Learning Agent Communication under Limited Bandwidth by Message Pruning

Hangyu Mao,[1] **Zhengchao Zhang,**[1] **Zhen Xiao,**[1] **Zhibo Gong,**[2] **Yan Ni**[1]
[1]Peking University, [2]Huawei Technologies Co., Ltd.
{hy.mao, zhengchaozhang, xiaozhen, niyan.ny}@pku.edu.cn, gongzhibo@huawei.com

Abstract

Communication is a crucial factor for the big multi-agent world to stay organized and productive. Recently, Deep Reinforcement Learning (DRL) has been applied to learn the communication strategy and the control policy for multiple agents. However, the practical ***limited bandwidth*** in multi-agent communication has been largely ignored by the existing DRL methods. Specifically, many methods keep sending messages incessantly, which consumes too much bandwidth. As a result, they are inapplicable to multi-agent systems with limited bandwidth. To handle this problem, we propose a gating mechanism to adaptively prune less beneficial messages. We evaluate the gating mechanism on several tasks. Experiments demonstrate that it can prune a lot of messages with little impact on performance. In fact, the performance may be greatly improved by pruning redundant messages. Moreover, the proposed gating mechanism is applicable to several previous methods, equipping them the ability to address bandwidth restricted settings.

Introduction

Communication is an essential human intelligence. It is a critical factor to keep the world in order. Recently, inspired by the communication among humans, Deep Reinforcement Learning (DRL) has been adopted to learn the communication among multiple artificial agents.

However, it remains an open question to apply the existing methods to real-world multi-agent systems because real-world applications usually impose many constraints on communication, e.g., the bandwidth limitation for transmitting messages and the protection of private messages. In order to develop practical communication strategies, these constraints need to be resolved.

In this paper, we focus on addressing the ***limited bandwidth*** problem in multi-agent communication. As we know, the bandwidth (and more generally, the resource) for transmitting the communication messages is limited (Roth, Simmons, and Veloso 2005; 2006; Becker et al. 2009; Wu, Zilberstein, and Chen 2011; Zhang and Lesser 2013). Thus, the agents should generate as few messages as possible on the premise of maintaining the performance.

We are interested in the limited bandwidth problem due to two reasons. On the one hand, it is ubiquitous in the real world. For example, in the wired packet routing systems, the links have a limited transmission capacity; in the wireless Internet of Things, the sensors have a limited battery capacity. Once we figure out a principled method to address this problem, many fields may benefit from this work. On the other hand, this problem has been largely ignored by the existing DRL methods. There is a great need to devote attention to this problem.

We take two steps to address this problem. Firstly, we aggregate the merits of the existing methods to form a basic model named *Actor-Critic Message Learner* (ACML). However, the basic ACML is still not practical because it does not change the communication pattern of the existing methods. That is to say, ACML and many previous methods send messages incessantly, regardless of whether the message is beneficial for the agent team. Secondly, we extend ACML with a gating mechanism to design a more flexible and practical Gated-ACML model. The gating mechanism is trained based on a novel auxiliary task, which tries to open the gate to encourage communication when the message is beneficial enough for the agent team, and close the gate to discourage communication otherwise. As a result, after the gating mechanism has been well-trained, it can prune unbeneficial messages adaptively to control the message quantity around a desired threshold. Consequently, Gated-ACML is applicable to multi-agent systems with limited bandwidth.

Our contributions are summarized as follows.

(1) We are the first to formally study the limited bandwidth problem using DRL. We give the first detailed analyses about how gating mechanism can serve this purpose.

(2) We apply the gating mechanism to several previous methods, equipping them the ability to address bandwidth restricted settings.

(3) We evaluate the gating mechanism on several simulators driven by the real-world tasks. Experiments show that it can prune a lot of messages with little impact on performance, or with great improvement on performance in specific scenarios. These are very fundamental findings in the community. It could be the best contribution of our work beyond the gating mechanism.

Related Work

Communication is vital for multi-agent systems. Recently, the communication channel implemented by Deep Neural Network (DNN) has been proven useful for learning beneficial messages (Sukhbaatar, Fergus, and others 2016; Foerster et al. 2016; Peng et al. 2017; Peng, Zhang, and Luo 2018; Mao et al. 2017; Kong et al. 2017; Mao et al. 2019; Lowe et al. 2017; Chu and Ye 2017; Kilinc and Montana 2018; Kim et al. 2019; Jiang and Lu 2018; Singh, Jain, and Sukhbaatar 2019; Kim, Cho, and Sung 2019).

CommNet (Sukhbaatar, Fergus, and others 2016) embeds a centralized communication channel into the actor network, and it processes other agents' messages by averaging them. AMP (Peng, Zhang, and Luo 2018) adopts a similar design, but it applies an attention mechanism to focus on specific messages from other agents. Relevant studies include but are not limited to (Foerster et al. 2016; Mao et al. 2017; Kong et al. 2017). However, the policy networks used in these methods take as input the messages from all agents to generate a single control action. Thus, the agents have to keep sending messages incessantly in every control cycle, without alternatives to reduce the message quantity. Due to emitting too many messages, they are inflexible to be applied to multi-agent systems with limited bandwidth.

We argue that the principled way to address the limited bandwidth problem is to apply some special mechanisms to adaptively decide whether to send (equivalently, whether to *prune*) the current message. From this perspective, the most relevant studies to our Gated-ACML are IC3Net (Singh, Jain, and Sukhbaatar 2019) and ATOC (Jiang and Lu 2018). Technically, the three methods adopt the so-called gating mechanism to generate a binary action to specify whether the agent should communicate with others. The fundamental difference is the research purpose: IC3Net aims to learn when to communicate at scale in both cooperative and competitive tasks; ATOC targets at learning to communicate with a dynamic agent topology; as far as we know, Gated-ACML is the first formal DRL method to address the limited bandwidth problem.

As a result of different research purposes, the implementations are very different. IC3Net allows multiple communication cycles in one step, and it generates the gating value for the next communication cycle. ATOC relies on the gating value to form dynamic local communication group, and it may classify one agent into many agent groups. Because these implementations do not explicitly consider bandwidth limitation, they may generate a lot of messages (as indicated by the experiments). In contrast, Gated-ACML adopts a global Q-value difference as well as a specially designed threshold to identify the beneficial messages, and it applies the gating value to prune the current messages of a global communication group. Therefore, it has more sophisticated ability to control how many messages will be pruned and to satisfy the needs of different applications.

Meanwhile, IC3Net and ATOC are only suitable for homogeneous agents due to their DNN structures, and they are designed for discrete action. By contrast, Gated-ACML is suitable for both homogeneous and heterogeneous agents, and it is designed for continuous action.

Background

DEC-POMDP. We consider a *fully cooperative* multi-agent setting that can be formulated as DEC-POMDP (Bernstein et al. 2002). It is formally defined as a tuple $\langle N, S, \vec{A}, T, R, \vec{O}, Z, \gamma \rangle$, where N is the number of agents; S is the set of state s; $\vec{A} = [A_1, ..., A_N]$ represents the set of *joint action* $\vec{a}$, and A_i is the set of *local action* a_i that agent i can take; $T(s'|s, \vec{a}) : S \times \vec{A} \times S \to [0, 1]$ represents the state transition function; $R : S \times \vec{A} \times S \to \mathbb{R}$ is the reward function shared by all agents; $\vec{O} = [O_1, ..., O_N]$ is the set of *joint observation* $\vec{o}$ controlled by the observation function $Z : S \times \vec{A} \to \vec{O}$; $\gamma \in [0, 1]$ is the *discount factor*.

In a given state s, agent i can only observe an observation o_i, and each agent takes an action a_i based on its observation o_i (and possibly messages from other agents), resulting in a new state s' and a shared reward r. The agents try to learn a policy $\pi_i(a_i|o_i) : O_i \times A_i \to [0, 1]$ that can maximize $\mathbb{E}[G]$ where G is the *discount return* defined as $G = \sum_{t=0}^{H} \gamma^t r^t$, and H is the time horizon. In practice, we map the observation history rather than the current observation to an action, namely, o_i represents the observation history of agent i in the rest of the paper.

Reinforcement Learning (RL). RL (Sutton and Barto 1998) is generally used to solve special DEC-POMDP problems where $N = 1$. In practice, we define the Q-value (or action-value) function as $Q^\pi(s, a) = \mathbb{E}_\pi[G|S = s, A = a]$, then the optimal policy π^* can be derived by $\pi^* = \arg\max_\pi Q^\pi(s, a)$. The actor-critic algorithm is one of the most effective RL methods because the actor and the critic can reinforce each other, so we design our Gated-ACML based on actor-critic method. Deterministic Policy Gradient (DPG) (Silver et al. 2014) is a special actor-critic algorithm where the actor adopts a deterministic policy $\mu_\theta : S \to A$ and the action space A is continuous. Deep DPG (DDPG) (Lillicrap et al. 2015) applies DNN $\mu_\theta(s)$ and $Q(s, a; w)$ to represent the actor and the critic, respectively. DDPG is an off-policy method. It adopts *target network* and *experience replay* to stabilize training and to improve data efficiency. Specifically, the critic's parameters w and the actor's parameters θ are updated based on:

$$L(w) = \mathbb{E}_{(s,a,r,s') \sim D}[\delta^2] \tag{1}$$

$$\delta = r + \gamma Q(s', a'; w^-)|_{a'=\mu_{\theta^-}(s')} - Q(s, a; w) \tag{2}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim D}[\nabla_\theta \mu_\theta(s) * \nabla_a Q(s, a; w)|_{a=\mu_\theta(s)}] \tag{3}$$

where $L(w)$ is the loss function of the critic; $J(\theta)$ is the objective function of the actor; D is the replay buffer containing recent experience tuples (s, a, r, s'); $Q(s, a; w^-)$ and $\mu_{\theta^-}(s)$ are the target networks whose parameters w^- and θ^- are periodically updated by copying w and θ. A merit of actor-critic algorithms is that the critic is only used during training, and it will be naturally removed during execution, so we only need to prune messages among the actors in our Gated-ACML.

Methods

We list the key notations as follows. a_i is the local action of agent i. $\vec{a}_{-i}$ is the joint action of other agents except for

agent i. $\vec{a}$ is the joint action of all agents, i.e., $\vec{a} = \langle a_i, \vec{a}_{-i} \rangle$. The observation history $\vec{o}$, o_i, $\vec{o}_{-i}$, and the policy μ_{θ_i} are denoted similarly. s' is the next state after s, and $\vec{o}'$, o'_i, $\vec{o}'_{-i}$, $\vec{a}'$, a'_i, $\vec{a}'_{-i}$ are denoted in a similar sense.

ACML

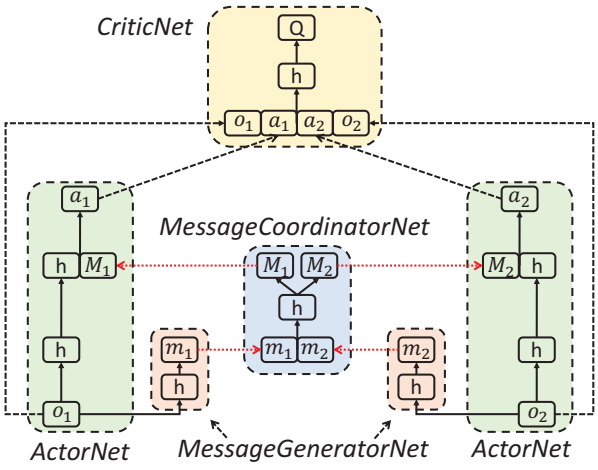


Figure 1: The proposed ACML. For clarity, we illustrate this model with a two-agent example. All components are implemented by DNN. h is the hidden layer of the DNN; m_i is the local message; M_i is the global message. The red arrows indicate the message exchange process.

Design. ACML is motivated by combining the merits of the existing methods. As Figure 1 shows, ACML adopts the following design: each agent is composed of an ActorNet and a MessageGeneratorNet, while all agents share the same CriticNet and MessageCoordinatorNet. All components are implemented by DNN. The shared MessageCoordinatorNet is similar to the communication channel of many previous methods such as CommNet, AMP and ATOC, while the shared CriticNet is the same as the well-known MADDPG (Lowe et al. 2017). By sharing the encoding of observation and action within the whole agent team, individual agents could build up relatively greater global perception, infer the intent of other agents, and cooperate on decision making (Jiang and Lu 2018).

In addition, aggregating these components properly could relieve the problems encountered by previous methods. For example, MADDPG and (Mao et al. 2019; Chu and Ye 2017) do not adopt the MessageCoordinatorNet, making them suffer from the partially observable problem; while AMP and ATOC do not adopt the CriticNet, making them suffer from the non-stationary problem during training (Hernandez-Leal et al. 2017). In contrast, ACML is fully observable and training stationary due to the careful design.

ACML works as follows during execution. (1) $m_i = \text{MessageGeneratorNet}(o_i)$, i.e., agent i generates the local message m_i based on its observation o_i. (2) All agents send the message m_i to the MessageCoordinatorNet. (3) $M_1, .., M_N = \text{MessageCoordinatorNet}(m_1, .., m_N)$, i.e., the MessageCoordinatorNet extracts the global message M_i for each agent i based on all local messages. (4) The MessageCoordinatorNet sends M_i back to agent i. (5) $a_i = \text{ActorNet}(o_i, M_i)$, i.e., agent i generates action a_i based on its local observation o_i and the global message M_i.

Training. The agents generate a_i based on o_i and M_i to interact with the environment, and the environment will feed a shared reward r to the agents. Then, the experience tuples $\langle o_i, \vec{o}_{-i}, a_i, \vec{a}_{-i}, r, o'_i, \vec{o}'_{-i} \rangle$ are used to train ACML. Specifically, as the agents exchange messages with each other, the actor and the shared critic can be represented as $\mu_{\theta_i}(o_i, M_i)$ and $Q(\vec{o}, \vec{a}; w)$, respectively. We can extend Equation (1 – 3) to multi-agent formulations as follows:

$$L(w) = \mathbb{E}_{(o_i, \vec{o}_{-i}, a_i, \vec{a}_{-i}, r, o'_i, \vec{o}'_{-i}) \sim D}[\delta^2] \tag{4}$$

$$\delta = r + \gamma Q(\vec{o}', \vec{a}'; w^-)|_{a'_i = \mu_{\theta_i^-}(o'_i)} - Q(\vec{o}, \vec{a}; w) \tag{5}$$

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{(o_i, \vec{o}_{-i}) \sim D}[\nabla_{\theta_i} \mu_{\theta_i}(o_i, M_i) \nabla_{a_i} Q(\vec{o}, \vec{a}; w)] \tag{6}$$

where $L(w)$, $J(\theta_i)$, w, θ_i, w^- and θ_i^- have similar meanings as the single-agent setting. As for the gradient to update the parameters θ^{cc} of communication channel (i.e., the MessageGeneratorNet and MessageCoordinatorNet), it can be further derived by the chain rule as:

$$\begin{aligned} \mathbb{E}_{(o_i, \vec{o}_{-i}) \sim D, i \sim [1,N]} [\nabla_{\theta^{cc}} M_i(m_1, ..., m_N; \theta^{cc}) * \\ \nabla_{M_i} \mu_{\theta_i}(o_i, M_i) * \\ \nabla_{a_i} Q(\vec{o}, \vec{a}; w)|_{a_i = \mu_{\theta_i}(o_i)}] \end{aligned} \tag{7}$$

Since ACML is end-to-end differentiable, the communication message and the control policy can be optimized jointly using back-propagation based on the above equations.

Gated-ACML

Motivation. As the execution process shows, ACML takes as input the encoding of messages from all agents to generate a control action. The agents have to send messages incessantly, regardless of whether the messages are beneficial to the performance of the agent team. This is a common problem of many previous methods such as CommNet (Sukhbaatar, Fergus, and others 2016), AMP (Peng, Zhang, and Luo 2018) and (Foerster et al. 2016; Mao et al. 2017; Kong et al. 2017). As a result, these methods usually consume a lot of bandwidths and resources, and they are unpractical for multi-agent systems with limited bandwidth.

We argue that the principled way to address this problem is to apply some special mechanisms to adaptively decide whether to ***prune*** the messages. Gated-ACML is motivated by this idea, and it adopts a gating mechanism to adaptively prune less beneficial messages among ActorNets, such that the agents can maintain the performance with as few messages as possible.

Design. As shown in Figure 2, besides the original components, each agent is equipped with an additional GatingNet. Specifically, Gated-ACML works as follows. (1) The agent generates a local message m_i as well as a probability score $p \in (0, 1)$ based on the observation o_i. (2) A gate value $g \in \{0, 1\}$ is generated by the indicator function $g \leftarrow \mathbb{I}(p > 0.5)$. That is to say, if $p > 0.5$, we set $g = 1$, otherwise, we set $g = 0$. (3) The agent sends $m_i \odot g$

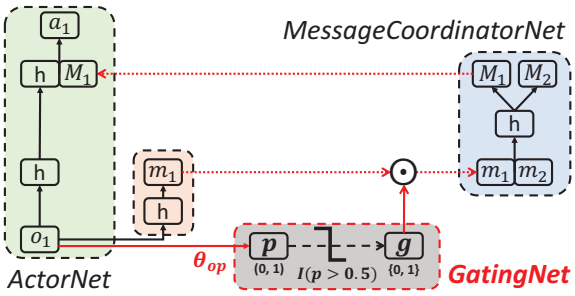


Figure 2: The actor part of the proposed Gated-ACML. For clarity, we only show one agent's structure, and we do not show the critic part because it is the same as that of ACML.

to the MessageCoordinatorNet, and the following process is the same as that of ACML.

As can be figured out, if $g = 1$, $m_i \odot g$ equals to the original message m_i, thus Gated-ACML is equivalent to ACML. In contrast, if $g = 0$, $m_i \odot g$ will be a zero vector, which means that the message can be pruned. In practice, if the agent has not received a message when the decision making time arrived, it will pad zeros automatically. This design is supported by the message-dropout technique (Kim, Cho, and Sung 2019), which has shown that replacing some messages by zero vectors with a certain probability can make the training robust against communication errors and message redundance that are common for large-scale communication (Kilinc and Montana 2018; Kim et al. 2019; Jiang and Lu 2018; Singh, Jain, and Sukhbaatar 2019; Kim, Cho, and Sung 2019). We find similar phenomena in our experiments.

Training with Auxiliary Task. In order to make the above design work, a suitable probability p must be trained for each observation o_i, otherwise Gated-ACML may degenerate to ACML in the extreme case where $\mathbb{I}(p > 0.5)$ always equals to 1. However, as the indicator function $g \leftarrow \mathbb{I}(p > 0.5)$ is non-differentiable, it makes the end-to-end back-propagation method inapplicable. We have tried approximate gradient (Hubara et al. 2016), sparse regularization (Makhzani and Frey 2015) and several other methods without success. To bypass the training of the non-differentiable indicator function, we decide to train its input p directly. To this end, we adopt the auxiliary task technique (Jaderberg et al. 2016) to provide training signal for p explicitly.

Recall that we want to prune the messages on the premise of maintaining the performance. Because the performance of RL could be measured by the Q-value, i.e., the expected long term cumulative rewards, we design the following auxiliary task. Let p indicate the probability that $\Delta Q(o_i) = Q(\langle o_i, a_i^C \rangle, \langle \vec{o}_{-i}, \vec{a}_{-i}^C \rangle) - Q(\langle o_i, a_i^I \rangle, \langle \vec{o}_{-i}, \vec{a}_{-i}^C \rangle)$ is larger than T, where a_i^C is the action generated based on communication, a_i^I is the action generated independently (i.e., without communication), and T is a threshold controlling how many messages should be pruned. In this setting, the label of this auxiliary task can be formulated as:

$$Y(o_i) = \mathbb{I}(\Delta Q(o_i) > T) = \mathbb{I}(Q(\langle o_i, a_i^C \rangle, \langle \vec{o}_{-i}, \vec{a}_{-i}^C \rangle) - Q(\langle o_i, a_i^I \rangle, \langle \vec{o}_{-i}, \vec{a}_{-i}^C \rangle) > T) \quad (8)$$

where $\mathbb{I}$ is the indicator function. Then we can train p by minimizing the following loss function:

$$L_{\theta_{op}}(o_i) = -\mathbb{E}_{o_i}[\, Y(o_i) \log p(o_i|\theta_{op}) + (1 - Y(o_i)) \log(1 - p(o_i|\theta_{op}))\,] \quad (9)$$

where θ_{op} is the parameters between the observation o_i and the probability p as shown in Figure 2.

Equation 8 implies that only agent i changes the behaviors between a_i^C and a_i^I to calculate $\Delta Q(o_i)$, while other agents are fixed to send messages all the time (i.e., keep $\vec{a}_{-i}^C$ unchanged) during training. To make sure that all agents can be trained properly, we train each agent based on the above equations in a round-robin manner, namely, i cycles from 1 to N and to 1 again. From the perspective of optimization, our method can be analogous to coordinate descent [1] if we take the agents as coordinates, since we optimize a specific coordinate hyperplane while fixing other coordinates (i.e., optimize agent i while fixing other agents [2]).

The insight of Equation 9 is that if the label of the auxiliary task is $Y(o_i) = 1$, namely, a_i^C can really obtain at least T Q-values more than a_i^I, the network should try to generate a probability $p(o_i; \theta_{op})$ that is larger than 0.5 to encourage communication (recall that $g \leftarrow \mathbb{I}(p > 0.5)$). In other words, Gated-ACML discourages communication by pruning the messages that contribute smaller Q-values than the threshold T. Therefore, after the gating mechanism has been well-trained, it can prune less beneficial messages adaptively to control the message quantity around a desired threshold specified by T. This is our key contribution beyond previous communication methods.

Key Implementation. The training method relies on correct label of the auxiliary task, so we should provide suitable $Q(\langle o_i, a_i^C \rangle, \langle \vec{o}_{-i}, \vec{a}_{-i}^C \rangle)$, $Q(\langle o_i, a_i^I \rangle, \langle \vec{o}_{-i}, \vec{a}_{-i}^C \rangle)$ and T as indicated by Equation 8.

For the Q-values, we firstly set $g = 1$ (i.e., without message pruning) to train other components except for the GatingNet based on Equation (4 – 7). After the model is trained well, we can get approximately correct ActorNet and CriticNet. Then, for a specific o_i, the ActorNet can generate a_i^C and a_i^I when we set $g = 1$ and $g = 0$, respectively. Afterwards, taking o_i, o_{-i} as well as the generated a_i^C, a_i^I and $\vec{a}_{-i}^C$ as input, the CriticNet can estimate approximately correct Q-values $Q(\langle o_i, a_i^C \rangle, \langle \vec{o}_{-i}, \vec{a}_{-i}^C \rangle)$ and $Q(\langle o_i, a_i^I \rangle, \langle \vec{o}_{-i}, \vec{a}_{-i}^C \rangle)$.

For the threshold T, we propose two methods to set a fixed T and a dynamic T, respectively. For a fixed T, we firstly sort the $\Delta Q(o_i)$ of the latest K observations o_i encountered during training, resulting in a sorted list of $\Delta Q(o_i)$, which is

[1] https://en.wikipedia.org/wiki/Coordinate_descent

[2] The assumption that $\vec{a}_{-i}^C$ keeps unchanged may be inconsistent with the reality because other agents j may take $\vec{a}_j^I$ instead of $\vec{a}_j^C$. Nevertheless, it is a lower bound case of our optimization objective. In practice, coordinate descent can optimize this lower bound with high efficiency in huge-scale multi-agent setting (Nesterov 2012).

denoted as $L_{\Delta Q(o_i)}$. Then, we set T by splitting $L_{\Delta Q(o_i)}$ in terms of the index. For example, if we want to prune $T_m\%$ messages, we set $T = L_{\Delta Q(o_i)}[K \times T_m\%]$. We do not split $L_{\Delta Q(o_i)}$ in terms of the value because $\Delta Q(o_i)$ usually has a non-uniform distribution. The advantage of a fixed T is that the actual number of the pruned messages is ensured to be close to the desired $T_m\%$. Besides, this method is friendly to a large K.

To calculate the dynamic T, we adopt the exponential moving average technique to update T as follows:

$$T_t = (1-\beta)T_{t-1} + \beta(\ Q_t(\langle o_i, a_i^C\rangle, \langle \vec{o}_{-i}, \vec{a}_{-i}^C\rangle) \\ -Q_t(\langle o_i, a_i^I\rangle, \langle \vec{o}_{-i}, \vec{a}_{-i}^C\rangle)\ ) \quad (10)$$

where β is a coefficient for discounting older T, and the subscript t represents the training timestep. We test some β in $[0.6, 0.9]$, and they all work well. The advantage of a dynamic T is that $Y(o_i)$ becomes an adaptive training label even for the same observation o_i. This is very important for the dynamically changing environments because T and $Y(o_i)$ can quickly adapt to these environments.

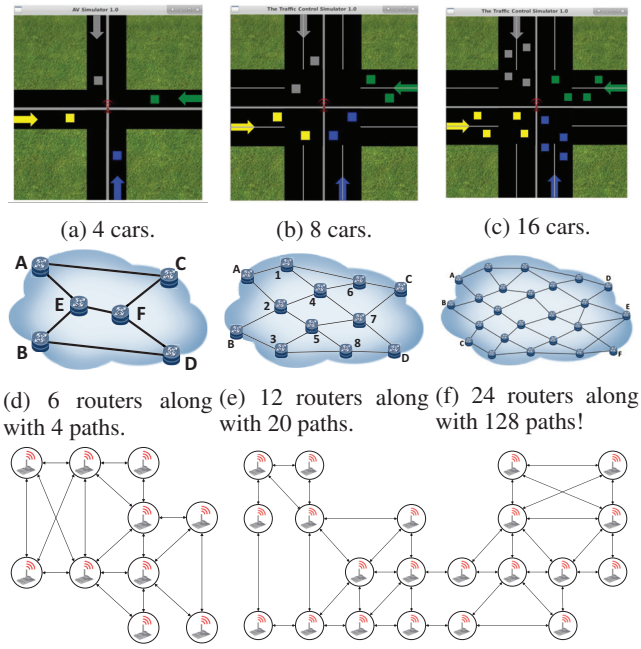
Apply Gating to Previous Methods

Although we introduce the proposed gating mechanism (i.e., the message pruning method) based on the basic ACML model, it is not specifically tailored to any specific DRL architecture. There are two directions to extend our method. On the one hand, we can apply the gating mechanism to previous methods such as CommNet and AMP, so that the resulting Gated-CommNet and Gated-AMP can also be applied in limited bandwidth setting. On the other hand, the basic Gated-ACML adopts a fully-connected MessageCoordinatorNet to process the received messages, so we can improve the fully-connected network with more advanced networks, such as the mean network in CommNet, the attention network in AMP and the BiRNN network in BiCNet (Peng et al. 2017). We mainly explore the former extension in the experiments because we focus on message pruning before sending the message rather than message processing after receiving the message.

Experimental Evaluations

Environments

Traffic Control. As shown in Figure 3a, the cars are driving on the road. The car collision occurs when the locations of two cars are overlapped, but it does not affect the simulation except for the reward these cars receiving. The cars are controlled by our method, and they try to learn a good driving policy to cooperatively drive through the junction. The simulation is terminated after 100 steps or when all cars successfully exit the junction. For each car, the ***observation*** encodes its current location and assigned route number. The ***action*** is a real number $a \in (0, 1)$, which indicates how far to move ahead the car on its route. For the reward, each car gets a reward $r_{time}^{\tau} = -0.1\tau$ at each timestep to discourage a traffic jam, where τ is the total timesteps since the car appeared in the simulator; in addition, a car collision incurs a penalty $r_{coll} = -10.0$ on the received reward, while an additional reward $r_{exit} = 30.0$ will be given if the car successfully exits the junction; thus, the ***total reward*** at time t is: $r(t) = \sum_{i=1}^{N^t} r_{time}^{\tau_i} + C^t r_{coll} + E^t r_{exit}$, where N^t, C^t and E^t are the numbers of cars present, car collisions and cars exiting at timestep t, respectively.



(a) 4 cars. (b) 8 cars. (c) 16 cars.

(d) 6 routers along with 4 paths. (e) 12 routers along with 20 paths. (f) 24 routers along with 128 paths!

(g) 10 access points along with 35 wireless links. (h) 21 access points along with 70 wireless links!

Figure 3: The evaluation tasks. (a-c) The simple, moderate and complex traffic control scenarios. (d-f) The simple, moderate and complex packet routing scenarios. (g-h) The simple and complex access point configuration scenarios.

Packet Routing. As shown in Figure 3d, the edge router has an aggregated flow that should be transmitted to other edge routers through available paths (e.g., B is set to transmit flow to D, and the available paths are $BEFD$ and BD). Each path is made up of several links, and each link has a *link utilization*, which equals to the ratio of the current flow on this link to the maximum capacity of this link. The routers are controlled by our algorithms, and they try to learn a good flow transmission policy to minimize the *Maximum Link Utilization in the whole network (MLU)*. The intuition behind this objective is that high link utilization is undesirable for dealing with bursty traffic. For each router, the ***observation*** includes the flow demands in its buffers, the estimated link utilization of its direct links during the last ten steps, the average link utilization of its direct links during the last control cycle, and the latest action taken by the router. The ***action*** is the flow rate assigned to each available path. The ***reward*** is $1 - MLU$ because we want to minimize *MLU*

Wifi Access Point Configuration. The cornerstone of any wireless network is the sensor and access point (AP). The primary job of an AP is to broadcast a wireless signal that sensors can detect and tune into. It is tedious to config-

Table 1: The average results of 10 experiments on traffic control tasks. For models named as Gated-*, dynamic thresholds with $\beta = 0.8$ are used. The "delay" indicates the timesteps to complete the simulation. The "# C" indicates the number of collisions.

	Simple Traffic Control (4 cars)				**Moderate Traffic Control (8 cars)**				**Complex Traffic Control (16 cars)**			
	reward	delay	# C	message	reward	delay	# C	message	reward	delay	# C	message
CommNet	-129.2	45.6	2.3	100.0%	-573.4	61.6	6.8	100.0%	-4278.8	82.1	16.5	100.0%
AMP	-97.1	41.8	**1.2**	100.0%	-1950.6	100.0	**0.9**	100.0%	-6391.5	100.0	**2.1**	100.0%
ACML	**-37.6**	**31.2**	1.9	100.0%	**-103.5**	**43.1**	1.7	100.0%	**-2824.9**	**66.4**	7.2	100.0%
ACML-mean	-32.9	29.3	2.0	100.0%	-96.1	40.2	3.9	100.0%	-2661.5	61.4	9.4	100.0%
ACML-attention	-24.5	28.8	1.6	100.0%	-91.8	39.6	1.3	100.0%	-2359.7	52.9	6.8	100.0%
Gated-CommNet	-88.4	41.1	1.7	22.8%	-476.7	47.2	2.5	34.3%	-3529.4	73.0	15.7	29.3%
Gated-AMP	-59.9	37.1	**1.3**	**18.6%**	-988.6	65.3	**1.6**	**23.7%**	-2870.5	65.5	**7.9**	**19.1%**
Gated-ACML	**-14.6**	**21.0**	2.4	23.9%	**-69.4**	**32.3**	2.1	29.8%	**-2101.1**	**48.6**	11.3	25.8%
ATOC	-19.7	25.9	1.9	37.3%	-77.5	35.6	2.4	63.7%	-2481.2	54.8	14.9	*112.5%*

Table 2: The average results of 10 experiments on packet routing and wifi access point configuration tasks. For models named as Gated-*, we adopt dynamic thresholds with $\beta = 0.8$. The "WAPC." is the abbreviation of Wifi Access Point Configuration.

	Simple Routing		**Moderate Routing**		**Complex Routing**		**Simple WAPC.**		**Complex WAPC.**	
	reward	message	reward	message	reward	message	reward	message	reward	message
CommNet	0.264	100.0%	0.164	100.0%	-	100.0%	0.652	100.0%	0.441	100.0%
AMP	0.266	100.0%	0.185	100.0%	-	100.0%	0.627	100.0%	0.418	100.0%
ACML	**0.317**	100.0%	**0.263**	100.0%	-	100.0%	**0.665**	100.0%	**0.480**	100.0%
ACML-mean	0.321	100.0%	0.267	100.0%	-	100.0%	0.673	100.0%	0.493	100.0%
ACML-attention	0.329	100.0%	0.271	100.0%	-	100.0%	0.689	100.0%	0.506	100.0%
Gated-CommNet	0.232	35.2%	0.144	**21.7%**	-	19.8%	0.595	53.1%	0.386	41.8%
Gated-AMP	0.241	46.7%	0.170	35.0%	-	81.7%	0.539	57.2%	0.350	**32.3%**
Gated-ACML	0.288	**33.6%**	**0.239**	27.9%	-	22.6%	**0.610**	**41.9%**	**0.411**	37.7%
ATOC	**0.297**	73.7%	0.102	*104.6%*	-	*326.1%*	0.418	*136.5%*	0.231	*393.4%*

ure the power of APs, and the AP behaviors differ greatly with various scenarios. The current optimization is highly depending on human expertise, which fails to handle dynamic environments. In the tasks shown in Figure 3g, the APs are controlled by our method, and they try to learn a good power configuration policy to maximize the Received Signal Strength Indicator(RSSI). In general, larger RSSI indicates better signal quality. For each AP, the ***observation*** includes radio frequency, bandwidth, the rate of package loss, the number of band, the current number of users in one specific band, the number of download bytes in ten seconds, the upload coordinate speed (Mbps), the download coordinate speed and the latency. The ***action*** is the power value ranging from 10.0 to 30.0. The ***reward*** is RSSI and the goal is to maximize accumulated RSSI.

Results

Results without Message Pruning. We firstly evaluate the ACML model. For traffic control, ACML outperforms CommNet and AMP as shown by the upper part of Table 1. The reason is that ACML has found better tradeoffs between small delay and small collision. Please recall the settings: the reward penalty of a great delay is much larger than that of a collision. We notice that ACML drives the cars with a relatively larger speed to complete the simulation with the smallest delay (i.e., 31.2, 43.1 and 66.4 for three scenarios, respectively). Besides, although its collision is not the smallest, ACML manages to achieve relatively smaller collisions (i.e., 1.9, 1.7 and 7.2 for three scenarios), since avoiding collision is also in favor of the total reward. In contrast, AMP has a great delay and CommNet has a great collision, which makes their total rewards unsatisfactory.

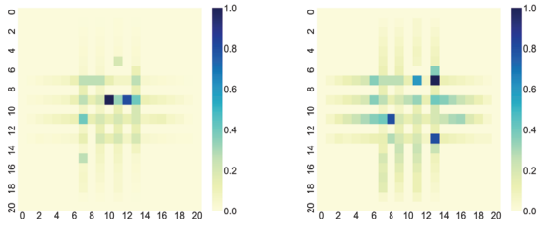
We further extend ACML with the average message operation in CommNet and the attention message operation in AMP. The resulting models are named as ACML-mean and ACML-attention, respectively. Both ACML-mean and ACML-attention improve ACML by a clear margin, and they perform much better than CommNet and AMP as shown in Table 1. The results imply that ACML is a general model, and it can be easily extended with advanced message processing operations to achieve better results.

In addition, the results of packet routing and wifi access point configuration shown at the upper part of Table 2 can be analyzed similarly to demonstrate that the performance of ACML is consistent in several multi-agent scenarios.

Results of Dynamic Threshold Message Pruning. For packet routing and access point configuration scenarios, as shown at the lower part of Table 2, Gated-ACML can prune a lot of messages (e.g., about $70\% \approx 1 - 33.6\%$) with little damage to the reward (e.g., about $10\% \approx 1 - 0.288/0.317$). It implies that the pruned messages are usually not beneficial enough to performance, which approves the effectiveness of our design. However, none of the tested methods can handle the complex routing task! In this case, they cannot provide correct label for the auxiliary loss function in Equation 9, so the message quantity is almost random. We leave the results here to show the limitation of our method and others.

For traffic control scenarios, as shown at the lower part

of Table 1, Gated-ACML emits much fewer messages than ACML, which is expected. What surprises us is that Gated-ACML even obtains more rewards than ACML. This is very appealing in the real-world applications, since we can get more rewards with less message exchange and less resource consumption. We find that this phenomenon is closely related to message redundance. Figure 4 shows the message distribution [3] generated by Gated-ACML. As can be observed, the messages are mostly distributed near the junction. We hypothesize that the messages near the junction are critical, while the messages far away from the junction are redundant. Because Gated-ACML has pruned the redundant messages, the agents are rarely disturbed, so they are more adaptive: their speed gradually decreases as they are approaching the junction, while their speed gradually increases as they are moving away from the junction. In contrast, ACML does not have the ability to prune the redundant messages, so the agents are badly disturbed, and they are very conservative: the cars are too cautious to drive with a large speed even when they are far away from the junction. Consequently, the delay of ACML is larger than that of Gated-ACML as shown in Table 1. Moreover, the message distribution is consistent with human experience. Because car collision easily occurs at the junction, keeping these messages is very important for avoiding car collision.



(a) For moderate traffic. (b) For complex traffic.

Figure 4: The message distribution of traffic control tasks.

For all evaluation scenarios, Gated-CommNet and Gated-AMP reveal the same trend as Gated-ACML. It means that the gating mechanism with dynamic threshold is generally applicable to several DRL methods and multi-agent tasks.

We also modify ATOC to make it suitable for continuous action and heterogeneous agents. The results are shown at the last row of Table 1 and 2. ATOC works well in traffic control tasks, but performs badly in routing and wifi tasks. The main reason is that ATOC decomposes all agents into small agent groups by distance between agents. It is in favor of tasks defined on 2D plane (e.g., traffic control). However, the routers or APs are entangled with each other by other factors but not distance between them. Critically, the results show that ATOC could hardly be used in limited bandwidth setting: it emits much more messages than our methods (e.g., about tenfold messages of ours in complex wifi scenario). This is because agents in ATOC can belong to many agent groups, and thus generate many messages. The results show the urgent demand of principled message pruning methods.

[3] We firstly discretize the 2D continuous plane into a 21-by-21 discrete plane. Then, we count the number of messages in each discrete cell. Finally, we normalize these numbers.

Table 3: The results of Gated-ACML in packet routing scenarios. We adopt a fixed threshold $T = L_{\Delta Q_{(o_i)}}[K \times T_m\%]$.

$T_m\%$	Simple Routing *pruned* message	Simple Routing reward *decrease*	Moderate Routing *pruned* message	Moderate Routing reward *decrease*
10.0%	12.19%	**-8.46%**	11.60%	**-7.03%**
20.0%	24.07%	**-13.59%**	22.77%	**-12.14%**
30.0%	27.65%	**-4.88%**	29.98%	**-3.25%**
70.0%	66.73%	9.27%	68.54%	10.06%
80.0%	**79.14%**	**14.01%**	76.81%	13.25%
90.0%	87.22%	18.60%	85.11%	19.50%
100.0%	100.00%	59.35%	100.00%	65.42%

Results of Fixed Threshold Message Pruning. The results are shown in Table 3. It can be noticed that the number of pruned messages is close to the desired $T_m\%$ for both routing scenarios. It is the advantage of fixed threshold as mentioned before, but it has not been implemented by any previous methods as far as we know. In addition, as shown at the upper part of the table, when the quantity of *pruned* messages is smaller than 30%, the reward *decrease* is a negative value, which means that the reward is increased actually. This is because some messages are extremely noisy in the routing system, and pruning these messages will be beneficial for the performance. Please note that similar phenomena have been observed in traffic control scenarios. However, as we are pruning more messages, the reward decrease is becoming larger and larger as shown at the lower part of the table, especially when all messages are pruned. It indicates that the remaining messages are very critical for maintaining the performance, and Gated-ACML has learnt to share these important messages with other agents. In contrast, even if a large number of messages (e.g., 79.14%) have been pruned, the reward decrease is not so great (e.g., 14.01%). Again, it implies that the pruned messages are less beneficial to the performance, and Gated-ACML with fixed threshold has mastered an effective strategy to prune these messages.

Conclusions

We have proposed a gating mechanism, which consists of several key designs like auxiliary task with appropriate training signal, dynamic and fixed thresholds, to address the limited bandwidth that has been largely ignored by previous DRL methods. The gating mechanism prunes less beneficial messages in an adaptive manner, so that the performance can be maintained or even improved with much fewer messages. Furthermore, it is applicable to several previous methods and multi-agent scenarios with good performance. To the best of our knowledge, it is the first method to achieve these in the multi-agent reinforcement learning community.

Acknowledgments

The authors would like to thank the anonymous reviewers for their comments. This work was supported by the Na-

tional Natural Science Foundation of China under Grant No.61872397. The contact author is Zhen Xiao.

References

Becker, R.; Carlin, A.; Lesser, V.; and Zilberstein, S. 2009. Analyzing myopic approaches for multi-agent communication. *Computational Intelligence* 25(1):31–50.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of markov decision processes. *Mathematics of operations research* 27(4):819–840.

Chu, X., and Ye, H. 2017. Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1710.00336*.

Foerster, J.; Assael, I. A.; de Freitas, N.; and Whiteson, S. 2016. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2137–2145.

Hernandez-Leal, P.; Kaisers, M.; Baarslag, T.; and de Cote, E. M. 2017. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks. In *Advances in neural information processing systems*, 4107–4115.

Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2016. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

Jiang, J., and Lu, Z. 2018. Learning attentional communication for multi-agent cooperation. *Advances in neural information processing systems*.

Kilinc, O., and Montana, G. 2018. Multi-agent deep reinforcement learning with extremely noisy observations. *arXiv preprint arXiv:1812.00922*.

Kim, D.; Moon, S.; Hostallero, D.; Kang, W. J.; Lee, T.; Son, K.; and Yi, Y. 2019. Learning to schedule communication in multi-agent reinforcement learning. *International Conference on Learning Representations*.

Kim, W.; Cho, M.; and Sung, Y. 2019. Message-dropout: An efficient training method for multi-agent deep reinforcement learning. *Thirty-Third AAAI Conference on Artificial Intelligence*.

Kong, X.; Xin, B.; Liu, F.; and Wang, Y. 2017. Revisiting the master-slave architecture in multi-agent deep reinforcement learning. *arXiv preprint arXiv:1712.07305*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, O. P.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, 6379–6390.

Makhzani, A., and Frey, B. J. 2015. Winner-take-all autoencoders. In *Advances in Neural Information Processing Systems*, 2791–2799.

Mao, H.; Gong, Z.; Ni, Y.; and Xiao, Z. 2017. Accnet: Actor-coordinator-critic net for "learning-to-communicate" with deep multi-agent reinforcement learning. *arXiv preprint arXiv:1706.03235*.

Mao, H.; Zhang, Z.; Xiao, Z.; and Gong, Z. 2019. Modelling the dynamic joint policy of teammates with attention multi-agent ddpg. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 1108–1116. International Foundation for Autonomous Agents and Multiagent Systems.

Nesterov, Y. 2012. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization* 22(2):341–362.

Peng, P.; Yuan, Q.; Wen, Y.; Yang, Y.; Tang, Z.; Long, H.; and Wang, J. 2017. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*.

Peng, Z.; Zhang, L.; and Luo, T. 2018. Learning to communicate via supervised attentional message processing. In *Proceedings of the 31st International Conference on Computer Animation and Social Agents*, 11–16. ACM.

Roth, M.; Simmons, R.; and Veloso, M. 2005. Reasoning about joint beliefs for execution-time communication decisions. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 786–793. ACM.

Roth, M.; Simmons, R.; and Veloso, M. 2006. What to communicate? execution-time decision in multi-agent pomdps. In *Distributed Autonomous Robotic Systems 7*. Springer. 177–186.

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic policy gradient algorithms. In *ICML*.

Singh, A.; Jain, T.; and Sukhbaatar, S. 2019. Learning when to communicate at scale in multiagent cooperative and competitive tasks. *International Conference on Learning Representations*.

Sukhbaatar, S.; Fergus, R.; et al. 2016. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, 2244–2252.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

Wu, F.; Zilberstein, S.; and Chen, X. 2011. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence* 175(2):487–511.

Zhang, C., and Lesser, V. 2013. Coordinating multi-agent reinforcement learning with limited communication. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 1101–1108. International Foundation for Autonomous Agents and Multiagent Systems.