Value-based CTDE Methods in Symmetric Two-team Markov Game: from Cooperation to Team Competition

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this paper, we identify the best training scenario to train a team of agents to compete against multiple possible strategies of opposing teams. We restrict ourselves to the case of a symmetric two-team Markov game which is a competition between two symmetric teams. We evaluate cooperative value-based methods in a mixed cooperative-competitive environment. We selected three training methods based on the centralised training and decentralised execution (CTDE) paradigm: QMIX, MAVEN and QVMix. To train such teams, we modified the StarCraft Multi-Agent Challenge environment to create competitive scenarios where both teams could learn and compete simultaneously in a partially observable environment. For each method, we considered three learning scenarios differentiated by the variety of team policies encountered during training. Our results suggest that training against multiple evolving strategies achieves the best results when, for scoring their performances, teams are faced with several strategies, whether the stationary strategy is better than all trained teams or not.

1 Introduction

Many applications exist where two teams of multiple agents compete, such as in games (Pommerman [Resnick et al., 2018]) or in robotics (RoboCup [Kitano et al., 1997]). In certain use-cases, agents are not fully aware of the entire game environment, such as in Hide and Seek [Baker et al., 2019] or Quake III Arena in Capture the Flag mode [Jaderberg et al., 2019]. To solve the challenges faced within these games, reinforcement learning (RL) is a solution. RL is a paradigm of machine learning where an agent interacts with its environment by selecting an action based on its observation of the environment state and receives a reward based on environment rules [Sutton and Barto, 1998]. The agent then learns to select actions that provide it with the best sum of rewards. To train such agents without learning environment rules, we distinguish between two families of methods. In value-based methods, an agent learns the best sum of rewards (value) it can achieve by selecting a given action at a given state while, in policy-based methods, the agent directly learns which action (policy) it must take in a given state to maximise the sum of rewards. Single-agent RL (SARL) extends to multi-agent RL (MARL) when multiple agents coexist and learn in the same environment. We distinguish different types of MARL based on the goals of agents. They can share the same goal and cooperate, or compete for an opposite goal to the detriment of other agents. In this paper, we evaluate value-based methods designed for cooperative settings to train agents in a partially observable mixed cooperative-competitive environment where two teams of cooperative agents compete to achieve an opposing goal.

Cooperative scenarios, such as in the StarCraft Multi-Agent Challenge (SMAC) [Samvelyan et al., 2019], can be considered as a decentralised partially observable Markov decision process (Dec-

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

POMDP) [Oliehoek et al., 2016] which is a framework where agents share the same rewards to learn to cooperate. The partial observability of agents is a problem in such an environment because it prevents the control from being centralised, meaning that it is not possible to control the team as a single agent and train it with SARL methods. Therefore, decentralisation is required and each agent must select actions on its own. A first solution is to train agents independently with SARL methods. Another possibility is the centralised training with decentralised execution (CTDE) paradigm, where information about all agents is exploited during training but not during execution. Recent CTDE methods have shown better results than independent methods in Dec-POMDP. For policy-based methods, we denote COMA [Foerster et al., 2018], MADDPG [Lowe et al., 2017] and LIIR [Du et al., 2019], among others. For value-based methods, we denote QMIX [Rashid et al., 2018], MAVEN [Mahajan et al., 2019] and QVMix [Leroy et al., 2020], QTRAN [Son et al., 2019], weighted QMIX [Rashid et al., 2020], QPLEX [Wang et al., 2021], among others.

Reinforcement learning has not been the first choice to solve complex competitive two-player games. For example, we think of Deep Blue [Campbell et al., 2002] or Alpha Go [Silver et al., 2016]. However, with the rise of RL, it has become possible to play such games using RL algorithms [Silver et al., 2018]. Complex games, with partial observability, or sometimes with more than two competing agents, have also been solved using RL methods such as Quake III Arena in Capture the Flag mode [Jaderberg et al., 2019] or StarCraft [Vinyals et al., 2019]. To train such agents to compete, a paradigm called self-play is used. Agents play against themselves, allowing them to face evolving strategies and to improve their skills. In more complex games, a population of learning agents is created by duplicating some of them during training to enforce the diversity of opponents. This allows agents to improve their skills while remaining competent against previously encountered strategies. In most of the mixed cooperative-competition papers mentioned before, agents are trained with independent SARL method in self-play or within a population, except in Baker et al. [2019] who chose a CTDE method close to COMA [Foerster et al., 2018] by exploiting the state of the environment during training.

In this paper, we mix cooperative and competitive methods to train teams to compete in a symmetric two-team Markov game where two teams of the same agent composition compete. Specifically, we study the difference between three learning scenarios to train CTDE value-based methods: learning against a stationary team policy, against a single evolving team strategy (self-play), and against multiple evolving team strategies within a population of learning teams. To perform these experiments and to study the performance of each learning scenario, we present a new, competitive environment by modifying SMAC which has been designed for cooperation. We chose symmetrical competition to ensure fair and balanced competition in addition to the possibility of controlling either team with the same agents. In two different SMAC environments, we trained teams with three value-based CTDE methods, QMIX, MAVEN and QVMix, each with the three learning scenarios and we then analysed how they perform when faced with multiple opposing strategies. We selected value-based methods because they provide the highest win rates in SMAC [Samvelyan et al., 2019]. Our results suggest that when competing with several possible strategies, teams trained in a population achieve the best performance, but a selection process is required to select the best team. We reached this conclusion irrespective of whether or not the stationary strategy was better than all trained teams.

In the following, we first formally define a symmetric two-team Markov game and describe CTDE value-based methods in cooperative settings in Section 2. In Section 3, we detail the different learning scenarios, their performance criteria to evaluate them and how we trained teams in the new competitive SMAC environment. Results are presented in Section 4, followed by the conclusion in Section 5.

2 Background

Our framework is a special case of a Markov game [Littman, 1994]. Specifically, we are in a symmetric, mixed cooperative-competitive, partially observable two-team Markov game described by a tuple $[\mathcal{S}, O, \mathcal{Z}, \mathcal{U}, n, R, P, \gamma]$, where two teams of n agents compete by choosing an action at every timestep t. Note that there can be different types of agent, but both team compositions must be the same, which implies that the same number of agents of the same type are present in both teams. An agent of the jth team is denoted by $a_{i,j} (i \in \{1, ..., n\}, j \in \{1, 2\})$ when it is not shortened to a, and its action space is defined by $\mathcal{U}_{a_{i,j}}$. Due to team symmetry, agents $a_{i,1}$ and $a_{i,2}$ are the same type and

their action space is the same ($\forall i$). We therefore denote $\mathcal{U} = \mathcal{U}_{a_{1,1}} \times .. \times \mathcal{U}_{a_{n,1}} = \mathcal{U}_{a_{1,2}} \times .. \times \mathcal{U}_{a_{n,2}}$, the combined set of n action spaces.

The state of the environment at timestep t is $s_t \in \mathcal{S}$ where $\mathcal{S}$ is the set of states, and $o_t^a \in \mathcal{Z}$ is the observation of this state perceived by the agent a, and $\mathcal{Z}$ is the set of observations. The observation function $O : \mathcal{S} \times \{1, ..., 2n\} \rightarrow \mathcal{Z}$ determines o_t^a. At each timestep t, each agent simultaneously executes an action $u_t^a \in \mathcal{U}_a$ such that the state s_t transits to a new state s_{t+1} with a probability defined by the transition function $P(s_{t+1}|s_t, \boldsymbol{u_t}) : \mathcal{S}^2 \times \mathcal{U}^2 \rightarrow \mathbb{R}^+$ where $\boldsymbol{u_t} = \bigcup_{j \in \{1,2\}} \boldsymbol{u_t^j} = \bigcup_{i \in \{1,..,n\}, j \in \{1,2\}} u_t^{a_{i,j}}$. After all agents' actions are performed, common team rewards $\{r_t^1, r_t^2\} = R(s_{t+1}, s_t, \boldsymbol{u_t}) : \mathcal{S}^2 \times \mathcal{U}^2 \rightarrow \mathbb{R}^2$ are assigned to the agents. An agent a chooses its action based on its current observation $o_t^a \in \mathcal{Z}$ and its history $\tau_t^a \in (\mathcal{Z} \times \mathcal{U}_a)^{t-1}$. Its policy is the function $\pi^a(u_t^a|\tau_t^a, o_t^a) : (\mathcal{Z} \times \mathcal{U}_a)^t \rightarrow \mathbb{R}^+$ that maps, from its history and the current observation at timestep t, the probability of taking action u_t^a at timestep t. We denote the team policy: $\boldsymbol{\pi_j} = \bigcup_{i \in \{1,..,n\}} \pi^{a_{i,j}}$.

During an episode, the cumulative discounted reward obtained from timestep t over the next T timesteps by team j is defined by $R_t^j = \sum_{k=0}^{T-1} \gamma^k r_{t+k}^j$ where $\gamma \in [0, 1)$ is the discount factor. The goal of each agent is to maximise its expected R_0^j over a number of timesteps T defined by the episode time limit. The optimal team policy maximises the expected cumulative discounted reward: $\boldsymbol{\pi_j^*} = \text{argmax}_{\boldsymbol{\pi_j}} \mathbb{E}[R_0^j|\boldsymbol{\pi}]$ that depends on both team policies $\boldsymbol{\pi} = (\boldsymbol{\pi_1}, \boldsymbol{\pi_2})$. To evaluate the team policy, we consider the state-value function $V^{\boldsymbol{\pi_j}}(s) = \mathbb{E}[R_t^j|s_t = s, \boldsymbol{\pi}]$, also called the V function. Because rewards are shared among a team, the V function is the same for its agents: $V^{\boldsymbol{\pi_j}}(s) = V^{\pi^{a_{1,j}}}(s) = V^{\pi^{a_{n,j}}}(s)$. We also define the state-joint-action value function $Q^{\boldsymbol{\pi_j}}(s, \boldsymbol{u}) = \mathbb{E}[R_t^j|s_t = s, \boldsymbol{u_t^j} = \boldsymbol{u}, \boldsymbol{\pi}]$, also called the Q function, which characterises the expected return of the team when performing actions $\boldsymbol{u}$ at timestep t and following policy $\boldsymbol{\pi_j}$ afterwards. The corresponding individual state-action value function of an agent policy is defined by $Q^{\pi^a}(s, u) = \mathbb{E}[R_t|s_t = s, u_t^a = u, \boldsymbol{\pi}] = Q^{\boldsymbol{\pi}}(s, \boldsymbol{u})$, and denoted later as Q_a for the sake of conciseness.

It is possible to obtain the Dec-POMDP [Oliehoek et al., 2016] definition from that of the two-team Markov game by constraining one team to have a stationary policy. The other team would then be the only team learning and the two-team Markov game can be considered as a Dec-POMDP. Its formal definition is easily derived from that of the two-team Markov game by eliminating team considerations and ignoring the j. In our experiments, teams are trained with value-based methods designed for Dec-POMDP. In this type of environment, it is not possible to learn only the state-joint-action value function $Q(s_t, \boldsymbol{u_t})$ because agents select their action based only on (o_t, τ_t) and do not perceive s_t. They therefore learn their own Q_a. A solution is known as independent Q learning (IQL) [Tan, 1993] and consists of training agents to learn Q_a independently, ignoring the presence of other learning agents. Another type of solution is to exploit more information during training but only the available (o_t, τ_t) during execution. It is a paradigm called centralised training with decentralised execution (CTDE). The algorithms tested in this paper, QMIX [Rashid et al., 2018], MAVEN [Mahajan et al., 2019] and QVMix [Leroy et al., 2020] exploit this paradigm. All three methods learn $Q(s_t, \boldsymbol{u_t})$ as a factorisation of s_t and Q_a $\forall a$ during training. The Q_a are no longer Q functions but are utility functions used to select actions which are retained for the execution. There exists a condition to factorise $Q(s_t, \boldsymbol{u_t})$ and it is called the Individual-Global-Max condition (IGM): $\text{argmax}_{\mathbf{u_t}} Q(s_t, \mathbf{u_t}) = \bigcup_a \text{argmax}_{u_t^a} Q_a$ [Son et al., 2019]. In QMIX, Q_a factorise $Q(s_t, \boldsymbol{u_t})$ with a hypernetwork. The weights of each Q_a in the factorisation are computed based on s_t and are constrained to be positive to ensure IGM. MAVEN follows the same principle of QMIX with an additional hierarchical policy network that modify each Q_a in order to influence agents' behaviour and to improve their exploration capacity. QVMix is an extension of the Deep Quality-Value (DQV) family of algorithms [Sabatelli et al., 2018, 2020] applied to QMIX where agents learn both the Q and the V functions. Those value-based methods initially developed for solving Dec-POMDP are thoroughly detailed in Appendix A. Concurrently to this work, other methods have been developed which achieve better results in Dec-POMDP. We denote QTRAN [Son et al., 2019], weighted QMIX [Rashid et al., 2020], QPLEX [Wang et al., 2021], among others.

3 Learning scenarios, performance criteria and training parameters

Our goal is to train a team to be resilient to different adversarial strategies. We test three different learning scenarios differentiated by the diversity of opponents' strategies encountered during training. We hereafter define the learning scenarios and performance criteria, followed by the definition of the environment and the details of the training.

In the first learning scenario, teams are trained against a stationary strategy, which we refer to as a heuristic. As introduced in Section 2, if one team has a stationary policy, the two-team Markov game is a Dec-POMDP, the framework for which CTDE methods are designed. The second scenario is called self-play where the team is trained by playing against itself facing a strategy that continuously improves at the same learning speed. The team thus observes episodes from both sides. This scenario has proven its value in competitive MARL (e.g. [Baker et al., 2019]). In the third scenario, several teams are trained at the same time within a population of teams trained with the same method. Teams either play against themselves or against other learning teams and therefore, self-play is the special case of a training population composed of a single team. This learning scenario has proven itself when the number of winning strategies is large, such as in StarCraft [Vinyals et al., 2019]. The latter trained a growing population of agents but in this paper, the size of the training population is fixed to five to reduce computational complexity. Per 100 environment timesteps generated by our game simulator, we note that in the case of learning against a heuristic, 100 samples (s_t, u_t, r_t, s_{t+1}) can be exploited for learning per team, 200 if the learning scenario is self-play and 40 if, as a learning scenario, we have a population of five teams that performed the same number of actions.

In our experiments, teams were trained with these three learning scenarios and with the three methods presented in Section 2, QMIX, MAVEN and QVMix, in two different environments. We hereafter describe the same experimental process that we conducted for both environments. For each method, each learning scenario was executed ten times and stopped after each team had exploited 10 million samples. For each method, this resulted in ten teams trained against the heuristic (**H**) requiring 10 million environment timesteps, ten teams trained in self-play (**S**) requiring 5 million environment timesteps and ten training populations of five teams leading to 50 teams trained within populations (**P**) requiring 50 million environment timesteps. We therefore, in total, trained 210 teams of nine different types to analyse the learning scenarios in both environments.

After the training, teams were evaluated with the Elo rating system [Elo, 1978] which ranks individuals in a group using Elo scores computed for each of them. Between two individuals, the one with the highest Elo score has the highest probability of winning a game against the other. From these two Elo scores, one can compute the winning probability. Player's scores are updated based on the outcome of the game and their current Elo scores. The Elo rating system is formally described in Appendix B.

For each environment, we assemble several groups of teams, which we refer to as test populations, to evaluate our trained teams based on the Elo scores achieved in these test populations. To evaluate the learning scenarios, we group teams trained with the same method in the same test population. That leads to three test populations of 70 teams per environment. To find the best learning scenario/training method pair, we group all trained teams in a single test population. This test population has therefore 210 teams. To also take into account the heuristic in the assessment of the trained teams, we created four additional test populations from the four previously defined by adding the heuristic to them. In practice, to compute the Elo scores in these 16 test populations, each team plays 20 games against all the other teams in a randomised order. Every team starts with an Elo score of 1000 and we set the maximum Elo score update to 10, which is sufficiently small for our population sizes. In Section 4, we analyse the distribution of Elo scores after every team had finished its testing games.

During training, team neural network parameters are recorded every 200.000 timesteps until the 10 millionth timestep. The test populations are composed of the agents whose network parameters correspond to those recorded at the 10 millionth timestep. The other saved networks allow one to evaluate teams' performance during training and their timestep efficiency. To perform this evaluation, these intermediate trained teams play against the heuristic and against teams trained with different learning scenarios but the same training method. We analyse the win rates along training timesteps of these different matchups when teams play 24 games against each other. For example, and for the same method, each ten teams trained in self-play played 24 games against all the 50 teams trained within a population for each saved parameter.

To perform our experiments, we created a new environment by modifying the StarCraft Multi-Agent Challenge (SMAC) [Samvelyan et al., 2019] to create a competitive environment. SMAC has been developed to train a team to cooperate in a competitive environment but only against the built-in StarCraft AI which has a stationary strategy like a heuristic. We adapted[1] both SMAC and the associated learning framework PyMARL [Samvelyan et al., 2019]. It is now possible to control both opposing teams in SMAC, as well as train them simultaneously. In competitive SMAC, the goal remains unchanged, and it is to defeat the opposing team, by inflicting sufficient damage to reduce the hit points of all opponents to zero. The winner team is the one that ends up with the highest sum of remaining hit points at the end. If both teams end up with the same total of hit points, it is a draw.

For our experiments, we have chosen the $3m$ and $3s5z$ maps. A map is the name given to the different scenarios in SMAC, and from an RL perspective, a different map is a different environment. In the $3m$ map, two teams of three marines compete for a maximum 100 timesteps. In the $3s5z$ map, two teams of three stalkers and five zealots compete for a maximum 150 timesteps. More details about these two maps and competitive SMAC can be found in Appendix C. While both maps appear to be simple, the task of learning everything from scratch is not. When learning against the heuristic, teams do not learn how to find their opponents, because they approach them. When they learn in self-play or within a population, they first need to understand where to find opponents. In this paper, the heuristic implemented is based on two possible choices for an agent. Either it does not see any opponent, and it moves toward the starting point of the opponent's team until it reaches the opposite side of the map and stops. Or, when there are enemies within its shooting range, it attacks the nearest one. This heuristic is slightly different from StarCraft's built-in AI originally used to train teams to cooperate in SMAC. The built-in AI selects targets based on a priority score. It will choose to attack the closest unit with the highest priority and this will remain the target until its priority drops or it can no longer be attacked. A unit's priority score is based on its type and current action. For example, if two of the same units attack and the targeted unit stops attacking, its priority score will drop and the built-in AI will select the other unit to attack. This is the main difference with our heuristic which will attack the nearest unit regardless of its action and its priority. Our heuristic is arguably more complex to beat than the one provided in SMAC.

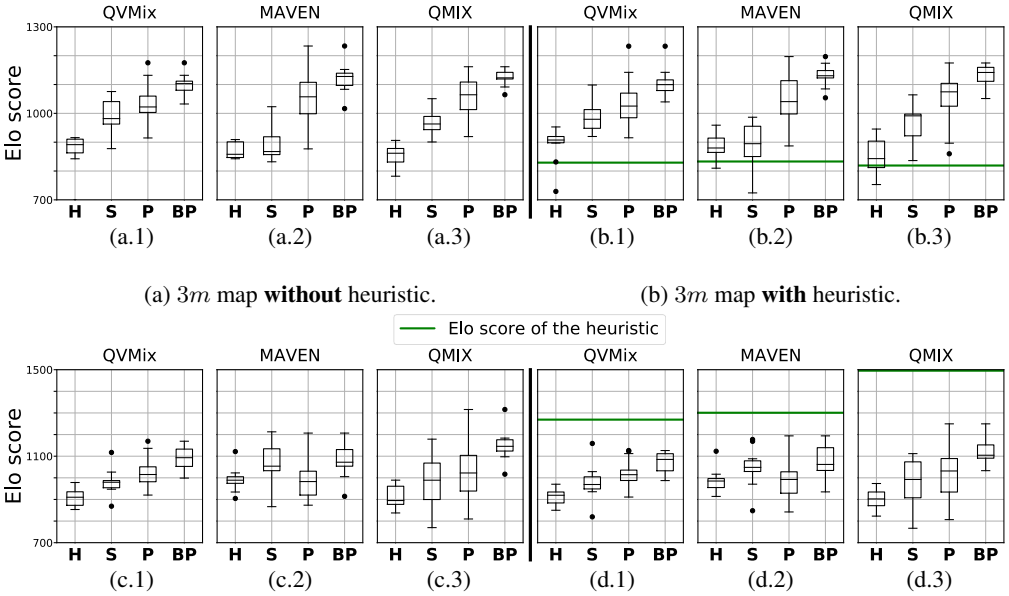
To train teams, we enforced their starting positions to be balanced when training against the heuristic. Concerning the training within a population, each team had an equal chance of playing against any team in the population from any starting position, including itself. For all learning scenarios and methods, network architectures and parameters remained the same to ensure fair comparison. Learning parameters were determined by default configurations provided by their different authors. More details about the training parameters can be found in Appendix D. The main differences with the original implementations are that epsilon anneal time is set to 2 million instead of 0.5 million and that the networks are updated every eight episodes in $3m$ and every episode in $3s5z$. Also note that, as in the CTDE method literature, individual networks share the same parameters between agents of the same team to improve learning speed. When evaluating the Elo scores or when computing the win rates, agents greedily select actions that maximise their individual Q_a function. As described earlier, training a population of teams requires much more time than training a team in self-play or against the heuristic, and we present training times in Appendix E.

4 Performance of learning scenarios

We present the Elo scores of teams with box plots in Figure 1 when the test population contains teams trained with the same method. We first focus on performance when the heuristic is not in the test population (Fig. 1a, 1c). The first observation is that the best teams are the ones trained within a population, except with MAVEN in $3s5z$ (Fig. 1c.2) for which the differences between learning scenarios are smaller. We discuss these differences later. The second observation is that the population scenario has the highest variance. To understand this, we also plot the box plots corresponding to the ten teams that achieved the highest Elo score of each training population, denoted by **BP**. These box plots confirm that there is a difference between teams of the same training population and a selection must be performed to find the best one, so as to optimise the performances of this learning scenario. Training against the heuristic is the worst scenario, arguably because agents overfit the heuristic's strategy. However, the heuristic is not in these test populations and its impact is the concern of additional analysis, detailed later. The performance of teams trained in self-play lies in between the

[1] The code in supplementary material will be published.

two other learning scenarios. While some teams achieve Elo scores close to the best teams of each training population, the scores of others are lower than the lowest scores of teams trained within population.



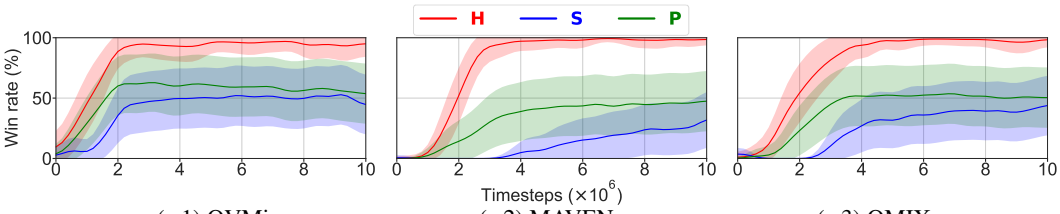
(a) $3m$ map **without** heuristic. (b) $3m$ map **with** heuristic.

(c) $3s5z$ map **without** heuristic. (d) $3s5z$ map **with** heuristic.

Figure 1: Elo score box plots of 12 test populations. Half of the experiments were performed in the $3m$ map, shown at the top (1a, 1b) and the other half in the $3s5z$ map, shown at the bottom (1c, 1c). In each test population, teams are trained with the same method, which is either QVMix (1a.1, 1b.1,1c.1, 1d.1), MAVEN (1a.2, 1b.2,1c.2, 1d.2) or QMIX (1a.3, 1b.3,1c.3, 1d.3). In 1b and 1d, the heuristic is present in the test population and a green line represents its Elo score. Box plots represent the distribution of the ELO scores of the teams trained either against the heuristic (**H**), in self-play (**S**), within a population (**P**) or the best of each training population (**BP**). For most methods, teams trained within a population achieved the highest Elo scores. Box plots present the median, the first quantile ($Q1$) and the third quantile ($Q3$). The reach of whiskers is defined by $1.7 * (Q3 - Q1)$.

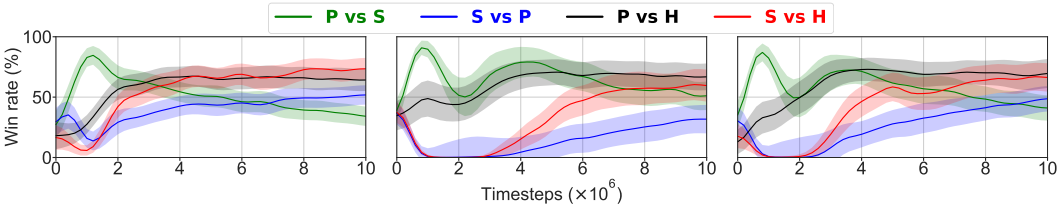
The same experiment is performed with the addition of the heuristic in the three test populations and corresponding box plots are presented in Figures 1b and 1d. The heuristic scores are different depending on the map. In the $3m$ map, most of the teams achieve a higher Elo score than the heuristic, whereas in $3s5z$, the heuristic dominates all teams. In the $3s5z$ map and for all three learning scenarios, Elo scores slightly decreased with the addition of the heuristic. The conclusion is straightforward and the heuristic is better than all teams. However, one should note that the score ordering between the teams remained the same between Figure 1d and 1c. In $3m$, one can see that the Elo score of teams trained against the heuristic (**H**) is higher in Fig. 1b than the ones in Figure 1a, as a direct consequence of the introduction into the test populations of a team against which they win. When compared with the previous test populations (Fig. 1a.1, 1a.2 and 1a.3), teams trained with QVMix achieved higher Elo scores than without the heuristic in the test population (Fig. 1b.1), while when trained with MAVEN and QMIX, they achieved lower Elo scores (Fig. 1b.2, 1b.3). In all cases, the higher values of box plots are not significantly different but lower values are, meaning that some teams performed poorly against the heuristic. Finally, teams trained within a population remain the most successful in most of our experiments, no matter if the heuristic is the best or almost the worst team.

In Figure 2a, we present the evolution of win rates against the heuristic along training timesteps by each team in the $3m$ map. For all methods, teams trained against the heuristic are the best against it on average. This explains why their Elo score improves when the heuristic is included in the test population. This is also the case for QVMix teams trained in self-play and within a population that performed better and learned faster than teams trained with MAVEN and QMIX with these two learning scenarios. However, in the $3s5z$ map, it can be seen in Figure 2c that the win rates against



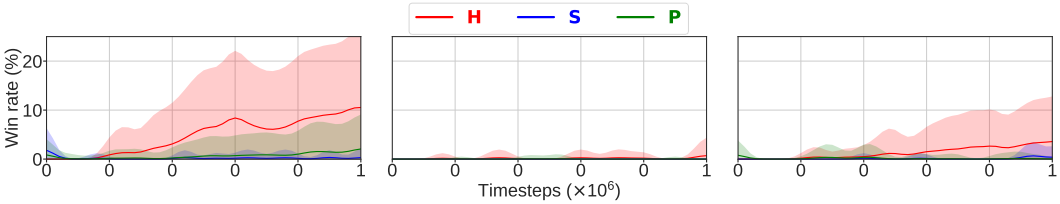
(a.1) QVMix (a.2) MAVEN (a.3) QMIX

(a) Win rates achieved against the heuristic by teams trained in the $3m$ map.



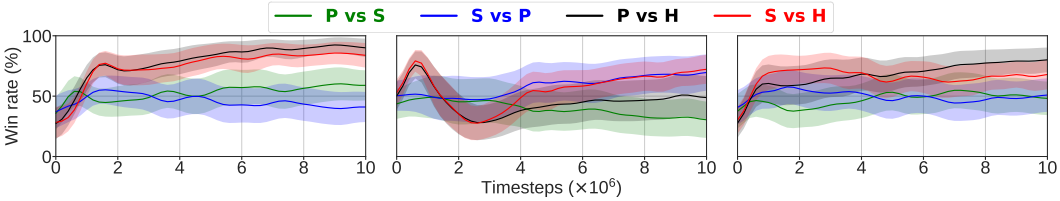
(b.1) QVMix (b.2) MAVEN (b.3) QMIX

(b) Win rates achieved by teams trained in the $3m$ map against themselves.



(c.1) QVMix (c.2) MAVEN (c.3) QMIX

(c) Win rates achieved against the heuristic by teams trained in the $3s5z$ map. Note the change in scale of the y-axis.



(d.1) QVMix (d.2) MAVEN (d.3) QMIX

(d) Win rates achieved by teams trained in the $3s5z$ map against themselves.

Figure 2: Means of win rate achieved along training timesteps by confronting teams trained with the same method against the heuristic (2a,2c) or against other teams trained with a different learning scenario (2b, 2d). Teams are trained either with QVMix (2a.1, 2b.1,2c.1, 2d.1), MAVEN (2a.2, 2b.2,2c.2, 2d.2) or QMIX (2a.3, 2b.3,2c.3, 2d.3). Tests were performed in the $3m$ map, shown at the top, and in the $3s5z$ maps, shown at the bottom. In 2a and 2c, win rates against the heuristic are presented in red, blue and green for teams trained against the heuristic, in self-play and within a population, respectively. In 2b and 2d, win rates of teams trained within a population against teams trained in self-play are presented in green and against teams trained against the heuristic in black. Win rates of teams trained in self-play against teams trained within a population are presented in blue and against teams trained against the heuristic in red. Training against the heuristic achieves the best win rates against the heuristic with the lowest variance, but is the worst scenario when competing against other trained teams. The error band is half the standard deviation.

the heuristic are very low, to not say equal to zero (note the change of scale in these Figures). Only the win rates of teams trained against the heuristic, especially with QVMix and QMIX, increase at the end of the training but with a high variance in comparison to the $3m$ map (Fig. 2a). For QVMix, the win rates of teams trained with a population also increase at the end of the training phase. A first conclusion is that the time we have budgeted for training in the $3s5z$ map is insufficient to achieve a high win rate. However, we find this beneficial because it shows that, even when the heuristic is better than all teams, training against it, with the same training timesteps allowance, is not the best learning scenario when teams have to be good against several strategies.

In the $3m$ map, the standard deviation of green and blue win rates, representing population and self-play learning scenarios respectively, is high and confirms the results of Elo score box plots that there are performance gaps between teams trained within the same learning scenario. Observations also suggest that teams trained in self-play would achieve the same win rates as teams trained within a population if they were trained for longer, suggesting a difference of training sample efficiency. As teams first need to learn to cross the map to meet opponents and to learn to fight them, this difference is because training within a population, against several strategies, increases the probability of creating episodes where two opposing agents meet and fight at the beginning of training.

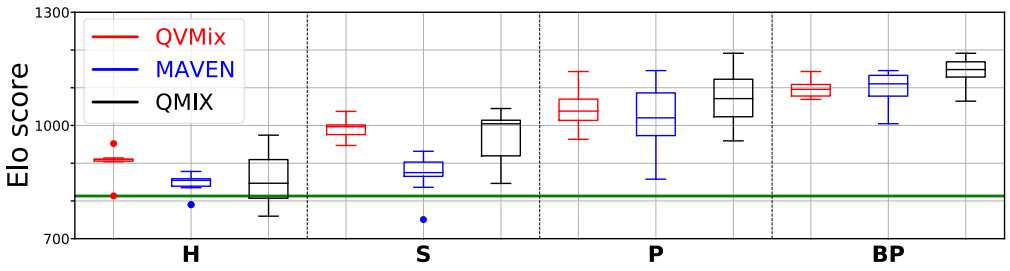
Win rates from the confrontations between trained teams are presented in Figures 2b and 2d. This confirms the previous results with box plots that training against the heuristic is the worst scenario, as an average win rate above 60% is achieved against them as shown in the black and red curves. Green and blue curves enable one to analyse self-play against population-learning scenarios. The draw rates can be obtained by subtracting from 1 the sum of these two curves. At the beginning of the training phase, teams trained within a population are better than self-play ones in the $3m$ map. This high win rate decreases with training in favour of the win rate of teams trained in self-play, and for QVMix and QMIX, until it becomes higher than the latter, also suggesting a difference of training sample efficiency. Moreover, although the training sample efficiency is lower for teams trained in self-play, the number of environment timesteps required to train them is five-times lower in our setting. Note that we insist on the fact that it is on average that the self-play teams become better. In the $3s5z$ map, this overlap phenomenon does not occur and the average win rates fluctuate around 50% with the green curves staying closely above the blue ones at the end. The proximity of performance between teams trained in self-play and within a population is arguably due to the environment and the $3m$ map which does not offer the possibility of winning with very different strategies. As the $3s5z$ map appears to be more complex, with the lack of performance against the heuristic as evidence, teams trained within a population remain better on average.

On average, the results of the teams trained within a population with MAVEN in the 3s5z map differ from the other experiments. Some results are slightly worse in regard to those of the other experiments. The reason for this is not clear, as we executed the experiments several times. However, this does not affect the conclusions of our experiments that remain clear. Finally, it would be necessary to repeat these experiments more times or to analyse the behaviour of the agents in depth to find the problem, which is beyond the scope of this paper.

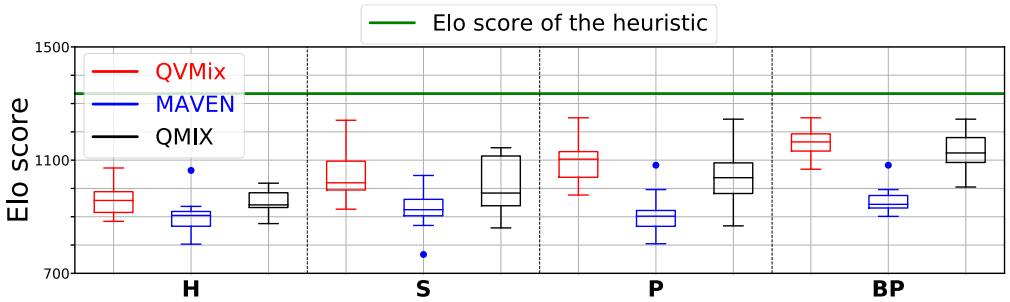
In Figure 3, we present the box plots of Elo scores obtained by all trained teams and the heuristic in a single test population for both maps. We observe that the learning scenario ranking remains the same as in other experiments. In the $3m$ map, QMIX achieves the highest Elo scores while the low MAVEN Elo scores are worse than the ones of QMIX and QVMix when teams are trained in self-play or within a population. QVMix produces results with a lower variance than QMIX. In the $3s5z$ map, QVMix is the one achieving the highest Elo scores and it is also MAVEN that achieves the lowest ones. We conducted the same experiment without the heuristic, which led to the same conclusion (see Figure 8 in Appendix F). Despite its mechanism of exploration, MAVEN is not able to outperform QMIX and QVMix and it is also the case in [Mahajan et al., 2019] and [Leroy et al., 2020] where they show that MAVEN outperforms QMIX but not QVMix in more complex Dec-POMDP environments.

5 Conclusion and future works

In this paper, we evaluated learning scenarios to train teams to face multiple strategies in a symmetric two-team Markov game. Teams are trained with three CTDE value-based methods: QMIX, MAVEN and QVMix and with three learning scenarios differentiated by the variety of strategies that teams will encounter during their training. Specifically, they are trained by playing against a stationary strategy,



(a) $3m$



(b) $3s5z$

Figure 3: Elo score box plots of two test populations, in $3m$ at the top and in $3s5z$ at the bottom, composed of the heuristic and teams trained using three methods and three learning scenarios. The training method is either QVMix (red), MAVEN (blue) or QMIX (black). Box plots represent the distribution of the ELO scores of teams trained either against the heuristic (**H**), in self-play (**S**), within a population (**P**) or the best of each population (**BP**). Box plots present the median, the first quantile ($Q1$) and the third quantile ($Q3$). The reach of whiskers is defined by $1.7 * (Q3 - Q1)$.

against themselves, or within a population of teams trained with the same method. To perform our experiments, we modified the cooperative environment SMAC to allow teams to compete and train at the same time, and trained teams in two different SMAC environments. These nine types of trained teams are evaluated at the end of their training with the Elo rating system. Different groups are formed to identify which learning scenario is the best and which learning scenario/training method pair is the best. We also analysed the win rates of several matchups during training to support the results provided by the Elo scores. Our results showed that training teams within a population of learning teams is the best learning scenario, when each team plays the same number of timesteps for training purposes. We reached this conclusion irrespective of whether or not the stationary strategy was better than all trained teams. Finally, a selection procedure is required because teams from the same training population do not perform equally.

This work is a first investigation of two-team competition with CTDE methods, and we hereafter suggest five future research directions. First, we suggest performing the same experiments on more complex environments, for example by adding more units in the map or by changing the layout of the map. Another research direction would be to study how diversity in the training population impacts the performance. In our experiments our training populations are composed of agents trained with the same method, but its diversity could be increased by, for examples, adding the heuristic in the population, confronting agents against older learned policies, varying the size of the population or training teams with different methods in the same population. We also propose to perform behavioural and policy analyses so as to better understand why some teams achieve a better Elo score and, in particular, why we observe a difference in performances inside a single training population. Moreover, the asymmetric two-team Markov game is an obvious fourth direction, with the challenge of training asymmetric teams. Finally, in this paper, we focus on value-based methods and do not challenge them against policy-based methods, which could lead to interesting results.

References

Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jakob Foerster, Julian Togelius, Kyunghyun Cho, and Joan Bruna. Pommerman: A multi-agent playground. In *CEUR Workshop Proceedings*, volume 2282. CEUR-WS, 2018.

Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. Robocup: A challenge problem for ai. *AI magazine*, 18(1):73–73, 1997.

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In *International Conference on Learning Representations*, 2019.

Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

Frans A Oliehoek, Christopher Amato, et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *NIPS*, 2017.

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

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. *Proceedings of the 35th International Conference on Machine Learning, Stockholm*, 2018.

Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. MAVEN: Multi-agent variational exploration. In *Advances in Neural Information Processing Systems*, 2019.

Pascal Leroy, Damien Ernst, Pierre Geurts, Gilles Louppe, Jonathan Pisane, and Matthia Sabatelli. QVMix and QVMix-Max: extending the deep quality-value family of algorithms to cooperative multi-agent reinforcement learning. *AAAI-21 Workshop on Reinforcement Learning in Games*, 2020.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *Proceedings of the 36th International Conference on Machine Learning, Long Beach, California*, 2019.

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

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. {QPLEX}: Duplex dueling multi-agent q-learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Rcmk0xxIQV.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134 (1-2):57–83, 2002.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419): 1140–1144, 2018.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

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

Matthia Sabatelli, Gilles Louppe, Pierre Geurts, and Marco A Wiering. Deep quality-value (DQV) learning. *Advances in Neural Information Processing Systems, Deep Reinforcement Learning Workshop. Montreal*, 2018.

Matthia Sabatelli, Gilles Louppe, Pierre Geurts, and Marco Wiering. The deep quality-value family of deep reinforcement learning algorithms. In *International Joint Conference on Neural Networks (IJCNN)*, 2020.

Arpad E. Elo. *The rating of chessplayers, past and present*. Ishi press international, 1978.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2 2015.

Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. *AAAI Fall Symposium Series*, 2015.

David Ha, Andrew Dai, and Quoc V. Le. HyperNetworks. *5th International Conference on Learning Representations, Toulon*, 2016.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS Workshop on Deep Learning*, 2014.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12:1057–1063, 1999.