Deep Multi-Agent Reinforcement Learning CS234 Final Report

Liu Yang [1] Pei-Chen Wu [1] Guanzhi Wang [1]

Abstract

In this report, we tackle the StarCraft Multi-Agent Challenge (SMAC) (Samvelyan et al., 2019). The challenge is a reflection of many real-world settings in which a team of agents must coordinate their behaviour while acting in a decentralized way. We follow the popular centralized-training/decentralized-execution framework and enable the state-of-the-art algorithms QMIX (Rashid et al., 2018), SMIX(λ) (Yao et al., 2019) and MAVEN (Mahajan et al., 2020) as our baselines. SMIX(λ) and MAVEN are both extensions of QMIX, where SMIX(λ) aims to expand the hypothesis space and MAVEN aims to enhance the exploration capability. We propose a novel method named as Communication-Attended Observation (CAO) which tries to enhance the agents' exploration in feature space. By allowing the agent to access the observations of its visible allies, we introduce a Transformer-style self-attention mechanism (Vaswani et al., 2017) to augment the agent's awareness about the environment. We integrate CAO with three baselines and observe considerable performance boost for all of them.

1. Introduction

In the last few years, deep multi-agent reinforcement learning (MARL) has become a highly active area of research. Many real-world problems that might be tackled by RL are inherently multi-agent in nature, for example, the coordination of self-driving cars, autonomous drones, and multi-agent robot systems are becoming increasingly critical. Therefore, it is essential to develop MARL solutions that can handle decentralisation constraints and deal with high dimensional inputs, the exponentially growing joint action space of many agents. A particular challenging of problems in this area is partially observable, cooperative, multi-agent learning, in which teams of agents must learn to coordinate their behaviour while conditioning only on their private observations. In order to address this problem, the StarCraft Multi-Agent Challenge (SMAC) has been proposed (Samvelyan et al., 2019).

SMAC is based on the popular real-time strategy game StarCraft II and focuses on decentralised micromanagement challenges where each unit is controlled by an independent agent that must act based on local observations. Groups of these agents must be trained to solve challenging combat scenarios, battling an opposing army under the centralised control of the game's built-in scripted AI. Several MARL algorithms have been proposed, such as independent Q-learning (IQL) (Tan, 1993), counterfactual multi-agent policy gradients (COMA) (Foerster et al., 2017), value decomposition networks (VDN) (Sunehag et al., 2017), and QMIX (Rashid et al., 2018), Multi-Agent Variational Exploration (MAVEN) (Mahajan et al., 2020) and SMIX(λ) (Yao et al., 2019).

IQL decomposes a multi-agent problem into a collection of simultaneous single-agent problems. COMA uses a centralised critic credit to train decentralised actors and estimates a counterfactual advantage function for each agent to address multi-agent credit assignment. In between above two extremes, VDN allows centralized value-function learning with decentralised execution.

As a result, QMIX significantly outperforms existing value-based MARL methods, IQL and VDN. Similar to VDN, QMIX lies between the extremes of IQL and COMA, but can represent a much richer class of action-value functions. QMIX employs a network that estimates joint action-value as a complex non-linear combination of per-agent values that condition only on local observations. It can train decentralised policies in a centralised end-to-end fashion. In contrast to VDN, QMIX only needs to ensure that a global argmax performed on Q_{tot} yields the same result as a set of individual argmax operations performed on each Q_a. To this end, it suffices to enforce a monotonicity constraint 1, on the relationship between Q_{tot} and each Q_a.

In our final project, we choose QMIX (Rashid et al., 2018)

[1]Stanford University. Correspondence to: Liu Yang <liuyeung@stanford.edu>, Pei-Chen Wu <pcwu1023@stanford.edu>, Guanzhi Wang <guanzhi@stanford.edu>.

as our baseline for handling SMAC challenges. We have also conducted more novel approaches called SMIX(λ) (Yao et al., 2019) and MAVEN (Mahajan et al., 2020). SMIX(λ) improves the centralized value function estimation with an off policy-based learning method which removes the need of explicitly relying on the centralized greedy behavior assumption during training, and incorporates the λ-return (Frans & Christopher, 2018) to better balance the bias and variance trade-off and to better account for the environment's non-Markovian property. MAVEN hybridises value and policy-based methods by introducing a latent space for hierarchical control. The value-based agents condition their behaviour on the shared latent variable controlled by a hierarchical policy. This allows MAVEN to achieve committed, temporally extended exploration, which is key to solve complex multi-agent tasks.

After running several SMAC scenarios with QMIX, we found out that QMIX struggles on some hard scenarios, especially, asymmetric scenarios, in which the enemy army outnumbers the allied army by one or more units. MAVEN performs substantially better than QMIX on the Super Hard map, corridor (Mahajan et al., 2020). However, it struggles on the easy map, 2s3z, with 20% test win rate, which is much lower than QMIX and SMIX, those have 100% and 40% test win rate, respectively. In addition, MAVEN also struggles on some harder maps, such as MMM2 and 3s_vs_5z. In contrast to QMIX and MAVEN, SMIX(λ) performs better on few hard maps, such as 3s_vs_5z and 2c_vs_64zg, and still performs good on easier maps.

In order to solve this difficult MARL problem, we proposed another idea to tackle it, which is communication-attended observation (CAO). CAO augments an agent's observation by the observations of its visible allies. In other words, the idea is to augment a specific agent's knowledge of the environment by its allies' partial observations. We have integrated CAO with QMIX, MAVEN, and SMIX(λ) in order to use latent variable to coordinate the agents' behaviors in latent space. We want to leverage the relaxation of requirement of decentralized execution by making agents retrieve the observation between each others while locating within their visible range. CAO improves and outperforms the original QMIX, SMIX(λ) and MAVEN on some maps, such as 5m_vs_6m and 2c_vs_64zg. For 1c3s5z map, CAO has enhanced SMIX(λ) significantly. In conclusion, experiment results show that CAO boosts the performance of baselines and has better convergence when comparing with them. The detailed implementations of CAO have been addressed in this final report.

1.1. Simulator/Data

SMAC and PyMARL are popular and important framework for MARL. SMAC makes use of the StarCraft II Learning Environment (SC2LE) (Vinyals et al., 2017) to communicate with the StarCraft II engine. SC2LE provides full control of the game by making it possible to send commands and receive observations. The goal of SC2LE is to learn to play the full game of StartCraft II at levels of both macro and micro managements. SMAC just focus on the part of cooperative multi-agent micromanagement. Training data should be acquired by interacting with the environment in unsupervised way.

PyMARL is built on PyTorch features implementation of several MARL algorithms and serves as a template for dealing with some of the unique challenges of deep MARL in practice. We are using SMAC and PyMARL for our final project to help us investigate MARL problems and develop our solutions.

1.2. Background

A fully cooperative multi-agent task can be described as a Dec-POMDP (Sutton & Barto, 2016) given by a tuple $G = \langle S, U, P, r, Z, O, n, \gamma \rangle$, where $s \in S$ is the true state of environment. At each time step, each agent $a \in A \equiv \{1, ..., n\}$ chooses an action $u^a \in U$, forming a joint action $\boldsymbol{u} \in \boldsymbol{U} \equiv U^n$. This causes a transition of the environment according to the state transition function $P(s'|s, \boldsymbol{u}) : S \times \boldsymbol{U}$.

In contrast to partially-observable stochastic games, all agents in a Dec-POMDP share the same team reward function $r(s, \boldsymbol{u}) : S \times \boldsymbol{U} \rightarrow \Re$, where $\gamma \in [0, 1)$ is the discount factor. Dec-POMDPs consider partially observable scenarios in which an observation function $O(s, a) : S \times A \rightarrow Z$ determines the observations $z^a \in Z$ that each agent draws individually at each time step. Each agent has an action-observation history $\tau^a \in T \equiv (Z \times U)^*$, on which it conditions a stochastic policy $\pi^a(u^a|\tau^a) : T \times U \rightarrow [0, 1]$. The joint policy π admits a joint action-value function: $Q^\pi(s_t, \boldsymbol{u}_t) = E_{s_{t+1:\infty}, u_{t+1:\infty}}[R_t|s_t, \boldsymbol{u}_t]$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ is the discounted return.

2. Related Work

2.1. QMIX

QMIX using an architecture consisting of agent networks, a mixing network, and a set of hypernetworks. Figure 1 illustrates the overall setup. For each agent a, there is one agent network that represents its individual value function $Q_a(\tau^a, u^a)$. Agent networks are represented as DRQNs that receive the current individual observation o_t^a and the last action u_{t-1}^a as input at each time step, as shown in Figure 1c. The mixing network is a feed-forward neural network that takes the agent network outputs as input and mixes them monotonically, producing the values of Q_{tot}, as shown in Figure 1a, which illustrates the mixing network and the hypernetworks.

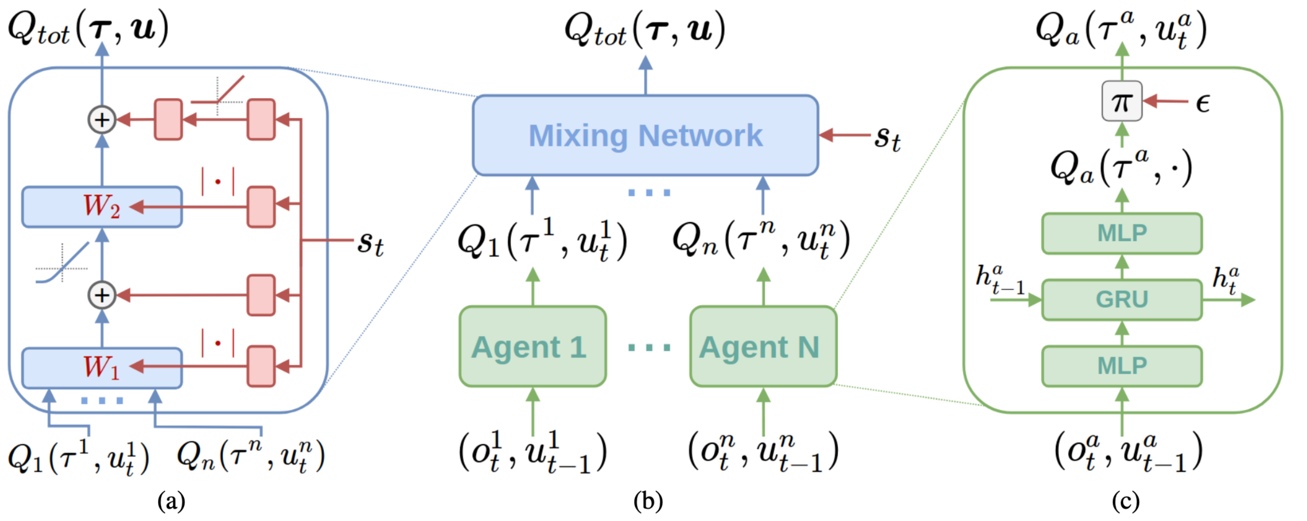


Figure 1. (a) Mixing network. The hypernetworks (in red) produce the weights and biases for mixing network layers (in blue). (b) The overall QMIX architecture. (c) Agent network.

QMIX proposed a constraint to ensure the monotonic relationship between Q_{tot} and Q_a:

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \forall a \in A. \tag{1}$$

By imposing such constraint, the argmax operation over Q_{tot} has the same effect as the argmax operations for each Q_a individually. The weights of the mixing network are restricted to be non-negative to enforce this monotonicity constraint.

For consistency QMIX needs to ensure that a global $argmax$ performed on Q_{tot} yields the same result as a set of individual $argmax$ operations performed on each Q_a:

$$\underset{\boldsymbol{u}}{\operatorname{argmax}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) = \begin{pmatrix} \operatorname{argmax}_{\boldsymbol{u}^1} Q_1(\boldsymbol{\tau}^1, \boldsymbol{u}^1) \\ \vdots \\ \operatorname{argmax}_{\boldsymbol{u}^n} Q_n(\boldsymbol{\tau}^n, \boldsymbol{u}^n) \end{pmatrix} \tag{2}$$

This allows each agent a to participate in a decentralised execution solely by choosing greedy actions with respect to its Q_a.

For the cost function, QMIX is trained end-to-end to minimise the following loss:

$$L(\theta) = \sum_{i=1}^{b} \left[(y_i^{tot} - Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}, s; \theta))^2 \right] \tag{3}$$

where b is the batch size of transitions sampled from the replay buffer, $y^{tot} = r + \gamma \max_{\boldsymbol{u}'} Q_{tot}(\boldsymbol{\tau}', \boldsymbol{u}', s'; \theta^-)$ and θ^- are the parameters of a target network as in DQN.

2.2. SMIX(λ)

SMIX(λ) is a SARSA(λ) (Sutton et al., 2014) style off-policy method that aims at learning a centralized value function within the framework of centralized training with decentralized execution (CTDE) for better MARL.

The overall architecture of these generalized versions of SMIX(λ) is given in Figure 4. The left and right dashed boxes show the value-based SMIX(λ) and actor-critic based SMIX(λ) algorithms, and the two solid boxes represent the modules in the centralized training and decentralized execution respectively. Each agent's Q network or π network only has access to its own observation (and its own observation history), and the centralized Q network or centralized critic network aggregates all agents' decentralized information.

The key idea of SMIX(λ) is to further simplify the coefficient ρ_i in (4), so as to reduce the variance of the importance sampling estimator and to potentially bypass the curse of dimensionality involved in calculating $\pi(\cdot|\boldsymbol{\tau})$.

$$Q(\boldsymbol{\tau}, \boldsymbol{a}) \leftarrow Q(\boldsymbol{\tau}, \boldsymbol{a}) + E_\mu \left[\sum_{t \geq 0} \gamma^t \left(\Pi_{i=1}^t \rho_i \right) \delta_t^\pi \right] \tag{4}$$

Specifically, relaxing each coefficient $\rho_i = 1.0$ in (4) and use an experience replay memory to store the most recent off-policy data. Then, using the $\lambda-$return as the TD target estimator, which is defined as follows:

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)} \tag{5}$$

$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n E_\pi Q(\boldsymbol{\tau}_{t+n}, \boldsymbol{a}_{t+n}; \theta^-)$ is the n-step return and θ^- are the parameters of the target network.

In implementation, SMIX(λ) is trained end-to-end and the loss function for the centralized value function Q_{tot}^π has the following form:

$$L_t(\theta) = \sum_{i=1}^{N_b}\left[\left(y_i^{tot} - Q_{tot}^\pi(\boldsymbol{\tau}, \boldsymbol{a}; \theta)\right)^2\right] \tag{6}$$

where $y_i^{tot} = G_t^\lambda$ is defined in (6) and is estimated through it experience replaying, N_b is the batch size.

2.3. MAVEN

Decentralised execution and monotonicity constraint restrict QMIX to provably poor exploration and suboptimality. MAVEN (Mahajan et al., 2020) has been proposed in order to shed light on a problem unique to decentralised MARL that arises due to inefficient exploration. Figure 5 illustrates the complete setup for MAVEN. The lefthand side of the diagram describes the learning framework for the latent space policy and the joint action values. MAVEN parametrises the hierarchical policy by θ, the agent utility network with η, the hypernet map from latent variable z used to condition utilities by ϕ, and the mixer net with ψ. η can be associated with a feature extraction module per agent and ϕ can be associated with the task of modifying the utilities for a particular mode of exploration.

MAVEN models the hierarchical policy $\pi_z(\cdot|s_0;\theta)$ as a transformation of a simple random variable $x \sim p(x)$ through a neural network parameterised by θ; thus $z \sim g_\theta(x, s_0)$, where s_0 is initial state. Fixing z gives a joint action-value function $Q(\boldsymbol{u}, s; z, \phi, \eta, \psi)$ which implicitly defines a greedy deterministic policy $\pi_A(\boldsymbol{u}|s; z, \phi, \eta, \psi)$.

The corresponding Q-learning loss $L_{QL}(\phi, \eta, \psi)$:

$$E_{\pi A}\left[\left(Q(\boldsymbol{u}_t, s_t; z) - \left[r(\boldsymbol{u}_t, s_t) + \gamma \max_{\boldsymbol{u}_{t+1}} Q(\boldsymbol{u}_{t+1}, s_{t+1}; z)\right]\right)^2\right] \tag{7}$$

where t is the time step. Next, fixing ϕ, η, ψ, the hierarchical policy over $\pi_z(\cdot|s_0;\theta)$ is trained on the cumulative trajectory reward $R(\tau, z|\phi, \eta, \psi) = \sum_t r_t$ where τ is the joint trajectory.

The overall objective is presented in the following equation:

$$\max_{\upsilon,\phi,\eta,\psi,\theta} J_{RL}(\theta) + \lambda_{MI} J_V(\upsilon, \phi, \eta, \psi) - \lambda_{QL} L_{QL}(\phi, \eta, \psi),$$

where λ_{MI} , λ_{QL} are positive multipliers.

3. Approach

We enable QMIX (Rashid et al., 2018), SMIX(λ) (Yao et al., 2019) and MAVEN (Mahajan et al., 2020) as the baselines. We propose a method named as Communication-Attended Observation (CAO) which augments an agent's observation by the observations of its visible allies. We integrates CAO into the 3 baselines and for each we can observe performance gain in certain degree.

This section is organized into 2 parts. In section 3.1, we go through the architectures of QMIX (Rashid et al., 2018), SMIX(λ) (Yao et al., 2019) and MAVEN (Mahajan et al., 2020), we discuss the advantage and weakness of these 3 methods respectively. In section.3.2, we illustrate the proposed Communication-Attended Observation (CAO), we explain its intuition and architecture in details.

3.1. Baseline Analysis: QMIX, SMIX(λ) and MAVEN

Section.2.1 has explained the QMIX algorithm (Rashid et al., 2018) in details. Basically the advantage of QMIX is its simplicity and effectiveness. We find that QMIX generally obtains decent performance for a bunch of maps, it rarely degrades performance to zero winning rate. Note in Figure 1.(a), the generation of *Mixing Network* is conditioned on the realtime state s_t, which makes QMIX robust to the environment change. However, Eq.(2) greatly restricts the hypothesis space of QMIX and as pointed by MAVEN (Mahajan et al., 2020): a specific QMIX $agent_i$ blindly optimizes its objective Q_i independent of other agents, leading to a single monotonic policy invariant to actions of all the other agents. QMIX is also known as lack of exploration, simply enlarging ϵ in greedy search (see Figure 1.(c)) can't solve this problem as MAVEN (Mahajan et al., 2019) proves larger ϵ can actually increase the probability of adopting sub-optimal policy for certain cases.

Section.2.2 has explained SMIX(λ) in details. SMIX(λ) inherits the architecture of QMIX but tries to enhance the exploration capability by two ways (see Figure 4). By Eq.(4), SMIX(λ) defines a Q function fit for off policy learning via importance sampling. By using λ-return G_t^λ as y_i^{tot} (see Eq.(6)), SMIX(λ) relaxes the restriction introduced by Eq.(3) where y_i^{tot} is determined by the *Q-hypernet* within the QMIX hypothesis space. Basically SMIX(λ) has a larger hypothesis space than that of QMIX, thus SMIX(λ) is expected to perform well in learning more complex Q function. However, SMIX(λ) makes a very strong assumption that the importance sampling weight $\rho_i = 1$, which equals to assume the proposal policy is exactly the same as the optimal policy. Secondly, SMIX(λ) tries to prove its temporal Q value asymptotically converges to that of Q(λ) (Harutyunyan et al., 2016). But the proof is conducted by induction completely ignoring the accumulation errors. Most likely these drawbacks explain why we don't observe SMIX(λ) produces significant performance boost compared to QMIX.

Section.2.3 has explained MAVEN (Mahajan et al., 2020) in details. MAVEN proposes a novel idea that to coordinate the agents behavior in latent space conditional on a certain map. MAVEN introduces concept of generative model in tackling multi-agent reinforcement learning by maximizing the mutual information between the latent variable and the behavior of agents. However, as illustrated in Figure 5, MAVEN generates the latent variable z by the initial state s_0 of one episode, then z won't be changed through out the episode. MAVEN views z as a kind of *mode* variable but this *mode* is not sufficient to adapt to the changing environment within the episode execution.

In summary, SMIX(λ) and MAVEN try to enhance the hypothesis space and exploration capability in different ways. However, they either make strong assumptions or design an insufficiently flexible latent variable. We think of enhancing the exploration capability in an alternative perspective, that is, we broaden the sight of a specific agent by its allies.

3.2. Communication-Attended Observation

We relax the requirement of decentralized execution a little by making an agent could retrieve the observation of another agent locating within its visible range. We argue that this relaxation is reasonable because visibility generally guarantees communication either in games or in real life.

Figure 2 illustrates the architecture of the proposed Communication-Attended Observation mechanism. The intuition is to augment the partial observation of a specific agent by the partial observations of its allies. Intuitively an enemy which is out of an agent's horizon may be within that of a visible ally, our COA mechanism makes the agent aware the existence of the enemy indirectly via its allies' observations.

In Figure 2, $F_{AL^1}, F_{AL^2}, \cdots, F_{AL^n}$ are the partial observation features of the n allies'. F_{agent} is the partial observation feature of the agent. When conduct a kind of self-attention similar as Transformer (Vaswani et al., 2017) does. That is, given a observation feature F, assume we have 3 functions $Key(F), Query(F), Value(F)$ which are *key, query, value* vectors of F respectively.

The attention weight agent with respect to the kth ally is given by:

$$W_k = Key(F_{AL^k})^T Query(F_{agent}) \tag{8}$$

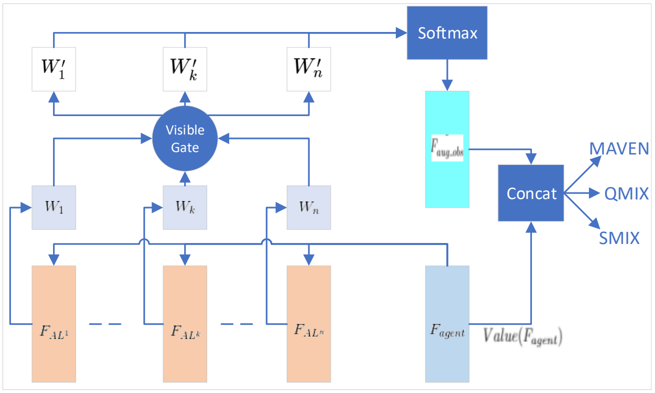


Figure 2. Architecture of the proposed Communication-Attended Observation (CAO) mechanism.

The Visible Gate adjusts the attention weights according to the visibility of allies by:

$$W'_k = \begin{cases} W_k, & k^{th}\ ally\ visible \\ -\infty, & otherwise \end{cases} \tag{9}$$

The final attention weight $\boldsymbol{w}$ is obtained by applying softmax operation to W' by:

$$w_i = \frac{e^{W'_i}}{\sum_{k=1}^{n} e^{W'_k}} \tag{10}$$

From Eq.(10), we see the attention weights of those invisible allies' have been driven to 0. Thus invisible allies won't contribute to the augmented observations.

The augmented observation feature is determined by:

$$F_{aug\_obs} = \sum_{k=1}^{n} w_k Value(F_{AL^k}) \tag{11}$$

The Communication-Attended Observation is obtained by concatenating the augmented observation F_{aug_obs}, the *value* of F_{agent}, the agent id and its last action.ncall Specifically:

$$F_{CAO} = [F_{aug\_obs}; Value(F_{agent}); agent{-}id; last{-}action] \tag{12}$$

$$F_{CAO} = [F_{aug\_obs}; Value(F_{agent})] \tag{13}$$

The Communication-Attended Observation F_{CAO} is handed over to the downstream logic, which can be QMIX, SMIX(λ) and MAVEN. Note our CAO mechanism aims to broadens the agent's exploration in feature space. On one side, we make more informative agent by sharing partial observations. On the other side, the sharing only happens between visible allies, the execution still keeps the decentralized characteristics.

4. Experiment results

We conduct experiments for QMIX, SMIX(λ) and MAVEN, w/ and w/o Communication Attended Observation (CAO). Following (Rashid et al., 2018), the evaluation metric is *test win rate*, defined by the percentage of 20 independent episodes in which the method defeats all enemy units within the time limit. All experiment results are shown in Figure 3.

The results for QMIX w/o and w/ CAO are shown in Figure 3(a) and 3(b). For scenario 5m_vs_6m (black in color), CAO boosts the performance of QMIX by a large margin (13.2 absolute percentage points). This shows the effectiveness of CAO, which enables the information propagation among allies. For example, assume agent 1 can see agent 2 but not agent 3, while agent 2 can see both agent 1 and agent 3. Then it's possible for agent 1 to get some kind of information from agent 3 via agent 2. And this is how the communication is attended and how the information propagation is achieved. For some easy scenarios like 3m, 8m, 1c3s5z (yellow, orange and blue in color), QMIX w/ CAO achieves similar/slightly better results than those of the vinillar QMIX (both are close to 1). CAO causes performance degradation a little bit for scenario 2s3z and 3s5z (green and red in color). The reason for this might be CAO augments the input space and thus more training data/iterations are needed to achieve similar results as the vinilla QMIX.

The results for SMIX(λ) w/o and w/ CAO are shown in Figure 3(c) and 3(d). From the experiments results, we can see SMIX(λ) does not improve QMIX actually. This is because we fix the hyperparameter space of SMIX(λ) for fair comparison and SMIX(λ) is not robust enough with a single set of hyperparameters (different maps need different hyperparameters to achieve optimal results). However, we still find CAO improves the performance of SMIX(λ) a lot. For scenario 3m (orange in color), SMIX(λ) w/ CAO converges faster than SMIX(λ) itself. For scenario 1c3s5z, SMIX(λ) w/ CAO has a much better performance (improves *test win rate* by 27.3 absolute percentage points). CAO is thus still an effective scheme/module that can be inserted to SMIX(λ).

The results for MAVEN w/o and w/ CAO are shown in Figure 3(e) and 3(f). Similarly to SMIX(λ), MAVEN does not improve QMIX actually due to the fixed hyperparameters. Such observation shows QMIX is still a strong and robust baseline for multi-agent reinforcement learning. From Figure 3(e) and 3(f), we can see CAO makes MAVEN converge much faster. For scenario 3m and 8m, MAVEN w/ CAO converges in less than 0.5m steps, while MAVEN itself converges in 1.5m steps. On interesting finding is for scenario 2s3z, MAVEN w/ CAO has a much higher peak value (improves *test win rate* by 33.1 absolute percentage points). However, for both MAVEN w/o CAO and w/CAO, *test win rate* decreases to an extremely low value after reaching the peak value. This is because the training process of the discriminator in MAVEN is unstable sometimes and we aim to improve its stability in the future.

5. Conclusion

We proposed Communication-Attended Observation (CAO) which augments an agent's observation by the observations of its visible allies. We relax the requirement of decentralized execution compared with prior works and enable information propagation among allies. We argue that this relaxation is reasonable because visibility generally guarantees communication either in games or in real life.

We integrates CAO into three prior works (QMIX, SMIX(λ) and MAVEN). Experiment results consistently show CAO either boosts the performance (improves *test win rate*) or makes the backbone method converges faster.

In the future, we aim to conduct additional experiments to compare the methods under more hard scenarios. We would like to improve CAO mechanism with a more coordinated strategy among agents.

6. Contributions of team members

All of the three team member have equal contributions to the report. For the entire project, Liu focuses on methodology proposal and implementation; Guanzhi focuses on experiment running and analysis; PeiChen focuses on environment setup.

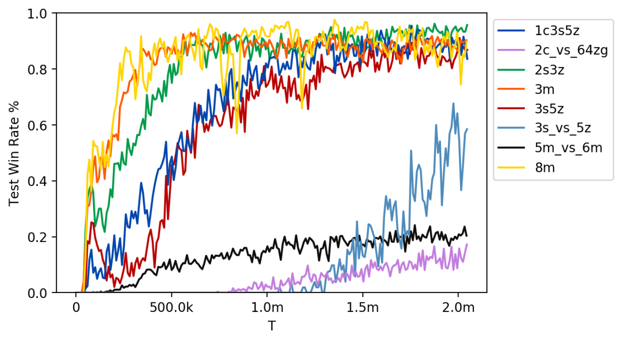
References

Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. *CoRR*, abs/1705.08926, 2017.

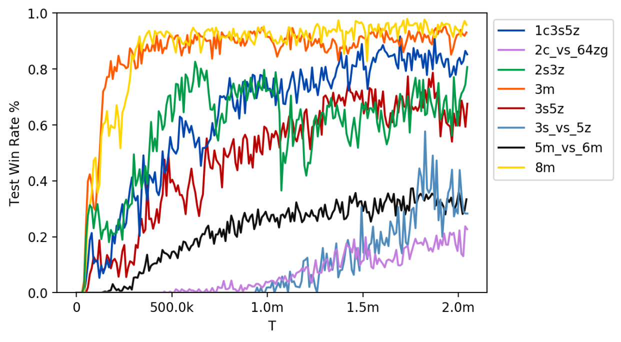
Frans, A., O. and Christopher, A. Reinforcement learning: An introduction. *MIT press*, 2018.

Harutyunyan, A., Bellemare, M. G., Stepleton, T., and Munos, R. Q(λ) with off-policy corrections. In Ortner, R., Simon, H. U., and Zilles, S. (eds.), *Algorithmic Learning Theory*, pp. 305–320, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46379-7.

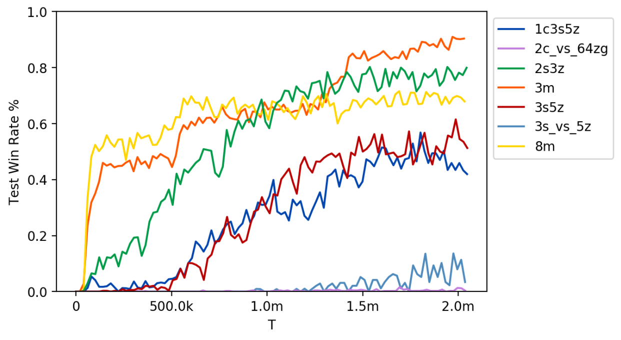
Mahajan, A., Rashid, T., Samvelyan, M., and Whiteson, S. Maven: Multi-agent variational exploration. In *Advances in Neural Information Processing Systems*, pp. 7611–7622, 2019.



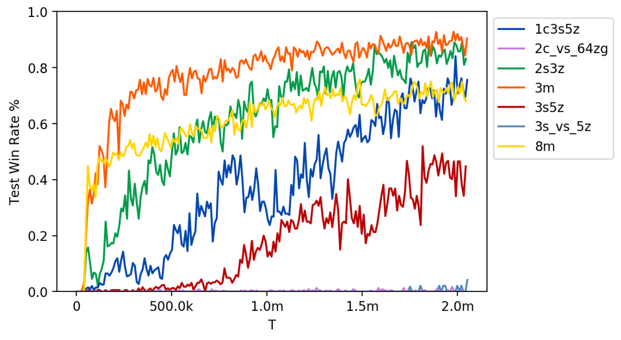
(a) QMIX w/o CAO.



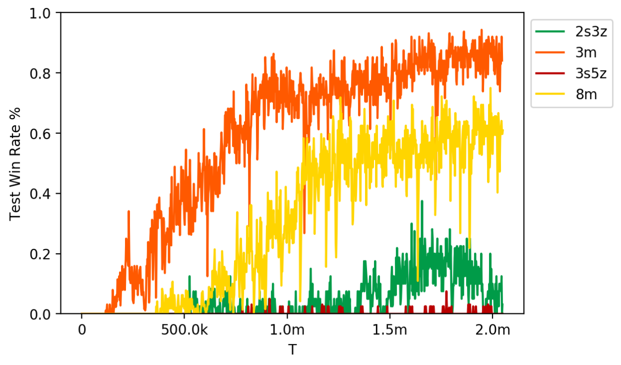
(b) QMIX w/ CAO.



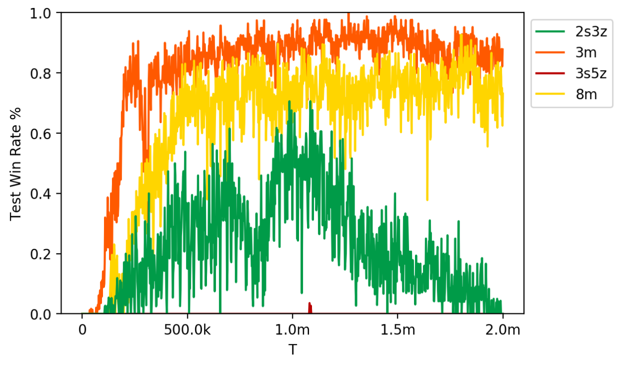
(c) SMIX(λ) w/o CAO.



(d) SMIX(λ) w/ CAO.



(e) MAVEN w/o CAO.



(f) MAVEN w/ CAO.

Figure 3. Experiment results for QMIX, SMIX(λ) and MAVEN, w/ and w/o Communication Attended Observation (CAO).

Mahajan, A., Rashid, T., Samvelyan, M., and Whiteson, S. Maven: Multi-agent variational exploration. *arXiv:1910.07483v2*, 2020.

Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. *International Conference on Machine Learning*, 2018.

Samvelyan, M., Rashid, T., de Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G. J., Hung, C., Torr, P. H. S., Foerster, J. N., and Whiteson, S. The starcraft multi-agent challenge. *CoRR*, abs/1902.04043, 2019.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, Wojciech, M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, Joel, Z., Tuyls, K., and Graepel, T. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296v1*, 2017.

Sutton, R., Mahmood, Ashique, R., Precup, D., and Hasselt, H. A new q(λ) with interim forward view and monte carlo equivalence. *PMLR*, 2014.

Sutton, Richard, S. and Barto, Andrew, G. A concise introduction to decentralized pomdps. *Springer*, 2016.

Tan, M. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth inter-*

national conference on machine learning, pp. 330–337, 1993.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T. P., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., and Tsing, R. Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017.

Yao, X., Wen, C., Wang, Y., and Tan, X. Smix(λ): Enhancing centralized value functions for cooperative multi-agent reinforcement learning. *arXiv:1911.04094V3*, 2019.

A. Appendix

Code base

We have made our code publicly available.

QMIX and SMIX + CAO:

https://github.com/xiaoxiaoheimei/cs234-2020win-project.

MAVEN + CAO:

https://github.com/xiaoxiaoheimei/cs234-win-final-MAVEN-base

Our implementations are based on the offical repos of QMIX, SMIX and MAVEN.

QMIX: https://github.com/oxwhirl/pymarl

SMIX: https://github.com/chaovven/SMIX

MAVEN: https://github.com/AnujMahajanOxf/MAVEN

We have kept the old commits. So it's easy to track our changes. CAO is defined here.

Name	Ally Units	Enemy Units
2s3z	2 Stalkers & 3 Zealots	2 Stalkers & 3 Zealots
3s5z	3 Stalkers & 5 Zealots	3 Stalkers & 5 Zealots
1c3s5z	1 Colossus, 3 Stalkers & 5 Zealots	1 Colossus, 3 Stalkers & 5 Zealots
3m	3 Marines	3 Marines
8m	8 Marines	8 Marines
5m_vs_6m	5 Marines	6 Marines
27m_vs_30m	27 Marines	30 Marines
MMM2	1 Medivac, 2 Marauders & 7 Marines	1 Medivac, 3 Marauders & 8 Marines
3s_vs_5z	3 Stalkers	5 Zealots
2c_vs_64zg	2 Colossi	64 Zerglings

Table 1. Maps in SMAC challenges.

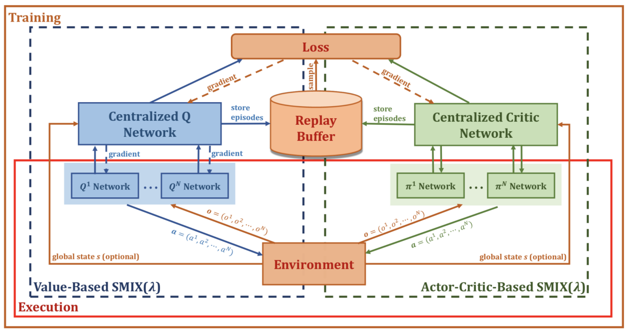

Training
Loss
gradient
sample
gradient
Centralized Q Network
store episodes
Replay Buffer
store episodes
Centralized Critic Network
gradient
gradient
Q^1 Network
Q^N Network
π^1 Network
π^N Network
$o = (o^1, o^2, \dots, o^N)$
$a = (a^1, a^2, \dots, a^N)$
$o = (o^1, o^2, \dots, o^N)$
$a = (a^1, a^2, \dots, a^N)$
global state s (optional)
global state s (optional)
Environment
Value-Based SMIX(λ)
Actor-Critic-Based SMIX(λ)
Execution

Figure 4. Value-based SMIX(λ) and actor-critic-based SMIX(λ) architecture

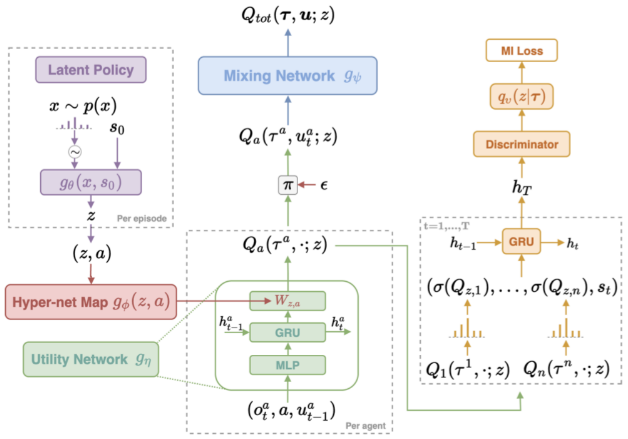


Figure 5. Architecture for MAVEN