A Review of Cooperative Multi-Agent Deep Reinforcement Learning

Afshin OroojlooyJadid 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 mostly focused on recent papers on Multi-Agent Reinforcement Learning (MARL) than the older papers, unless it was necessary. Several ideas and papers are proposed with different notations, and we tried our best to unify them with a single notation and categorize them by their relevance. In particular, we have focused on five common approaches on modeling and solving multi-agent reinforcement learning problems: (I) independent-learners, (II) fully observable critic, (III) value function decomposition, (IV) consensus, (IV) learn to communicate. Moreover, we discuss some new emerging research areas in MARL along with the relevant recent papers. In addition, some of the recent applications of MARL in real world are discussed. Finally, a list of available environments for MARL research are provided and the paper is concluded with proposals on the possible research directions.

1 Introduction

Multi-Agent Reinforcement Learning (MARL) is a system of 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 a given goal. Basically, the agents are learnable units which want to learn policy on the fly in order to maximize the long-term reward through the interaction with the environment. Due to the complexities of the environments and the combinatorial nature of the problem, training the agents is a challenging problem and most MARL problems are categorized as NP-Hard problems [11].

With the motivation of recent success on deep reinforcement learning (RL)—super-human level control on Atari games [92], mastering the game of Go [121], chess [122], robotic [65], health care planning [82], power grid [42], routing [98], and inventory optimization [101]—several researches have been focused on deep MARL. One naive approach to train a MARL problem could be training all the agents using a centralized controller, which in fact turns to a single agent RL problem to obtain the joint action of all agents to execute in each time step. However, in this approach, the number of actions 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 for that results in the loss of the network. Moreover, usually in MARL problems, each agent accesses only to 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 MARL system such as (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 models with the cooperative goal, and most of the relevant papers with either full or partial observability are reviewed. Note that, [89, 20, 19] provide reviews on cooperative games and general MARL papers published till 2012. Also, [27] provides a survey over the utilization of transfer learning in MARL. We mostly focused on the recent works on the cooperative Deep MARL. Furthermore, MARL problems usually include large state/action spaces. Therefor, the classical tabular RL algorithm are not efficient to solve them, and we mostly focus on the approximated RL algorithms.

The rest of the paper is organized as the following: in Section 1.1 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 MARL environment is described in Section 1.2. Section 2 explains the IQL-type algorithm, Section 3 reviews the papers with a fully observable critic model, Section 4 includes the value decomposition papers, Section 5 explains the consensus approach, Section 6 reviews the learn-to-communicate approach, Section 7 explains some of the emerging research directions, Section 8 provides some applications of the MARL models in real-world, Section 9 very briefly mentions some of the available multi-agent environments, and finally Section 10 concludes the paper.

1.1 Single-Agent RL Formulation

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 it in the environment to receive reward r_t and transfer to the new state $s_{t+1} \in \mathcal{S}$. The goal of the agent is to determine a policy that maximizes the long-term reward. The value function starting from state s and following policy π denoted by $V^\pi(s)$ is given by

$$V^\pi(s) = \mathbb{E}_\pi \Big[\sum_{t=1}^\infty \gamma^{t-1} r_{t+1} | s_0 = s \Big], \tag{1}$$

where, $\gamma \in [0, 1]$ is the discounting factor. This problem is known as the Markov Decision Process (MDP) and provides a framework to study problems in a given environment. Within a given state transition probability distribution $p(s_{t+1}|s_t, a_t)$ and reward matrix $r(s_t, a_t)$, Bellman [10] showed that the following equation holds for all state s_t in any time step t:

$$V^\pi(s_t) = \sum_{a \in \mathcal{A}(s_t)} \pi(a|s_t) \sum_{s' \in \mathcal{S}} p(s'|s_t, a) \left[r(s_t, a) + \gamma V^\pi(s') \right], \tag{2}$$

where here and in the rest of the paper we use s' to denote the successor state of s.

Instead of averaging, by maximizing over the actions the optimal state-value and optimal policy can be obtained by:

$$V^{\pi^*}(s_t) = \max_a \sum_{s'} p(s'|s_t, a) \left[r(s_t, a) + \gamma V^{\pi^*}(s') \right]. \tag{3}$$

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

$$Q^{\pi^*}(s_t, a_t) = \sum_{s'} p(s'|s_t, a_t) \left[r(s_t, a_t) + \gamma \max_{a'} Q^{\pi^*}(s', a') \right]. \tag{4}$$

One can obtain an optimal policy π^* utilizing the above equations. The relevant methods are called *value-based* methods. However, in the real world usually $p(s_{t+1}|s_t, a_t)$ is not available and one cannot obtain the optimal policy using (3), or (4). In order to address this issue approximating the state-value, the Q-value, or the policy has been a common practice. Therefore, a function is used to approximate the state-value (called value-function) or the policy (called policy-function) for any possible state. Similarly, Q-value is approximated by a Q-function. The function approximator with parameters θ can be a simple linear regression model or a deep neural network. The goal is to maximize the utility function,

$$J(\theta) = \mathbb{E}_{\theta,s} \sum_{t=0}^\infty \gamma^t r(s_t, a_t; \theta). \tag{5}$$

There are two main approaches to solve this problem. In the first approach, a value function or state/value is being learned to approximate the expected sum of the discounted long-term reward. Given these approximations, the optimal policy can be achieved. In the second approach, the policy is directly learned which determines the probability of choosing an action for a given state. We explain a brief overview of the first and second approaches in Sections 1.1.1 and 1.1.2, respectively. In either of these approaches, the distribution of θ and s are unknown, so that the expectation is approximated by averaging over samples from some trajectories. Note that there is a large body of literature on the single-agent RL algorithms, and we only reviewed those algorithms which are actively used in multi-agent literature too. So, we did not explain advanced algorithms like TRPO [115] and PPO [116] which are not so common in MARL literature. For more details about other single-agent RL algorithms see [76, 134].

1.1.1 Value Approximation

Within the value approximation, a function to estimate $V(s)$ or $Q(s,a)$ is learned. It is showed that with large enough number of observation, a linear approximation converges to a local optimal [12, 135]. Nevertheless, the linear 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 approximators and especially neural networks in recent years [91, 13, 40]. Among the recent works on the value-based methods, deep Q-network (DQN) algorithm [92] got a lot of attention due to its supper human-level performance on the Atari-2600 games [9], in which it only observes the video of the game. DQN utilizes the replay memory buffer [81] and a moving target network to stabilize the training. The replay memory 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 replay memory. DQN algorithm uses the convolutional neural network (CNN) to the state s_t as the input and then pass the flatted output of the CNN to a fully connected neural network such that it includes $|A|$ outputs to approximate the Q-value for each possible action. This chain of the neural networks with weights θ is then trained by taking a mini-batch of size m to minimize loss function:

$$L(\theta) = \frac{1}{m} \sum_{i=1}^{m} (y_i - Q(s_i, a_i; \theta))^2 \tag{6}$$

$$y_i = \begin{cases} r_i, & d_t = \text{True}, \\ r_i + \gamma \max_{a'} Q(s_i', a', \theta^-) & d_t = \text{False}, \end{cases} \tag{7}$$

where, θ^- is the weights of the *target network* which is updated by θ every C iterations. Later, new DQN-based approaches proposed for solving RL problems. For example, inspired by [44] which proposed double Q-learning, the Double-DQN algorithm proposed in [144]. Similarly, another heuristic formula to obtain y_i is proposed as Dueling DQN [152]. In addition, another extension of theDQN algorithm is proposed by combining recurrent neural network with DQN. DRQN [45] is a DQN algorithm which uses an LSTM instead of a feed-forward network and is applied to a partially observable environment. In all these algorithms, 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 getting an `argmax` over the Q-values for the state. Typically, the value of ϵ is annealed during the training. To see other extensions of the value-based algorithm and other exploration algorithms, see [76].

1.1.2 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 merit function, and try to optimize this function over the policy parameter through a training process. This class of RL algorithms is called policy-based methods. For example, in the policy gradient method, a stochastic policy by parameters $\theta \in \mathbb{R}^d$ is learned. First, 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, a). Then, the stochastic policy can be obtained by:

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

which is the softmax function. The goal is to directly learn the parameter θ using a gradient-based algorithm. Let us $J(\theta)$ measures the value function for the policy π_θ starting from s_0. Then the *policy gradient theorem* provides an analytical expression for $J(\theta)$ as the following:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi(a|s; \theta) Q_{\pi_\theta}(s, a) \right], \tag{8}$$

Then the policy-based methods update the parameter θ as below:

$$\theta^{t+1} = \theta^t + \alpha \widehat{\nabla_\theta J(\theta)}, \tag{9}$$

where, $\widehat{\nabla_\theta J(\theta)}$ is an approximation of the true gradient, and α is the learning rate.

There are two main benefits of the policy-gradient method over value-based methods [134]: (i) By following the policy gradient algorithm, there is a chance of converging to a deterministic policy, though with a value-based algorithm, there is always a $\epsilon\%$ of taking a random action. (ii) With the value-based algorithms there is no way of learning a stochastic policy. This is important when the optimal policy is stochastic which might be the case in competitive

games. Considering those benefits, numerous policy gradient methods are proposed in the literature and depending on the gradient approximation strategy, they have different applications and effectiveness. We briefly review some of the most important ones which we recall them frequently in the next sections. For a review on other policy gradient-based algorithms see [135, 76].

REINFORCE algorithm is one of the first successful policy-gradient algorithms which proposed in 2000 [135]. Particularly, REINFORCE applies Monte Carlo method and uses the actual return $G := \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$ (T is the end of the episode) as an approximation for the $Q_{\pi_\theta}(s, a)$ in equation (8).

The Actor-Critic (AC) model extends REINFORCE algorithm by adding another approximator, called critic, to learn the Q-values. The critic receives state s_t and returns the approximation of the $Q_w(s_t, a_t)$. The critic is trained by calculating the TD-error $\delta_t = r(s_t, a_t) + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t)$, and updating w by $w = w + \alpha_w \delta_t \nabla_w Q_w(s_t, a_t)$, in which α_w is the critic's learning rate. A widely useful extension of REINFORCE and actor-critic algorithm is to subtract a baseline from the return (or the approximated return) to reduce the approximation variance. In this order, advantage function $A(s_t, a_t)$ can be used instead of the Q-value in which value function $V(s)$ is used as the baseline. Thus, the critic learns the value-function $V(s)$ and then the advantage $A(s_t, a_t)$ is achieved by $r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)$. Following this technique, several AC-based methods were proposed. A3C proposed in [93] uses the advantage function and runs several instances of the actor-critic model with the advantage function and asynchronously gathers the gradients and updates the weights of a master node. Afterward, the master node broadcast the new weights to the sender node and in this way all nodes are updated asynchronously. A2C [1] uses the same framework, but updates the weights in synchronous manner.

1.2 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 a to denote the vector of actions for all agents, a_{-i} is the vector of all agents except agent i, τ_i represents observation-action history of agent i, and τ 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. 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; Therefore, 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 [36] (assuming the full observability) also does not hold for the multi-agent system:

$$Q_i^*(s, a_i | \boldsymbol{\pi}_{-i}) = \sum_{\boldsymbol{a}_{-i}} \boldsymbol{\pi}_{-i}(\boldsymbol{a}_{-i}, s) \left[r(s, a_i, \boldsymbol{a}_{-i}) + \gamma \sum_{s'} P(s'|s, u_i, \boldsymbol{a}_{-i}) \max_{a_i'} Q_i^*(s, a_i') \right], \qquad (10)$$

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, since the policy of each agent changes during the training; so, one cannot use experience replay without dealing with the non-stationarity. Without experience replay, the DQN algorithm [92] and its extensions or basically neither of the approximated value-based algorithms can be directly used. 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 [11]. A similar equation to (10) 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:

$$\forall \pi_i \in \Pi_i, \forall s \in \mathcal{S}, J_i(\pi_i^*, \pi_{-i}^*) \geq J_i(\pi_i, \pi_{-i}^*); \forall i \in \{1, \ldots, N\} \tag{11}$$

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

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

policy $\hat{\pi}$ is called Pareto-optimal.

2 IQL and Extensions

The most naive approach 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 [139], 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 [153] (or it can be the newer extensions like DQN [92], DRQN [45], etc.). IQL is an appealing algorithm since (i) it does not have the scalability problem with increasing the number of agents, (ii) each agent only needs its local history of observations during the training and the inference time. The tabular IQL usually works well in practice for small size problems [89, 89]; however, in the case of function approximation, especially deep neural network (DNN) it fails. One of the main reasons for this failure is the need for the replay memory to stabilize the training with DNNs [36]. In an extension of IQL, Distributed Q-learning [70] 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:

$$q_i^{t+1}(s,a) = \begin{cases} q_i^t(s,a) & \text{if } s \neq s_t \text{ or } a \neq a_t, \\ \max\{q_i^t(s,a), r(s_t, \boldsymbol{a}_t) + \gamma \max_{a' \in A} q_i^t(\delta(s_t, \boldsymbol{a}_t), a')\} & \text{otherwise,} \end{cases} \tag{12}$$

in which $q_i^t(s,a) = \max_{\boldsymbol{a}=\{a^1, \ldots, a^N\}, a^j=a} Q(s, \boldsymbol{a})$ 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. Thus, it results in an overestimated Q-values. This issue and the course of dimensionality results in poor performance in the problems with high dimension.

Hysteretic Q-learning [88] 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 team-mate explorations. [16] also propose to use a variable learning rate to improve the performance of tabular IQL. In another extension for the IQL, [39] 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 [92] utilized *reply memory* 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 DQN algorithm instead of each single Q-learner. [138] implemented this idea and was one of the first papers which took the benefit of the neural network as a general powerful approximator in 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 [9]) 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 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.

Replay memory is one of the core elements in 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. Basically, the policy that generates the data for the replay memory is different than the current policy so that the learned policy of each agent can be misleading. In order to address this issue, [35] disables the replay memory part of the algorithm, or in [75] the old transitions are discarded and the replay memory 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, [36] propose two algorithms to stabilize the reply memory 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:

$$L(\theta_i) = \sum_{k=1}^{b} \frac{\boldsymbol{\pi}_{-i}^{t_c}(\boldsymbol{a}_{-i}, s)}{\boldsymbol{\pi}_{-i}^{t_i}(\boldsymbol{a}_{-i}, s)} \left[(y_i^{DQN} - Q(s, a_i; \theta_i))^2 \right], \tag{13}$$

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 replay memory 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 replay memory 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 [110] 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.

[100] proposes another extension of the replay memory 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 [45] and the Hysteretic Q-learning [88]. Also, to alleviate the non-stationarity of MARL, the idea of Concurrent Experience Replay Trajectories (CERTs) is proposed, in which the experience replay memory gathers the experiences of all the 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 [46], the experiences in the replay memory 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.

3   Fully Observable Critic

Non-stationarity of the environment is the main issue in MARL problems. One of the common approaches to address this issue is using a fully observable critic. The fully observable critic involves the observation 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. So, once the critic is fully observable, the non-stationarity is resolved and it can be used as a good leader for local actors. Using this idea, [83] proposes 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, each critic concatenates all states-actions together as the input and using the local reward obtains the corresponding Q-value. The critic is trained by minimizing a DQN-like loss function:

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

in which $\bar{\pi}_j$ 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 [79], DQN [92], REINFORCE [135], and TRPO [115] algorithm in a set of grounded communication environments from particle environment [43], 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 problem 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. [109] proposes 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 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 replay buffer 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 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, [25] considers multi-agent cooperative problems with N agents and proposes three actor-critic algorithms based on MADDPG. The first one assumes all the 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 each agent 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 and in sum $N + 1$ 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 [116], and PS-TRPO algorithms (PS-TRPO is the TRPO algorithm [115] with parameter sharing, see [131] for more details).

[87] presents another algorithm based on MADDPG for a cooperative game, called ATT-MADDPG which considers the same setting as in [83]. It enhances the MADDPG algorithm by adding an attention layer in the critic network. In the model, 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 model combines the MADDPG algorithm with k-head Q-value model [145]. 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 [151] 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 [94]), 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 replay memory 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 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, [51] 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 [24, 54] 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 [147], 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) [43]. MAAC is compared with COMA [38] (will be discussed shortly), MADDPG, and their updated version with SAC, as well as an independent learner with DDPG [79], over two environments: treasure collection and rover-tower. MAAC obtains better result than the other algorithms such that the performance gap becomes shorter as the number of agents increases.

[55] assumes 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 receive 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 approach. The goal is to maximize the sum of the reward of all agents. During the training, DGN allows the gradients of one agent flow to K-neighbor agents and its receptive field to stimulate cooperation. In particular, DGN consists of three phases: (i) an observation encoder, (ii) 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 [50], 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 not allow 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 $(\{o_{tot}\}_{j \in \mathbb{N}(i)}^t)$ to get an action. The results of their algorithm are compared with DQN, CommNet [130], and MeanField Q-Learning [161], on Jungle, Battle, and Routing environments.

[161] 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 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.

[38] 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 their weights with other agents, and uses gated recurrent unit (GRU) [23] to utilize the history of the observation. They present the results of their algorithm on StarCraft game [110] and compare COMA with central-V, central-QV, and two implementations of independent actor-critic (IAC).

[160] considers 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 junctions. 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 [38], 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 [38], 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 are 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 [38] and independent actor-critic learners model.

[111] extends A3C [93] 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 [104], 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 reward. During the training, all agents share the weights (as it is done in A3C) for both actor and critic models and 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 [64] a MARL 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 in order 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 to 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 u_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.

4 Value Function Factorization

Consider a cooperative MARL 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, [132] shows that naive centralized RL methods fail to find the global optimum, even if we are able to solve the problems with such 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 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. Although, it is not easy to divide the received reward among the agents based on their contribution. 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* [3] 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* [99]. 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 [34] and multi-agent setting [31], though defining the potential function is challenging and usually needs specific domain knowledge. In order to address this issue, [32] combines 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.

[132] considers a fully cooperative MARL (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.,

$$Q_{tot} = \sum_{i=1}^{N} Q_i(\tau_i, u_i, \theta_i) \tag{14}$$

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

$$\arg\max_{\boldsymbol{u}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) = \begin{pmatrix} \arg\max_{u_1} Q_1(\tau_1, u_1) \\ \vdots \\ \arg\max_{u_N} Q_N(\tau_N, u_N) \end{pmatrix}, \tag{15}$$

in which τ 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 [152] 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 [108] considers the same problem as VDN does, and proposed an algorithm which is basically an improvement over VDN [132]. 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:

$$\frac{\partial Q_{tot}}{\partial Q_i} \geq 0, \forall i, \tag{16}$$

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 [45], 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 MARL problems, the existing restrictions for these two do not hold for all problems. To the end of relaxing these restrictions, [68] proposes 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 $\sum_{i=1}^N Q_i(\tau_i, u_i, \theta_i)$. 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

$$V_{tot} = \max_{\boldsymbol{u}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) - \sum_{i=1}^N Q_i(\tau_i, \bar{u}_i), \tag{17}$$

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

$$\sum_{i=1}^N Q_i(\tau_i, u_i, \theta_i) - Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) + V_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) = \begin{cases} 0 & \boldsymbol{u} = \bar{\boldsymbol{u}} \\ \geq 0 & \boldsymbol{u} \neq \bar{\boldsymbol{u}} \end{cases} \tag{18}$$

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 [47] and modified predator-prey [127] 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, [90] considers a MARL 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. Basically, 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 problem can convergence 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 [1] 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.

5 Consensus

The idea of the centralized critic, which is discussed in Section 3, 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, in order to reach 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 [170, 55]. 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.

[146] studies a problem in which each agent has a local observation, executes its own 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 maximizing 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 [61, 60] the authors proposed 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:

$$Q_i^{t+1}(s, a) = Q_i^t(s, a) + \alpha_{s,a}^t \left(r_i(s_t, a_t) + \gamma \min_{a' \in \mathcal{A}} Q_i^t(s_{t+1}, a') - Q_i^t(s, a) \right)$$

$$- \beta_{s,a}^t \sum_{j \in \mathcal{N}_i^t} \left(Q_i^t(s, a) - Q_j^t(s, a) \right), \tag{19}$$

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 condition, they prove that their algorithm obtains the result that the agent could achieve if the problem was solved centrally.

In [103] 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 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 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.

[85] proposes 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 overall tasks. Each agent runs its own instance of the environment with a specific task, without intervening 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 in 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 actor networks 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 [170] a MARL 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 randomly 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

$$\mu_i^t = (1 - \beta_{wt}^t)\mu_i^t + \beta_{wt}r_i^{t+1}, \tag{20}$$

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:

$$w_i^{t+1} = \sum_{j \in \mathcal{N}_i} c_{ij} \tilde{w}_j^t. \tag{21}$$

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 (20):

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

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 [83] 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 [149] a double-averaging scheme is proposed for the task of *policy evaluation* for MARL problems. The setting is following [170], 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 details, first, the duality theory has been utilized to reformulate the MARL policy evaluation problem, which is supposed to minimize 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* [106] and the SAG algorithm [114]. Under linear function approximation, it is proved that the proposed algorithm converges linearly under some conditions.

[169] considers the MARL problem with continuous state and action space. The rest of the setting is similar to the [170] (i.e., global state, global action, and local reward). Again, an AC-based algorithm is proposed for the considered 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 [120] which requires off-policy exploration. However, in the setting of [169] the off-policy information of each agent is not known to other agents, so the approach used in DPG [120, 79] cannot be applied here. Instead, a gradient update based on the expected policy-gradient (EPG) [26] 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 the mentioned 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 similar setting as [170], [133] proposes 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 [170] comes from the fact that the critic step is conducted in an off-policy setting using emphatic temporal differences $ETD(\lambda)$ policy evaluation method [137]. 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 $f : \mathcal{S} \to \mathbb{R}^+$ 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 [66] 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 [164], the authors proved the convergence of the proposed method for the distributed actor-critic method when linear function approximation is utilized.

[167] proposes 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 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 [84], an off-policy distributed policy evaluation algorithm is proposed. In this paper, a linear function has been used to approximate the long-term 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 [1] [136] 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

---

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

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 GTD algorithm is a stochastic Arrow-Hurwicz [2] [6] algorithm applied to the dual problem of the original optimization problem. Then, inspiring from [22], 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 [84], in [126] 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 [84], 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 [74]. 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 which combines the features of the states is used as the estimator for the value function. The problem is modeled as a constrained optimization problem (consensus constraint), and then following the same process as [84], a primal-dual algorithm was proposed to solve it. A rigorous analysis based on the ordinary differential equation (ODE) [15] is provided for the proposed algorithm. Particularly, they showed that the set of KKT solutions of the Lagrangian dual problem can be compactly written by an ODE in the form of $\dot{x} = -Ax - b$. In order 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 considered a stock market problem where $N = 5$ different agents have different policies for trading the stocks. DGTD is utilized to estimate the average of long term discounted profit of all agents. The results are compared with a single GTD algorithm in the case the sum of the reward is available. The comparison results show that each agent can successfully approximate the global value function.

[21] considers 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 variance-reduced algorithm called AVRG [163] 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. [84]) 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 [171]. In contrast to the [170], where the actor step is performed locally and the consensus update is proposed for the critic, in [171] 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.

6 Learn to Communicate

As mentioned earlier in Section 5, 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 message, and the destination agent. Usually, the communication-actions do not interfere with the environment, i.e., the messages do not affect the next state or reward. [62] proposed one of first learning to communicate algorithms, in which tabular Q-learning agents learn messages to communicate with other

---

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

agents in the predator-prey environment. The same approach with a tabular RL is followed in [146]. 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, [35] 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 [45] 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 continues vector for the communication action so that there is no action selector for the communication action m_i^t, and instead, a regularizer unit discretize it, if necessary. They provide numerical results on switch riddle prisoner game and three communication game with mnist dataset. The results are compared with no-communication and parameter sharing version of RIAL and DIAL methods.

[58] extends 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 results of their algorithm with different parameters is presented. Following a similar line, [73] 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 [123]. 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 [154] 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 high success rate and show that the learned communications are interpretable. In another work in this direction, in [29] a fully cooperative two-agent game is considered for the task of *image guessing*. In particular, two bots, namely 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 [123] the REINFORCE algorithm [154] is used to train both agents. Note that [58] allows "yes/no" actions within multiple rounds of communication, [73] consists of one single round with continues messages, and [29] combines them such that multiple rounds of continues communication are allowed.

Similarly, [95] studies 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 [73] 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 model 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 model 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 [52]. The results of the algorithm are compared with a no-communication approach in the mentioned game.

[130] considers a fully cooperative MARL task in which each agent observes a local state and is able to send a continues 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 [35], 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 [129].

To extend CommNet, [49] proposes 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 [102] the authors introduce a bi-directional communication network (BiCNet) using 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 combat game. In particular, there exists two vectorized networks, namely actor and critic network 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 are calculated and the weights of the networks get updated accordingly using Adam algorithm. In order to reduce the variance, they applied the deterministic off-policy AC algorithm [120]. The proposed algorithm applied to the multi-agent StarCraft combat game [110]. It is shown that BiCNet is able to discover several effective ways to collaborate during the game.

[124] considers the MARL 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 [3]. 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

---

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

time t, the action a_i^t and c_i^{t+1} are:

$$g_i^{t+1} = f(h_i^t), \tag{22}$$

$$h_i^{t+1}, l_i^{t+1} = \text{LSTM}\left(e(o_i^t) + c_i^t, h_i^t, l_i^t\right), \tag{23}$$

$$c_i^{t+1} = \frac{1}{N-1} C \sum_{i \neq j} h_j^{t+1} g_j^{t+1}, \tag{24}$$

$$a_i^t = \pi(h_i^t), \tag{25}$$

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 [154]. In order to analyze the performance of IC3Net, predator-pray, traffic junction [130], and StarCraft with explore and combat tasks [110] are considered. The results are compared with CommNet [130], no-communication model, and no-communication model with only global reward.

In [53], 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:

$$c_i^t = \sum_{j \neq i} \left[D_{KL} \left[p(a_j^t | a_i^t, o_i^t) || p(a_j^t | o_i^t) \right] \right] \tag{26}$$

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 (26). 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 [107].

[30] proposes 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 by 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 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 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 [4], traffic junction in which they control the cars, and House3D [158] as well as the CommNet [130] when it was possible.

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, [54] fixes 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, the communication might impair the 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_{mu} 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 channel, and as a result, the information can be transferred among a larger number of agents. The actor and critic models are trained following 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, and MADDPG was not a benchmark.

7   Emerging Areas

In this section, we discuss a few recent papers which either combine the approaches in sections 3-6 or they propose a model that does not quite fit in either of the previous sections.

[37] considers 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 [119] a different setting of the MARL 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 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. [67, 148]), 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 MARL, two possible scenarios are synchronous and asynchronous. In the synchronous hierarchical MARL, all high-level agents take actions 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 MARL. Nonetheless, obtaining high-level cooperation in asynchronous case is challenging. In the following we study some recent papers in hierarchical MARL.

In [140] 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) [67] 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 [130] is combined with Ind-hDQN. In this way, Ind-hDQN's neural network is modified to include the average of the hth 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 [108] 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 replay buffer, 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) [100] 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 [86]) 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 [113].

In the literature of MARL, there are several papers who consider the games with Nash equilibrium. In a MARL setting 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 [69] provides a detailed review of the older papers.

[165] discusses 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, [165] also utilizes rFMQ [89], 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, a two-player game with Nash equilibrium is presented, as well as the boat navigation problem with two actions. SCC-rFMQ is compared with MADDPG, Sequential Monte Carlo Methods (SMC) [72], and rFMQ [89]. SMC [72] 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, actor updates the policy distribution. In this way, for each state, the actor provides a probability distribution over the continues action space and based on the importance of sampling, the action is selected.

8 Applications

The MARL problem has numerous application in the real world. In this section, we review some of the application-oriented papers, in which the problem is modeled as a MARL problem. 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.

A multi-agent Q-learning algorithm has been proposed in [150] 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.

[105] proposes 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. Similar problem with the RL approach is studied in [2]. Also, recently [168] provided CityFlow, a new traffic-signal environment to be used for MARL researches. In a related area, [80] considers rid-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 largest rid-sharing company in China. In a similar direction, [17] studies the air-traffic control problem as a MARL problem to ensure safe separation between aircraft. Each airplane, as 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 [159] 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 rang, and 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 [71] 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.

[166] considers 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 by 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. [157] also considers a similar problem and propose another value-based algorithm. Moreover, in [156] 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, [162] consider a packet routing problem in the wireless sensor networks and model it as a MARL 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 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 the similar domain, [28] propose a 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.

In [97] 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 policy gradient algorithm, an algorithm is proposed to update the prediction and motion planning modules in an end-to-end manner. The results on MNIST data-set is provided in which each agent only observes a few pixels of the image.

[7] consider the liquidation of a large amount of stock in a limited time T. The liquidation process usually is done with cooperation 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 MARL with (i) competitive objectives: when each agent want 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 MARL 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.

Finally, as an application in civil engineering, [5] propose a MARL model 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 non-stationary issue in MARL. 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 subsystem. This algorithm particularly extends the policy gradient algorithm to the cases with large amount 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 results they cover a broad range of engineering systems including (i) a simple stationary parallelseries 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.

9 Environments

Environments are core elements in any MDP problem. 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 [9], MoJoCo (simulates the detailed physics movements of human and some animals body) [142], OpenAI Gym which gathers these together [18], PyGame Learning Environment (similar to Arcade) [141], OpenSim (builds musculoskeletal structures of human) [117], DeepMind Lab (3D navigation and puzzle-solving) [8], ViZDoom (3D Shooting and Navigation Doom using only the visual information) [63], Malmo (based on Minecraft) [57], MINOS (Home indoor 3D Navigation) [112], House3D (3D Navigation in indoor area) [158], and MazeLab [172] 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 3-7 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 [110] has achieved a lot of attention in the recent years [36, 143, 124, 38, 108, 102]. 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 continues action spaces. Multi-agent particle environment [94] gathers a list of navigation tasks, e.g., the predator-prey for both discrete and continues actions. Some of the games allow choosing a communication action too. Harvest-gathering [53] is a similar game with communication-action. Neural MMO [128] 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, [168] provided CityFlow, a new traffic-signal environment to be used for MARL researches. In addition, [155] introduce a framework to control cars within a mixed system of human-like driver and AI agents.

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. [56] provides 12 environments navigation/maze-like games. Unity [59] 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 [125] extends the Unity engine and provides an 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 [96] also recently started to support multi-agent environments and some of the known algorithms are also added in rllib repository [78, 77]. Ray supports entering/leaving the agents from the problem which is a common case in traffic control suites.

10 Conclusion and Future Research Directions

In this review, we categorize MARL problems 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 1 a summary the of most influential papers in each category is presented. In this summary we have gathered the 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 the communication status, i.e., 0 means there is no communication among the agents, and 1 otherwise. In forth column, AC means the proposed algorithm is actor-critic based, and Q is used when the proposed algorithm is value-based. In

	Reference	Com	AC/Q	Conv	State (Trn, Exe)	Action (Trn, Exe)	Reward (Trn, Exe)
IQL	Tan, 1993 [139]	0	Q	0	(L,L)	(L,L)	(G,G)
	Lauer and Riedmiller, 2000 [70]	0	Q	0	(G,G)	(L,L)	(G,G)
	Matignon et al., 2007 [88]	0	Q	0	(G,G)	(G,L)	(G,G)
	Tampuu et al., 2017 [138]	0	Q	0	(G,G)	(L,L)	(G,G)
	Omidshafiei et al., 2017 [100]	0	Q	0	(G,G)	(G,L)	(G,G)
	Fuji et al., 2018 [39]	0	Q	0	(G,G)	(G,L)	(G,G)
Fully Observable Critic	Wang et al., 2019 [151]	1	AC	0	(G,L)	(G,L)	(L,L)
	Foerster et al., 2018 [38]	0	AC	0	(G,L)	(G,L)	(G,G)
	Ryu et al., 2018 [109]	0	AC	0	(G,L)	(G,L)	(L/G,L/G)
	Sartoretti et al., 2019 [111]	0	AC	0	(G,G)	(L,L)	(L,L)
	Chu and Ye, 2017 [25]	0	AC	0	(L,L)	(L,L)	(L/G,L/G)
	Yang et al., 2018 [160]	0	AC	0	(L,L)	(L,L)	(L,L)
	Jiang et al., 2018 [55]	1	Q	0	(L,L)	(L,L)	(L,L)
	Iqbal and Sha, 2019 [51]	1	AC	0	(G,L)	(G,L)	(G,L)
	Yang et al., 2018 [161]	1	AC/Q	1	(G,G)	(G,L)	(G,L)
	Kim et al., 2019 [64]	1	AC	0	(G,L)	(G,L)	(G,G)
	Lowe et al., 2017 [83]	0	AC	0	(G,L)	(G,L)	(L,L)
	Mao et al., 2019 [87]	0	AC	0	(G,L)	(G,L)	(L,L)
VD	Rashid et al., 2018 [108]	0	Q	0	(G,L)	(G,L)	(G,G)
	Sunehag et al., 2018 [132]	0	Q	0	(L,L)	(L,L)	(G,G)
	Mguni et al., 2018 [90]	0	Q	1	(L,L)	(L,L)	(G,G)
	Kyunghwan Son and Yi, 2019 [68]	0	Q	1	(L,L)	(L,L)	(G,G)
Consensus	Zhang et al., 2018 [170]	1	AC	1	(G,G)	(G,L)	(L,L)
	Kar et al., 2013 [60]	1	Q	1	(G,G)	(L,L)	(L,L)
	Lee et al., 2018 [74]	1	Q	1	(G,G)	(L,L)	(L,L)
	Macua et al., 2015 [84]	1	Q	0	(L,L)	(L,L)	(G,G)
	Macua et al., 2018 [85]	1	AC	0	(L,L)	(L,L)	(L,L)
	Cassano et al., 2018 [21]	1	Q	1	(L/G,L/G)	(L,L)	(L/G,L/G)
	Zhang et al., 2018 [169]	1	AC	1	(G,G)	(G,L)	(L,L)
	Zhang and Zavlanos, 2019 [171]	1	AC	1	(G,G)	(G,G)	(L,L)
Learn to comm	Varshavskaya et al., 2009 [146]	1	AC	1	(L,L)	(L,L)	(L,L)
	Peng et al., 2017 [102]	1	AC	0	(G,G)	(G,G)	(L,L)
	Foerster et al., 2016 [35]	1	Q	0	(L,L)	(L,L)	(G,G)
	Sukhbaatar et al., 2016 [130]	1	AC	0	(L,L)	(L,L)	(G,G)
	Singh et al., 2018 [124]	1	AC	0	(L,L)	(L,L)	(L,L)
	Lazaridou et al., 2017 [73]	1	AC	0	(G,G)	(L,L)	(G,G)
	Das et al., 2017 [29]	1	AC	0	(L,L)	(G,G)	(L,L)

Table 1: The proposed algorithms for MARL and the relevant setting

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.

Beside this table, in what follows, we list a few potential research directions and open problems in the realm of MARL.

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 setting, the latter one seems to be more fit to the multi-agent setting. In MARL literature, there exist a few algorithms utilizing the off-policy [169, 133, 84]; however, there is still room for research on off-policy in MARL in algorithm design, theory, and applications.

Safe MARL: Safe RL is defined as the training of agent to learn a policy which maximizes the long-term 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 [41]. Nonetheless, the research on safe MARL is very scarce. For example, in [118], a safe RL algorithm is proposed for Multi-Agent Autonomous Driving. Therefore, another straightforward research direction for MARL would be 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 the ability and skill. However, in real-world applications, it is likely that we face a MARL 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 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 [14, 33, 48]. 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) on MARL problems. For instance, TRPO [115], which has been shown to be very efficient in single-agent RL problems, might be helpful for MARL 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.

References

[1] Synchronous advantage actor-critic. `synchronousAdvantageActor-Critic`. Accessed: 2019-07-22.

[2] 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.

[3] 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.

[4] 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.

[5] CP Andriotis and KG Papakonstantinou. Managing engineering systems with large state and action spaces through deep reinforcement learning. *Reliability Engineering & System Safety*, 191:106483, 2019.

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

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

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

[17] Marc Brittain and Peng Wei. Autonomous air traffic controller: A deep multi-agent reinforcement learning approach. *arXiv preprint arXiv:1905.01303*, 2019.

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

[19] 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.

[20] 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.

[21] Lucas Cassano, Kun Yuan, and Ali H Sayed. Multi-agent fully decentralized off-policy learning with linear convergence rates. *arXiv preprint arXiv:1810.07792*, 2018.

[22] 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.

[23] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[24] 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.

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

[26] Kamil Ciosek and Shimon Whiteson. Expected policy gradients for reinforcement learning. *arXiv preprint arXiv:1801.03326*, 2018.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] Sam Devlin, Logan Yliniemi, Daniel Kudenko, and Kagan Tumer. Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 165–172. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] Jakob N Foerster, Christian A Schroeder de Witt, Gregory Farquhar, Philip HS Torr, Wendelin Boehmer, and Shimon Whiteson. Multi-agent common knowledge reinforcement learning. *arXiv preprint arXiv:1810.11702*, 2018.

[38] 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.

[39] 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.

[40] 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.

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

[42] 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.

[43] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

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

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

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

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

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

[53] 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`.

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

[55] Jiechuan Jiang, Chen Dun, and Zongqing Lu. Graph convolutional reinforcement learning for multi-agent cooperation. *arXiv preprint arXiv:1810.09202*, 2018.

[56] Shuo Jiang. Multi-Agent Reinforcement Learning Environment. `https://github.com/Bigpig4396/Multi-Agent-Reinfor` Accessed: 2019-07-28.

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

[58] 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.

[59] 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.

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

[61] 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, 2013.

[62] 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.

[63] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZ-Doom: 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. URL `http://arxiv.org/abs/1605.02097`. The best paper award.

[64] 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. *arXiv preprint arXiv:1902.01554*, 2019.

[65] 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.

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

[67] 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.

[68] Wan Ju Kang David Earl Hostallero Kyunghwan Son, Daewoo Kim 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.

[69] 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.

[70] 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.

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

[72] 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.

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

[74] 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.

[75] 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.

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

[77] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E Gonzalez, Michael I Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. *arXiv preprint arXiv:1712.09381*, 2017.

[78] 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. *arXiv preprint arXiv:1712.09381*, 2017.

[79] 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. *arXiv preprint arXiv:1509.02971*, 2015.

[80] 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.

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

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

[88] 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.

[89] 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.

[90] 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.

[91] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[92] 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.

[93] 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.

[94] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

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

[96] 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.

[97] Hossein K. Mousavi, MohammadReza Nazari, Martin Takác, and Nader Motee. Multi-agent image classification via reinforcement learning. *CoRR*, abs/1905.04835, 2019. URL `http://arxiv.org/abs/1905.04835`.

[98] 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.

[99] 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`.

[100] 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.

[101] Afshin Oroojlooyjadid, MohammadReza Nazari, Lawrence Snyder, and Martin Takáč. A deep q-network for the beer game: A reinforcement learning algorithm to solve inventory optimization problems. *arXiv preprint arXiv:1708.05924*, 2017.

[102] 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. *arXiv preprint arXiv:1703.10069*, 2017.

[103] 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.

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

[105] 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.

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

[107] 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, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/rabinowitz18a.html`.

[108] 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, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/rashid18a.html`.

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

[110] 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.

[111] 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, 2019.

[112] 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.

[113] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

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

[115] 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.

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

[117] 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.

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

[119] Tianmin Shu and Yuandong Tian. Mˆ 3rl: Mind-aware multi-agent management reinforcement learning. *arXiv preprint arXiv:1810.00147*, 2018.

[120] 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.

[121] 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.

[122] 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.

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

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

[125] Yuhang Song, Jianyi Wang, Thomas Lukasiewicz, Zhenghua Xu, Mai Xu, Zihan Ding, and Lianlong Wu. Arena: A general evaluation platform and building toolkit for multi-agent intelligence. *arXiv preprint arXiv:1905.08085*, 2019.

[126] 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.

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

[128] 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.

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

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

[131] 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.

[132] 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.

[133] Wesley Suttle, Zhuoran Yang, Kaiqing Zhang, Zhaoran Wang, Tamer Basar, and Ji Liu. A multi-agent off-policy actor-critic algorithm for distributed reinforcement learning. *arXiv preprint arXiv:1903.06372*, 2019.

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

[135] 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.

[136] 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`.

[137] 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.

[138] 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.

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

[140] 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.

[141] Norman Tasfi. Pygame learning environment. `https://github.com/ntasfi/PyGame-Learning-Environment`, 2016.

[142] 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.

[143] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies for starcraft micromanagement. 2016.

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

[145] 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.

[146] 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.

[147] 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.

[148] 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.

[149] 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.

[150] 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, 2016.

[151] Rose E Wang, Michael Everett, and Jonathan P How. R-maddpg for partially observable environments and limited communication. 2019.

[152] 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 2016. PMLR. URL `http://proceedings.mlr.press/v48/wangf16.html`.

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

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

[155] 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.

[156] 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.

[157] 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.

[158] 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.

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

[160] Jiachen Yang, Alireza Nakhaei, David Isele, Hongyuan Zha, and Kikuo Fujimura. Cm3: Cooperative multi-goal multi-stage multi-agent reinforcement learning. *arXiv preprint arXiv:1809.05188*, 2018.

[161] 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, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

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

[163] 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.

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

[165] 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, 2018.

[166] 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.

[167] 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.

[168] 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, 2019.

[169] 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, 2018.

[170] 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, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[171] Yan Zhang and Michael M. Zavlanos. Distributed off-policy actor-critic reinforcement learning with policy consensus. *CoRR*, abs/1903.09255, 2019.

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