A REVIEW OF COOPERATIVE MULTI-AGENT DEEP REINFORCEMENT LEARNING

Afshin Oroojlooy and Davood Hajinezhad
{afshin.oroojlooy, davood.hajinezhad}@sas.com
SAS Institute Inc., Cary, NC, USA

ABSTRACT

Deep Reinforcement Learning has made significant progress in multi-agent systems in recent years. In this review article, we have focused on presenting recent approaches on Multi-Agent Reinforcement Learning (MARL) algorithms. In particular, we have focused on five common approaches on modeling and solving cooperative multi-agent reinforcement learning problems: (I) independent learners, (II) fully observable critic, (III) value function factorization, (IV) consensus, and (IV) learn to communicate. First, we elaborate on each of these methods, possible challenges, and how these challenges were mitigated in the relevant papers. If applicable, we further make a connection among different papers in each category. Next, we cover some new emerging research areas in MARL along with the relevant recent papers. Due to the recent success of MARL in real-world applications, we assign a section to provide a review of these applications and corresponding articles. Also, a list of available environments for MARL research is provided in this survey. Finally, the paper is concluded with proposals on the possible research directions.

Keywords: Reinforcement Learning, Multi-agent systems, Cooperative.

1 Introduction

Multi-Agent Reinforcement Learning (MARL) algorithms are dealing with systems consisting of several agents (robots, machines, cars, etc.) which are interacting within a common environment. Each agent makes a decision in each time-step and works along with the other agent(s) to achieve an individual predetermined goal. The goal of MARL algorithms is to learn a policy for each agent such that all agents together achieve the goal of the system. Particularly, the agents are learnable units that aim to learn an optimal policy on the fly to maximize the long-term cumulative

discounted reward through the interaction with the environment. Due to the complexities of the environments or the combinatorial nature of the problem, training the agents is typically a challenging task and several problems which MARL deals with them are categorized as NP-Hard problems, e.g. manufacturing scheduling (Gabel and Riedmiller 2007, Dittrich and Fohlmeister 2020), vehicle routing problem (Silva et al. 2019, Zhang et al. 2020b), some multi-agent games (Bard et al. 2020) are only a few examples to mention.

With the motivation of recent success on deep reinforcement learning (RL)—super-human level control on Atari games (Mnih et al. 2015), mastering the game of Go (Silver et al. 2016), chess (Silver et al. 2017), robotic (Kober et al. 2013), health care planning (Liu et al. 2017), power grid (Glavic et al. 2017), routing (Nazari et al. 2018), and inventory optimization (Oroojlooyjadid et al. 0)—on one hand, and the importance of multi-agent system (Wang et al. 2016b, Leibo et al. 2017) on the other hand, several researches have been focused on deep MARL. One naive approach to solve these problems is to convert the problem to a single-agent problem and make the decision for all the agents using a centralized controller. However, in this approach, the number of actions typically exponentially increases, which makes the problem intractable. Besides, each agent needs to send its local information to the central controller and with increasing the number of agents, this approach becomes very expensive or impossible. In addition to the communication cost, this approach is vulnerable to the presence of the central unit and any incident that results in the loss of the network. Moreover, usually in multi-agent problems, each agent accesses only some local information, and due to privacy issues, they may not be allowed to share their information with the other agents.

There are several properties of the system that is important in modeling a multi-agent system: (i) centralized or decentralized control, (ii) fully or partially observable environment, (iii) cooperative or competitive environment. Within a centralized controller, a central unit takes the decision for each agent in each time step. On the other hand, in the decentralized system, each agent takes a decision for itself. Also, the agents might cooperate to achieve a common goal, e.g. a group of robots who want to identify a source or they might compete with each other to maximize their own reward, e.g. the players in different teams of a game. In each of these cases, the agent might be able to access the whole information and the sensory observation (if any) of the other agents, or on the other hand, each agent might be able to observe only its local information. In this paper, we have focused on the decentralized problems with the cooperative goal, and most of the relevant papers with either full or partial observability are reviewed. Note that Weiß (1995), Matignon et al. (2012), Buşoniu et al. (2010), Bu et al. (2008) provide reviews on cooperative games and general MARL algorithms published till 2012. Also, Da Silva and Costa (2019) provide a survey over the utilization of transfer learning in MARL. Zhang et al. (2019b)

provide a comprehensive overview on the theoretical results, convergence, and complexity analysis of MARL algorithms on Markov/stochastic games and extensive-form games on competitive, cooperative, and mixed environments. In the cooperative setting, they have mostly focused on the theory of consensus and policy evaluation. In this paper, we did not limit ourselves to a given branch of cooperative MARL such as consensus, and we tried to cover most of the recent works on the cooperative Deep MARL. In Nguyen et al. (2020), a review is provided for MARL, where the focus is on deep MARL from the following perspectives: non-stationarity, partial observability, continuous state and action spaces, training schemes, and transfer learning. We provide a comprehensive overview of current research directions on the cooperative MARL under six categories, and we tried our best to unify all papers through a single notation. Since the problems that MARL algorithms deal with, usually include large state/action spaces and, the classical tabular RL algorithms are not efficient to solve them, we mostly focus on the approximated cooperative MARL algorithms.

The rest of the paper is organized as the following: in section 2, we discuss the taxonomy and organization of the MARL algorithms we reviewed. in Section 3.1 we briefly explain the single agent RL problem and some of its components. Then the multi-agent formulation is represented and some of the main challenges of multi-agent environment from the RL viewpoint is described in Section 3.2. Section 4 explains the independent Q-learner type algorithm, Section 5 reviews the papers with a fully observable critic model, Section 6 includes the value decomposition papers, Section 7 explains the consensus approach, Section 8 reviews the learn-to-communicate approach, Section 9 explains some of the emerging research directions, Section 10 provides some applications of the multi-agent problems and MARL algorithms in real-world, Section 11 very briefly mentions some of the available multi-agent environments, and finally Section 13 concludes the paper.

2 Taxonomy

In this section, we provide a high-level explanation about the taxonomy and the angle we looked at the MARL.

A simple approach to extend single-agent RL algorithms to multi-agent algorithms is to consider each agent as an independent learner. In this setting, the other agents' actions would be treated as part of the environment. This idea was formalized in Tan (1993) for the first time where the Q-learning algorithm was extended for this problem which is called *independent Q-Learning (IQL)*. The biggest challenge for IQL is *non-stationarity*, as the other agents' actions toward local interests will impact the environment transitions.

To address the non-stationarity issue, one strategy is to assume that all critics observe the same state (global state) and actions of all agents, which we call it *fully observable critic* model. In this setting, the critic model learns the true state-value and when is paired with the actor can be used toward finding the optimal policy. When the reward is shared among all agents, only one critic model is required; however, in the case of private local reward, each agent needs to train a local critic model for itself.

Consider multi-agent problem settings, where the agents aim to maximize a single joint reward or assume it is possible to reduce the multi-agent problem into a single agent problem. A usual RL algorithm may fail to find the global optimal solution in this simplified setting. The main reason is that in such settings, the agents do not know the true share of the reward for their actions and as a result, some agents may get lazy over time (Sunehag et al. 2018). In addition, exploration among the agents with poor policy aggravates the team reward, thus restrain the agents with good policies to proceed toward optimal policy. One idea to remedy this issue is to figure out the share of each individual agent into the global reward. This solution is formalized as the *Value Function Factorization*, where a decomposition function is learned from the global reward.

Another drawback in the fully observable critic paradigm is the communication cost. In particular, with increasing the number of learning agents, it might be a prohibitive task to collect all state/action information in a critic due to communication bandwidth and memory limitations. The same issue occurs for the actor when the observation and actions are being shared. Therefore, the question is how to address this communication issue and change the topology such that the local agents can cooperate and communicate in learning optimal policy. The key idea is to put the learning agents on a sparsely connected network, where each agent can communicate with a small subset of agents. Then the agents seek an optimal solution under the constraint that this solution is in *consensus* with its neighbors. Through communications, eventually, the whole network reaches a unanimous policy which results in the optimal policy.

For the consensus algorithms or the fully observable critic model, it is assumed that the agent can send their observation, action, or rewards to the other agents and the hope is that they can learn the optimal policy by having that information from other agents. But, one does not know the true information that is required for the agent to learn the optimal policy. In other words, the agent might be able to learn the optimal policy by sending and receiving a simple message instead of sending and receiving the whole observation, action, and reward information. So, another line of research which is called, *Learn to Communicate*, allows the agents to learn what to send, when to send, and send that to which agents. In particular, besides the action to the environment, the agents learn another action called communication action.

Despite the fact that the above taxonomy covers a big portion of MARL, there are still some algorithms that either do not fit in any of these categories or are at the intersection of a few of them. In this review, we discuss some of these algorithms too.

In Table 1, a summary of different categories are presented. Notice that in the third column we provide only a few representative references. More papers will be discussed in the following sections.

Categories	**Description**	**References**
Independent Learners	Each learning agent is an independent learner without considering its influence into the environment of other agents	Tan (1993), Fuji et al. (2018), Tampuu et al. (2017), Foerster et al. (2017)
Fully Observable Critic	All critic models observe the same state (global state) in order to address the non-stationarity issue	Lowe et al. (2017), Ryu et al. (2018), Mao et al. (2019)
Value Function Factorization	Distinguish the share of each individual agent into the global reward to avoid having lazy agents	Sunehag et al. (2018), Rashid et al. (2018), Son et al. (2019)
Consensus	To avoid communication overhead, agent communicate through sparse communication network and try to reach consensus	Macua et al. (2018), Zhang et al. (2018c), Cassano et al. (2021)
Learn to Communicate	To improve the communication efficiency, we allow the agents to learn what to send, when to send, and send that to which agents	Foerster et al. (2016), Jorge et al. (2016), Mordatch and Abbeel (2018b)

Table 1: A summary of the MARL taxonomy in this paper.

3 Background, Single-Agent RL Formulation, and Multi-Agent RL Notation

In this section, we first go over some background of reinforcement learning and the common approaches to solve that for the single-agent problem in Section 3.1. Then, in Section 3.2 we introduce the notation and definition of the Multi-agent sequential decision-making problem and the challenges that MARL algorithms need to address.

3.1 Single Agent RL

RL considers a sequential decision making problem in which an agent interacts with an environment. The agent at time period t observes state $s_t \in \mathcal{S}$ in which $\mathcal{S}$ is the state space, takes action $a_t \in \mathcal{A}(s_t)$ where $\mathcal{A}(s_t)$ is the valid action space for state s_t, and executes that in the environment to receive reward $r(s_t, a_t, s_{t+1}) \in \mathbb{R}$ and then transfer to the new state $s_{t+1} \in \mathcal{S}$. The process runs for the stochastic T time-steps, where an episode ends. Markov Decision Process (MDP) provides a framework to characterize and study this problem where the agent has full observability of the state.

The goal of the agent in an MDP is to determine a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, a mapping of the state space $\mathcal{S}$ to the action space $\mathcal{A}$ [1], that maximizes the long-term cumulative discounted rewards:

$$J = \mathbb{E}_{\pi,s_0}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) | a_t = \pi(.|s_t)\right], \tag{1}$$

where, $\gamma \in [0, 1]$ is the discounting factor. Accordingly, the value function starting from state s and following policy π denoted by $V^\pi(s)$ is given by

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) | a_t \sim \pi(.|s_t), s_0 = s\right], \tag{2}$$

and given action a, the Q-value is defined as

$$Q^\pi(s, a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) | a_t \sim \pi(.|s_t), s_0 = s, a_0 = a\right], \tag{3}$$

Given a known state transition probability distribution $p(s_{t+1}|s_t, a_t)$ and reward matrix $r(s_t, a_t)$, Bellman (Bellman 1957) showed that the following equation holds for all state s_t at any time step t, including the optimal values too:

$$V^\pi(s_t) = \sum_{a \in \mathcal{A}(s_t)} \pi(a|s_t) \sum_{s' \in \mathcal{S}} p(s'|s_t, a) \left[r(s_t, a) + \gamma V^\pi(s')\right], \tag{4}$$

where s' denotes the successor state of s_t; which will be used interchangeably with s_{t+1} throughout the paper. Thorough maximizing over the actions, the optimal state-value and optimal policy can be obtained:

$$V^{\pi^*}(s_t) = \max_a \sum_{s'} p(s'|s_t, a) \left[r(s_t, a) + \gamma V^{\pi^*}(s')\right]. \tag{5}$$

Similarly, the optimal Q-value for each state-action can be obtained by:

$$Q^{\pi^*}(s_t, a_t) = \sum_{s'} p(s'|s_t, a_t) \left[r(s_t, a_t) + \gamma \max_{a'} Q^{\pi^*}(s', a')\right]. \tag{6}$$

One can obtain an optimal policy π^* through learning directly $Q^{\pi^*}(s_t, a_t)$. The relevant methods are called *value-based* methods. However, in the real world usually the knowledge of the environment i.e., $p(s'|s_t, a_t)$ is not available and one cannot obtain the optimal policy using (5) or (6). In order to address this issue, learning the state-value, or the Q-value, through sampling has been a common practice. This approximation requires only samples of state, action, and reward that are obtained from the interaction with the environment. In the earlier approaches, the value for each state/state-action was stored in a table and was updated through an iterative approach. The *value*

[1] A policy can be deterministic or stochastic. Action a_t is the direct outcome of a deterministic policy, i.e., $a_t = \pi(s_t)$. In stochastic policies, the outcome of the policy is the probability of choosing each of the actions, i.e., $\pi(a|s_t) = Pr(a_t = a|s_t)$, and then an additional method is required to choose an action among them. For example, a greedy method chooses the action with the highest probability. In this paper, when we refer to an action resulted from a policy we mostly use the notation for the stochastic policy, $a \sim \pi(.|s)$.

iteration and *policy iteration* are two famous algorithms in this category that can attain the optimal policy. Although, these approaches are not practical for tasks with enormous state/action spaces due to the curse of dimensionality. This issue can be mitigated through *function approximation*, in which parameters of a function need to be learned by utilizing supervised learning approaches. The function approximator with parameters θ results in policy $\pi_\theta(a|s)$. The function approximator with parameters θ can be a simple linear regression model or a deep neural network. Given the function approximator, the goal of an RL algorithm can be re-written to maximize the utility function,

$$J(\theta) = \mathbb{E}_{a\sim\pi_\theta(.|s),s\sim\rho_{\pi_\theta}} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t; \theta), \tag{7}$$

where the expectation is taken over the actions and the distribution for state occupancy.

In a different class of approaches, called *policy-based*, the policy is directly learned which determines the probability of choosing an action for a given state. In either of these approaches, the goal is to find parameters θ to maximize utility function $J(\theta)$ through learning with sampling. We explain a brief overview of the value-based and policy-based approaches in Sections 3.1.1 and 3.1.2, respectively. Note that there is a large body of literature on the single-agent RL algorithms, and we only reviewed those algorithms which are actively being used in multi-agent literature too. So, to keep the coherency, we prefer not to explore advanced algorithms like soft actor-critic (SAC) (Haarnoja et al. 2018) and TD3 (Fujimoto et al. 2018), which are not so common in MARL literature. For more details about other single-agent RL algorithms see Li (2017) and Sutton and Barto (2018).

For all the above notations and descriptions, we assume full observability of the environment. However, in cases that the agent accesses only some part of the state, it can be categorized as a decision-making problem with partial observability. In such circumstances, MDP can no longer be used to model the problem; instead, partially observable MDPs (POMDP) is introduced as the modeling framework. This situation happens in a lot of multi-agent systems and will be discussed throughout the paper.

3.1.1 Value Approximation

In value approximation, the goal is to learn a function to estimate $V(s)$ or $Q(s, a)$. It is showed that with a large enough number of observations, a linear function approximation converges to a local optimal (Bertsekas and Tsitsiklis 1996, Sutton et al. 2000). Nevertheless, the linear function approximators are not powerful enough to capture all complexities of a complex environment. To address this issue, there has been a tremendous amount of research on non-linear function approximators and especially neural networks in recent years (Mnih et al. 2013, Bhatnagar et al. 2009, Gao et al. 2019). Among the recent works on the value-based methods, the deep Q-network (DQN)

algorithm (Mnih et al. 2015) attracted a lot of attention due to its human-level performance on the Atari-2600 games (Bellemare et al. 2013), in which it only observes the video of the game. DQN utilizes the experience replay (Lin 1992) and a moving target network to stabilize the training. The experience replay holds the previous observation tuple $(s_t, a_t, r_t, s_{t+1}, d_t)$ in which d_t determines if the episode ended with this observation. Then the approximator is trained by taking a random mini-batch from the experience replay. Utilizing the experience replay results in sample efficiency and stabilizing the training, since it breaks the temporal correlations among consecutive observations. The DQN algorithm uses a deep neural network to approximate the Q-value for each possible action $a \in \mathcal{A}$. The input of the network is a function of the state s_t, e.g., concatenation/average of the last four observed states. The original paper used a combination of convolutional neural network (CNN) and fully connected (FC) as the neural network approximator; although, it can be a linear or non-linear function approximator, like any combination of FC, CNN, or recurrent neural networks (RNN). The original DQN algorithm uses a function of state s_t as the input to a CNN and its output is the input to an FC neural network. This neural network with weights θ is then trained by taking a mini-batch of size m from the experience replay to minimize the following loss function:

$$L(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(y_i - Q(s_i, a_i; \theta)\right)^2, \tag{8}$$

$$y_i = \begin{cases} r_i, & d_t = \text{True}, \\ r_i + \gamma \max_{a'} Q(s'_i, a', \theta^-) & d_t = \text{False}, \end{cases} \tag{9}$$

where, θ^- is the weights of the *target network* which is updated by θ every C iterations. Later, new DQN-based approaches were proposed for solving RL problems. For example, inspired by Hasselt (2010) which proposed double Q-learning, the Double-DQN algorithm proposed in Van Hasselt et al. (2016) to alleviate the over-estimation issue of Q-values. Similarly, Dueling double DQN (Wang et al. 2016c) proposed learning a network with two heads to obtain the advantage value $A(s, a) = Q(s, a) - V(s)$ and $V(s)$ and use that to get the Q-values. In addition, another extension of the DQN algorithm is proposed by combining recurrent neural networks with DQN. DRQN (Hausknecht and Stone 2015) is a DQN algorithm which uses a Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) layer instead of a fully connected network and is applied to a partially observable environment. In all these variants, usually, the ϵ-Greedy algorithm is used to ensure the exploration. That is, in each time-step with a probability of ϵ the action is chosen randomly, and otherwise, it is selected greedily by taking an `argmax` over the Q-values for the state. Typically, the value of ϵ is annealed during the training. Choosing the hyper-parameters of the ϵ-greedy algorithm, target update frequency, and the experience replay can widely affect the speed and quality of the training. For example, it is shown that a large experience replay buffer

can negatively affect performance. See Zhang and Sutton (2017), Liu and Zou (2018), Fedus et al. (2020) for more details.

In the final policy, which is used for scoring, usually the ϵ is set to zero, which results in a deterministic policy. This mostly works well in practice; although, it might not be applicable to stochastic policies. To address this issue, softmax operator and Boltzmann softmax operator are added to DQN (Lipton et al. 2016, Pan et al. 2020) to get the probability of choosing each action,

$$\text{Boltzmann}\left(Q(s_t,a)\right) = \frac{e^{\beta Q(s_t,a)}}{\sum_{a\in\mathcal{A}(s_t)} e^{\beta Q(s_t,a)}}, \forall u \in \mathcal{A}(s_t),$$

in which β is the temperature parameter to control the rate of stochasticity of actions. To use the Boltzmann softmax operator also one needs to find a reasonable temperature parameter value and as a result cannot be used to find the general optimal stochastic policy. To see other extensions of the value-based algorithm and other exploration algorithms see Li (2017).

3.1.2 Policy Approximation

In the value-based methods, the key idea is learning the optimal value-function or the Q-function, and from there a greedy policy could be obtained. In another direction, one can parametrize the policy, make a utility function, and try to optimize this function over the policy parameter through a supervised learning process. This class of RL algorithms is called policy-based methods, which provides a probability distribution over actions. For example, in the policy gradient method, a stochastic policy by parameters $\theta \in \mathbb{R}^d$ is learned. First, we define

$$h(s_t, a_t; \theta) = \theta^T \phi(s_t, a_t),$$

where, $\phi(s_t, a_t) \in \mathbb{R}^d$ is called the *feature vector* of the state-action pair (s_t, a_t). Then, the stochastic policy can be obtained by:

$$\pi(a_t|s_t;\theta) = \frac{e^{h(s_t,a_t;\theta)}}{\sum_b e^{h(s_t,b;\theta)}},$$

which is the softmax function. The goal is to directly learn the parameter θ using a gradient-based algorithm. Let us define $J(\theta)$ to measure the expected value for policy π_θ for trajectory $\tau = s_0, a_0, r_0, \ldots, s_T, a_T, r_T$:

$$J(\theta) = E_\tau \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$$

Then the *policy gradient theorem* provides an analytical expression for the gradient of $J(\theta)$ as the following:

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\nabla_\theta \log p(\tau;\theta) G(\tau)\right], \tag{10}$$

$$= \mathbb{E}_{a\sim\pi_\theta(.|s), s\sim\rho_{\pi_\theta}} \left[\nabla_\theta \log \pi(a|s;\theta) Q_{\pi_\theta}(s,a)\right]. \tag{11}$$

, in which $G(\tau)$ is the return of the trajectory. Then the policy-based methods update the parameter θ as below:

$$\theta^{t+1} = \theta^t + \alpha\widehat{\nabla_\theta J(\theta)}, \tag{12}$$

where, $\widehat{\nabla_\theta J(\theta)}$ is an approximation of the true gradient, and α is the learning rate. Depends on how to estimate $\nabla_\theta \log p(\tau;\theta)$, $\log \pi(a|s;\theta)$, or $G(\tau)$, there are several variants of policy gradient algorithms. In the following, we explore some of these variants which are mostly used for MARL algorithms.

REINFORCE algorithm is one of the first policy-gradient algorithms (Sutton et al. 2000). Particularly, REINFORCE applies Monte Carlo method and uses the actual return $G_t := \sum_{t'=t}^{T} \gamma^{t'} r(s_{t'}, a_{t'})$ as an approximation for $G(\tau)$ in equation (10), and re-writes the gradient $\nabla_\theta \log p(\tau;\theta)$ as $\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$. This provides an unbiased estimation of the gradient; although, it has a high variance which makes the training quite hard in practice. To reduce the variance, it has been shown that subtracting a baseline b from $G(\tau)$ is very helpful and is being widely used in practice. If the baseline is a function of state s_t, the gradient estimator will be still an unbiased estimator, which makes $V(s)$ an appealing candidate for the baseline function. Intuitively, with a positive $G_t - V(s_t)$, we want to move toward the gradients since it results in a cumulative reward which is higher than the average cumulative reward for that state, and vice versa. There are several other extensions of REINFORCE algorithms, each tries to minimize the gradient estimation. Among them, REINFORCE with reward-to-go and baseline usually outperforms the REINFORCE with baseline. For more details about other extensions see Sutton and Barto (2018).

REINFORCE uses the actual return from a random trajectory, which might introduce a high variance into the training. In addition, one needs to wait until the end of the episode to obtain the actual cumulative discounted reward. Actor-critic (AC) algorithm extends REINFORCE by eliminating this constraint and minimizes the variance of gradient estimation. In AC, instead of waiting until the end of the episode, a critic model is used to approximate the value of state s_t by $Q(s_t, a_t) = r(s_t, a_t) + \gamma V(s_{t+1})$. As a natural choice for the baseline, picking $V(s_t)$ results in utilizing the advantage function $A(s_t, a_t) = r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)$. The critic model is trained by calculating the TD-error $\delta_t = r(s_t, a_t) + \gamma V_w(s_{t+1}) - V_w(s_t)$, and updating w by $w = w + \alpha_w \delta_t \nabla_w V_w(s_t)$, in which α_w is the critic's learning rate. Therefore, there is no need to wait until the end of the episode and after each time-step one can run a train-step. With AC, it is also straight-forward to train an agent for non-episodic environments.

Following this technique, several AC-based methods were proposed. Asynchronous advantage actor-critic (A3C) contains a master node connected to a few worker nodes (Mnih et al. 2016). This algorithm runs several instances of the actor-critic model and asynchronously gathers the

gradients to update the weights of a master node. Afterward, the master node broadcasts the new weights to the worker node, and in this way, all nodes are updated asynchronously. Synchronous advantage actor-critic (A2C) algorithm uses the same framework but synchronously updates the weights.

Neither REINFORCE, AC, A2C, and A3C guarantee a steady improvement over the objective function. Trust region policy gradient (TRPO) algorithm (Schulman et al. 2015) is proposed to address this issue. TRPO tries to obtain new parameters θ' with the goal of maximizing the difference between $J(\theta') - J(\theta)$, where θ is the parameter of the current policy. Under the trajectory generated from the new policy $\pi_{\theta'}$, it can be shown that

$$J(\theta') - J(\theta) = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t A_{\pi_\theta}(s_t, a_t) \right] \tag{13a}$$

$$= \sum_{t=0}^{\infty} \mathbb{E}_{s_t \sim p_{\theta'}(s_t)} \left[\mathbb{E}_{a_t \sim p_{\theta'}(a_t)} \left[\frac{p_{\theta'}(a_t|s_t)}{p_\theta(a_t|s_t)} \gamma^t A_{\pi_\theta}(s_t, a_t) \right] \right], \tag{13b}$$

$$\sim \sum_{t=0}^{\infty} \mathbb{E}_{s_t \sim p_{\theta}(s_t)} \left[\mathbb{E}_{a_t \sim p_{\theta'}(a_t)} \left[\frac{p_{\theta'}(a_t|s_t)}{p_\theta(a_t|s_t)} \gamma^t A_{\pi_\theta}(s_t, a_t) \right] \right], \tag{13c}$$

where the expectation of s_t in (13b) is over the $p_{\theta'}(s_t)$, though we do not have θ'. To address this issue, TRPO approximates (13b) by substituting $s_t \sim p_{\theta'}(s_t)$ with $s_t \sim p_\theta(s_t)$ assuming that $\pi_{\theta'}$ is close to π_θ, resulting in (13c). Under the assumption $|\pi_{\theta'}(a_t|s_t) - \pi_\theta(a_t|s_t)| \leq \epsilon, \ \forall s_t$, Schulman et al. (2015) show that

$$J(\theta') - J(\theta) \geq \sum_{t=0}^{\infty} \mathbb{E}_{s_t \sim p_{\theta}(s_t)} \left[\mathbb{E}_{a_t \sim p_{\theta'}(a_t)} \left[\frac{p_{\theta'}(a_t|s_t)}{p_\theta(a_t|s_t)} \gamma^t A_{\pi_\theta}(s_t, a_t) \right] \right] - \sum_t 2t\epsilon C \tag{14}$$

which C is a function of $r_{\max}$ and T. Thus, if ϵ is small enough, TRPO guarantees monotonic improvement under the assumption of the closeness of the policies. TRPO uses Kullback–Leibler divergence $D_{KL}(p_1(x)||p_2(x)) = \mathbb{E}_{x \sim p_1(x)} \left[\log \frac{p_1(x)}{p_2(x)} \right]$ to measure the amount of changes in the policy update. Therefore, it sets $|\pi_{\theta'}(a_t|s_t) - \pi_\theta(a_t|s_t)| \leq \sqrt{0.5 D_{KL}(\pi_{\theta'}(a_t|s_t)||\pi_\theta(a_t|s_t))} \leq \epsilon$ and solves a constrained optimization problem. For solving this optimization problem, TRPO approximates the objective function by using the first order term of the corresponding Taylor series. Similarly, the constraint is approximated using the second order term of the corresponding Taylor series. This results in a policy gradient update which involves calculating the inverse of the Fisher matrix (F^{-1}) multiplier and a specific learning to make sure that the bound ϵ is hold. Neural networks may have millions of parameters, which make it impossible to directly achieve F^{-1}. To address this issue, the conjugate gradient algorithm (Hestenes et al. 1952) is used.

In general, despite TRPO's benefits, it is relatively a complicated algorithm, it is expensive to run, and it is not compatible with architectures including dropout kind of noises, or parameter sharing among different networks (Schulman et al. 2017). To address these issues, Schulman et al. (2017)

proposed proximal policy optimization (PPO) algorithm in which it avoids using the Fisher matrix and its computation burden. PPO defines $r_t(\theta) = \frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)}$ and it obtains the gradient update by

$$\mathbb{E}_{\tau\sim p_\theta}\left[\sum_{t=0}^{\infty}\min(r_t(\theta)A_{\pi_\theta}(s_t,a_t),\text{clip}(r_t(\theta),1-\epsilon,1+\epsilon)A_{\pi_\theta}(s_t,a_t))\right], \tag{15}$$

which in practice works as good as TRPO in most cases.

Despite the issue of data efficiency, policy-based algorithms provide better convergence guarantees over the value-based algorithms (Yang et al. 2018b, Zhang et al. 2020a, Agarwal et al. 2020). This is still true with the policy gradient which utilizes neural networks as function approximation (Liu et al. 2019, Wang et al. 2020b). In addition, compared to the value-based algorithms, policy-based approaches can be easily applied to the continuous control problem. Furthermore, for most problems, we do not know the true form of the optimal policy, i.e., deterministic or stochastic. The policy gradient has the ability to learn either a stochastic or deterministic policy; however, in value-based algorithms, one needs to know the form of the policy at the algorithm's design time, which might be unknown. This results in two benefits of the policy-gradient method over value-based methods (Sutton and Barto 2018): (i) when the optimal policy is a stochastic policy (like Tic-tac-toe game), policy gradient by nature is able to learn that. However, the value-based algorithms have no way of learning the optimal stochastic policy. (ii) if the optimal policy is deterministic, by following the policy gradient algorithms, there is a chance of converging to a deterministic policy. However, with a value-based algorithm, one does not know the true form of the optimal policy so that he cannot choose the optimal exploration parameter (like ϵ in ϵ greedy method) to be used in the scoring time. In the first benefit, note that one may use a softmax operator over the Q-value to provide the probabilities for choosing each action; but, the value-based algorithms cannot learn the probabilities by themselves as a stochastic policy. Similarly, one may choose a non-zero ϵ for the score time, but one does not know the optimal value for such ϵ, so this method may not result in the optimal policy. Also, on the second benefit note that this issue is not limited to the ϵ-greedy algorithm. With added soft-max operator to a value-based algorithm, we get the probability of choosing each action. Even in this setting, the algorithm is designed to get the true values for each action, and there is no known mapping of true values to the optimal probabilities for choosing each action, which does not necessarily result in 0 and 1 actions. Similarly, the other variants of the soft-max operator like Boltzmann softmax which uses a temperature parameter, do not help either. Although, the temperature parameter can help to get determinism; still in practice we do not if the optimal solution is deterministic to do that.

3.2 Multi-Agent RL Notations and Formulation

We denote a multi-agent setting with tuple $< N, \mathcal{S}, \mathcal{A}, R, P, \mathcal{O}, \gamma >$, in which N is the number of agents, $\mathcal{S}$ is state space, $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_N\}$ is the set of actions for all agents, P is the transition probability among the states, R is the reward function, and $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_N\}$ is the set of observations for all agents. Within any type of the environment, we use $\boldsymbol{a}$ to denote the vector of actions for all agents, a_{-i} is the set of all agents except agent i, τ_i represents observation-action history of agent i, and $\boldsymbol{\tau}$ is the observation-action of all agents. Also, $\mathcal{T}$, $\mathcal{S}$, and $\mathcal{A}$ are the observation-action space, state space, and action space, respectively. Then, in a cooperative problem with N agents with full observability of the environment, each agent i at time-step t observes the global state s_t and uses the local stochastic policy π_i to take action a_i^t and then receives reward r_i^t. If the environment is *fully cooperative*, at each time step all agents observe a joint reward value r_t, i.e., $r_1^t = \cdots = r_N^t = r^t$. If the agents are not able to fully observe the state of the system, each agent only accesses its own local observation o_i^t.

Similar to the single-agent case, each agent is able to learn the optimal Q-value or the optimal stochastic policy. However, since the policy of each agent changes as the training progresses, the environment becomes non-stationary from the perspective of any individual agent. Basically, $P(s'|s, a_i, \pi_1, \ldots, \pi_N) \neq P(s'|s, a_i, \pi_1', \ldots, \pi_N')$ when any $\pi_i \neq \pi_i'$ so that we lose the underlying assumption of MDP. This means that the experience of each agent involves different co-player policies, so we cannot fix them and train an agent such that any attempt to train such models results in fluctuations of the training. This makes the model training quite challenging. Therefore, the adopted Bellman equation for MARL (Foerster et al. 2017) (assuming the full observability) also does not hold for the multi-agent system:

$$Q_i^*(s, a_i|\boldsymbol{\pi}_{-i}) = \sum_{\boldsymbol{a}_{-i}} \boldsymbol{\pi}_{-i}(\boldsymbol{a}_{-i}, s) \left[r(s, a_i, \boldsymbol{a}_{-i}) + \gamma \sum_{s'} P(s'|s, a_i, \boldsymbol{a}_{-i}) \max_{a_i'} Q_i^*(s, a_i') \right], \quad (16)$$

where $\boldsymbol{\pi}_{-i} = \Pi_{j \neq i} \pi_j(a_j|s)$. Due to the fact that $\boldsymbol{\pi}_{-i}$ changes over time as the policy of other agents changes, in MARL one cannot obtain the optimal Q-value using the classic Bellman equation.

On the other hand, the policy of each agent changes during the training, which results in a mix of observations from different policies in the experience replay. Thus, one cannot use the experience replay without dealing with the non-stationarity. Without experience replay, the DQN algorithm (Mnih et al. 2015) and its extensions can be hard to train due to the sample inefficiency and correlation among the samples. The same issue exists within AC-based algorithms which use a DQN-like algorithm for the critic. Besides, in most problems in MARL, agents are not able to observe the full state of the system, which are categorized as decentralized POMDP (Dec-POMDP). Due to the partial observability and the non-stationarity of the local observations, Dec-POMDPs are even

harder problems to solve and it can be shown that they are in the class of NEXP-complete problems (Bernstein et al. 2002). A similar equation to (16) can be obtained for the partially observable environment too.

In the Multi-agent RL, the noise and variance of the rewards increase which results in the instability of the training. The reason is that the reward of one agent depends on the actions of other agents, and the conditioned reward on the action of a single agent can exhibit much more noise and variability than a single agent's reward. Therefore, training a policy gradient algorithm also would not be effective in general.

Finally, we define the following notation which is used in a couple of papers in Nash equilibrium. A joint policy π^* defines a Nash equilibrium if and only if:

$$\forall \pi_i \in \Pi_i, \forall s \in \mathcal{S}, v_i^{(\pi_i^*, \pi_{-i}^*)}(s) \geq v_i^{(\pi_i, \pi_{-i}^*)}(s); \forall i \in \{1, \ldots, N\} \tag{17}$$

in which $v_i^{(\pi_i, \pi_{-i})}(s)$ is the expected cumulative long-term return of agent i in state s and Π_i is the set of all possible policies for agent i. Particularly, it means that each agent prefers not to change its policy if it wants to attain the long-term cumulative discounted reward. Further, if the following holds for policy $\hat{\pi}$:

$$v_i^{(\hat{\pi})}(s) \geq v_i^{(\pi_i)}(s), \quad \forall i \in \{1, \ldots, N\}, \ \forall \pi_i \in \Pi_i, \ \forall s \in \mathcal{S},$$

policy $\hat{\pi}$ is called Pareto-optimal. We introduce notation for Nash equilibrium only as much that we needed for representing a few famous papers on the cooperative MARL. For more details on this topic see Yang and Wang (2020).

4 Independent Learners

One of the first proposed approaches to solve the multi-agent RL problem is to treat each agent independently such that it considers the rest of the agents as part of the environment. This idea is formalized in independent Q-Learning (IQL) algorithm (Tan 1993), in which each agent accesses its local observation and the overall agents try to maximize a joint reward. Each agent runs a separate Q-learning algorithm (Watkins and Dayan 1992) (or it can be the newer extensions like DQN (Mnih et al. 2015), DRQN (Hausknecht and Stone 2015), etc.). IQL is an appealing algorithm since (i) it does not have the scalability and communication problem that the central control method encounters with increasing the number of agents, (ii) each agent only needs its local history of observations during the training and the inference time. Although it fits very well to the partially observable settings, it has the non-stationarity of environment issue. The tabular IQL usually works well in practice for small size problems (Matignon et al. 2012, Zawadzki et al. 2014); however, in the case of function approximation, especially deep neural network (DNN), it may

not work very well. One of the main reasons for this weak performance is the need for the experience replay to stabilize the training with DNNs (Foerster et al. 2017). In an extension of IQL, Distributed Q-learning (Lauer and Riedmiller 2000) considers a decentralized fully cooperative multi-agent problem such that all agents observe the full state of the system and do not know the actions of the other agents, although in the training time it assumes the joint action is available for all agents. The joint action is executed in the environment and it returns the joint reward that each agent receives. This algorithm updates the Q-values only when there is a guaranteed improvement, assuming that the low returns are the result of a bad exploration of the teammates. In other words, it maximizes over the possible actions for agent i, assuming other agents selected the local optimal action, i.e., for a given joint action $\boldsymbol{a}_t = (a_1^t, \ldots, a_N^t)$, it updates the Q-values of agent i by:

$$q_i^{t+1}(s,a) = \begin{cases} q_i^t(s,a) & \text{if } s \neq s_t \text{ or } a \neq a_t, \\ \max\left\{q_i^t(s,a), r(s_t,\boldsymbol{a}_t) + \gamma \max_{a' \in A} q_i^t\left(\delta(s_t,\boldsymbol{a}_t), a'\right)\right\} & \text{otherwise,} \end{cases} \tag{18}$$

in which $q_i^t(s,a) = \max_{\boldsymbol{a}=\{a^1,\ldots,a^N\},a^i=a} Q(s,\boldsymbol{a})$ and $\delta(s_t,\boldsymbol{a}_t)$ is the environment function which results in s_{t+1}. Therefore, Distributed Q-learning completely ignores the low rewards that causes an overestimated Q-values. This issue besides the curse of dimensionality results in poor performance in the problems with high dimension.

Hysteretic Q-learning (Matignon et al. 2007) considers the same problem and tries to obtain a good policy assuming that the low return might be the result of stochasticity in the environment so that it does not ignore them as Distributed Q-Learning does. In particular, when the TD-error is positive, it updates the Q-values by the learning rate α, and otherwise, it updates the Q-values by the learning rate $\beta < \alpha$. Thus, the model is also robust to negative learning due to the teammate explorations. Bowling and Veloso (2002) also propose to use a variable learning rate to improve the performance of tabular IQL. In another extension for the IQL, Fuji et al. (2018) propose to train one of the agents at each time and fix the policy of other agents within a periodic manner in order to stabilize the environment. So, during the training, other agents do not change their policies and the environment from the view-point of the single agent is stationary.

DQN algorithm Mnih et al. (2015) utilized *experience reply* and *target network* and was able to attain super-human level control on most of the Atari games. The classical IQL uses the tabular version, so one naive idea could be using the DQN algorithm instead of each single Q-learner. Tampuu et al. (2017) implemented this idea and was one of the first papers which took the benefit of the neural network as a general powerful approximator in an IQL-like setting. Specifically, this paper analyzes the performance of the DQN in a decentralized two-agent game for both competitive and cooperative settings. They assume that each agent observes the full state (the video of the game), takes an action by its own policy and the reward values are also known to both agents. The paper is mainly built on the *Pong* game (from Atari-2600 environment (Bellemare et al. 2013)) in

which by changing the reward function the competitive and cooperative behaviors are obtained. In the competitive version, each agent that drops the ball loses a reward point, and the opponent wins the reward point so that it is a zero-sum game. In the cooperative setting, once either of the agents drops the ball, both agents lose a reward point. The numerical results show that in both cases the agents are able to learn how to play the game very efficiently, that is in the cooperative setting, they learn to keep the ball for long periods, and in the competitive setting, the agents learn to quickly beat the competitor.

Experience replay is one of the core elements of the DQN algorithm. It helps to stabilize the training of the neural network and improves the sample efficiency of the history of observations. However, due to the non-stationarity of the environment, using the experience replay in a multi-agent environment is problematic. Particularly, the policy that generates the data for the experience replay is different than the current policy so that the learned policy of each agent can be misleading. In order to address this issue, Foerster et al. (2016) disable the experience replay part of the algorithm, or in Leibo et al. (2017) the old transitions are discarded and the experience replay uses only the recent experiences. Even though these approaches help to reduce the non-stationarity of the environment, but both limit the sample efficiency. To resolve this problem, Foerster et al. (2017) propose two algorithms to stabilize the experience reply in IQL-type algorithms. They consider a fully cooperative MARL with local observation-action. In the first approach, each transition is augmented with the probability of choosing the joint action. Then, during the loss calculation, the importance sampling correction is calculated using the current policy. Thus, the loss function is changed to:

$$L(\theta_i) = \sum_{k=1}^{b} \frac{\boldsymbol{\pi}_{-i}^{t_c}(\boldsymbol{a}_{-i}, s)}{\boldsymbol{\pi}_{-i}^{t_i}(\boldsymbol{a}_{-i}, s)} \left[(y_i^{DQN} - Q(s, a_i; \theta_i))^2 \right], \tag{19}$$

in which θ_i is the policy parameters for agent i, t_c is the current time-step, and t_i is the time of collecting i^{th} sample. In this way, the effect of the transitions generated from dissimilar policies is regularized on gradients. In the second algorithm, named FingerPrint, they propose augmenting the experience replay with some parts of the policies of the other agents. However, the number of parameters in DNN is usually large and as a result, it is intractable in practice. Thus, they propose to augment each instance in the experience replay by the iteration number e and the ϵ of the ϵ-greedy algorithm. In the numerical experiments, they share the weights among the agent, while the id of each agent is also available as the input. They provide the results of two proposed algorithms plus the combination of them on the StarCraft game (Samvelyan et al. 2019) and compare the results by a classic experience replay and one no-experience replay algorithm. They conclude that the second algorithm obtains better results compared to the other algorithm.

Omidshafiei et al. (2017) propose another extension of the experience replay for the MARL. They consider multi-task cooperative games, with independent partially observable learners such that each agent only knows its own action, with a joint reward. An algorithm, called HDRQN, is proposed which is based on the DRQN algorithm (Hausknecht and Stone 2015) and the Hysteretic Q-learning (Matignon et al. 2007). Also, to alleviate the non-stationarity of MARL, the idea of Concurrent Experience Replay Trajectories (CERTs) is proposed, in which the experience replay gathers the experiences of all agents in any period of one episode and also during the sampling of a mini-batch, it obtains the experiences of one period of all agents together. Since they use LSTM, the experiences in the experience replay are zero-padded (adds zero to the end of the experiments with smaller sizes to make the size of all experiments equal). Moreover, in the multi-task version of HRDQN, there are different tasks that each has its own transition probability, observation, and reward function. During the training, each agent observes the task ID, while it is not accessible in the inference time. To evaluate the model, a two-player game is utilized, in which agents are rewarded only when all the agents simultaneously capture the moving target. In order to make the game partially observable, a flickering screen is used such that with 30% chance the screen is flickering. The actions of the agents are moving north, south, west, east, or waiting. Additionally, actions are noisy, i.e. with 10% probability the agent might act differently than what it wanted.

5 Fully Observable Critic

Non-stationarity of the environment is the main issue in multi-agent problems and MARL algorithms. One of the common approaches to address this issue is using a fully observable critic. The fully observable critic involves the observations and actions of all agents and as a result, the environment is stationary even though the policy of other agents changes. In other words, $P(s'|s, a_1, \ldots, a_N, \pi_1, \ldots, \pi_N) = P(s'|s, a_1, \ldots, a_N, \pi'_1, \ldots, \pi'_N)$ even if $\pi_i \neq \pi'_i$, since the environment returns an equal next-state regardless of the changes in the policy of other agents. Following this idea, there can be 1 or N critic models: (i) in a fully cooperative problems, one central critic is trained, and (ii) when each agent observes a local reward, each agent may need to train its own critic model, resulting to N critic models. In either case, once the critic is fully observable, the non-stationarity of critic is resolved and it can be used as a good leader for local actors.

Using this idea, Lowe et al. (2017) propose a model-free multi-agent reinforcement learning algorithm to the problem in which agent i at time step t of execution accesses its own local observation o_i^t, local actions a_i^t, and local rewards r_i^t. They consider cooperative, competitive, and mixed competitive and cooperative games, and proposed Multi-agent DDPG (MADDPG) algorithm in which each agent trains a DDPG algorithm such that the actor $\pi_i(o_i; \theta_i)$ with policy weights θ_i observes the local observations while the critic Q_i^μ is allowed to access the observations, actions, and the

target policies of all agents in the training time. Then, the critic of each agent concatenates all state-actions together as the input and using the local reward obtains the corresponding Q-value. Either of N critics are trained by minimizing a DQN-like loss function:

$$L(\mu_i) = \mathbb{E}_{\boldsymbol{o}^t,a,r,\boldsymbol{o}^{t+1}}\left[\left(Q_i(\boldsymbol{s}^t, a_1^t, \ldots, a_N^t; \mu_i) - y\right)^2\right],$$
$$y = r_i^t + \gamma Q_i\left(\boldsymbol{o}^t, \bar{a}_1^{t+1}, \ldots, \bar{a}_N^{t+1}; \bar{\mu}_i\right)|_{\bar{a}_j^{t+1}=\bar{\pi}(o_j^{t+1})},$$

in which $\boldsymbol{o}^t$ is observation of all agents, $\bar{\pi}_j$ is the target policy, and $\bar{\mu}$ is the target critic. As a result, the *critic of each agent deals with a stationary environment*, and in the inference time, it only needs to access the local information. MADDPG is compared with the decentralized trained version of DDPG (Lillicrap et al. 2016), DQN (Mnih et al. 2015), REINFORCE Sutton et al. (2000), and TRPO (Schulman et al. 2015) algorithm in a set of grounded communication environments from particle environment (Haarnoja et al. 2018), e.g., predator-prey, arrival task, etc. The continues space-action predator-prey environment from this environment is usually considered as a benchmark for MARL algorithms with local observation and cooperative rewards. In the most basic version of the predator-prey, there are two predators which are randomly placed in a 5×5 grid, along with one prey which also randomly is located in the grid. Each predator observes its direct neighbor cells, i.e., 3×3 cells, and the goal is to catch the prey together to receive a reward, and in all other situations, each predator obtains a negative reward.

Several extensions of MADDPG algorithm are proposed in the literature and we review some of them in the rest of this section. Ryu et al. (2018) propose an actor-critic model with local actor and critic for a DEC-POMDP problem, in which each agent observes a local observation o_i, observes its own reward $r_i : \mathcal{S} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_N \to \mathbb{R}$, and learns a deterministic policy $\mu_{\theta_i} : \mathcal{O}_i \to \mathcal{A}_i$. The goal for agent i is to maximize its own discounted return $R_i = \sum_{t=0}^{\infty} \gamma^t r_i^t$. An extension of MADDPG with a generative cooperative policy network, called MADDPG-GCPN, is proposed in which there is an extra actor network μ_i^c to generate action samples of other agents. Then, the critic uses the experience replay filled with sampled actions from GCPN, and not from the actor of the other agents. So, there is no need to share the target policy of other agents during the training time. Further, the algorithm is modified in a way such that the critic can use either immediate individual or joint reward during the training. They presented a new version of the predator-prey game in which each agent receives an individual reward plus a shared one if they catch the prey. The experimental analysis on the predator-prey game and a controlling energy storage systems problem shows that the standard deviation of obtained Q-values is lower in MADDPG-GCPN compared to MADDPG.

As another extension of MADDPG, Chu and Ye (2017) consider multi-agent cooperative problems with N agents and proposes three actor-critic algorithms based on MADDPG. The first one assumes that all agents know the global reward and shares the weights between agents so that

it ~~actually~~ includes one actor and one critic network. The second algorithm assumes the global reward is not shared and each agent indeed updates its own critic using the local reward so that there are N critic networks. Although, agents share their weights so that there is only one actor network which means $N + 1$ networks are trained. The third algorithm also assumes non-shared global reward, though uses only two networks, one actor network and one critic network such that the critic has N heads in which head $i \in \{1, \ldots, N\}$ provides the Q-value of the agent i. They compare the results of their algorithms on three new games with MADDPG, PPO (Schulman et al. 2017), and PS-TRPO algorithms (PS-TRPO is the TRPO algorithm (Schulman et al. 2015) with parameter sharing, see Sukthankar and Rodriguez-Aguilar (2017) for more details).

Mao et al. (2019) present another algorithm based on MADDPG for a cooperative game, called ATT-MADDPG which considers the same setting as in Lowe et al. (2017). It enhances the MADDPG algorithm by adding an attention layer in the critic network. In the algorithm, each agent trains a critic which accesses the actions and observations of all agents. To obtain the Q-value, an attention layer is added on the top of the critic model to determine the corresponding Q-value. In this way, at agent i, instead of just using $[o_1^t, \ldots, o_N^t]$ and $[a_i^t, a_{-i}^t]$ for time step t, ATT-MADDPG considers K combinations of possible action-vector a_{-i}^t, and obtains the corresponding K Q-values. Also, using an attention model, it obtains the weights of all K action-sets such that the hidden vector h_i^t of the attention model is generated via the actions of other agents (a_{-i}^t). Then, the attention weights are used to obtain the final Q-value by a weighted sum of the K possible Q-values. Indeed, their algorithm combines the MADDPG with the k-head Q-value (Van Seijen et al. 2017). They provide some numerical experiments on cooperative navigation, predator-prey, a packet-routing problem, and compare the performance with MADDPG and few other algorithms. Moreover, the effect of small or large K is analyzed within each environment. In Wang et al. (2019) again the problem setting is considered as the MADDPG, though they assume a limit on the communication bandwidth. Due to this limitation, the agents are not able to share all the information, e.g., they cannot share their locations on "arrival task" (from Multi-Agent Particle Environment (Mordatch and Abbeel 2018a)), which limits the ability of the MADDPG algorithm to solve the problem. To address this issue, they propose R-MADDPG, in which a recurrent neural network is used to remember the last communication in both actor and critic. In this order, they modified the experience replay such that each tuple includes $(o_i^t, a_i^t, o_i^{t+1}, r_i^t, h_i^t, h_i^{t+1})$, in which h_t^i is the hidden state of the actor network. The results are compared with MADDPG over the "arrival task" with a communication limit, where each agent can select either to send a message or not, and the message is simply the position of the agent. With the fully observable state, their algorithm works as well as MADDPG. On the other hand, within the partially observable environment, recurrent actor (with

fully connected critic) does not provide any better results than MADDPG; though, applying both recurrent actor and recurrent critic, R-MADDPG obtains higher rewards than MADDPG.

Since MADDPG concatenates all the local observations in the critic, it faces the curse of dimensionality with increasing the number of agents. To address this issue, Iqbal and Sha (2019) proposed Multiple Actor Attention-Critic (MAAC) algorithm which efficiently scales up with the number of agents. The main idea in this work is to use an attention mechanism Choi et al. (2017), Jiang and Lu (2018) to select relevant information for each agent during the training. In particular, agent i receives the observations, $o = (o_1, ..., o_N)$, and actions, $a = (a_1, ..., a_N)$ from all agents. The value function parameterized by ψ, $Q_i^{\psi}(o, a)$, is defined as a function of agent i's observation-action, as well as the information received from the other agents:

$$Q_i^{\psi}(o, a) = f_i(g_i(o_i, a_i), x_i),$$

where f_i is a two-layer perceptron, g_i is a one-layer embedding function, and x_i is the contribution of other agents. In order to fix the size of x_i, it is set equal to the weighted sum of other agents' observation-action:

$$x_i = \sum_{j \neq i} \alpha_j v_j = \sum_{j \neq i} \alpha_j h(V g_j(o_j, a_j)),$$

where v_j is a function of the embedding of agent j, encoded with an embedding function and then linearly transformed by a shared matrix V, and h is the activation function. Denoting $e_j = g_j(o_j, a_j)$, using the query-key system (Vaswani et al. 2017), the attention weight α_j is proportional to:

$$\alpha_j \propto \exp(e_j^T W_k^T W_q e_i),$$

where W_q transforms e_i into a "query" and W_k transforms e_j into a "key". The critic step updates the ψ through minimizing the following loss function:

$$\mathcal{L}_Q(\psi) = \sum_{i=1}^{N} \mathbb{E}_{(o,a,r,o') \sim D} \left[(Q_i^{\psi}(o, a) - y_i)^2 \right],$$

where,

$$y_i = r_i + \gamma \mathbb{E}_{a' \sim \pi_{\bar{\theta}}(o')} \left[Q_i^{\bar{\psi}}(o', a') - \alpha \ \log(\pi_{\bar{\theta}_i}(a_i'|o_i')) \right],$$

where $\bar{\psi}$ and $\bar{\theta}$ are the parameters of the target critics and target policies respectively. In order to encourage exploration, they also use the idea of soft-actor-critic (SAC) (Haarnoja et al. 2018). MAAC is compared with COMA (Foerster et al. 2018) (will be discussed shortly), MADDPG, and their updated version with SAC, as well as an independent learner with DDPG (Lillicrap et al. 2016), over two environments: treasure collection and rover-tower. MAAC obtains better results than the other algorithms such that the performance gap becomes shorter as the number of agents increases.

Jiang et al. (2020) assume a graph connection among the agents such that each node is an agent. A partially observable environment is assumed in which each agent observes a local observation, takes a local action, and receives a local reward, while agents can share their observations with their neighbors and the weights of all agents are shared. A graph convolutional reinforcement learning for cooperative multi-agent is proposed. The multi-agent system is modeled as a graph such that agents are the nodes, and each has some features which are the encoded local observations. A multi-head attention model is used as the convolution kernel to obtain the connection weights to the neighbor nodes. To learn Q-function an end-to-end algorithm, named DGN, is proposed, which uses the centralized training and distributed execution (CTDE) approach. The goal is to maximize the sum of the reward of all agents. During the training, DGN allows the gradients of one agent to flow K-neighbor agents and its receptive field to stimulate cooperation. In particular, DGN consists of three phases: (i) an observation encoder, (ii) a convolutional layer, and (iii) Q-network. The encoder (which is a simple MLP, or convolution layer if it deals with images) receives observation o_i^t of agent i at time t and encodes it to a feature vector h_i^t. In phase (ii), the convolutional layer integrates the local observations of K neighbors to generate a latent feature h'^t_i. In order to obtain the latent vector, an attention model is used to make the input independent of the number of input features. The attention model gets all feature vectors of the K-neighbors, generates the attention weights, and then calculates the weighted sum of the feature vectors to obtain h'^t_i. Another convolution layer may be added to the model to increase the receptive field of each agent, such that h'^t_i are the inputs of that layer, and h''^t_i are the outcomes. Finally, in phase (iii), the Q-network provides the Q-value of each possible action. Based on the idea of DenseNet (Huang et al. 2017), the Q-network gathers observations and all the latent features and concatenates them for the input. (Note that this procedure is followed for each agent, and the weights of the network are shared among all agents.) In the loss function, besides the Q-value loss function, a penalized KL-divergence is added. This function measures the changes between the current attention weights and the next state attention weights and tries to avoid drastic changes in the attention weight. Using the trained network, in the execution time each agent i observes o_i^t plus the observation of its K neighbors ($\{o_{tot}\}^t_{j\in\mathbb{N}(i)}$) to get an action. The results of their algorithm are compared with DQN, CommNet (Sukhbaatar et al. 2016), and MeanField Q-Learning (Yang et al. 2018a), on Jungle, Battle, and Routing environments.

Yang et al. (2018a) considers the multi-agent RL problems in the case that there exists a huge number of agents collaborating or competing with each other to optimize some specific long-term cumulative discounted rewards. They propose *Mean Field Reinforcement Learning* framework, where every single agent only considers an average effect of its neighborhoods, instead of exhaustive communication with all other agents within the population. Two algorithms namely Mean Field

Q-learning (MF-Q) and Mean Field Actor-Critic (MF-AC) are developed following the mean-field idea. There exists a single state visible to all agents, and the local reward and the local action of all agents are also visible to the others during the training. Applying Tylor' theorem, it is proved that $Q^i(s, a)$ can be approximated by $Q^i(s, a^i, \bar{a}^i)$, where a concatenates the action of all agents, a^i is the action of agent i, and $\bar{a}^i$ denotes the average action from the neighbors. Furthermore, utilizing the contraction mapping technique, it is shown that mean-field Q-values converge to the Nash Q-values under some particular assumptions. The proposed algorithms are tested on three different problems: Gaussian Squeeze and the Ising Model (a framework in statistical mechanics to mathematically model ferromagnetism), which are cooperative problems, and the battle game, which is a mixed cooperative-competitive game. The numerical results show the effectiveness of the proposed method in the case of many-agent RL problems.

(Foerster et al. 2018) proposes COMA, a model with a single centralized critic which uses the global state, the vector of all actions, and a joint reward. This critic is shared among all agents, while the actor is trained locally for each agent with the local observation-action history. The joint reward is used to train $Q(s^t, [a_i^t, a_{-i}^t])$. Then, for the agent i, with the fixed a_{-i}^t the actor uses a counterfactual baseline

$$b(s^t, a_{-i}^t) = \sum_{\hat{a}_i^t} \pi_i(\hat{a}_i^t | o_i^t) Q(s^t, [\hat{a}_i^t, a_{-i}^t]),$$

to obtain contribution of action a_i^t via advantage function $A(s, a_i^t) = Q(s, [a_{-i}^t, a_i]) - b(s^t, a_{-i}^t)$. Also, each actor shares its weights with other agents, and uses gated recurrent unit (GRU) (Cho et al. 2014) to utilize the history of the observation. They present the results of their algorithm on StarCraft game (Samvelyan et al. 2019) and compare COMA with central-V, central-QV, and two implementations of independent actor-critic (IAC).

Yang et al. (2020) consider a multi-agent cooperative problem in which in addition to the cooperative goal, each agent needs to attain some personal goals. Each agent observes a local observation, local reward corresponding to the goal, and its own history of actions, while the actions are executed jointly in the environment. This is a common situation in problems like autonomous car driving. For example, each car has to reach a given destination and all cars need to avoid the accident and cooperate in intersections. The authors propose an algorithm (centralized training, decentralized execution) called CM3, with two phases. In the first phase, one single network is trained for all agents to learn personal goals. The output of this network, a hidden layer, is passed to a given layer of the second network to initialize it, and the goal of the second phase is to attain the global goal. Also, since the collective optimal solution of all agents is not necessarily optimal for every individual agent, a credit assignment approach is proposed to obtain the global solution of all agents. This credit assignment, motivated by Foerster et al.

(2018), is embedded in the design of the second phase. In the first phase of the algorithm, each agent is trained with an actor-critic algorithm as a single agent problem learning to achieve the given goal of the agent. In this order, the agent is trained with some randomly assigned goal very well, i.e., agent i wants to learn the local policy π to maximize $J_i(\pi)$ for any arbitrary given goal. All agents share the weights of the policy, so the model is reduced to maximize $J_{local}(\pi) = \sum_{i=1}^{N} J_i(\pi)$. Using the advantage approximation, the update is performed by $\nabla_\theta J_{local}(\pi) = \mathbb{E}\left[\sum_{i=0}^{N} \nabla_\theta \log \pi_i(a_i|o_i, g_i)(R(s, a_i, g_i) + \gamma V(o_i^{t+1}, g_i) - V(o_i^t, g_i))\right]$, in which g_i is the goal of agent i. The second phase starts by the pre-trained agents, and trains a new global network in the multi-agent setting to achieve a cooperative goal by a comprehensive exploration. The cooperative goal is the sum of local rewards, i.e., $J_{global}(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_g^t\right]$, in which $R_g^t = \sum_{i=1}^{N} r(o_i^t, a_i^t, g_i)$. In the first phase, each agent only observes the part of the state that involves the required observation to complete the personal task. In the second phase, additional observations are given to each agent to achieve the cooperative goal. This new information is used to train the centralized critic and also used in the advantage function to update the actor's policy. The advantage function uses the counterfactual baseline (Foerster et al. 2018), so that the global objective is updated by $\nabla_\theta J_{global}(\pi) = \mathbb{E}\left[\sum_{i=0}^{N} \nabla_\theta \log \pi_i(a_i|o_i, g_i)(Q(s, a, g) - b(s, a^{-i}, g))\right]$. Finally, a combined version of the local and global models is used in this phase to train the model with the centralized critic. They present the experiments on an autonomous vehicle negotiation problem and compare the results with COMA (Foerster et al. 2018) and independent actor-critic learners model.

Sartoretti et al. (2019b) extend A3C (Mnih et al. 2016) algorithm for a centralized actor and a centralized critic. The proposed algorithm is based on centralized training and decentralized execution, in a fully observable environment. They consider a construction problem, TERMES (Petersen 2012), in which each agent is responsible to gather, carry, and place some blocks to build a certain structure. Each agent observes the global state plus its own location, takes its own local action, and executes it locally (no joint action selection/execution). Each agent receives its local sparse reward, such that the reward is +1 if it puts down a block in a correct position and -1 if it picks up a block from a correct position. Any other actions receive 0 rewards. During the training, all agents share the weights (as it is done in A3C) for both actor and critic models to train a central agent asynchronously. In the execution time, each agent uses one copy of the learned policy without communicating with other agents. The goal of all agents is to achieve the maximum common reward. The learned policy can be executed in an arbitrary number of agents, and each agent can see the other agents as moving elements, i.e., as part of the environment.

Finally, in Kim et al. (2019) a multi-agent problem is considered under the following two assumptions: 1) The communication bandwidth among the agents is limited. 2) There exists a shared

communication medium such that at each time step only a subset of agents is able to use it to broadcast their messages to the other agents. Therefore, communication scheduling is required to determine which agents are able to broadcast their messages. Utilizing the proposed framework, which is called SchedNet, the agents are able to schedule themselves, learn how to encode the received messages, and also learn how to pick actions based on these encoded messages. SchedNet focuses on centralized training and distributed execution. Therefore, in training the global state is available to the critic, while the actor is local for each agent and the agents are able to communicate through the limited channel. To control the communication, a Medium Access Control (MAC) protocol is proposed, which uses a weight-based scheduler (WSA) to determine which nodes can access the shared medium. The local actor i contains three networks: 1) message encoder, which takes the local observation o_i and outputs the messages m_i. 2) weight generator, which takes the local observation o_i and outputs the weight w_i. Specifically, the w_i determines the importance of observations in node i. 3) action selector. This network receives the observation o_i, encoded messages m_i plus the information from the scheduling module, which selects K agents who are able to broadcast their messages. Then maps this information to the action a_i. A centralized critic is used during the training to criticize the actor. In particular, the critic receives the global state of the environment and the weigh vector W generated by the wight generator networks as an input, and the output will be $Q(S, W)$ as well as $V(S)$. The first one is used to update the actor wight w_i while the second one is used for adjusting the weights of two other networks, namely action selector and message encoder. Experimental results on the predator-prey, cooperative-communication, and navigation task demonstrate that intelligent communication scheduling can be helpful in MARL.

6 Value Function Factorization

Consider a cooperative multi-agent problem in which we are allowed to share all information among the agents and there is no communication limitation among the agents. Further, let's assume that we are able to deal with the huge action space. In this scenario, a centralized RL approach can be used to solve the problem, i.e., all state observations are merged together and the problem is reduced to a single agent problem with a combinatorial action space. However, Sunehag et al. (2018) shows that naive centralized RL methods fail to find the global optimum, even if we are able to solve the problems with such a huge state and action space. The issue comes from the fact that some of the agents may get lazy and not learn and cooperate as they are supposed to. This may lead to the failure of the whole system. One possible approach to address this issue is to determine the role of each agent in the joint reward and then somehow isolate its share out of it. This category of algorithms is called *Value Function Factorization*.

In POMDP settings, if the optimal reward-shaping is available, the problem reduces to train several independent learners, which simplifies the learning. Therefore, having a reward-shaping model would be appealing for any cooperative MARL. However, in practice it is not easy to divide the received reward among the agents since their contribution to the reward is not known or it is hard to measure. Following this idea, the rest of this section discusses the corresponding algorithms.

In the literature of tabular RL, there are two common approaches for reward shaping: (i) *difference rewards* (Agogino and Tumer 2004) which tries to isolate the reward of each agent from the joint reward, i.e. $\hat{r}_i = r - r_{-i}$, where r_{-i} denotes the other agents' share than agent i in the global reward, (ii) *Potential-based reward shaping* (Ng et al. 1999). In this class of value function factorization methods, term $r + \Phi(s') - \Phi(s)$ is used instead of mere r, in which $\Phi(s)$ defines the desirability of the agent to be at state s. This approach is also extended for online POMDP settings (Eck et al. 2016) and multi-agent setting (Devlin and Kudenko 2011), though defining the potential function is challenging and usually needs specific domain knowledge. In order to address this issue, Devlin et al. (2014) combine these approaches and proposes two tabular algorithms. Following this idea, several value function factorization algorithms are proposed to automate reward-shaping and avoid the need for a field expert, which are summarized in the following.

Sunehag et al. (2018) consider a fully cooperative multi-agent problem (so a single shared reward exists) in which each agent observes its own state and action history. An algorithm, called VDN, is proposed to decompose the value function for each agent. Intuitively, VDN measures the impact of each agent on the observed joint reward. It is assumed that the joint action-value function Q_{tot} can be additively decomposed into N Q-functions for N agents, in which each Q-function only relies on the local state-action history, i.e.,

$$Q_{tot} = \sum_{i=1}^{N} Q_i(\tau_i, a_i, \theta_i) \tag{20}$$

In other words, for the joint observation-action history $\boldsymbol{\tau}$, it assumes validity of the individual-global-max (IGM) condition. Individual action-value functions $[Q_i : \mathcal{T} \times \mathcal{A}]_{i=1}^{N}$ satisfies IGM condition for the joint action-value function Q_{tot}, if:

$$\arg\max_{\boldsymbol{a}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{a}) = \begin{pmatrix} \arg\max_{a_1} Q_1(\tau_1, a_1) \\ \vdots \\ \arg\max_{a_N} Q_N(\tau_N, a_N) \end{pmatrix}, \tag{21}$$

in which $\boldsymbol{\tau}$ is the vector of local observation of all agents, and u is the vector of actions for all agents. Therefore, each agent observes its local state, obtains the Q-values for its action, selects an action, and then the sum of Q-values for the selected action of all agents provides the total Q-value of the problem. Using the shared reward and the total Q-value, the loss is calculated and then the gradients are backpropagated into the networks of all agents. In the numerical experiments, a recurrent

neural network with dueling architecture (Wang et al. 2016c) is used to train the model. Also, two extensions of the model are analyzed: (i) shared the policy among the agent, by adding the one-hot-code of the agent id to state input, (ii) adding information channels to share some information among the agents. Finally, VDN is compared with independent learners, and centralized training, in three versions of the two-player 2D grid.

QMIX (Rashid et al. 2018) considers the same problem as VDN does, and proposed an algorithm which is in fact an improvement over VDN (Sunehag et al. 2018). As mentioned, VDN adds some restrictions to have the additivity of the Q-value and further shares the action-value function during the training. QMIX also shares the action-value function during the training (a centralized training algorithm, decentralized execution); however, adds the below constraint to the problem:

$$\frac{\partial Q_{tot}}{\partial Q_i} \geq 0, \forall i, \tag{22}$$

which enforces positive weights on the mixer network, and as a result, it can guarantee (approximately) monotonic improvement. Particularly, in this model, each agent has a Q_i network and they are part of the general network (Q_{tot}) that provides the Q-value of the whole game. Each Q_i has the same structure as DRQN (Hausknecht and Stone 2015), so it is trained using the same loss function as DQN. Besides the monotonicity constraint over the relationship between Q_{tot} and each Q_i, QMIX adds some extra information from the global state plus a non-linearity into Q_{tot} to improve the solution quality. They provide numerical results on StarCraft II and compare the solution with VDN.

Even though VDN and QMIX cover a large domain of multi-agent problems, the assumptions for these two methods do not hold for all problems. To address this issue, Son et al. (2019) propose QTRAN algorithm. The general settings are the same as VDN and QMIX (i.e., general DEC-POMDP problems in which each agent has its own partial observation, action history, and all agents share a joint reward). The key idea here is that the actual Q_{tot} may be different than $\sum_{i=1}^{N} Q_i(\tau_i, a_i, \theta_i)$. However, they consider an alternative joint action-value Q'_{tot}, assumed to be factorizable by additive decomposition. Then, to fill the possible gap between Q_{tot} and Q'_{tot} they introduce

$$V_{tot} = \max_{\boldsymbol{a}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{a}) - \sum_{i=1}^{N} Q_i(\tau_i, \bar{a}_i), \tag{23}$$

in which $\bar{a}_i$ is $\arg\max_{a'_i} Q_i(\tau_i, a'_i)$. Given, $\bar{\boldsymbol{a}} = [\bar{a}_i]_{i=1}^{N}$, they prove that

$$\sum_{i=1}^{N} Q_i(\tau_i, a_i, \theta_i) - Q_{tot}(\boldsymbol{\tau}, \boldsymbol{a}) + V_{tot}(\boldsymbol{\tau}, \boldsymbol{a}) = \begin{cases} 0 & \boldsymbol{a} = \bar{\boldsymbol{a}} \\ \geq 0 & \boldsymbol{a} \neq \bar{\boldsymbol{a}} \end{cases} \tag{24}$$

Based on this theory, three networks are built: individual Q_i, Q_{tot}, and the joint regularizer V_{tot} and three loss functions are demonstrated to train the networks. The local network at each agent is just a

regular value-based network, with the local observation which provides the Q-value of all possible actions and runs locally at the execution time. Both Q_{tot} and the regularizer networks use hidden features from the individual value-based network to help sample efficiency. In the experimental analysis, the comparisons of QTRAN with VDN and QMIX on Multi-domain Gaussian Squeeze (HolmesParker et al. 2014) and modified predator-prey (Stone and Veloso 2000) is provided.

Within the cooperative setting with the absence of joint reward, the reward-shaping idea can be applied too. Specifically, assume at time step t agent i observes its own local reward r_i^t. In this setting, Mguni et al. (2018) considers a multi-agent problem in which each agent observes the full state, takes its local action based on a stochastic policy. A general reward-shaping algorithm for the multi-agent problem is discussed and proof for obtaining the Nash equilibrium is provided. In particular, a meta-agent (MA) is introduced to modify the agent's reward functions to get the convergence to an efficient Nash equilibrium solution. The MA initially does not know the parametric reward modifier and learns it through the training. Specifically, MA wants to find the optimal variables w to reshape the reward functions of each agent, though it only observes the corresponding reward of chosen w. With a given w, the MARL algorithm can converge while the agents do not know anything about the MA function. The agents only observe the assigned reward by MA and use it to optimize their own policy. Once all agents execute their actions and receive the reward, the MA receives the feedback and updates the weight w. Training the MA with a gradient-based algorithm is quite expensive, so in the numerical experiments, a Bayesian optimization with an expected improvement acquisition function is used. To train the agents, an actor-critic algorithm is used with a two-layer neural network as the value network. The value network shares its parameters with the actor network, and an A2C algorithm (Mnih et al. 2016) is used to train the actor. It is proved that under a given condition, a reward modifier function exists such that it maximizes the expectation of the reward modifier function. In other words, a Markov-Nash equilibrium (M-NE) exists in which each agent follows a policy that provides the highest possible value for each agent. Then, convergence to the optimal solution is proved under certain conditions. To demonstrate the performance of their algorithm, a problem with 2000 agents is considered in which the desire location of agents changes through time.

7 Consensus

The idea of the centralized critic, which is discussed in Section 5, works well when there exists a small number of agents in the communication network. However, with increasing the number of agents, the volume of the information might overwhelm the capacity of a single unit. Moreover, in the sensor-network applications, in which the information is observed across a large number of scattered centers, collecting all this local information to a centralized unit, under some limitations

such as energy limitation, privacy constraints, geographical limitations, and hardware failures, is often a formidable task. One idea to deal with this problem is to remove the central unit and allow the agents to communicate through a sparse network, and share the information with only a subset of agents, with the goal of reaching a consensus over a variable with these agents (called neighbors). Besides the numerous real-world applications of this setting, this is quite a fair assumption in the applications of MARL (Zhang et al. 2018c, Jiang et al. 2020). By limiting the number of neighbors to communicate, the amount of communication remains linear in the order of the number of neighbors. In this way, each agent uses only its local observations, though uses some shared information from the neighbors to stay tuned with the network. Further, applying the consensus idea, there exist several works which prove the convergence of the proposed algorithms when the linear approximators are utilized. In the following, we review some of the leading and most recent papers in this area.

Varshavskaya et al. (2009) study a problem in which each agent has a local observation, executes its local policy, and receives a local reward. A tabular policy optimization agreement algorithm is proposed which uses Boltzmann's law (similar to soft-max function) to solve this problem. The agreement (consensus) algorithm assumes that an agent can send its local reward, a counter on the observation, and the taken action per observation to its neighbors. The goal of the algorithm is to maximize the weighted average of the local rewards. In this way, they guarantee that each agent learns as much as a central learner could, and therefore converges to a local optimum.

In Kar et al. (2013b,a) the authors propose a decentralized multi-agent version of the tabular Q-learning algorithm called $\mathcal{QD}$-learning. In this paper, the global reward is expressed as the sum of the local rewards, though each agent is only aware of its own local reward. In the problem setup, the authors assume that the agents communicate through a time-invariant un-directed weakly connected network, to share their observations with their direct neighbors. All agents observe the global state and the global action, and the goal is to optimize the network-averaged infinite horizon discounted reward. The $\mathcal{QD}$-learning works as follows. Assume that we have N agents in the network and agent i can communicate with its neighbors $\mathcal{N}_i$. This agent stores $Q_i(s,a)$ values for all possible state-action pairs. Each update for the agent i at time t includes the regular Q-learning plus the deviation of the Q-value from its neighbor as below:

$$Q_i^{t+1}(s,a) = Q_i^t(s,a) + \alpha_{s,a}^t \left(r_i(s_t,a_t) + \gamma \min_{a' \in \mathcal{A}} {Q_i}^t(s_{t+1},a') - {Q_i}^t(s,a) \right) \\ -\beta_{s,a}^t \sum_{j \in \mathcal{N}_i^t} \left(Q_i^t(s,a) - Q_j^t(s,a) \right), \tag{25}$$

where $\mathcal{A}$ is the set of all possible actions, and $\alpha_{s,a}^t$ and $\beta_{s,a}^t$ are the stepsizes of the $\mathcal{QD}$-learning algorithm. It is proved that this method converges to the optimal Q-values in an asymptotic behavior

under some specific conditions on the step-sizes. In other words, under the given conditions, they prove that their algorithm obtains the result that the agent could achieve if the problem was solved centrally.

In Pennesi and Paschalidis (2010) a distributed Actor-Critic (D-AC) algorithm is proposed under the assumption that the states, actions, and rewards are local for each agent; however, each agent's action does not change the other agents' transition models. The critic step is performed locally, meaning that each agent evaluates its own policy using the local reward receives from the environment. In particular, the state-action value function is parameterized using a linear function and the parameters are updated in each agent locally using the Temporal Difference algorithm together with the eligibility traces. The Actor step on the other hand is conducted using information exchange among the agents. First, the gradient of the average reward is calculated. Then a gradient step is performed to improve the local copy of the policy parameter along with a consensus step. A convergence analysis is provided, under diminishing step-sizes, showing that the gradient of the average reward function tends to zero for every agent as the number of iterations goes to infinity. In this paper, a sensor network problem with multiple mobile nodes has been considered for testing the proposed algorithm. In particular, there are M target points and N mobile sensor nodes. Whenever one node visits a target point a reward will be collected. The ultimate goal is to train the moving nodes in the grid such that the long-term cumulative discounted reward becomes maximized. They have been considered a 20×20 grid with three target points and sixteen agents. The numerical results prove that the reward improves over time while the policy parameters reach consensus.

Macua et al. (2018) propose a new algorithm, called Diffusion-based Distributed Actor-Critic (Diff-DAC) for single and multi-task multi-agent RL. In the setting, there are N agents in the network such that there is one path between any two agents, each is assigned either the same task or a different task than the others, and the goal is to maximize the weighted sum of the value functions over all tasks. Each agent runs its own instance of the environment with a specific task, without intervening with the other agents. For example, each agent runs a given cart-pole problem where the pole length and its mass are different for different agents. Basically, one agent does not need any information, like state, action, or reward of the other agents. Agent i learns the policy with parameters θ_i, while it tries to reach consensus with its neighbors using diffusion strategy. In particular, the Diff-DAC trains multiple agents in parallel with different and/or similar tasks to reach a single policy that performs well on average for all tasks, meaning that the single policy might obtain a high reward for some tasks but performs poorly for the others. The problem formulation is based on the average reward, average value-function, and average probability transition. Based on that, they provide a linear programming formulation of the tabular problem and provide its Lagrangian

relaxation and the duality condition to have a saddle point. A dual-ascent approach is used to find a saddle point, in which (i) a primal solution is found for a given dual variable by solving an LP problem, and then (ii) a gradient ascend is performed in the direction of the dual variables. These steps are performed iteratively to obtain the optimal solution. Next, the authors propose a practical algorithm utilizing a DNN function approximator. During the training of this algorithm, agent i first performs an update over the weights of the critic as well as the actor network using local information. Then, a weighted average is taken over the weights of both networks which assures these networks reach consensus. The algorithm is compared with a centralized training for the Cart-Pole game.

In Zhang et al. (2018c) a multi-agent problem is considered with the following setup. There exists a common environment for all agents and the global state s is available to all of them, each agent takes its own local action a_i, and the global action $a = [a_1, a_2, \cdots, a_N]$ is available to all N agents, each agent receives r_i reward after taking an action and this reward is visible only in agent i. In this setup, the agents can do time-varying random communication with their direct neighbors ($\mathcal{N}$) to share some information. Two AC-based algorithms are proposed to solve this problem. In the first algorithm, each agent has its own local approximation of the Q-function with weights w_i, though a fair approximation needs global reward r_t (not local rewards $r_t^i, \forall i = 1, \ldots, N$). To address this issue, it is assumed that each agent shares parameter w_i with its neighbors, and in this way a consensual estimate of Q_w can be achieved. To update the critic, the temporal difference is estimated by

$$\delta_i^t = r_i^{t+1} - \mu_i^t + Q(s_{t+1}, a_{t+1}, w_i^t) - Q(s_t, a_t, , w_i^t),$$

in which

$$\mu_i^t = (1 - \beta_{wt}^t)\mu_i^t + \beta_{wt} r_i^{t+1}, \tag{26}$$

i.e. the moving average of the agent i rewards with parameter β_{it}, and the new weights w_i are achieved locally. To achieve the consensus a weighted sum (with weights coming from the consensus matrix) of the parameters of the neighbors' critics are calculated as below:

$$w_i^{t+1} = \sum_{j \in \mathcal{N}_i} c_{ij} \tilde{w}_j^t. \tag{27}$$

This weighted sum provides the new weights of the critic i for the next time step. To update the actor, each agent observes the global state and the local action to update its policy; though, during the training, the advantage function requires actions of all agents, as mentioned earlier. In the critic update, the agents do not share any rewards info and neither actor policy; so, in some sense, the agents keep the privacy of their data and policy. However, they share the actions with all agents, so that the setting of the problem is pretty similar to MADDPG algorithm; although, MADDPG assumes the local observation in the actor. In the second algorithm, besides sharing the

critic weights, the critic observes the moving average estimate for rewards of the neighbor agents and uses that to obtain a consensual estimate of the reward. Therefore, this algorithm performs the following update instead of (26):

$$\tilde{\mu}_i^t = (1 - \beta_{wt}^t)\mu_i^t + \beta_{wt} r_i^{t+1},$$

in which $\tilde{\mu}_i^t = \sum_{j \in \mathcal{N}_i} c_{ij} \tilde{\mu}_j^t$. Note that in the second algorithm agents share more information among their neighbors. From the theoretical perspective, The authors provide a global convergence proof for both algorithms in the case of linear function approximation. In the numerical results, they provide the results on two examples: (i) a problem with 20 agents and |S|=20, (ii) a completely modified version of cooperative navigation (Lowe et al. 2017) with 10 agents and $|S| = 40$, such that each agent observes the full state and they added a given target landmark to cover for each agent; so agents try to get closer to the certain landmark. They compare the results of two algorithms with the case that there is a single actor-critic model, observing the rewards of all agents, and the centralized controller is updated there. In the first problem, their algorithms converged to the same return value that the centralized algorithms achieve. In the second problem, it used a neural network and with that non-linear approximation and their algorithms got a small gap compared to the solutions of the centralized version.

In Wai et al. (2018), a double-averaging scheme is proposed for the task of *policy evaluation* for multi-agent problems. The setting is following Zhang et al. (2018c), i.e., the state is global, the actions are visible to all agents, and the rewards are private and visible only to the local agent. In detail, first, the duality theory has been utilized to reformulate the multi-agent policy evaluation problem, which is supposed to minimize the mean squared projected Bellman error (MSPBE) objective, into a convex-concave with a finite-sum structure optimization problem. Then, in order to efficiently solve the problem, the authors combine the *dynamic consensus* (Qu and Li 2017) and the SAG algorithm (Schmidt et al. 2017). Under linear function approximation, it is proved that the proposed algorithm converges linearly under some conditions.

Zhang et al. (2018b) consider the multi-agent problem with continuous state and action space. The rest of the setting is similar to the Zhang et al. (2018c) (i.e., global state, global action, and local reward). Again, an AC-based algorithm is proposed for this problem. In general, for the continuous spaces, stochastic policies lead to a high variance in gradient estimation. Therefore, to deal with this issue deterministic policy gradient (DPG) algorithm is proposed in Silver et al. (2014) which requires off-policy exploration. However, in the setting of Zhang et al. (2018b) the off-policy information of each agent is not known to other agents, so the approach used in DPG (Silver et al. 2014, Lillicrap et al. 2016) cannot be applied here. Instead, a gradient update based on the expected policy-gradient (EPG) (Ciosek and Whiteson 2020) is proposed, which uses a global estimate of Q-value, approximated by the consensus update. Thus, each agent shares parameters w_i of each

Q-value estimator with its neighbors. Given these assumptions, the convergence guarantees with a linear approximator are provided and the performance is compared with a centrally trained algorithm for the same problem.

Following a similar setting as Zhang et al. (2018c), Suttle et al. (2020) propose a new distributed off-policy actor-critic algorithm, such that there exists a global state visible to all agents, each agent takes an action which is visible to the whole network, and receives a local reward which is available only locally. The main difference between this work and Zhang et al. (2018c) comes from the fact that the critic step is conducted in an off-policy setting using emphatic temporal differences $ETD(\lambda)$ policy evaluation method (Sutton et al. 2016). In particular, $ETD(\lambda)$ uses state-dependent discount factor (γ) and state-dependent bootstrapping parameter (λ). Besides, in this method there exists an interest function $f : \mathcal{S} \rightarrow \mathbb{R}^+$ that takes into account the user's interest in specific states. The algorithm steps are as the following: First, each agent performs a consensus step over the critic parameter. Since the behavior policy is different than the target policy for each agent, they apply importance sampling (Kroese and Rubinstein 2012) to re-weight the samples from the behavior policy in order to correspond them to the target policy. Then, an inner loop starts to perform another consensus step over the importance sampling ratio. In the next step, a critic update using $ETD(\lambda)$ algorithm is performed locally and the updated weights are broadcast over the network. Finally, each agent performs the actor update using local gradient information for the actor parameter. Following the analysis provided for $ETD(\lambda)$ in Yu (2015), the authors proved the convergence of the proposed method for the distributed actor-critic method when linear function approximation is utilized.

Zhang et al. (2017) propose a consensus RL algorithm, in which each agent uses its local observations as well as its neighbors within a given directed graph. The multi-agent problem is modeled as a control problem, and a consensus error is introduced. The control policy is supposed to minimize the consensus error while stabilizes the system and gets the finite local cost. A theoretical bound for the consensus error is provided, and the theoretical solution for having the optimal policy is discussed, which indeed needs environment dynamics. A practical actor-critic algorithm is proposed to implement the proposed algorithm. The practical version involves a neural network approximator via a linear activation function. The critic measures the local cost of each agent, and the actor network approximates the control policy. The results of their algorithm on leader-tracking communication problem are presented and compared with the known optimal solution.

In Macua et al. (2015), an off-policy distributed policy evaluation algorithm is proposed. In this paper, a linear function has been used to approximate the long-term cumulative discounted reward of a given policy (target policy), which is assumed to be the same for all agents, while different agents follow different policies along the way. In particular, a distributed variant of the Gradient

Temporal Difference (GTD) algorithm [2] (Sutton et al. 2009) is developed utilizing a primal-dual optimization scenario. In order to deal with the off-policy setting, they have applied the importance sampling technique. The state space, action space, and transition probabilities are the same for every node, but their actions do not influence each other. This assumption makes the problem stationary. Therefore, the agents do not need to know the state and the action of the other agents. Regarding the reward, it is assumed that there exists only one global reward in the problem. First, they showed that the GTD algorithm is a stochastic Arrow-Hurwicz [3] (Arrow et al. 1958) algorithm applied to the dual problem of the original optimization problem. Then, inspiring from Chen and Sayed (2012), they proposed a diffusion-based distributed GTD algorithm. Under sufficiently small but constant step-sizes, they provide a mean-square-error performance analysis which proves that the proposed algorithms converge to a unique solution. In order to evaluate the performance of the proposed method, a 2-D grid world problem with 15 agents is considered. Two different policies are evaluated using distributed GTD algorithm. It is shown that the diffusion strategy helps the agents to benefit from the other agents' experiences.

Considering similar setup as Macua et al. (2015), in Stanković and Stanković (2016) two multi-agent policy evaluation algorithms were proposed over a time-varying communication network. A given policy is evaluated using the samples derived from different policies in different agents (i.e. off-policy). Same as Macua et al. (2015), it is assumed that the actions of the agents do not interfere with each other. Weak convergence is provided for both algorithms.

Another variant of the distributed GTD algorithm was proposed in Lee et al. (2018). Each agent in the network is following a local policy π_i and the goal is to evaluate the global long-term reward, which is the sum of the local rewards. In this work, it is assumed that each agent can observe the global joint state. A linear function, that combines the features of the states, is used to estimate the value function. The problem is modeled as a constrained optimization problem (consensus constraint), and then following the same procedure as Macua et al. (2015), a primal-dual algorithm was proposed to solve it. A rigorous convergence analysis based on the ordinary differential equation (ODE) (Borkar and Meyn 2000) is provided for the proposed algorithm. To keep the stability of the algorithm, they add some box constraints over the variables. Finally, under diminishing step-size, they prove that the distributed GTD (DGTD) converges with probability one. One of the numerical examples is a stock market problem, where $N = 5$ different agents have different policies for trading the stocks. DGTD is utilized to estimate the average long-term discounted profit of all agents. The results are compared with a single GTD algorithm in the case the sum of the reward

[2] GTD algorithm is proposed to stabilize the TD algorithm with linear function approximation in an off-policy setting.

[3] Arrow-Hurwicz is a primal-dual optimization algorithm that performs the gradient step on the Lagrangian over the primal and dual variables iteratively

is available. The comparison results show that each agent can successfully approximate the global value function.

Cassano et al. (2021) consider two different scenarios for the policy evaluation task: (i) each agent is following a policy (behavior policy) different than others, and the goal is to evaluate a target policy (i.e. off-policy). In this case, each agent has only knowledge about its own state and reward, which is independent of the other agents' state and reward. (ii) The state is global and visible to all agents, the reward is local for each agent, and the goal is to evaluate the target team policy. They propose a Fast Diffusion for Policy Evaluation (FDPE) for the case with a finite data set, which combines off-policy learning, eligibility traces, and linear function approximation. This algorithm can be applied to both scenarios mentioned earlier. The main idea here is to apply a variance-reduced algorithm called AVRG (Ying et al. 2018) over a finite data set to get a linear convergence rate. Further, they modified the cost function to control the bias term. In particular, they use h-stage Bellman Equation to derive H-truncated λ-weighted Mean Square Projected Bellman Error ($H\lambda$- MSPBE) compare to the usual cases (e.g. Macua et al. (2015)) where they use Mean Square Projected Bellman Error (MSPBE). It is shown that the bias term can be controlled through (H, λ). Also, they add a regularization term to the cost function, which can be useful in some cases.

A distributed off-policy actor-critic algorithm is proposed in Zhang and Zavlanos (2019). In contrast to the Zhang et al. (2018c), where the actor step is performed locally and the consensus update is proposed for the critic, in Zhang and Zavlanos (2019) the critic is performed locally, and the agents asymptotically achieve consensus on the actor parameter. The state space and action space are continuous and each agent has the local state and action; however, the global state and the global action are visible to all agents. Both policy function and value function are linearly parameterized. A convergence analysis is provided for the proposed algorithm under diminishing step-sizes for both actor and critic steps. The effectiveness of the proposed method was studied on the distributed resource allocation problem.

8 Learn to Communicate

As mentioned earlier in Section 7, some environments allow the communication of agents. The consensus algorithms use the communication bandwidth to pass raw observation, policy weight/gradients, critic weight/gradients, or some combination of them. A different approach to use the communication bandwidth is to learn a communication-action (like a message) to allow agents to be able to send information that they want. In this way, the agent can learn the time for sending a message, the type of the message, and the destination agents. Usually, the communication-actions do not interfere with the environment, i.e., the messages do not affect the next state or reward. Kasai et al. (2008) proposed one of the first learning to communicate algo-

rithms, in which tabular Q-learning agents learn messages to communicate with other agents in the predator-prey environment. The same approach with a tabular RL is followed in Varshavskaya et al. (2009). Besides these early works, there are several recent papers in this area which utilize the function approximator. In this section, we discuss some of the more relevant papers in this research area.

In one of the most recent works, Foerster et al. (2016) consider a problem to learn how to communicate in a fully cooperative (recall that in a fully cooperative environment, agents share a global reward) multi-agent setting in which each agent accesses a local observation and has a limited bandwidth to communicate to other agents. Suppose that $\mathcal{M}$ and $\mathcal{U}$ denote message space and action space respectively. In each time-step, each agent takes action $u \in \mathcal{U}$ which affects the environment, and decides for action $m \in \mathcal{M}$ which does not affect the environment and only other agents observe it. The proposed algorithm follows the centralized learning decentralized execution paradigm under which it is assumed in the training time agents do not have any restriction on the communication bandwidth. They propose two main approaches to solve this problem. Both approaches use DRQN (Hausknecht and Stone 2015) to address partial observability, and disabled experience replay to deal with the non-stationarity. The input of Q-network for agent i at time t includes o_i^t, h_i^t (the hidden state of RNN), $\{u_j^{t-1}\}_j$, and $\{m_j^{t-1}\}_j$ for all $j \in \{1, \ldots, N\}$. When parameter sharing is used, i is also added in the input which helps learn specialized networks for agent i within parameter sharing. All of the input values are converted into a vector of the same size either by a look-up table or an embedding (separate embedding of each input element), and the sum of these *same-size vectors* is the final input to the network. The network returns $|\mathcal{M}| + |\mathcal{U}|$ outputs for selecting actions u and m. The network includes two layers of GRU, followed by two MLP layers, and the final layer with $|U| + |M|$ representing $|U|$ Q-values. $|M|$ is different on two algorithms and is explained in the following. First they propose *reinforced inter-agent learning* (RIAL) algorithm. To select the communication action, the network includes additional $|M|$ Q-values to select discrete action m_t^i. They also proposed a practical version of RIAL in which the agents share the policy parameters so that RIAL only needs to learn one network. The second algorithm is *differentiable inter-agent learning* (DIAL), in which the message is continuous and the message receiver provides some feedback—in form of gradient—to the message sender, to minimize the DQN loss. In other words, the receiver obtains the gradient of its Q-value w.r.t the received message and sends it back to the sender so that the sender knows how to change the message to optimize the Q-value of the receiver. Intuitively, agents are rewarded for the communication actions, if the receiver agent correctly interprets the message and acts upon that. The network also creates a continuous vector for the communication action so that there is no action selector for the communication action m_i^t, and instead, a regularizer unit discretizes it, if necessary. They provide

numerical results on the switch riddle prisoner game and three communication games with mnist dataset. The results are compared with the no-communication and parameter sharing version of RIAL and DIAL methods.

Jorge et al. (2016) extend DIAL in three directions: (i) allow communication of arbitrary size, (ii) gradually increase noise on the communication channels to make sure that the agents learn a symbolic language, (iii) agents do not share parameters. They provide the results of their algorithm on a version of "Guess Who?" game, in which two agents, namely "asking" and "answering", participate. The game is around guessing the true image that the answering agent knows, while the asking agent has n images and by asking $n/2$ questions should guess the correct image. The answering agent returns only "yes/no" answers, and after $n/2$ questions the asking agent guesses the target image. The result of their algorithm with different parameters is presented. Following a similar line, Lazaridou et al. (2017) considers a problem with two agents and one round of communication to learn an interpretable language among the sender and receiver agents. The sender receives two images, while it knows the target image, and sends a message to the receiver along with the images. If the receiver guesses the correct image, both win a reward. Thus, they need to learn to communicate through the message. Each individual agent converts the image to a vector using VGG ConvNet (Simonyan and Zisserman 2014). The sender builds a neural network on top of the input vector to select one of the available symbols (values 10 and 100 are used in the experiments) as the message. The receiver embeds the message into the same size as of the images' vector, and then through a neural network combines them together to obtain the guess. Both agents use REINFORCE algorithm (Williams 1992) to train their model and do not share their policies with each other. There is not any pre-designed meaning associated with the utilized symbols. Their results demonstrate a high success rate and show that the learned communications are interpretable. In another work in this direction, in Das et al. (2017) a fully cooperative two-agent game is considered for the task of *image guessing*. In particular, two bots, namely a questioner bot (Q-BOT) and an answerer bot (ABOT) communicate in natural language and the task for Q-BOT is to guess an unseen image from a set of images. At every round of the game, Q-BOT asks a question, A-BOT provides an answer. Then the Q-BOT updates its information and makes a prediction about the image. The action space is common among both agents consisting of all possible output sequences under a token vocabulary V, though the state is local for each agent. For A-BOT the state includes the sequence of questions and answers, the caption provided for the Q-BOT besides the image itself; while the state for Q-BOT does not include the image information. There exists a single reward for both agents in this game. Similar to Simonyan and Zisserman (2014) the REINFORCE algorithm (Williams 1992) is used to train both agents. Note that Jorge et al. (2016) allow "yes/no" actions within multiple rounds of communication, Lazaridou et al. (2017) consist of one single round with

continuous messages, and Das et al. (2017) combine them such that multiple rounds of continuous communication are allowed.

Similarly, Mordatch and Abbeel (2018b) study a joint reward problem, in which each agent observes locations and communication messages of all agents. Each agent has a given goal vector g accessed only privately (like moving to or gazing at a given location), and the goal may involve interacting with other agents. Each agent chooses one physical action (e.g., moving or gazing to a new location) and chooses one of the K symbols from a given vocabulary list. The symbols are treated as abstract categorical variables without any predefined meaning and agents learn to use each symbol for a given purpose. All agents have the same action space and share their policies. Unlike Lazaridou et al. (2017) there is an arbitrary number of agents and they do not have any predefined rules, like speaker and listener, and the goals are not specifically defined such as the correct utterance. The goal of the model is to maximize the reward while creating an interpretable and understandable language for humans. To this end, a soft penalty also is added to encourage small vocabulary sizes, which results in having multiple words to create a meaning. The proposed model uses the state variable of all agents and uses a fully connected neural network to obtain the embedding Φ_s. Similarly, Φ_c is obtained as an embedding of all messages. Then, it combines the goal of the agent i, Φ_s, and Φ_c through a fully connected neural network to obtain ψ_u and ψ_c. Then, the physical action u is equal to $\psi_u + \epsilon$ and the communication message is $c \sim G(\psi_c)$, in which $\epsilon \sim N(0, 1)$ and $G(c) = -\log(-\log(c))$ is a Gumble-softmax estimator (Jang et al. 2016). The results of the algorithm are compared with a no-communication approach in the mentioned game.

Sukhbaatar et al. (2016) consider a fully cooperative multi-agent problem in which each agent observes a local state and is able to send a continuous communication message to the other agents. They propose a model, called CommNet, in which a central controller takes the state observations and the communication messages of all agents, and runs multi-step communications to provide actions of all agents in the output. CommNet assumes that each agent receives the messages of all agents. In the first round, the state observations s_i of agent i are encoded to h_i^0, and the communication messages c_i^0 are zero. Then in each round $0 < t < K$, the controller concatenates all h_i^{t-1} and c_i^{t-1}, passes them into function $f(.)$, which is a linear layer followed by a non-linear function and obtains h_i^t and c_i^t for all agents. To obtain the actions, h_i^K is decoded to provide a distribution over the action space. Furthermore, they provide a version of the algorithm that assumes each agent only observes the messages of its neighbor. Note that, compared to Foerster et al. (2016), commNet allows multiple rounds of communication between agents, and the number of agents can be different in different episodes. The performance of CommNet is compared with independent learners, the fully connected, and discrete communication over a traffic junction and Combat game from Sukhbaatar et al. (2015).

To extend CommNet, Hoshen (2017) propose Vertex Attention Interaction Network (VAIN), which adds an attention vector to learn the importance weight of each message. Then, instead of concatenating the messages together, the weighted sum of them is obtained and used to take the action. VAIN works well when there are sparse agents who interact with each agent. They compare their solution with CommNet over several environments.

In Peng et al. (2017) the authors introduce a bi-directional communication network (BiCNet) using a recurrent neural network such that heterogeneous agents can communicate with different sets of parameters. Then a multi-agent vectorized version of AC algorithm is proposed for a combat game. In particular, there exists two vectorized networks, namely actor and critic networks which are shared among all agents, and each component of the vector represents an agent. The policy network takes the shared observation together with the local information and returns the actions for all agents in the network. The Bi-directional recurrent network is designed in a way to be served as a local memory too. Therefore, each individual agent is capable of maintaining its own internal states besides sharing the information with its neighbors. In each iteration of the algorithm, the gradient of both networks is calculated and the weights of the networks get updated accordingly using the Adam algorithm. In order to reduce the variance, they applied the deterministic off-policy AC algorithm (Silver et al. 2014). The proposed algorithm was applied to the multi-agent StarCraft combat game (Samvelyan et al. 2019). It is shown that BiCNet is able to discover several effective ways to collaborate during the game.

Singh et al. (2018) consider the multi-agent problem in which each agent has a local reward and observation. An algorithm called Individualized Controlled Continuous Communication Model (IC3Net) is proposed to learn to *what and when to communicate*, which can be applied to cooperative, competitive, and semi-cooperative environments [4]. IC3Net allows multiple continues communication cycles and in each round uses a gating mechanism to decide to communicate or not. The local observation o_i^t are encoded and passed to an LSTM model, which its weights are shared among the agents. Then, the final hidden state h_i^t of the LSTM for agent i in time step t is used to get the final policy. A Softmax function $f(.)$ over h_i^t returns a binary action to decide whether to communicate or not. Considering message c_i^t of agent i at time t, the action a_i^t and c_i^{t+1}

[4] Semi-cooperative environments are those that each agent looks for its own goal while all agents also want to maximize a common goal.

are:

$$g_i^{t+1} = f(h_i^t), \tag{28}$$

$$h_i^{t+1}, l_i^{t+1} = \texttt{LSTM}\left(e(o_i^t) + c_i^t, h_i^t, l_i^t\right), \tag{29}$$

$$c_i^{t+1} = \frac{1}{N-1} C \sum_{i \neq j} h_j^{t+1} g_j^{t+1}, \tag{30}$$

$$a_i^t = \pi(h_i^t), \tag{31}$$

in which l_i^t is the cell state of the LSTM cell, C is a linear transformator, and $e(.)$ is an embedding function. Policy π and gating function f are trained using REINFORCE algorithm (Williams 1992). In order to analyze the performance of IC3Net, predator-pray, traffic junction (Sukhbaatar et al. 2016), and StarCraft with explore and combat tasks (Samvelyan et al. 2019) are considered. The results are compared with CommNet (Sukhbaatar et al. 2016), no-communication model, and no-communication model with only global reward.

In Jaques et al. (2019), the authors aim to avoid centralized learning in multi-agent RL problems when each agent observes local state o_i^t, takes a local action, and observes local reward z_i^t from the environment. The key idea is to define a reward called intrinsic reward for influencing the other agents' actions. In particular, each agent simulates the potential actions that it can take and measures the effect on the behavior of other agents by having the selected action. Then, the actions which have a higher effect on the action of other agents will be rewarded more. Following this idea, the reward function $r_i^t = \alpha z_i^t + \beta c_i^t$ is used, where c_i^t is the casual influence reward on the other agents, α, and β are some trade-off weights. c_i^t is computed by measuring the KL difference in the policy of agent j when a_i is known or is unknown as below:

$$c_i^t = \sum_{j \neq i} \left[D_{KL} \left[p(a_j^t | a_i^t, o_i^t) || p(a_j^t | o_i^t) \right] \right] \tag{32}$$

In order to measure the influence reward, two different scenarios are considered: (i) a centralized training, so each agent observes the probability of another agent's action for a given counterfactual, (ii) modeling the other agents' behavior. The first case can be easily handled by the equation (32). In the second case, each agent is learning $p(a_j^t | a_i^t, o_i^t)$ through a separate neural network. In order to train these neural networks, the agents use the history of observed actions and the cross-entropy loss functions. The proposed algorithm is analyzed on harvest and clean-up environments and is compared with an A3C baseline and baseline which allows agents to communicate with each other. This work is partly relevant to *Theory of Mind* which tries to explain the effect of agents on each other in multi-agent settings. To see more details see Rabinowitz et al. (2018).

Das et al. (2019) propose an algorithm, called TarMAC, to learn to communicate in a multi-agent setting, where the agents *learn to what to sent* and also learn to *communicate to which agent*.

They show that the learned policy is interpretable, and can be extended to competitive and mixed environments. To make sure that the message gets enough attention from the intended agents, each agent also encodes some information in the continuous message to define the type of the agent that the message is intended for. This way, the receiving agent can measure the relevance of the message to itself. The proposed algorithm follows a centralized training with a decentralized execution paradigm. Each agent accesses local observation, observes the messages of all agents, and the goal is maximizing the team reward R, while the discrete actions are executed jointly in the environment. Each agent sends a message including two parts, the signature $(k_i^t \in R^{d_k})$ and the value $(v_i^t \in R^{d_v})$. The signature part provides the information of the intended agent to receive the message, and the value is the message. Each recipient j receives all messages and learns variable $q_j^t \in R^{d_k}$ to receive the messages. Multiplying q_j^t and k_i^t for all $i \in \{1, \ldots, N\}$ results in the attention weights of α_{ij} for all messages from agents $i \in \{1, \ldots, N\}$. Finally, the aggregated message c_i^t is the weighted sum of the message values, which the weights are the obtained attention values. This aggregated message and the local observation o_i^t are the input of the local actor. Then, a regular actor-critic model is trained which uses a centralized critic. The actor is a single layer GRU layer, and the critic uses the joint actions $\{a_1, \ldots, a_N\}$ and the hidden state $\{h_1, \ldots, h_N\}$ to obtain the Q-value. Also, the actors share the policy parameters to speed up the training, and multi-round communication is used to increase efficiency. The proposed method (with no attention, no communication version of the algorithm) is compared with SHAPES (Andreas et al. 2016), traffic junction in which they control the cars, and House3D (Wu et al. 2018) as well as the CommNet (Sukhbaatar et al. 2016) when it was possible. In the same direction of DIAL, Freed et al. (2020) proposed a centralized training and decentralized execution algorithm based on stochastic message encoding/decoding to provide a discrete communication channel that is mathematically equal to a communication channel with additive noise. The proposed algorithm allows gradients backpropagate through the channel from the receiver of the message to the sender. The base framework of the algorithm is somehow similar to DIAL (Foerster et al. 2016); although, unlike DIAL, the proposed algorithm is designed to work under (known and unknown) additive communication noises.

In the algorithm, the sender agent generates a real-valued message z and passes it to a randomized encoder in which the encoder adds a uniform noise $\epsilon \sim U(-1/M, 1/M)$ to the continues message to get $\tilde{z}$. Then, to get one of the $M = 2^C$ possible discrete messages, where it is an integer version of the message by mapping it into 2^C possible ranges. The discrete message m is sent to the receiver, where a randomized decoder tries to reconstruct the original continuous message z from m. The function uses the mapping of 2^C possible ranges to extract message $\hat{z}$, and then deducts a uniform noise to get an approximation of the original message. The uniform noise in the decoder is generated from the same distribution which the sender used to add the noise to the

message. It is proved that with this encoder/decoder, $\hat{z} = z + \epsilon'$ that mathematically is equivalent to a system where the sender sends the real-valued message to the receiver through a channel which adds a uniform noise from a known distribution to the message. In addition, they have provided another version of the encoder/decoder functions to handle the case in which the noise distribution is unknown and it is a function of the message and the state variable, i.e., $\hat{m} = P(.|m, \mathcal{S})$.

In the numerical experiments, an actor-critic algorithm is used to train the weights of the networks, in which the critic observes the full state of the system and the actors share their weights. The performance of the algorithm is analyzed on two environments: (i) hidden-goal path-finding, in which in a 2D-grid each agent is assigned with a given goal cell and needs to arrive at that goal, with 5 actions: move into four directions or stay. Each agent observes its location and the goal of other agents. So, they need to find out about the location of other agents and the location of their own goal through communication with other agents, (ii) coordinated multi-agent search, where there are two agents in a 2D-grid problem and they are able to see all goals only when they are adjacent to the goal or on the goal cell. So, the agents need to communicate with others to get some information about their goals. The results of the proposed algorithm are compared with (i) reinforced communication learning (RCL) based algorithm (like RIAL in Foerster et al. (2016) in which the communication action is treated like another action of the agent and is trained by RL algorithms) with noise, (ii) RCL without noise for all cases, (iii) real-valued message is passed to the agents, (iv) and no-communication for one of the environments.

All the papers discussed so far in this section assumed the existence of a communication message and basically, they allow each agent to learn what to send. In a different approach, Jiang and Lu (2018) fix the message type and only allows each agent to decide to start a communication with the agents in its receptive field. They consider the problem that each agent observes the local observation, takes a local action, and receives a local reward. The key idea here is that when there is a large number of agents, sharing the information of all agents and communication might not be helpful since it is hard for the agent to differentiate the valuable information from the shared information. In this case, communication might impair learning. To address this issue, an algorithm, called ATOC is proposed in which an attention unit learns when to integrate the shared information from the other agents. In ATOC, each agent encodes the local observation, i.e. $h_i^t = \mu_I(o_i^t; \theta_\mu)$ in which θ_μ is the weights of a MLP. Every T time step, agent i runs an attention unit with input h_i^t to determine whether to communicate with the agents in its receptive field or not. If it decides to communicate, a communication group with at most m collaborator is created and does not change for T time-steps. Each agent in this group sends the encoded information h_i^t to the communication channel, in which they are combined and $\tilde{h}_i^t$ is returned for each agent $i \in \mathcal{M}_\rangle$, where $\mathcal{M}_\rangle$ is the list of the selected agents for the communication channel. Then, agent i merges

$\tilde{h}_i^t$ with h_i^t, passes it to the MLP and obtains $a_i^t = \mu_{II}(h_i^t, \tilde{h}_i^t; \theta_\mu)$. Note that one agent can be added in two communication channels, and as a result, the information can be transferred among a larger number of agents. The actor and critic models are trained in the same way as the DDPG model, and the gradients of the actor (μ_{II}) are also passed in the communication channel, if relevant. Also, the difference of the Q-value with and without communication is obtained and is used to train the attention unit. Some numerical experiments on the particle environment are done and ATOC is compared with CommNet, BiCNet, and DDPG (ATOC without any communication). Their experiments involve at least 50 agents so that MADDPG algorithm could not be a benchmark.

9 Other approaches and hybrid algorithms

In this section, we discuss a few recent papers which either combine the approaches in sections 5-8 or propose a model that does not quite fit in either of the previous sections.

Schroeder de Witt et al. (2019) consider a problem in which each agent observes a local observation, selects an action which is not known to other agents, and receives a joint reward, known to all agents. Further, it is assumed that all agents access a common knowledge, and all know that any agent knows this information, and each agent knows that all agents know that all agents know it and so on. Also, there might be subgroups of the agents who share more common knowledge, and the agents inside each group use a centralized policy to take action and each agent plays its own action in a decentralized model. Typically, subgroups of the agents have more common knowledge and the selected action would result in higher performance than the actions selected by larger groups. So, having groups of smaller size would be interesting. However, there is a computational trade-off between the selecting smaller or larger subgroups since there is numerous possible combination of agents to form smaller groups. This paper proposes an algorithm to address this challenge, i.e., divide the agents to a new subgroup or take actions via a larger joint policy. The proposed algorithm, called MACKRL, provides a hierarchical RL that in each level of hierarchy decides either to choose a joint action for the subgroup or propose a partition of the agents into smaller subgroups. This algorithm is very expensive to run since the number of possible jointed-agents increases exponentially and the algorithm becomes intractable. To address this issue, a pairwise version of the algorithm is proposed in which there are three levels of hierarchies, the first for grouping agents, the second one for either action-selection or sub-grouping, and the last one for action selection. Also, a Central-V algorithm is presented for training actor and critic networks.

In Shu and Tian (2019) a different setting of the multi-agent system is considered. In this problem, a manager along with a set of self-interested agents (workers) with different skills and preferences work on a set of tasks. In this setting, the agents like to work on their preferred tasks (which may

not be profitable for the entire project) unless they offered the right bonus for doing a different task. Furthermore, the manager does not know the skills and preferences (or any distribution of them) of each individual agent in advance. The problem goal is to train the manager to control the workers by inferring their minds and assigning incentives to them upon the completion of particular goals. The approach includes three main modules. (i) Identification, which uses workers' performance history to recognize the identity of agents. In particular, the performance history of agent i is denoted by $\mathbb{P}_i = \{P_i^t = (\rho_{igb}^t) : t = 1, 2, \cdots, T\}$, where ρ_{igb}^t is the probability of worker i finishes the goal g in t steps with b bonuses. In this module, these matrices are flattened into a vector and encoded to history representation denoted by h_i. (ii) Modeling the behavior of agents. A worker's mind is modeled by its performance, intentions, and skills. In mind tracker module, the manager encodes both current and past information to updates its beliefs about the workers. Formally, let's $\Gamma_i^t = (s_i^\tau, a_i^\tau, g_i^\tau, b_i^\tau) : \tau = 1, 2, \cdots, t$ denotes the trajectory of worker i. Then mind tracker module M receives Γ_i^t as well as history representation h_i from the first module and outputs m_i as the mind for agent i. (iii) Training the manager, which includes assigning the goals and bonuses to the workers. To this end, the manager needs to have all workers as a context defined as $c^{t+1} = C(\{(s_i^{t+1}, m_i^t, h_i) : i = 1, 2, \cdots, N\})$, where C pools all workers information. Then utilizing both individual information and the context, the manager module provides the goal policy π^g and bonus policy π^b for all workers. All three modules are trained using the A2C algorithm. The proposed algorithm is evaluated in two environments: Resource Collection and Crafting in 2D Minecraft. The results demonstrate that the manager can estimate the workers' mind through monitoring their behavior and motivate them to accomplish the tasks they do not prefer.

Next, we discuss MARL in a hierarchical setting. To do so, let us briefly introduce the hierarchical RL. In this setting, the problem is decomposed into a hierarchy of tasks such that easy-to-learn tasks are at the lower level of the hierarchy and a strategy to select those tasks can be learned at a higher level of the hierarchy. Thus, in the hierarchical setting, the decisions at the high level are made less frequently than those at the lower level, which usually happens at every step. The high-level policy is mainly focused on long-run planning, which involves several one-step tasks in the low-level of the hierarchy. Following this approach, in single-agent hierarchical RL (e.g. Kulkarni et al. (2016), Vezhnevets et al. (2017)), a meta-controller at the high-level learns a policy to select the sequence of tasks and a separate policy is trained to perform each task at the low-level.

For the hierarchical multi-agent systems, two possible scenarios are synchronous and asynchronous. In the synchronous hierarchical multi-agent systems, all high-level agents take action at the same time. In other words, if one agent takes its low-level actions earlier than other agents, it has to wait until all agents finish their low-level actions. This could be a restricted assumption if the number of agents is quite large. On the other hand, there is no restriction on asynchronous

hierarchical multi-agent systems. Nonetheless, obtaining high-level cooperation in asynchronous cases is challenging. In the following, we study some recent papers in hierarchical MARL.

In Tang et al. (2018) a cooperative problem with sparse and delayed rewards is considered, in which each agent accesses a local observation, takes a local action, and submit the joint action into the environment to get the local rewards. Each agent has some low-level and high-level actions to take such that the problem of the task selection for each agent can be modeled as a hierarchical RL problem. To solve this problem, three algorithms are proposed: Independent hDQN (Ind-hDQN), hierarchical Communication networks (hCom), and hierarchical hQmix. Ind-hDQN is based on the hierarchical DQN (hDQN) (Kulkarni et al. 2016) and decomposes the cooperative problem into independent goals and then learns them in a hierarchical manner. In order to analyze Ind-hDQN, first, we describe hDQN—for the single-agent—and then explain Ind-hDQN for multi-agent setting. In hDQN, the meta-controller is modeled as a semi-MDP (SMDP) and the aim is to maximize

$$\tilde{r}_t = R(s_{t+\tau}|s_t, g_t) = r_t + \cdots + r_{t+\tau},$$

where, g_t is the selected goal by the meta-controller and τ is the stochastic number of periods to achieve the goal. Via $\tilde{r}_t$, a DQN algorithm learns the meta-controller policy. This policy decides which low-level task should be taken at each time step. Then, the low-level policy learns to maximize the goal-dependent reward $\hat{r}_t$. In Ind-hDQN it is assumed that agent i knows local observation o_i^t, its meta-controller learns policy $\pi_i(g_i^t|o_i^t)$, and in the low-level it learns policy $\hat{\pi}_i(a_i^t|g_i^t)$ to interact with the environment. The low-level policy is trained by the environment's reward signals r_i^t and the meta-controller's policy is trained by the intrinsic reward $\hat{r}_i^t$. Since Ind-hDQN trains independent agents, it can be applied to both synchronous and asynchronous settings.

In the second algorithm, named hCom, the idea of CommNet (Sukhbaatar et al. 2016) is combined with Ind-hDQN. In this way, Ind-hDQN's neural network is modified to include the average of the h^{th} hidden layers of other agents, i.e., it is added as the $(h+1)^{\text{th}}$ layer of each agent. Similar to Ind-hDQN, hCom works for both synchronous and asynchronous settings. The third algorithm, hQmix, is based on Qmix (Rashid et al. 2018) to handle the case that all agents share a joint reward r_t. To this end, the Qmix architecture is added to the meta-controller and as a result, the Qmix allows training separated Q-values for each agent. This is possible by learning Q_{tot} as is directed in the Qmix. hQmix only is applicable for synchronous settings, since Q_{tot} is estimated over joint-action of all agents. In each of the proposed algorithms, the neural network's weights of the policy are shared among the tasks that have the same input and output dimensions. Moreover, the weights of the neural network are shared among the agents for the low-level policies. Thus, only one low-level network is trained; although, it can be used for different tasks and by all agents. In addition, a new experience replay, Augmented Concurrent Experience Replay (ACER), is proposed. ACER saves

transition tuple $(o_i^t, g_i^t, \tilde{r}_i^t, \tau, o_i^{t+\tau})$ for meta-controller and saves $AE_i(t,\tau) = \{(o_i^{t+k}, g_i^t, \tilde{r}_i^{t+k}, \tau - k, o_i^{t+\tau})\}_{k=0}^{\tau-1}$ to train the low-level policy. ACER also uses the idea of Concurrent Experience Replay Trajectories (CERTs) (Omidshafiei et al. 2017) such that experiences are stored in the rows of episodes and columns of time steps to ensure availability of concurrent mini-batches. They have analyzed their algorithm in Multiagent Trash Collection tasks (an extension of environment Makar et al. (2001)) and Fever Basketball Defense. In the experiments, the low-level learning is done for several homogeneous agents so that it can be considered as a single-agnet learning problem. The results are compared with Ind-DQN and Ind-DDQN with prioritized experience replay (Schaul et al. 2016).

In the literature of MARL, there are several papers which consider the games with Nash equilibrium. In a multi-agent system, Nash equilibrium is achieved, if all agents get their own highest possible value-function and not willing to change it. Here we only discuss a few recent papers since Lanctot et al. (2017) provide a detailed review of the older papers and Yang and Wang (2020) present a full review from the game theoretical perspective on the multi-agent formulation and MARL algorithms.

Zhang et al. (2018a) discuss the coordination problem in the multi-agent cooperative domains with a fully observable state and continues actions to find the Pareto-optimal Nash-equilibrium. The paper proposes an algorithm named Sample Continuous Coordination with recursive Frequency Maximum Q-Value (SCC-rFMQ) which includes two main parts: (i) given state s, a set of discrete actions from the continues set $A_i(s)$ are selected for agent i, (ii) the action evaluation and training the policy are performed. The first phase involves selecting a set of good actions which are seen before while performs exploration. In this way, it follows a Coordination Re-sample (CR) strategy that preserves $n/3$ best previous actions for each agent. The rest of the actions are selected according to the variable probability distribution, i.e. get actions randomly via $N(a_{max}(s), \sigma)$, in which $a_{max}(s)$ is the action that gives maximum Q-value for state s. Let use $a^*(s)$ to denote the action of the best seen Q-value. If $a_{max}(s) \neq a^*(s)$, exploration rate $\sigma_i(s)$ is reset to the initial value of 1/3; otherwise if $V(s) < Q_i(s, a_{max})$, CR shrinks $\sigma_i(s)$ with a given rate; and expands it otherwise. Using the new $\sigma_i(s)$, new actions are selected with $N(a_{max}, \sigma)$ and resets $A_i(s)$ and $Q(s,a)$.

Beside CR, Zhang et al. (2018a) also utilize rFMQ (Matignon et al. 2012), which extends Q-Learning with the frequency value $F(s,a)$. In this way, in addition to $Q(s,a)$, $Q_{max}(s,a)$ is also obtained and updated through the learning procedure. To select actions, an evaluation-value function $E(s,a)$ is obtained to run greedily such that $E(s,a) = (1 - F(s,a))Q(s,a) + F(s,a)Q_{max}(s,a)$ and $F(s,a)$ estimates the percentage of time that action a results in observing the maximum reward, for state s. The estimation of the frequency $F(s,a)$ is recursively updated via a separate learning rate. In order to show the effectiveness of SCC-rFMQ, the results of a climbing game,

Category	Problem	Goal	Algorithm
Web service	Task scheduling for web services Wang et al. (2016a)	Minimize time and cost	Tabular Q-learning
Traffic control	Control multi traffic signals Prabuchandran et al. (2014)	Minimize queue on a neighborhood	Tabular Q-learning
Traffic control	Control multi-intersection traffic signal Wei et al. (2019a)	Minimize total travel time	IQL
Traffic control	Control multi-intersection traffic signal Gong et al. (2019)	Minimize cumulative delay	IQL with DDQN
Traffic control	Control multi-intersection traffic signals Chu et al. (2019)	Minimize queue	IAC
Traffic control	Control multi-intersection's traffic signal Wei et al. (2019b)	Minimize queue	AC with attention
Traffic control	Control single and multi-intersection's traffic signal Zheng et al. (2019)	Minimize queue	IQL with ApeX-DQN
Traffic control	Ride-sharing management Lin et al. (2018)	Improve resource utilization	Q-learning and AC
Traffic control	Air-traffic control Brittain and Wei (2019)	Conflict resolution	CTDE A2C
Traffic control	Bike re-balancing problem Xiao (2018)	Improve trips frequency & bike usage	tabular Q-learning
Resource allocation	Online resource allocation Wu et al. (2011), Wu and Xu (2018)	maximize the utility of server	tabular Q-learning
Resource allocation	Packet routing in wireless sensor networks Ye et al. (2015)	Minimize consumed energy	Q-learning
Robot path planning	Multi agent path finding with static obstacles Sartoretti et al. (2019a)	Find the shortest path	IAC with A3C
Robot path planning	Multi Agent path finding with dynamic obstacles Wang et al. (2020a)	Find the shortest path	Double DQN
Production systems	Production control, job-shop scheduling Dittrich and Fohlmeister (2020)	Minimize average cycle time	DQN
Production systems	Transportation in semiconductor fabrication Ahn and Park (2021)	Minimize retrieval time	AC
Image classification	Image classification with swarm of robots Mousavi et al. (2019)	Minimizing classification error	REINFORCE
Stock market	Liquidating of a large amount of stock Bao and Liu (2019)	Maximize the liquidation sell value	DDPG
Stock market	Buy or sell stocks Lee et al. (2007)	Maximize the profit	Q-learning
Maintenance planning	Maintenance management Andriotis and Papakonstantinou (2019)	Minimize life-cycle cost	AC based

Table 2: A summary of applications for multi-agent problems with MARL algorithms which are reviewed in this paper.

a two-player game with Nash equilibrium are presented, as well as the boat navigation problem with two actions. SCC-rFMQ is compared with MADDPG, Sequential Monte Carlo Methods (SMC) (Lazaric et al. 2008), and rFMQ (Matignon et al. 2012). SMC (Lazaric et al. 2008) itself is an actor-critic algorithm with continues action space. In this algorithm, the actor takes action randomly such that the probability of extraction of each action is equal to the importance weight of the action. Also, the critic approximates an action-value function based on the observed reward of the playing action. Then, based on the function, the actor updates the policy distribution. In this way, for each state, the actor provides a probability distribution over the continuous action space and based on the importance of sampling, the action is selected.

10 Applications

The multi-agent problem and MARL algorithms have numerous applications in the real world. In this section, we review some of the application-oriented papers, in which the problem is modeled as a multi-agent problem and an MARL is utilized to solve that. The main focus will be on describing the problem and what kind of approach is used to solve the problem so that there are not much of technical details about the problem and algorithm. Table 2 provides a summary of some of the iconic reviewed papers in this section. As it is shown, the IQL approach is the most utilized approach in the application papers. For more details about each paper see the corresponding section.

10.1 Web Service Composition

A multi-agent Q-learning algorithm has been proposed in Wang et al. (2016a) for *dynamic web service composition* problem. In this problem, there exists a sequence of tasks that need to be done in order to accomplish the web service composition problem. The key idea in this work is to decompose the main task into independent sub-tasks. Then a tabular-based Q-learning algorithm is applied to find the optimal strategy for the sub-task. In addition, to improve the efficiency of the proposed method they proposed the experience sharing strategy. In this case, there exists a supervisor agent, who is responsible to communicate with all other agents to spread the existing knowledge in a particular agent among all other ones.

10.2 Traffic Control

Prabuchandran et al. (2014) propose a tabular Q-learning algorithm to control *multiple traffic signals* on neighbor junctions to maximize the traffic flow. Each agent (the traffic light) accesses the local observations, including the number of lanes and the queue length at each lane, decides about the green light duration, and shares the queue length with its neighbors. The cost of each agent is the average of the queue length at its neighbors so that it tries to minimize the queue length of its own queue as well as all its neighbors. The algorithm is compared with two classical approaches in a simulated environment of two areas with 9 and 12 junctions in India. A similar problem with the RL approach is studied in Abdoos et al. (2011). Also, recently Zhang et al. (2019a) provided CityFlow, a new traffic-signal environment to be used for MARL researches.

CityFlow has been used in many traffic signal control problems. In an intersection, there are some predefined sets of phases—determined based on the structure of the intersection—and the goal of the problem can be translated into deciding about the sequence of these phases to minimize the total travel of all vehicles. However, total travel time is not a direct function of state and actions in an intersection, so that usually auxiliary objective functions like minimizing queue length, waiting time, or delay time are considered. A variety of traffic statistics such as the number of moving/waiting cars in each lane, the queue length, the number of waiting cars in each lane, etc., can be used as the state s_t. The action set is usually defined as the set of all possible phases. Typically, the reward is defined as a combination of several components such as queue length, the waiting time of the cars, intersection pressure, etc. See Wei et al. (2019c) for a detailed review.

Wei et al. (2019a) consider the multi-intersection traffic signal control problem and propose an IQL type algorithm to solve it. Each intersection is considered as an RL agent, which observes the current phase, the number of cars on the outgoing road, and the number of cars in each segment of the incoming road. The action is to decide about the next active phase, and the reward of each intersec-

tion is the negative of the corresponding pressure. There is no parameter sharing among the agents, and each agent trains its own weights. Numerical experiments on several synthetic and real-world traffic cases are conducted to show the performance of the algorithm. In a similar paper, Gong et al. (2019) consider the same problem and proposes an IQL based algorithm. To obtain the state, first, each intersection is divided into several chunks to build a matrix in which each chunk includes a binary variable indicating the existence of a vehicle. Then, to get the state of each intersection, the matrix of the considered intersection and its upstream and downstream intersections are obtained and concatenated together to mitigate the complexity of the multi-agent problem. The reward is defined as the difference between the waiting times of all vehicles between two consecutive cycles, and the action is the next phase to run. The goal of the model is to minimize the cumulative delay of all vehicles. To solve the problem an IQL approach is proposed in which the agents are trained with the double dueling deep Q network (Wang et al. 2016c) algorithm, where a CNN network along with an FC layer is used to obtain the advantage values. To explore the performance of the algorithm, a commercial traffic simulator, Aimsun Next, is used as the environment and the real-world data from Florida is used in which eight traffic signals are controlled by the proposed algorithm.

Cooperation among the agents plays a pivotal role in the traffic signal control problem since the action for every individual agent will directly influence the other agents. There have been some efforts to remedy this issue. For example, Prashanth and Bhatnagar (2010) consider a central controller that watches and controls all other agents. This strategy suffers from the curse of dimensionality. Another approach is to assume that agents could share their states among the neighbors (Arel et al. 2010). For example, Chu et al. (2019) show that sharing local information could be very helpful though keep the algorithm practical. They propose MA2C, a fully cooperative in which each intersection trains an independent advantage actor-critic where it allows sharing the observations and probability simplex of the policy with the neighbor agents. So, the agent has some information about regional traffic and can try to alleviate that rather than focusing on a self-oriented policy to reduce the traffic in a single intersection. To balance the importance of the local and shared information, a spatial discount factor is considered to scale the effect of the shared observation and rewards. Each agent represents its state by the cumulative delay of the first vehicle on the intersection from time $t-1$ to time t, as well as the number of approaching cars within a given distance to the intersection. Each agent chooses its next phase and is reward locally by the weighted sum of the queue length along each incoming lane and wait time of cars in each lane of the intersection. MA2C is evaluated on a large traffic grid with both synthetic and real-world traffic data and the results are compared with IA2C, and IQL.

Even though some algorithms considered sharing some information among the agents, still it is not known how important that information are for each agent. To address this issue Wei et al. (2019b) proposed CoLight, an attention-based RL algorithm. Each intersection learns weights for every other agent, and the weighted sum of the neighbors' state is used by each agent. In CoLight, the state includes the current one-hot-coded phase as well as the number of cars in each lane of the roads, the action is choosing the next phase, and the goal is to minimize the average queue length on each intersection. All intersections share the parameters so that only one network needs to be trained. Synthetic and real-world data-set is used to show the performance of the algorithm, including traffic network in Jinan with 12 intersections, Hangzhou with 16 intersections, and Manhattan with 196 intersections.

None of the mentioned algorithms can address a city-level problem, i.e., thousands of the intersection. The main issues are (i) the local reward maximization does not guarantee global reward, and gathering the required data is quite challenging, (ii) the action of each intersection affects the others so that the coordination is required to minimize the total travel time. To address these issues, Chacha Chen et al. (2020) proposed MPLight, an RL algorithm for large-scale traffic signal control systems. The state for each agent consists of the current phase and the 12 possible pressure values of the 12 traffic movements. The intersections with a smaller number of movements are zero-padded. The action is to choose one of eight possible phases, and the local reward is the pressure of the intersection. A DQN algorithm is proposed with parameter sharing among the intersections. Both synthetic and real-world data-sets are used, from which Manhattan with 2510 signals is the largest analyzed network.

In Zheng et al. (2019) an RL-based algorithm, called FRAP, was proposed for the traffic signal control problem. The key property of FRAP is invariancy to symmetric operations such as rotation and flip. Toward this end, two principles of competition are utilized: (i) that larger traffic movement indicates higher demand, and (ii) the line with higher traffic (demand) is prioritized to the line with lower traffic. In FRAP, the state is defined as the phase and number of vehicles at each lane, the action is choosing the next phase, and the reward is the queue length. The proposed model includes three parts: (i) phase demand modeling, which provides the embedded sum of all green-traffic movements on each phase, (ii) phase pair embedding, in which the movement-conflict and phase-demand matrices are built and embedded to the required sizes, and (iii) phase pair competition, which runs two convolutions neural network and then some fully connected layers to obtain the final Q-values to choose the action.

In a related area, Lin et al. (2018) consider ride-sharing management problem and propose a customized Q-learning and an actor-critic algorithm for this problem. The algorithms are evaluated on a built simulation with Didi Chuxinghe, which is the largest ride-sharing company in China.

The goal is to match the demand and supply to improve the utilization of transportation resources. In a similar direction, Brittain and Wei (2019) study the air-traffic control problem with a MARL algorithm to ensure the safe separation between aircraft. Each airplane, as a learning agent, shares the state information with N closest agents. Each state includes distance to the goal, speed, acceleration, distance to the intersection, and the distance to the N closest airplanes. Each agent decides about the change of speed from three possible choices and receives a penalty if it is in a close distance of another airplane. The goal of the model is to identify and address the conflicts between air-crafts in high-density intersections. The A2C algorithm is used, while a centralized training decentralized execution approach is followed. The actor and critic network share the weights. BlueSky air traffic control simulator is used as the environment and the results of a case with 30 airplanes are provided.

In Xiao (2018), a distributed tabular Q-learning algorithm was proposed for *bike rebalancing* problem. Using distributed RL, they improve the current solutions in terms of frequency of trips and the number of bikes are being moved on each trip. In the problem setup, the action is the number of bikes to move. The agents receive a positive reward if the bike stock is within a particular range, and a negative reward otherwise. Also, there exists a negative reward for every bike moves in each hour. Each agent acts independently; however, there exists a controller called a knowledge repository (KR) that shares the learning information among the agents. More specifically, the KR is designed to facilitate the transfer learning (Lazaric 2012) among the agents. Using only distributed RL the success ratio (the number of successfully rebalanced stations over the total stations) is improved about 10% to 35%. Furthermore, combining with the transfer learning, the algorithm rebalances the network 62.4% better than the one without transfer learning.

10.3 Resource Allocation

Zhang et al. (2009) consider an online *distributed resource allocation* problem, such that each agent is a server and observes the information of its neighbors. In each time step, each agent receives a task and has to decide to allocate it locally, or has to pick a neighbor and send it to that neighbor. Once the task is finished, the agent is rewarded with some utility of the task. Due to the communication bandwidth constraints, the number of tasks that each agent can send to its neighbors is limited and the goal is to cooperatively maximize the utility of the whole cluster. An algorithm based on tabular Q-learning is proposed and its results are compared with a centralized controller's result. Wu et al. (2011) also consider a similar problem and propose another value-based algorithm. Moreover, in Wu and Xu (2018) a similar problem is studied in which there are n schedulers who dispatches jobs of k customers in m machines. They also propose a value-based algorithm and use a gossip mechanism to transfer utilities among the neighbor agents. In a

similar domain, Ye et al. (2015) consider a packet routing problem in the wireless sensor networks, model it as a multi-agent problem. In this problem, the sensor data should be sent to a base station to analyze them, though usually each sensory unit has a limited capacity to store data, and there are strict communication bandwidth limits. This problem has several applications in surveillance, video recording, processing and communicating. When a packet is sent from a given sensory unit, the distance to the destination and the size of the packet determines the required energy to send the packet, and one of the goals of the system is to minimize the sum of consumed energy. They propose an MARL algorithm based on the Q-learning in which each agent selects some cooperating neighbors in a given radius and then can communicate with them. The results are compared with several classical algorithms. Within a similar domain, Dandanov et al. (2017) propose an RL algorithm to deal with the antenna tilt angle in mobile antenna, to get a compromise between the mobile coverage and the capacity of the network. The reward matrix of the problem is built and transition probability among the states are known so that optimal value for each state-action is achieved.

10.4 Robot Path Planning

Multi-agent path finding (MAPF) problem is an NP-hard problem (Ma et al. 2019, LaValle 2006). The goal is to find the path for several agents in a system with obstacles for going from given source and destinations for each agent. The algorithms to solve MAPF, in general, can be categorized into three classes: coupled, decoupled, and dynamically-coupled methods. The Coupled approaches treat MAPF as a single high-dimensional agent, which results in the exponential growth of complexity. On the other hand, decoupled approaches obtain a separate path for each agent, and adjust the paths with the goal of zero-collisions. Decoupled approaches are able to quickly obtain solutions for large problems (Leroy et al. 1999, Van Den Berg et al. 2011). One of the common approaches to path adjustments is "velocity planning" (Cui et al. 2012, Chen et al. 2017), which modifies the velocity profile of each agent along with its path to avoid collisions. Similarly, priority planning can be used to allow utilizing the fastest path and speed for the agents with higher priority (Ma et al. 2016, Cáp et al. 2013). Even though decoupled approaches can be used for a large number of agents, they are not very effective since they consider a small portion of the joint configuration space and search through low-dimensional spaces (Sanchez and Latombe 2002). Dynamically coupled approaches are proposed to solve these issues. These approached lie between coupled and decoupled approaches. For example, Conflict-Based Search (CBS) finds the optimal or semi-optimal paths without searching in high-dimensional spaces by building a set of constraints and planning for each agent (Barer et al. 2014, Sharon et al. 2015). Also, allowing on-demand growth in the search space during planning (Wagner and Choset 2015) is another approach. On the other hand, the location of obstacles may be considered static or dynamic

(Smierzchalski and Michalewicz 2005) which results in another two categorizations for algorithms, off-line and on-line planning, respectively.

Sartoretti et al. (2019a) consider MAPF problem with static obstacles and proposed PRIMAL, an IQL based approach for decentralized MAPF. PRIMAL uses RL and imitation learning (IL) to learn from an expert centralized MAPF planner. With PRIMAL, the agents use RL to learn efficient single-agent path planning, and IL is used to efficiently learn actions that can affect other agents and the whole team's benefit. This eliminates the need for explicit communication among agents during the execution. They considered a grid-world discrete state, and have defined the local state of each agent as the all information of a 10×10-cell block centered where the agent is located, along with the unit vector directing toward the goal. Actions are four possible moving (if they are allowed) plus staying still, and the reward is a penalty for each move and a higher penalty for staying still. Also, the collision results in a penalty and reaching the goal has a large positive reward. An algorithm based on A3C is used to train each agent locally, which may results in selfish behavior on each agent, leading to locally optimized actions for each agent. To address this issue three methods are proposed: (i) blocking penalty, (ii) imitation learning, (iii) randomized the size and obstacle density of the world. To use IL, a good heuristic algorithm is used to generate trajectories which are used along with the RL generated trajectories to train the model. It is shown that each of the three methods needs to be in the model and removing either of them results in a big loss in the accuracy of the model. The model is compared with the heuristic approaches which access the full observation of the environment. They also implemented PRIMAL on a small fleet of autonomous ground vehicles in a factory mockup.

Wang et al. (2020a) proposed globally guided reinforcement learning (G2RL) that uses a reward structure to generalize to any arbitrary environments. G2RL first calls a global guidance algorithm to get a path for each agent and then during the robot motion, a local double deep Q-Learning (DDQN) (Van Hasselt et al. 2016) based planner is used to generate actions to avoid the static and dynamic obstacles. The global guidance algorithm observes the whole map and all static obstacles. To make the RL agent capable of performing enough exploration, a dense reward function is proposed which encourages the agent to explore freely and tries to not force the agent to strictly follow the global guidance. The global motion planning is calculated once and remains the same during the motion. The dynamic obstacles may move in each time-step and their location become known to the agents when they are within a given distance to the agent. Similar to Sartoretti et al. (2019a), the agent has five possible actions. The goal is to minimize the number of steps which all agent need to go from the start point to the end point.

To train the RL agent, on each agent the local observation is passed into a transformer function, and the output is passed into a CNN-LSTM-FC network to get the action. In the local observation, the

static obstacle, the dynamic obstacle, and the proposed route by the global guidance are depicted with different colors. The agent uses the Double-DQN algorithm to obtain its local actions. Since an IQL-based approach is used to train the agent, it can be used in a system with any number of agents. PRIMAL is compared with several central controller type algorithms to demonstrate its performance in different cases.

10.5 Production Systems

In Dittrich and Fohlmeister (2020), a cooperative MARL is proposed for production control problem. The key idea in this paper is to use a decentralized control system to reduce the problem complexity and add the capability of real-time decision makings. However, this causes local optimal solutions. To overcome this limitation, cooperative behavior is considered for agents. To perform the idea, a central module that contains a deep Q-learning algorithm is considered. This DQN module, plus a decentralized multi-agent system (MAS) communicates to the manufacturing system (i.e., environment). The MAS module consists of two types of agents, namely, order agents and machine agents. The order agents make scheduling decisions based on the work plan, and the machine agents keep the machines information. These agents are able to collect some local data and the order agents have the capability of making some local decisions following the instructions from the DQN module. For each released order i, the subsequent g orders are grouped and denoted by G_i. The state is defined as the data related to the orders and the action represents the available machine tools for processing particular orders. The reward contains local reward and global reward. The local reward is defined in a way to encourage order agents for choosing the fastest route, where the global reward takes higher values when the deviation between the actual lead time and the target lead time is smaller. The proposed framework is tested on a job shop problem with three processing steps and the results are compared to a capacity-based solution.

One of the other studied problems with MARL is the overhead hoist transportation (OHT) problem in semiconductor fabrication (FAB) systems. Three main problems exist in FAB systems: (i) The dispatching problem looks for the assignment of idle OHTs to new loads, like moving a bundle of wafers. The goal of this problem is to minimize average delivery time or to maximize the resource utilization ratio for the material handling. On this problem, several constraints like deadlines or job priorities should be considered. (ii) The problem of determining the optimal path of the OHTs moving from a source machine to a destination machine. The goal of this problem is usually to minimize the total travel time or the total tardiness. (iii) The rebalancing problem, which aims to reallocate idle OHTs over different sections of FAB system. The goal is to minimize the time between the assignment of the load to an OHT and the start of the delivery, which is called retrieval time. Ahn and Park (2021) consider the third problem and propose a reinforcement learning algo-

rithm based on the graph neural networks to minimize the average retrieval time of the idle OHTs. Each agent is considered as the area that contains two machines to perform a process on the semiconductor. Each agent decides to move or not move an idle OHT from its zone to its neighboring zone. The local state of agent i at time t includes (i) the number of the idle OHTs at time t in zone i, (ii) the number of working idle OHTs at time t in zone i, (iii) and the number of loads waiting on zone i at time t. The observation of agent i at time t includes the s_t^j for all neighbor zones j which share their state with zone i. Then, a graph neural network is used to embed the observation of each agent to a vector of a given size. In the graph, each node represents a zone, and the node feature is the state of that zone. The edge of the graph is the number of OHTs moving from one node to another. The policy is a function of the embedded observation, including the state of the neighbor zones. An actor-critic algorithm is proposed, in which the policy parameters are shared among all the agents and the critic model uses the global state of the environment. Several numerical experiments are presented to show the effectiveness of the proposed model compared to some heuristic models.

10.6 Image Classification

In Mousavi et al. (2019) the authors show that a decentralized multi-agent reinforcement learning can be used for image classification. In their framework, multiple agents receive partial observations of the environment, communicate with the neighbors on a communication graph, and relocate to update their locally available information. Using an extension of the REINFORCE algorithm, an algorithm is proposed to update the prediction and motion planning modules in an end-to-end manner. The result on MNIST data-set is provided in which each agent only observes a few pixels of the image and has used an LSTM network to learn the policy. A similar problem and approach is followed in Mousavi et al. (2019) on the MNIST data-set.

10.7 Stock Market

Bao and Liu (2019) consider the liquidation of a large amount of stock in a limited time T. The liquidation process usually is done with the cooperation of several traders/brokers and massively impacts the market. The problem becomes more complex when other entities want to liquidate the same stock in the market. This problem can be modeled as a multi-agent system with (i) competitive objectives: when each agent wants to sell its own stock with the highest price, (ii) cooperative objective: when several agents want to cooperatively sell the stock of one customer at the highest price. The problem is modeled as multi-agent system in which each agent has to select to sell $a \in [0, 1]$ percent of stocks in each time step. If the agent selects to sell nothing, it takes the risk of dropped prices and at the end of T time periods, the trader has to sell all remaining

stocks even with zero price. An adapted version of the DDPG algorithm is proposed to solve this problem.

In a related problem, Lee et al. (2007) proposed MQ-Trader, which makes buy and sell suggestions in the stock exchange market. MQ-Trader consists of four cooperative Q-learning agents: buy and sell signal agents, which determine a binary action for buy/discard or sell/hold, respectively. These agents want to determine the right time to buy or sell stocks. The other agents, buy and sell order agents, decide about the buy price and sell price, respectively. These agents cooperate to maximize profitability. The intuition behind this system is to effectively divide the complex stock trading problem into simpler sub-problems. So, each agent needs to learn specialized knowledge for itself, i.e., buy/sell and price decisions. The state for the signal agents is represented by a matrix which is filled with a function of long time price history data. On the other hand, the order agents do not need the long history of the price and use the history of prices in the day to determine the price. The action for the order agents is to choose a best-price ratio over the moving average of the price. And, the reward is given as the obtained profit following the action. The performance of the algorithm is analyzed on KOSPI 200 which includes 200 major stocks in the Korea stock exchange market and its results are compared with some existing benchmarks.

10.8 Maintenance Management

As an application in civil engineering, Andriotis and Papakonstantinou (2019) propose a MARL algorithm for efficient maintenance management of structures, e.g. bridges, hospitals, etc. Each structure has $m > 1$ components and the maintenance plan has to consider all of them. The goal is to find the optimal maintenance plan for the whole structure, not the optimal policy for separated components. It is assumed that all components observe the global state, and a shared reward is known for all agents. An actor-critic algorithm called Deep Centralized Multi-agent Actor-Critic (DCMAC), proposed to solve this problem. DCMAC assumes that given the global state, actions of different components are conditionally independent. This way the authors deal with the non-stationary issue in multi-agent systems. Therefor, a centralized value-function is trained. Also, a centralized actor network outputs a set of actions $\{|A_1|, \ldots, |A_m|\}$, each for one component as well as one set of available actions describing the decisions for the subsystem. This algorithm particularly extends the policy gradient algorithm to the cases with a large number of discrete actions. Since the proposed algorithm is in off-policy setting, an important sampling technique is applied to deal with this issue. Under particular valid assumptions on engineering systems, the proposed algorithm can be extended to the case of Partially Observable MPD (POMDP)s. In the numerical experiments, they cover a broad range of engineering systems including (i) a simple stationary parallel series MDP system, (ii) a non-stationary system with k-out-of-n modes in both

MPD and POMDP environments, and (iii) a bridge truss system subject to non-stationary corrosion, simulated through an actual nonlinear structural model, in a POMDP environment. The results prove the effectiveness of the proposed algorithm.

11 Environments

Environments are core elements to train any non batch-MARL algorithm. Basically, the environment provides the scope of the problem and different problems/environments have been the motivation for developing the new RL algorithms. Interacting with the real world is usually expensive, time-consuming, or sometimes impossible. Thus, using the simulator of environments is the common practice. Using simulators helps to compare different algorithms and indeed provides a framework to compare different algorithms. With these motivations, several-single agent environments are developed. Among them, Arcade which provides Atari-2600 (Bellemare et al. 2013), MoJoCo (simulates the detailed physics movements of human and some animals body) (Todorov et al. 2012), OpenAI Gym which gathers these together (Brockman et al. 2016), PyGame Learning Environment (similar to Arcade) (Tasfi 2016), OpenSim (builds musculoskeletal structures of human) (Seth et al. 2011), DeepMind Lab (3D navigation and puzzle-solving) (Beattie et al. 2016), ViZDoom (3D Shooting and Navigation Doom using only the visual information) (Kempka et al. 2016), Malmo (based on Minecraft) (Johnson et al. 2016), MINOS (Home indoor 3D Navigation) (Savva et al. 2017), House3D (3D Navigation in indoor area) (Wu et al. 2018), and MazeLab (Zuo 2018) just are few to mention. Either of these environments at least include standard step and reset functions, such that `env.reset()` returns a random initial state and `env.step(`a_t`)` returns $s_{t+1}, r_t, d_t,$ `dict` in which `dict` is some additional information. This is a general structure which makes it possible to reproduce a given trajectories with a given policy.

In the multi-agent domain, there is a smaller number of available environments. In addition, there is a broad range of possible settings for sharing information among different agents. For example, some environments involve communication actions, some share the joint reward, some share the global state and each of these cases need special algorithms and not all of the algorithms in Sections 5-9 can be applied to each of these problems. Considering these limitations, there is a smaller number of environments in each setting. Among those, StarCraft II (Samvelyan et al. 2019) has achieved a lot of attention in the recent years (Foerster et al. 2017, Usunier et al. 2017, Singh et al. 2018, Foerster et al. 2018, Rashid et al. 2018, Peng et al. 2017). In this game, each agent only observes its own local information and receives a global reward, though different versions of the game with the globally observable state are also available. Finding dynamic goals also has been a common benchmark for both discrete and continuous action spaces. Multi-agent particle environment (Mordatch and Abbeel 2018a) gathers a list of navigation tasks, e.g., the predator-prey for

both discrete and continuous actions. Some of the games allow choosing a communication action too. Harvest-gathering (Jaques et al. 2019) is a similar game with communication-action. Neural MMO (Suarez et al. 2019) provides MMORPGs (Massively Multiplayer Online Role-Playing Games) environment like Pokemon in which agents learn combat and navigation while there is a large population of the same agents with the same goal. In the area of traffic management, Zhang et al. (2019a) provided CityFlow, a new traffic-signal environment to be used for MARL researches. In addition, Wu et al. (2017) introduced a framework to control cars within a mixed system of the human-like driver and AI agents. In the same direction as of those real-world like environments, Lussange et al. (2021) introduce a simulator of stock market for multi-agent systems.

In addition to the mentioned environments which are proposed by different papers, few projects gathered some of the environments together to provide a similar framework like OpenAI Gym for the multi-agent system. Jiang (2019) provide 12 environments navigation/maze-like games. Unity (Juliani et al. 2018) is a platform to develop single/multi-agent games which can be simple grid-world or quite complex strategic games with multiple agents. The resulted game can be used as an environment for training machine learning agents. The framework supports cooperative and competitive multi-agent environments. Unity gives the ability to create any kind of multi-agent environment that is intended; although it is not specifically designed for multi-agent systems. Arena (Song et al. 2020) extends the Unity engine and provides a specific platform to define and build new multi-agent games and scenarios based on the available games. The framework includes 38 multi-agent games from which 27 are new games. New scenarios or games can be built on top of these available games. Designing the games involve a GUI-based configurable social tree, and reward function can be selected from five reward scheme that are proposed to cover most of the possible cases in competitive/cooperative games. Similarly, Ray platform (Moritz et al. 2018) also recently started to support multi-agent environments, and some of the known algorithms are also added in rllib repository (Liang et al. 2017, 2018). Ray supports entering/leaving the agents from the problem which is a common case in traffic control suites.

12 Potential Research Directions

Off-policy MARL: In RL, off-policy refers to learning about a policy (called target policy), while the learning agent is behaving under a different policy (called behavior policy). This method is of great interest because it can learn about an optimal policy while it explores and also learns about multiple policies only by following a single policy. While the former advantage is helpful in both single and multi-agent settings, the latter one seems to be more fit for the multi-agent setting. In MARL literature, there exist a few algorithms utilizing the off-policy (Zhang et al.

2018b, Suttle et al. 2020, Macua et al. 2015); however, there is still room for research on off-policy in MARL algorithm design, theory, and applications.

Safe MARL: Safe RL is defined as the training of agent to learn a policy that maximizes the long-term cumulative discounted reward while ensures reasonable performance in order to respect safety constraints and avoid catastrophic situations during the training as well as execution of the policy. The main approaches in safe RL are based on introducing the risk concept in optimality conditions and regulating the exploration process to avoid undesirable actions. There is numerous research in safe RL for single-agent RL. See a comprehensive review in Garcıa and Fernández (2015). Nonetheless, the research on safe MARL is very scarce. For example, in Shalev-Shwartz et al. (2016), a safe RL algorithm is proposed for Multi-Agent Autonomous Driving. Similarly, in Diddigi et al. (2019) a framework for the constrained cooperative multi-agent games is proposed, in which to ensure the safety a constraints optimization method is utilized. To solve the problem a Lagrangian relaxation along with the actor-critic algorithm is proposed. Given the current limited research on this topic, another straightforward research direction for MARL would be the Safe MARL in order to provide more applicable policies in this setup.

Heterogeneous MARL: Most of the works we studied above are homogeneous MARL, meaning that all the agents in the network are identical in terms of ability and skill. However, in real-world applications, we likely face a multi-agent problem where agents have different skills and abilities. Therefore, an additional problem here would be how different agents should utilize the other agents' abilities to learn a more efficient policy. As a special case, consider the human-machine interaction. Particularly, humans are able to solve some RL problems very quickly using their experience and cognitive abilities. For example, in a 2D small space, they can find a very good approximation of the shortest path very quickly no matter how complex is the search space. On the other hand, machines have the ability to solve more complex problems in high-dimensional spaces. However, optimality comes at the cost of computational complexity so that oftentimes only a feasible solution is possible. The question that needs to be answered in this problem is the following: Is it possible to develop MARL algorithms that combine heterogeneous agents' abilities toward maximizing the long-term gain? Moreover, can this be done in a principled way that comes with performance guarantees?

Optimization in MARL: Without a doubt *optimization* is an indispensable part of the RL problems. Any progress in optimization methods may lead to more efficient RL and in turn MARL algorithms. In recent years, there has been a flurry of research on designing optimization algorithms for solving complex problems including nonconvex and nonsmooth optimization problems for multi-agent and distributed systems (Bianchi and Jakubowicz 2012, Di Lorenzo and Scutari 2016, Hong et al. 2017). However, the literature of MARL still lacks those algorithms. Future research

directions on MARL from the optimization perspective can be divided into two main branches: First, applying the existing optimization algorithms (or adapt them when necessary) to multi-agent problems. For instance, TRPO (Schulman et al. 2015), which has been shown to be very efficient in single-agent RL problems, might be helpful for multi-agent problems as well. Second, focusing on the theory part of the algorithms. Despite the decent performance of the numerical methods, which utilize the neural networks in MARL, there exists a huge gap between such numerical performance and some kind of convergence analysis. Therefore, this might be the time to think out of the box and focus on the theory part of the neural networks too.

Inverse MARL: One of the most vital components of RL is *reward* specification. While in some problems such as games it is trivial, in many other applications pre-specifying reward function is a cumbersome procedure and may lead to poor results. In such circumstances, modeling a skillful agent's behavior is utilized for ascertaining the reward function. This is called *Inverse Reinforcement Learning*. While this area attained significant attention in the single-agent RL problems (Arora and Doshi 2021), there is no remarkable contribution regarding the inverse RL for MARL. Therefore, a potential research avenue in MARL would be inverse MARL, how to define relevant components, address the possible challenges, and extend it to the potential applications.

Model-based MARL: Despite the numerous success stories for model-free RL and MARL, a very typical limitation of these algorithms is sample efficiency. Indeed, these algorithms require a tremendous number of samples to reach good performance. On the other hand, *model-based* RL has been shown to be very successful in a great range of applications (Moerland et al. 2020). For this type of RL algorithms, first, the environment model is learned, and then this model is utilized for prediction and control. In single-agent RL, there exists a significant amount of research regarding the model-based RL methods; see for instance Sun et al. (2019); however, their extension to MARL has not been explored widely. Therefore, investigating model-based MARL is another worthwhile research direction.

13 Conclusion

In this review, we categorize MARL algorithms into five groups, namely independent-learners, fully observable critic, value function decomposition, consensus, and learn to communicate. Then we provide an overview of the most recent papers in these classes. For each paper, first, we have highlighted the problem setup, such as the availability of global state, global action, reward, and the communication pattern among the agents. Then, we presented the key idea and the main steps of the proposed algorithm. Finally, we listed the environments which have been used for evaluating the performance of the algorithm. In addition, among the broad range of applications of MARL for

real-world problems, we picked a few representative ones and showed how MARL can be utilized for solving such complicated problems.

In Table 3 a summary the of most influential papers in each category is presented. In this summary, we have gathered the general settings of the considered problem and the proposed algorithm to show the gaps and the possible research directions. For example, the table shows that with value decomposition (VD), there is not any research that considers the local states, the local actions, and the local policies. In this table, the third column, *com*, shows the communication status, i.e., 0 means there is no communication among the agents, and 1 otherwise. In the fourth column, AC means the proposed algorithm is actor-critic based, and Q is used when the proposed algorithm is value-based. In the fifth column, *conv* stands for the convergence. Here, 1 is for the case that the convergence analysis is provided and otherwise it is 0. In the last three columns tuple (Trn, Exe) stands for (Training and Execution) and G and L are for the *global* and *local* availability respectively, and determine whether state, action, and policy of each agent is known to other agents.

References

Monireh Abdoos, Nasser Mozayani, and Ana LC Bazzan. Traffic light control in non-stationary environments based on multi agent q-learning. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1580–1585, Oct 2011. doi: 10.1109/ITSC.2011.6083114.

Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan. Optimality and approximation with policy gradient methods in markov decision processes. In *Conference on Learning Theory*, pages 64–66. PMLR, 2020.

Adrian K Agogino and Kagan Tumer. Unifying temporal and structural credit assignment problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 980–987. IEEE Computer Society, 2004.

Kyuree Ahn and Jinkyoo Park. Cooperative zone-based rebalancing of idle overhead hoist transportations using multi-agent reinforcement learning with graph representation learning. *IISE Transactions*, 0(0):1–17, 2021. doi: 10.1080/24725854.2020.1851823. URL https://doi.org/10.1080/24725854.2020.1851823.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016.

CP Andriotis and KG Papakonstantinou. Managing engineering systems with large state and action spaces through deep reinforcement learning. *Reliability Engineering & System Safety*, 191:

	Reference	Com	ComLim	AC/Q	Conv	State (Trn, Exe)	Action (Trn, Exe)	Reward (Trn, Exe)
IQL	Tan (1993)	0	0	Q	0	(L,L)	(L,L)	(G,G)
	Lauer and Riedmiller (2000)	0	0	Q	0	(G,G)	(L,L)	(G,G)
	Matignon et al. (2007)	0	0	Q	0	(G,G)	(G,L)	(G,G)
	Tampuu et al. (2017)	0	0	Q	0	(G,G)	(L,L)	(G,G)
	Omidshafiei et al. (2017)	0	0	Q	0	(G,G)	(G,L)	(G,G)
	Fuji et al. (2018)	0	0	Q	0	(G,G)	(G,L)	(G,G)
Fully Observable Critic	Wang et al. (2019)	1	1	AC	0	(G,L)	(G,L)	(L,L)
	Foerster et al. (2018)	0	0	AC	0	(G,L)	(G,L)	(G,G)
	Ryu et al. (2018)	0	0	AC	0	(G,L)	(G,L)	(L/G,L/G)
	Sartoretti et al. (2019b)	0	0	AC	0	(G,G)	(L,L)	(L,L)
	Chu and Ye (2017)	0	0	AC	0	(L,L)	(L,L)	(L/G,L/G)
	Yang et al. (2020)	0	0	AC	0	(L,L)	(L,L)	(L,L)
	Jiang et al. (2020)	1	0	Q	0	(L,L)	(L,L)	(L,L)
	Iqbal and Sha (2019)	1	0	AC	0	(G,L)	(G,L)	(G,L)
	Yang et al. (2018a)	1	0	AC/Q	1	(G,G)	(G,L)	(G,L)
	Kim et al. (2019)	1	1	AC	0	(G,L)	(G,L)	(G,G)
	Lowe et al. (2017)	0	0	AC	0	(G,L)	(G,L)	(L,L)
	Mao et al. (2019)	0	0	AC	0	(G,L)	(G,L)	(L,L)
VD	Rashid et al. (2018)	0	0	Q	0	(G,L)	(G,L)	(G,G)
	Sunehag et al. (2018)	0	0	Q	0	(L,L)	(L,L)	(G,G)
	Mguni et al. (2018)	0	0	Q	1	(L,L)	(L,L)	(G,G)
	Son et al. (2019)	0	0	Q	1	(L,L)	(L,L)	(G,G)
Consensus	Zhang et al. (2018c)	1	0	AC	1	(G,G)	(G,L)	(L,L)
	Kar et al. (2013a)	1	0	Q	1	(G,G)	(L,L)	(L,L)
	Lee et al. (2018)	1	0	Q	1	(G,G)	(L,L)	(L,L)
	Macua et al. (2015)	1	0	Q	0	(L,L)	(L,L)	(G,G)
	Macua et al. (2018)	1	0	AC	0	(L,L)	(L,L)	(L,L)
	Cassano et al. (2021)	1	0	Q	1	(L/G,L/G)	(L,L)	(L/G,L/G)
	Zhang et al. (2018b)	1	0	AC	1	(G,G)	(G,L)	(L,L)
	Zhang and Zavlanos (2019)	1	0	AC	1	(G,G)	(G,G)	(L,L)
Learn to comm	Varshavskaya et al. (2009)	1	0	AC	1	(L,L)	(L,L)	(L,L)
	Peng et al. (2017)	1	0	AC	0	(G,G)	(G,G)	(L,L)
	Foerster et al. (2016)	1	0	Q	0	(L,L)	(L,L)	(G,G)
	Sukhbaatar et al. (2016)	1	0	AC	0	(L,L)	(L,L)	(G,G)
	Singh et al. (2018)	1	0	AC	0	(L,L)	(L,L)	(L,L)
	Lazaridou et al. (2017)	1	0	AC	0	(G,G)	(L,L)	(G,G)
	Das et al. (2017)	1	0	AC	0	(L,L)	(G,G)	(L,L)

Table 3: The proposed algorithms for MARL and the relevant setting. AC stands for all actor-critic and policy gradient-based algorithms, Q represents any value-based algorithm, Com stands for communication, Com = 1 means the agents communicate directly, and Com = 0 means otherwise, ComLim stands for communication bandwidth limit, ComLim = 1 means there is a limit on the bandwidth, and ComLim = 0 means otherwise, Conv stands for convergence, and Conv = 1 means there is a convergence analysis for the proposed method, and Conv = 0 means otherwise, the tuple (Trn,Exe) shows the way that state, reward, or action are shared in (training, execution), e.g., (G,L) under state means that during the training the state is observable globally and during the execution, it is only accessible locally to each agent.

106483, 2019.

Itamar Arel, Cong Liu, Tom Urbanik, and Airton G Kohls. Reinforcement learning-based multi-agent system for network traffic signal control. *IET Intelligent Transport Systems*, 4(2):128–135, 2010.

Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, page 103500, 2021.

Joseph Arrow Arrow, Leonid Hurwicz, and Hirofumi Uzawa. *Studies in Linear and Non-linear Programming*. Stanford University Press, 1958.

Wenhang Bao and Xiao-yang Liu. Multi-agent deep reinforcement learning for liquidation strategy analysis. In *Workshops at the Thirty-Sixth ICML Conference on AI in Finance*, 2019.

Nolan Bard, Jakob N Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, et al. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, 2020.

Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Seventh Annual Symposium on Combinatorial Search*. Citeseer, 2014.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4): 819–840, 2002.

Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

Shalabh Bhatnagar, Doina Precup, David Silver, Richard S Sutton, Hamid R Maei, and Csaba Szepesvári. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems*, pages 1204–1212, 2009.

Pascal Bianchi and Jérémie Jakubowicz. Convergence of a multi-agent projected stochastic gradient algorithm for non-convex optimization. *IEEE Transactions on Automatic Control*, 58(2): 391–405, 2012.

Vivek S Borkar and Sean P Meyn. The o.d.e. method for convergence of stochastic approximation and reinforcement learning. *SIAM Journal on Control and Optimization*, 38(2):447–469, 2000.

Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

Marc Brittain and Peng Wei. Autonomous air traffic controller: A deep multi-agent reinforcement learning approach. In *Reinforcement Learning for Real Life Workshop in the 36th International Conference on Machine Learning, Long Beach*, 2019.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1*, pages 183–221. Springer, 2010.

Michal Cáp, Peter Novák, Martin Seleckỳ, Jan Faigl, and Jiff Vokffnek. Asynchronous decentralized prioritized planning for coordination in multi-robot system. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3822–3829. IEEE, 2013.

L. Cassano, K. Yuan, and A. H. Sayed. Multiagent fully decentralized value function learning with linear convergence rates. *IEEE Transactions on Automatic Control*, 66(4):1497–1512, 2021. doi: 10.1109/TAC.2020.2995814.

Hua Wei Chacha Chen, Nan Xu, Guanjie Zheng, Ming Yang, Yuanhao Xiong, Kai Xu, and Zhenhui Li. Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Jianshu Chen and Ali H Sayed. Diffusion adaptation strategies for distributed optimization and learning over networks. *IEEE Transactions on Signal Processing*, 60(8):4289–4305, 2012.

Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 285–292. IEEE, 2017.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014. URL http://aclweb.org/anthology/D/D14/D14-1179.pdf.

Jinyoung Choi, Beom-Jin Lee, and Byoung-Tak Zhang. Multi-focus attention network for efficient deep reinforcement learning. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Tianshu Chu, Jie Wang, Lara Codecà, and Zhaojian Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):1086–1095, 2019.

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

Kamil Ciosek and Shimon Whiteson. Expected policy gradients for reinforcement learning. *Journal of Machine Learning Research*, 21(52):1–51, 2020. URL http://jmlr.org/papers/v21/18-012.html.

Rongxin Cui, Bo Gao, and Ji Guo. Pareto-optimal coordination of multiple robots with safety guarantees. *Autonomous Robots*, 32(3):189–205, 2012.

Felipe Leno Da Silva and Anna Helena Reali Costa. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64:645–703, 2019.

Nikolay Dandanov, Hussein Al-Shatri, Anja Klein, and Vladimir Poulkov. Dynamic self-optimization of the antenna tilt for best trade-off between coverage and capacity in mobile networks. *Wireless Personal Communications*, 92(1):251–278, 2017.

Abhishek Das, Satwik Kottur, José MF Moura, Stefan Lee, and Dhruv Batra. Learning cooperative visual dialog agents with deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2951–2960, 2017.

Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. TarMAC: Targeted multi-agent communication. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1538–1546, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/das19a.html.

Sam Devlin and Daniel Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 225–232. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

Sam Devlin, Logan Yliniemi, Daniel Kudenko, and Kagan Tumer. Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the 2014 international confer-*

ence on Autonomous agents and multi-agent systems, pages 165–172. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

Paolo Di Lorenzo and Gesualdo Scutari. Next: In-network nonconvex optimization. *IEEE Transactions on Signal and Information Processing over Networks*, 2(2):120–136, 2016.

Raghuram Bharadwaj Diddigi, Sai Koti Reddy Danda, Shalabh Bhatnagar, et al. Actor-critic algorithms for constrained multi-agent reinforcement learning. *arXiv preprint arXiv:1905.02907*, 2019.

Marc-André Dittrich and Silas Fohlmeister. Cooperative multi-agent system for production control using reinforcement learning. *CIRP Annals*, 69(1):389–392, 2020.

Adam Eck, Leen-Kiat Soh, Sam Devlin, and Daniel Kudenko. Potential-based reward shaping for finite horizon online pomdp planning. *Autonomous Agents and Multi-Agent Systems*, 30(3): 403–445, 2016.

William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, pages 3061–3071. PMLR, 2020.

Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1146–1155. JMLR. org, 2017.

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

Benjamin Freed, Guillaume Sartoretti, Jiaheng Hu, and Howie Choset. Communication learning via backpropagation in discrete channels with unknown noise. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7160–7168, 2020.

Taiki Fuji, Kiyoto Ito, Kohsei Matsumoto, and Kazuo Yano. Deep multi-agent reinforcement learning using dnn-weight evolution to optimize supply chain performance. In *Proceedings of the 51st Hawaii International Conference on System Sciences*, page 8, 2018.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

Thomas Gabel and Martin Riedmiller. On a successful application of multi-agent reinforcement learning to operations research benchmarks. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 68–75. IEEE, 2007.

Qitong Gao, Davood Hajinezhad, Yan Zhang, Yiannis Kantaros, and Michael M Zavlanos. Reduced variance deep reinforcement learning with temporal logic specifications. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 237–248. ACM, 2019.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Mevludin Glavic, Raphaël Fonteneau, and Damien Ernst. Reinforcement learning for electric power system decision and control: Past considerations and perspectives. *IFAC-PapersOnLine*, 50(1):6918–6927, 2017.

Yaobang Gong, Mohamed Abdel-Aty, Qing Cai, and Md Sharikur Rahman. Decentralized network level adaptive signal control by multi-agent deep reinforcement learning. *Transportation Research Interdisciplinary Perspectives*, 1:100020, 2019.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

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

Magnus Rudolph Hestenes, Eduard Stiefel, et al. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC, 1952.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Chris HolmesParker, Matthew E. Taylor, Yusen Zhan, and Kagan Tumer. Exploiting structure and agent-centric rewards to promote coordination in large multiagent systems. In *Adaptive and Learning Agents Workshop*, 2014.

Mingyi Hong, Davood Hajinezhad, and Ming-Min Zhao. Prox-pda: The proximal primal-dual algorithm for fast distributed nonconvex optimization and learning over networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1529–1538. JMLR. org, 2017.

Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2701–2711. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6863-vain-attentional-multi-agent-predictive-modeling.pdf.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2961–2970, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/iqbal19a.html.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2016.

Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, Dj Strouse, Joel Z. Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3040–3049, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/jaques19a.html.

Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. In *Advances in Neural Information Processing Systems*, pages 7254–7264, 2018.

Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HkxdQkSYDB.

Shuo Jiang. Multi-Agent Reinforcement Learning Environment. https://github.com/Bigpig4396/Multi-Agent-Reinforcement-Learning-Environment, 2019. Accessed: 2019-07-28.

Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *IJCAI*, pages 4246–4247, 2016.

Emilio Jorge, Mikael Kågebäck, Fredrik D Johansson, and Emil Gustavsson. Learning to play guess who? and inventing a grounded language as a consequence. *arXiv preprint arXiv:1611.03218*, 2016.

Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

Soummya Kar, José MF Moura, and H Vincent Poor. $\mathcal{QD}$-learning: A collaborative distributed strategy for multi-agent reinforcement learning through consensus + innovations. *IEEE Transactions on Signal Processing*, 61(7):1848–1862, 2013a.

Soummya Kar, José MF Moura, and H Vincent Poor. Distributed reinforcement learning in multi-agent networks. In *2013 5th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 296–299. IEEE, 2013b.

Tatsuya Kasai, Hiroshi Tenmoto, and Akimoto Kamiya. Learning of communication codes in multi-agent reinforcement learning problem. In *2008 IEEE Conference on Soft Computing in Industrial Applications*, pages 1–6. IEEE, 2008.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, pages 341–348, Santorini, Greece, Sep 2016. IEEE. The best paper award.

Daewoo Kim, Sangwoo Moon, David Hostallero, Wan Ju Kang, Taeyoung Lee, Kyunghwan Son, and Yung Yi. Learning to schedule communication in multi-agent reinforcement learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SJxu5iR9KQ.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Dirk P Kroese and Reuven Y Rubinstein. Monte carlo methods. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(1):48–58, 2012.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203, 2017.

Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer, 2000.

Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.

Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In *Advances in neural information processing systems*, pages 833–840, 2008.

Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. Multi-agent cooperation and the emergence of (natural) language. In *ICLR*, 2017.

Donghwan Lee, Hyung-Jin Yoon, and Naira Hovakimyan. Primal-dual algorithm for distributed reinforcement learning: Distributed GTD. *2018 IEEE Conference on Decision and Control (CDC)*, pages 1967–1972, 2018.

Jae Won Lee, Jonghun Park, O Jangmin, Jongwoo Lee, and Euyseok Hong. A multiagent approach to q-learning for daily stock trading. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 37(6):864–877, 2007.

Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 464–473. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

Stephane Leroy, Jean-Paul Laumond, and Thierry Siméon. Multiple path coordination for mobile robots: A geometric algorithm. In *IJCAI*, volume 99, pages 1118–1123, 1999.

Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Joseph Gonzalez, Ken Goldberg, and Ion Stoica. Ray RLlib: A composable and scalable reinforcement learning library. In *Deep Reinforcement Learning symposium (DeepRL @ NeurIPS)*, 2017.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/liang18b.html.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR (Poster)*, 2016.

Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1774–1783. ACM, 2018.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Zachary C Lipton, Jianfeng Gao, Lihong Li, Xiujun Li, Faisal Ahmed, and Li Deng. Efficient exploration for dialog policy learning with deep bbq networks & replay buffer spiking. *CoRR abs/1608.05081*, 2016.

Boyi Liu, Qi Cai, Zhuoran Yang, and Zhaoran Wang. Neural trust region/proximal policy optimization attains globally optimal policy. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/227e072d131ba77451d8f27ab9afdfb7-Paper.

Ruishan Liu and James Zou. The effects of memory replay in reinforcement learning. In *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 478–485. IEEE, 2018.

Ying Liu, Brent Logan, Ning Liu, Zhiyuan Xu, Jian Tang, and Yangzhi Wang. Deep reinforcement learning for dynamic treatment regimes on medical registry data. In *2017 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 380–385. IEEE, 2017.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6382–6393, 2017.

Johann Lussange, Ivan Lazarevich, Sacha Bourgeois-Gironde, Stefano Palminteri, and Boris Gutkin. Modelling stock markets by multi-agent reinforcement learning. *Computational Economics*, 57(1):113–147, 2021.

Hang Ma, Craig Tovey, Guni Sharon, TK Kumar, and Sven Koenig. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Hang Ma, Daniel Harabor, Peter J Stuckey, Jiaoyang Li, and Sven Koenig. Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7643–7650, 2019.

Sergio Valcarcel Macua, Jianshu Chen, Santiago Zazo, and Ali H Sayed. Distributed policy evaluation under multiple behavior strategies. *IEEE Transactions on Automatic Control*, 60(5):1260–1274, May 2015. ISSN 0018-9286. doi: 10.1109/TAC.2014.2368731.

Sergio Valcarcel Macua, Aleksi Tukiainen, Daniel García-Ocaña Hernández, David Baldazo, Enrique Munoz de Cote, and Santiago Zazo. Diff-dac: Distributed actor-critic for average multitask deep reinforcement learning. In *Adaptive Learning Agents (ALA) Conference*, 2018.

Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the fifth international conference on Autonomous agents*, pages 246–253. ACM, 2001.

Hangyu Mao, Zhengchao Zhang, Zhen Xiao, and Zhibo Gong. 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*, pages 1108–1116. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

Laëtitia Matignon, Guillaume Laurent, and Nadine Le Fort-Piat. Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'07.*, pages 64–69, 2007.

Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, 2012.

David Mguni, Joel Jennings, Sergio Valcarcel Macua, Sofia Ceppi, and Enrique Munoz de Cote. Controlling the crowd: Inducing efficient equilibria in multi-agent systems. In *Advances in Neural Information Processing Systems 2018 MLITS Workshop*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *Advances in Neural Information Processing Systems, Deep Learning Workshop*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018a.

Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018b.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 561–577, 2018.

H. K. Mousavi, M. Nazari, M. Takáč, and N. Motee. Multi-agent image classification via reinforcement learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5020–5027, 2019. doi: 10.1109/IROS40897.2019.8968129.

Hossein K. Mousavi, Guangyi Liu, Weihang Yuan, Martin Takác, Héctor Muñoz-Avila, and Nader Motee. A layered architecture for active perception: Image classification using deep reinforcement learning. *CoRR*, abs/1909.09705, 2019. URL http://arxiv.org/abs/1909.09705.

Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9839–9849, 2018.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. URL http://dl.acm.org/citation.cfm?id=645528.657613.

Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 50(9):3826–3839, 2020.

Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2681–2690. JMLR. org, 2017.

Afshin Oroojlooyjadid, MohammadReza Nazari, Lawrence V. Snyder, and Martin Takáč. A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management*, 0(0):null, 0. doi: 10.1287/msom.2020.0939. URL https://doi.org/10.1287/msom.2020.0939.

Ling Pan, Qingpeng Cai, Qi Meng, Wei Chen, and Longbo Huang. Reinforcement learning with dynamic boltzmann softmax updates. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1992–1998. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/276. URL https://doi.org/10.24963/ijcai.2020/276. Main track.

Peng Peng, Ying Wen, Yaodong Yang, Quan Yuan, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. 2017.

Paris Pennesi and Ioannis Ch Paschalidis. A distributed actor-critic algorithm and applications to mobile sensor network coordination problems. *IEEE Transactions on Automatic Control*, 55(2): 492–497, Feb 2010. ISSN 0018-9286. doi: 10.1109/TAC.2009.2037462.

Kirstin Petersen. Termes: An autonomous robotic system for three-dimensional collective construction. *Robotics: Science and Systems VII*, page 257, 2012.

KJ Prabuchandran, Hemanth Kumar AN, and Shalabh Bhatnagar. Multi-agent reinforcement learning for traffic signal control. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 2529–2534. IEEE, 2014.

LA Prashanth and Shalabh Bhatnagar. Reinforcement learning with function approximation for traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):412–421, 2010.

Guannan Qu and Na Li. Harnessing smoothness to accelerate distributed optimization. *IEEE Transactions on Control of Network Systems*, 5(3):1245–1260, 2017.

Neil Rabinowitz, Frank Perbet, Francis Song, Chiyuan Zhang, S. M. Ali Eslami, and Matthew Botvinick. Machine theory of mind. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4218–4227, Stockholmsmassan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/rabinowitz18a.html.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4295–4304, Stockholmsmassan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/rashid18a.html.

Heechang Ryu, Hayong Shin, and Jinkyoo Park. Multi-agent actor-critic with generative cooperative policy network. *arXiv preprint arXiv:1810.09206*, 2018.

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.

Gildardo Sanchez and J-C Latombe. Using a prm planner to compare centralized and decoupled planning for multi-robot systems. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 2, pages 2112–2119. IEEE, 2002.

Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019a.

Guillaume Sartoretti, Yue Wu, William Paivine, TK Satish Kumar, Sven Koenig, and Howie Choset. Distributed reinforcement learning for multi-robot decentralized collective construction. In *Distributed Autonomous Robotic Systems*, pages 35–49. Springer, 2019b.

Manolis Savva, Angel X. Chang, Alexey Dosovitskiy, Thomas Funkhouser, and Vladlen Koltun. MINOS: Multimodal indoor simulator for navigation in complex environments. *arXiv:1712.03931*, 2017.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICLR (Poster)*, 2016.

Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

Christian Schroeder de Witt, Jakob Foerster, Gregory Farquhar, Philip Torr, Wendelin Boehmer, and Shimon Whiteson. Multi-agent common knowledge reinforcement learning. *Advances in Neural Information Processing Systems*, 32:9927–9939, 2019.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Ajay Seth, Michael Sherman, Jeffrey A Reinbolt, and Scott L Delp. Opensim: a musculoskeletal modeling and simulation framework for in silico investigations and exchange. *Procedia Iutam*, 2:212–232, 2011.

Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

Tianmin Shu and Yuandong Tian. M^{3}RL: Mind-aware multi-agent management reinforcement learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BkzeUiRcY7.

Maria Amélia Lopes Silva, Sérgio Ricardo de Souza, Marcone Jamilson Freitas Souza, and Ana Lúcia C Bazzan. A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. *Expert Systems with Applications*, 131:148–171, 2019.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/silver14.html.

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, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Learning when to communicate at scale in multiagent cooperative and competitive tasks. In *ICLR*, 2018.

Roman Smierzchalski and Zbigniew Michalewicz. Path planning in dynamic environments. In *Innovations in Robot Mobility and Control*, pages 135–153. Springer, 2005.

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. In *Proceedings of the 31st International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2019.

Yuhang Song, Andrzej Wojcicki, Thomas Lukasiewicz, Jianyi Wang, Abi Aryan, Zhenghua Xu, Mai Xu, Zihan Ding, and Lianlong Wu. Arena: A general evaluation platform and building toolkit for multi-agent intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7253–7260, Apr. 2020. doi: 10.1609/aaai.v34i05.6216. URL https://ojs.aaai.org/index.php/AAAI/article/view/6216.

Miloš S Stanković and Srdjan S Stanković. Multi-agent temporal-difference learning with linear function approximation: Weak convergence under time-varying network topologies. In *2016 American Control Conference (ACC)*, pages 167–172, July 2016. doi: 10.1109/ACC.2016.7524910.

Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

Joseph Suarez, Yilun Du, Phillip Isola, and Igor Mordatch. Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents. *arXiv preprint arXiv:1903.00784*, 2019.

Sainbayar Sukhbaatar, Arthur Szlam, Gabriel Synnaeve, Soumith Chintala, and Rob Fergus. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*, 2015.

Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

Gita Sukthankar and Juan A Rodriguez-Aguilar. *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers*, volume 10642. Springer, 2017.

Wen Sun, Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, and John Langford. Model-based rl in contextual decision processes: Pac bounds and exponential improvements over model-free approaches. In *Conference on Learning Theory*, pages 2898–2933. PMLR, 2019.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2085–2087. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

Wesley Suttle, Zhuoran Yang, Kaiqing Zhang, Zhaoran Wang, Tamer Başar, and Ji Liu. A multi-agent off-policy actor-critic algorithm for distributed reinforcement learning. *IFAC-PapersOnLine*, 53(2):1549–1554, 2020. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2020.12.2021. URL https://www.sciencedirect.com/science/article/pii/S2405896320326562. 21th IFAC World Congress.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

Richard S. Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference

learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 993–1000, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553501. URL http://doi.acm.org/10.1145/1553374.1553501.

Richard S Sutton, A Rupam Mahmood, and Martha White. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, 17(1): 2603–2631, 2016.

Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

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

Hongyao Tang, Jianye Hao, Tangjie Lv, Yingfeng Chen, Zongzhang Zhang, Hangtian Jia, Chunxu Ren, Yan Zheng, Changjie Fan, and Li Wang. Hierarchical deep multiagent reinforcement learning. *arXiv preprint arXiv:1809.09332*, 2018.

Norman Tasfi. Pygame learning environment. https://github.com/ntasfi/PyGame-Learning-Environment 2016.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies for starcraft micromanagement. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=r1LXit5ee.

Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

Harm Van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid reward architecture for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5392–5402, 2017.

Paulina Varshavskaya, Leslie Pack Kaelbling, and Daniela Rus. Efficient distributed reinforcement learning through agreement. In *Distributed Autonomous Robotic Systems 8*, pages 367–378. Springer, 2009.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3540–3549. JMLR. org, 2017.

Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

Hoi-To Wai, Zhuoran Yang, Princeton Zhaoran Wang, and Mingyi Hong. Multi-agent reinforcement learning via double averaging primal-dual optimization. In *Advances in Neural Information Processing Systems*, pages 9649–9660, 2018.

Binyu Wang, Zhe Liu, Qingbiao Li, and Amanda Prorok. Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *IEEE Robotics and Automation Letters*, 5(4):6932–6939, 2020a.

Hongbing Wang, Xiaojun Wang, Xingguo Hu, Xingzhi Zhang, and Mingzhu Gu. A multi-agent reinforcement learning approach to dynamic service composition. *Information Sciences*, 363: 96–119, 2016a.

Lingxiao Wang, Qi Cai, Zhuoran Yang, and Zhaoran Wang. Neural policy gradient methods: Global optimality and rates of convergence. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=BJgQfkSYDS.

Rose E Wang, Michael Everett, and Jonathan P How. R-maddpg for partially observable environments and limited communication. In *Reinforcement Learning for Real Life Workshop in the 36th International Conference on Machine Learning, Long Beach*, 2019.

Shiyong Wang, Jiafu Wan, Daqiang Zhang, Di Li, and Chunhua Zhang. Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination. *Computer Networks*, 101:158–168, 2016b.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016c. PMLR. URL http://proceedings.mlr.press/v48/wangf16.html.

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

Hua Wei, Chacha Chen, Guanjie Zheng, Kan Wu, Vikash Gayah, Kai Xu, and Zhenhui Li. Presslight: Learning max pressure control to coordinate traffic signals in arterial network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 1290–1298, 2019a.

Hua Wei, Nan Xu, Huichu Zhang, Guanjie Zheng, Xinshi Zang, Chacha Chen, Weinan Zhang, Yanmin Zhu, Kai Xu, and Zhenhui Li. Colight: Learning network-level cooperation for traffic signal control. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1913–1922, 2019b.

Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. A survey on traffic signal control methods. *arXiv preprint arXiv:1904.08117*, 2019c.

Gerhard Weiß. Distributed reinforcement learning. In Luc Steels, editor, *The Biology and Technology of Intelligent Autonomous Agents*, pages 415–428, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Cathy Wu, Aboudy Kreidieh, Kanaad Parvate, Eugene Vinitsky, and Alexandre M Bayen. Flow: Architecture and benchmarking for reinforcement learning in traffic control. *arXiv preprint arXiv:1710.05465*, 2017.

Jun Wu and Xin Xu. Decentralised grid scheduling approach based on multi-agent reinforcement learning and gossip mechanism. *CAAI Transactions on Intelligence Technology*, 3(1):8–17, 2018.

Jun Wu, Xin Xu, Pengcheng Zhang, and Chunming Liu. A novel multi-agent reinforcement learning approach for job scheduling in grid computing. *Future Generation Computer Systems*, 27 (5):430–439, 2011.

Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3d environment. *arXiv preprint arXiv:1801.02209*, 2018. URL https://openreview.net/forum?id=rkaT3zWCZ.

Ian Xiao. A distributed reinforcement learning solution with knowledge transfer capability for a bike rebalancing problem. *arXiv preprint arXiv:1810.04058*, 2018.

Jiachen Yang, Alireza Nakhaei, David Isele, Kikuo Fujimura, and Hongyuan Zha. Cm3: Cooperative multi-goal multi-stage multi-agent reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1lEX04tPr.

Yaodong Yang and Jun Wang. An overview of multi-agent reinforcement learning from game theoretical perspective. *arXiv preprint arXiv:2011.00583*, 2020.

Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5571–5580, Stockholmsmassan, Stockholm Sweden, 10–15 Jul 2018a. PMLR.

Zhuoran Yang, Kaiqing Zhang, Mingyi Hong, and Tamer Başar. A finite sample analysis of the actor-critic algorithm. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 2759–2764. IEEE, 2018b.

Dayong Ye, Minjie Zhang, and Yun Yang. A multi-agent framework for packet routing in wireless sensor networks. *sensors*, 15(5):10026–10047, 2015.

Bicheng Ying, Kun Yuan, and Ali H Sayed. Convergence of variance-reduced learning under random reshuffling. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2286–2290. IEEE, 2018.

Huizhen Yu. On convergence of emphatic temporal-difference learning. In *Conference on Learning Theory*, pages 1724–1751, 2015.

Erik Zawadzki, Asher Lipson, and Kevin Leyton-Brown. Empirically evaluating multiagent learning algorithms. *arXiv preprint arXiv:1401.8074*, 2014.

Chengwei Zhang, Xiaohong Li, Jianye Hao, Siqi Chen, Karl Tuyls, and Zhiyong Feng. Scc-rfmq learning in cooperative markov games with continuous actions. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2162–2164. International Foundation for Autonomous Agents and Multiagent Systems, 2018a.

Chongjie Zhang, Victor Lesser, and Prashant Shenoy. A multi-agent learning approach to online distributed resource allocation. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

Huaguang Zhang, He Jiang, Yanhong Luo, and Geyang Xiao. Data-driven optimal consensus control for discrete-time multi-agent systems with unknown dynamics using reinforcement learning method. *IEEE Transactions on Industrial Electronics*, 64(5):4091–4100, May 2017. ISSN 0278-0046. doi: 10.1109/TIE.2016.2542134.

Huichu Zhang, Siyuan Feng, Chang Liu, Yaoyao Ding, Yichen Zhu, Zihan Zhou, Weinan Zhang, Yong Yu, Haiming Jin, and Zhenhui Li. Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In *The World Wide Web Conference*, pages 3620–3624. ACM, 2019a.

Kaiqing Zhang, Zhuoran Yang, and Tamer Basar. Networked multi-agent reinforcement learning in continuous spaces. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 2771–2776. IEEE, 2018b.

Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5872–5881, Stockholmsmassan, Stockholm Sweden, 10–15 Jul 2018c. PMLR.

Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*, 2019b.

Kaiqing Zhang, Alec Koppel, Hao Zhu, and Tamer Basar. Global convergence of policy gradient methods to (almost) locally optimal policies. *SIAM Journal on Control and Optimization*, 58(6): 3586–3612, 2020a.

Ke Zhang, Fang He, Zhengchao Zhang, Xi Lin, and Meng Li. Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach. *Transportation Research Part C: Emerging Technologies*, 121:102861, 2020b.

Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

Y. Zhang and M. M. Zavlanos. Distributed off-policy actor-critic reinforcement learning with policy consensus. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 4674–4679, 2019. doi: 10.1109/CDC40024.2019.9029969.

Guanjie Zheng, Yuanhao Xiong, Xinshi Zang, Jie Feng, Hua Wei, Huichu Zhang, Yong Li, Kai Xu, and Zhenhui Li. Learning phase competition for traffic signal control. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1963–1972, 2019.

Xingdong Zuo. mazelab: A customizable framework to create maze and gridworld environments. https://github.com/zuoxingdong/mazelab, 2018.