For fair comparison, we try to train their policies in a way that their performance is improved when working with different scenarios. Specifically, in our experiments, MADDPG trained policies with all scenarios and optimized the objective as:

$$J(\theta_i) = \mathbb{E}_{c\sim\text{uniform}(\mathcal{C}),s\sim p^c,a\sim\mu}[R_i(s,a)] \tag{11}$$

As aforementioned, we do not know the true distribution before testing so MADDPG was trained with the uniformly distributed scenarios. Following the min-max idea of the standard version, M3DDPG maximized the objective in the worst-case scenario in the experiments as:

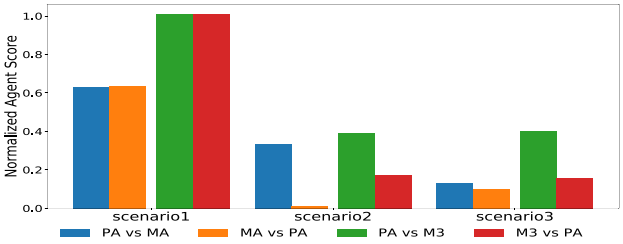
$$J(\theta_i) = \min_{c\in\mathcal{C},a_{j\neq i}} \mathbb{E}_{s\sim p^c,a_i\sim\mu_i}[R_i(s,a)] \tag{12}$$

By doing so, we can evaluate the effectiveness of our algorithm with multiple policies comparing with MADDPG and M3DDPG using only a single policy for each agent when the environment changes.

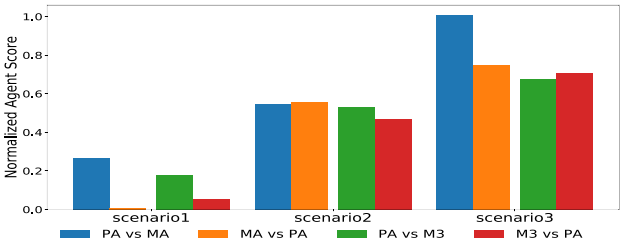
5.3 Results

We measure the performance of agents with policies learned by our PAMADDPG and agents with policies learned by MADDPG and M3DDPG in each environment. In the first two mixed cooperative and competitive domains, we switch the roles of both normal agent and adversary as in the MADDPG and M3DDPG papers to evaluate the quality of learned policies trained by different algorithms.

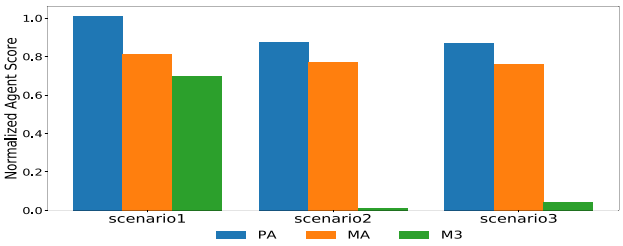
The results on the three environments are demonstrated in Fig. 2. As shown in the figure, each group of bar shows the $0-1$ normalized score for the environment,



(a) Keep-away



(b) Predator-prey



(c) Cooperative navigation

Fig. 3. Performance of PAMADDPG (PA), MADDPG (MA), and M3DDPG (M3) on different scenarios.

where a higher score shows better performance for the algorithm. In the first two environments, PAMADDPG outperforms M3DDPG and MADDPG because PAMADDPG achieves higher scores when playing normal agents (i.e., PA vs MA, PA vs M3) than the ones as adversaries (i.e., MA vs PA, M3 vs PA). Interestingly, PAMADDPG performs better when playing against MADDPG (i.e., PA vs MA, MA vs PA) than the case against M3DDPG (i.e., PA vs M3, M3 vs PA) in the Keep-away environment, while PAMADDPG shows better performance against M3DDPG than the case against MADDPG in the Predator-prey environment. Intuitively, this is because the Predator-prey environment is more competitive than the Keep-away environment so that M3DDPG who considers the worst-case situation works better than MADDPG when paired with our algorithm. In the Cooperative navigation environment, PAMADDPG consistently outperforms

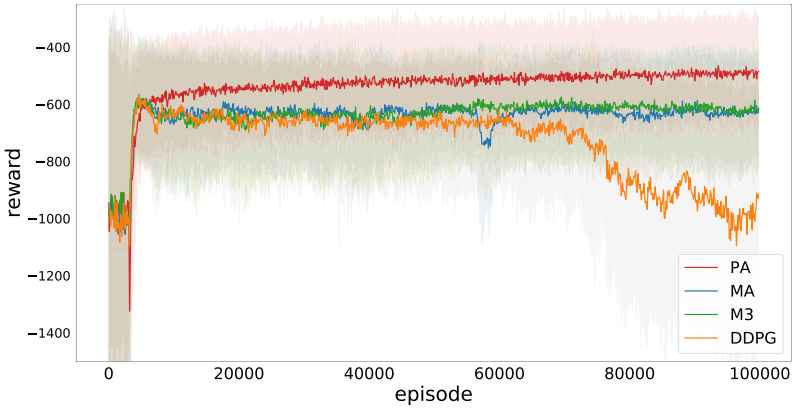


Fig. 4. Learning reward of PAMADDPG (PA), MADDPG (MA), M3DDPG (M3), and DDPG on the Cooperative navigation environment after 10,000 episodes.

MADDPG and M3DDPG. M3DDPG has the worst performance in terms of scores because this environment is a fully cooperative domain while M3DDPG makes unrealistic assumption that all the other agents act adversarially.

Figure 3 shows the results of our PAMADDPG comparing with MADDPG and M3DDPG when testing on different scenarios in each environment. In the Keep-away environment, PAMADDPG outperforms MADDPG and M3DDPG on Scenarios 2 and 3 while performs similarly on Scenario 1. This is because MADDPG and M3DDPG tends to converge to the policies fitting Scenario 1, which is expected to work poorly in Scenarios 2 and 3. In contrast, our PAMADDPG can adapt its policies to fit different scenarios during testing. In the Predator-prey environment, PAMADDPG outperforms MADDPG on Scenarios 1 and 3 but not Scenario 2, and M3DDPG on Scenarios 1 and 2 but not Scenario 3. Similar to the Keep-away environment, this is because MADDPG converges to the policies fitting Scenario 2 while M3DDPG converges to the policies fitting Scenario 3. As we can see from the figure, PAMADDPG achieves slightly less scores than MADDPG and M3DDPG on Scenarios 2 and 3 respectively. This is because the Predator-prey environment is very competitive and the policy predictors in PAMADDPG take time to form correct predictions. In the Cooperative navigation environment, our PAMADDPG outperforms MADDPG and M3DDPG for all the scenarios. Again, M3DDPG has the worst performance because this is a fully cooperative environment.

Figure 4 shows the average reward of different approaches on the Cooperative navigation environment during the training process. As we can see from the figure, our PAMADDPG algorithm converges to better reward than all the other methods. As expected, the reward of DDPG decreases after 80,000 episodes due to non-stationarity in multi-agent RL. As shown in the figure, the reward of MADDPG fluctuates about 60,000 episodes while the reward of PAMADDPG becomes stable after convergence.

6 Conclusion

In this paper, we addressed the non-stationarity problem in multi-agent RL and proposed the PAMADDPG algorithm. We model the non-stationarity in the environment as a finite set of scenarios. At training time, each agent learns multiple policies, one for each scenario, and trains a policy predictor that can be used to predict the best policy during execution. With the multiple policies and policy predictor, each agent is able to adapt its policy and choose the best one for the current scenario. We tested our algorithm on three common benchmark environments and showed that PAMADDPG outperforms MADDPG and M3DDPG in all the tested environment. In the future, we plan to conduct research on learning the scenarios directly from the environment.

References

1. Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., Abbeel, P.: Continuous adaptation via meta-learning in nonstationary and competitive environments. arXiv preprint arXiv:1710.03641 (2017)
2. Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., Mordatch, I.: Emergent complexity via multi-agent competition. arXiv preprint arXiv:1710.03748 (2017)
3. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the 34th International Conference on Machine Learning, pp. 1126–1135 (2017)
4. Foerster, J., Assael, I.A., de Freitas, N., Whiteson, S.: Learning to communicate with deep multi-agent reinforcement learning. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 2137–2145 (2016)
5. Foerster, J., Chen, R.Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., Mordatch, I.: Learning with opponent-learning awareness. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, pp. 122–130 (2018)
6. Foerster, J., et al.: Stabilising experience replay for deep multi-agent reinforcement learning. In: Proceedings of the 34th International Conference on Machine Learning, pp. 1146–1155 (2017)
7. Foerster, J.N., Assael, Y.M., de Freitas, N., Whiteson, S.: Learning to communicate to solve riddles with deep distributed recurrent q-networks. arXiv preprint arXiv:1602.02672 (2016)
8. Foerster, J.N., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S.: Counterfactual multi-agent policy gradients. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence (2018)
9. He, H., Boyd-Graber, J., Kwok, K., Daumé III, H.: Opponent modeling in deep reinforcement learning. In: Proceedings of the International Conference on Machine Learning, pp. 1804–1813 (2016)
10. Hernandez-Leal, P., Kaisers, M., Baarslag, T., de Cote, E.M.: A survey of learning in multiagent environments: dealing with non-stationarity. arXiv preprint arXiv:1707.09183 (2017)
11. Lazaridou, A., Peysakhovich, A., Baroni, M.: Multi-agent cooperation and the emergence of (natural) language. arXiv preprint arXiv:1612.07182 (2016)
12. Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. J. Mach. Learn. Res. **17**(1), 1334–1373 (2016)

13. Li, S., Wu, Y., Cui, X., Dong, H., Fang, F., Russell, S.: Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In: Proceedings of the AAAI Conference on Artificial Intelligence (2019)
14. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
15. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: Proceedings of the 11th International Conference on Machine Learning, pp. 157–163 (1994)
16. Long, P., Fanl, T., Liao, X., Liu, W., Zhang, H., Pan, J.: Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In: Proceedings of the 2018 IEEE International Conference on Robotics and Automation, pp. 6252–6259 (2018)
17. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O.P., Mordatch, I.: Multi-agent actor-critic for mixed cooperative-competitive environments. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 6379–6390 (2017)
18. Ma, J., Wu, F.: Feudal multi-agent deep reinforcement learning for traffic signal control. In: Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Auckland, New Zealand, pp. 816–824, May 2020
19. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
20. Papoudakis, G., Christianos, F., Rahman, A., Albrecht, S.V.: Dealing with non-stationarity in multi-agent deep reinforcement learning. arXiv preprint arXiv:1906.04737 (2019)
21. Raileanu, R., Denton, E., Szlam, A., Fergus, R.: Modeling others using oneself in multi-agent reinforcement learning. arXiv preprint arXiv:1802.09640 (2018)
22. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484 (2016)
23. Silver, D., et al.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815 (2017)
24. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: Proceeding of the 31st International Conference on Machine Learning, pp. 387–395 (2014)
25. Singh, A., Jain, T., Sukhbaatar, S.: Learning when to communicate at scale in multiagent cooperative and competitive tasks. arXiv preprint arXiv:1812.09755 (2018)
26. Sukhbaatar, S., Fergus, R., et al.: Learning multiagent communication with backpropagation. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 2244–2252 (2016)
27. Wu, C., Kreidieh, A., Parvate, K., Vinitsky, E., Bayen, A.M.: Flow: architecture and benchmarking for reinforcement learning in traffic control. arXiv preprint arXiv:1710.05465 (2017)