Multi-Agent Deep Reinforcement Learning: A Review

Qiong Hu[1], Hailin Yu[2], and Zhuyun Xiao[3]

[1]qiong.j.hu@gmail.com (405065032)
[2]yhltc@g.ucla.edu (305231131)
[3]zxiao2015@g.ucla.edu (504592699)

Abstract

Reinforcement learning has been extensively studied and employed in the past decades for a variety of domains, including robotics and intelligent agents. More specifically, it has been used in multi-agent learning settings. However, as the environment dimensions scale up reinforcement learning can become infeasible. Hence, deep reinforcement learning has been developed for efficient performance in more challenging high-dimensional situations. This review article starts off by presenting the history and current progress in multi-agent deep reinforcement learning (MADRL), followed by an overview of the broader field of reinforcement learning, deep reinforcement learning, and multi-agent reinforcement learning. Additionally, we discuss the evolution of algorithms in the MADRL framework. Different approaches to solving problems in a variety of settings relevant to MADRL will be explained and compared. Finally, we will highlight the merits and drawbacks of various MADRL methods.

Keywords: Deep reinforcement learning; Multi-agent systems

1 Introduction

Studying decision-making in multi-agent systems is of substantial interest in a variety of domains ranging from economics to robotics. For robotics and intelligent agents, reinforcement learning has long been an important method for addressing how agents choose actions in specific environments [23]. Classically, linear function approximation is used to generalize complex and infinite environments [27]. However, using non-linear function approximators in reinforcement learning is troublesome owing to the divergence when the updated schemes are not based on bound trajectories [4]. In recent years, as deep learning achieves significant progress in a variety of research domains such as computer vision and natural language processing, more interest in applying deep learning methods to reinforcement learning cases

surges. Deep reinforcement learning algorithm thus gains its momentum in scaling reinforcement learning to previously intractable problems, including learning how to play video games based on pixels and learning control policies for robots from camera inputs [2]. In particular, using deep reinforcement learning to compute multi-agent policies is promising as it can scale to large size multi-agent MDPs problems. This is contrary to traditional which quickly become computationally intensive [4].

In this review paper, we first give an overview of the basic background of reinforcement Learning (RL), deep learning (DL), deep reinforcement learning (DRL) [12, 2], and multi-agent learning (MAL). Then we discuss the different methods for MAL that could be adapted by MADRL, followed by a dive into the emerging field of multi-agent deep reinforcement learning (MADRL) [4]. This review paper also summarizes the important progresses, followed by discussions on the key algorithms and applications, on MADRL.

2 Background

2.1 Reinforcement Learning

Reinforcement learning addresses a set of problems in which we have an agent and a system. Typically, we use a state s to represent the agent, and a function f to represent the system, so we have

$$s' = f(s). \tag{1}$$

If the system or the agent has some input actions a,

$$s' = f(s, a). \tag{2}$$

If the system is only partially observable, which means we can only get observations z, then

$$z = h(s, a) \tag{3}$$

and we need to use z to estimate state s'.
We need to solve following problems:

1. State estimation/representation, in which we need to determine states s, the state often includes locations of the agent and velocity of the agent. There are many different methods to solve this problem, such as Bayes filters, Kalman filters, and Particle filters.

2. Control/Planning problems, in which we need to determine actions and/or a policy π, usually one action for each state. There are many different methods to solve Control/-Planning problems, for example, MDP solver, PRM, MPC and RRT.

3. Modeling/design, in which we need to determine the functions f and h. This is the most difficult part of Reinforcement Learning, because there is never an exact answer. As a result, this problem does not have a universal solution.

Firstly, let us look at a simple situation: the Markov Decision Process (MDP) formulation. In a system, we often have a target, like moving an agent to a certain place or making an agent to locate something. It becomes natural to define a Reward function r for our agent. Usually, r is determined by states s and actions a. So we have

$$r = R(s, a) \tag{4}$$

A reasonable choice for the reward function is to give the target states a higher reward compared to other states. When we do not want the agent go to certain states, we will assign them a very low reward.

In the real world, we often prefer an immediate reward, so we will assign a discounted factor γ between 0 and 1. When a reward r is obtained after a certain time t, the reward becomes $r * \gamma^t$.

With rewards for each state, the next step is how to specify rewards for a certain trajectory: $s0- > a0- > s1- > a1$.....? To answer this question, we can compute the sum of rewards r_{tot}:

$$r_{tot} = \sum_t r_t \gamma^t \tag{5}$$

Now we can determine the best policy which maximizes the long-term reward, that is the expected value of r_{tot}. We often use a value function to represent the expected long-term reward. Hence, these methods are called value-based methods. Specifically, for a certain state, the optimal value function is:

$$V^*(s) = \max_a \left[\sum_{s'} P(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right] \right] \tag{6}$$

and the best policy is :

$$\pi^*(s) = \arg\max_a \left[\sum_{s'} P(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right] \right] \tag{7}$$

in which $P(s'|s, a)$ represents the transition probability

There are two main methods to determine the best policy: Value iteration and policy iteration.

For value iteration, we initialize all value functions to be 0, and get a more accurate estimate every time.

Algorithm 1: Value Iteration Algorithm

Initialize $V_0(s) = 0 \; \forall s$
do
  $V_{i+1}(s) = \max_a \left[\sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V_i(s')] \right]$
until $V(s)$ *converges*;

For policy iteration, we initialize a random policy, then we identifies the optimal actions of current value function and update the policy each time.

Algorithm 2: Policy Iteration Algorithm

Initialize a random policy π_0

do

Calculate the value function V_i of policy π_i

$\pi_{i+1}(s) = \arg\max_a \left[\sum_{s'} P(s,a,s')[R(s,a,s') + \gamma V_i(s')]\right]$

until $\pi(s)$ *converges*;

Several differences exist between value and policy iteration. Specifically, value iteration has more steps with smaller gains at each step, suggesting that its iterations can stop before convergence and still remain fairly accurate. On the contrary, policy iteration takes fewer steps with a larger change at each step.

In reality, one cannot always have perfect knowledge of the reward function r and the transition probability $\mathcal{P}$. Therefore, Q-learning and policy gradient method are used. Here, the goal is to maximize the utility function [18]:

$$J(\theta) = \mathop{\mathbb{E}}_{\theta,s}\left[\sum_t R(s,a,\theta)\gamma^t\right] \tag{8}$$

For Q-learning, we minimize loss function:

$$L(\theta) = \left(r + \gamma \max Q(s',a',\theta^-) - Q(s,a,\theta)\right)^2 \tag{9}$$

where, θ^- is the weights of the target network which is updated by θ every C iterations.

For policy gradient, we update the parameter θ by calculating the gradient of $J(\theta)$:

$$\theta' = \theta - \alpha \nabla J(\theta) \tag{10}$$

2.2 Deep Learning

Recently, deep learning emerges as a popular machine learning method that can perform well with massive amounts of data. A typical task in machine learning is to learn a function or a classifier. However, traditional machine learning methods such as supported vector machine learning (SVM) or boosting, make strong assumptions about the underlying mechanism giving rise to the data. However, in practice, modeling assumptions can be violated for traditional methods. In constant, deep learning is tremendously flexible adapting to different data structures like images and text. As a result, deep learning has become widely popular and has already been successful in different areas, for example, CNN has beaten human in image classification [10].

Deep learning methods use a deep neural network which imitates the structure of human brains. A deep neural network has many layers and each has some neurons. With a non-linear activation, the whole network can represent and approximate a general function. The inputs data go through the whole network and then we get output data. We use output

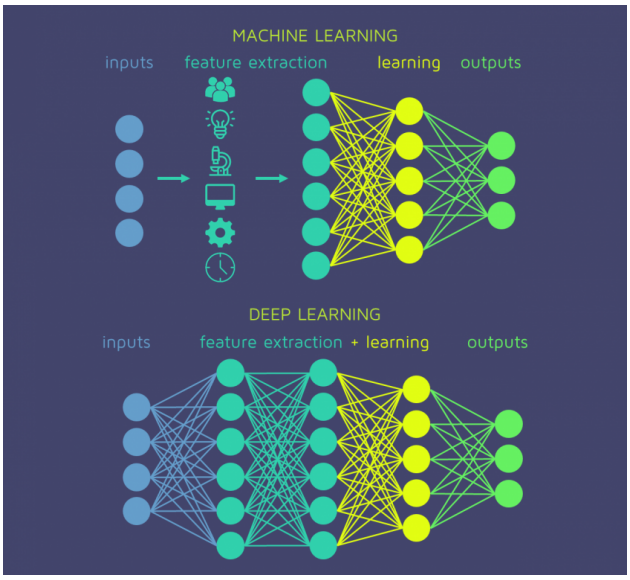

Figure 1: Difference between traditional machine learning and deep learning [5]

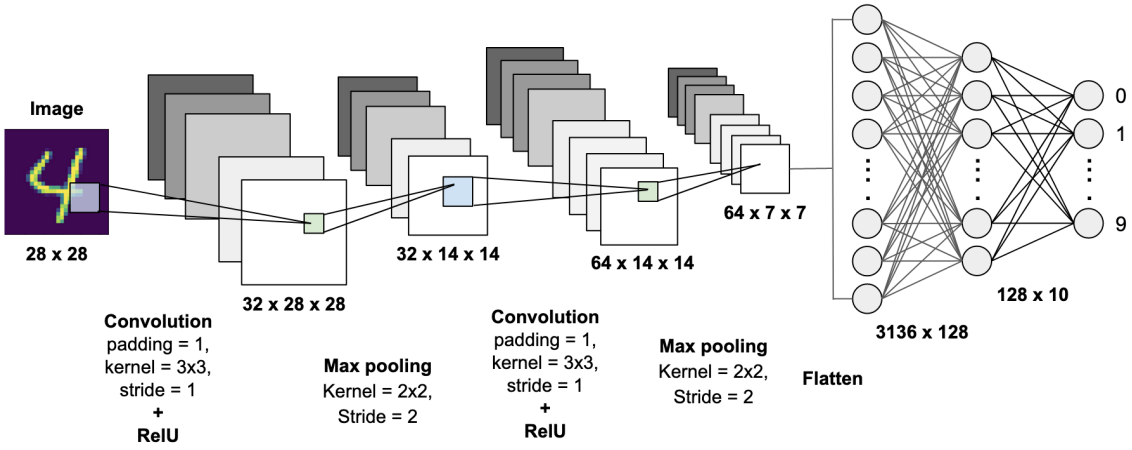

Figure 2: An example of convolution neural network (CNN).

data, labels and loss functions to calculate the loss, and then update weights of neurons in each layer using gradient descent:

$$w' = w - \alpha \nabla Loss(w) \tag{11}$$

In the deep neural network, individual layer will learn its own features.

2.3 Deep Reinforcement Learning

Deep Reinforcement Learning emerges as a new area, perhaps the first successful example is AlphaGo [21]. In 2016, AlphaGo defeated the top human players in the "Go" game, a task

which had been originally seen as an impossible for q computer.

As reinforcement learning involves large amounts of data, it is a natural framework for deep learning. As a result, researchers combined both paradigms together to give rise to a novel area of research.

In deep reinforcement learning, we typically apply a DNN to learn the value function directly, meaning the input is one of the states and the output is a value function of the state.

2.4 Multi-Agent Learning

So far what we have discussed all pertain to single agent learning. However, in real world applications, we always encounter problems in multi-agent settings, such as auto-driving, game AI and robotics control. There are two main difficulties that come with multi-agent problems. The first is extremely large number of states. To put this in perspective, for a single agent situation, we already have a large space N, and for M agents, we will have a space N^M. The other difficulty is how to define a joint value function. Why we need a joint value function? For a single agent, we can easily compare two Q values, for example, $Q_1 = 2$ is bigger than $Q_2 = 1$. However, when it comes to the multi-agent framework, we cannot easily compare two vector Q values, for example, $Q_1 = [2, 0]$ and $Q_2 = [1, 1]$.

To solve the second problem, several methods have been created. One of such methods is independent Q-learning (IQL) [26], in which every agent learn its own Q function and see other agents as part of the environment. Such method is easily computed for a lot of agents, however, the problem is that individual agents learning makes the environment appear non-stationary to other agents.

Another method is to learn a fully observable centralised critic [13], which estimates Q-values for the joint action u conditioned on the central state s. As a result, each agent will influence the centralised critic and vice versa. One can train the centralised critic by changing the action of one agent while keep other agents unchanged. This method considers the value function as an integrated entity, so each agent's learning will not make the environment appear non-stationary. However, the problem arises as the calculation becomes too complex for problems with a large number of agents.

Another method that lies between the above two is called value decomposition (VD) [22, 19]. It considers both individual Q function and total Q function. The idea of it is that total Q function can be expressed as a function of individual Q functions, such as:

$$Q_{tot} = \sum Q_i \tag{12}$$

The function can be as simple as above or become very complex, while it needs to meet one requirement [22]:

$$dQ_{tot}/dQ_i > 0 \tag{13}$$

which means when an individual Q function becomes larger, the corresponding total Q function should also increase.

2.5 Multi-Agent Deep Reinforcement Learning

Roughly speaking, the multi-agent deep reinforcement learning (MADRL) setting refers to methods where a DNN is incorporated to learn both an individual Q-function and a total Q-function. The motivation for MADRL is described next. A more comprehensive review can be found in [16].

Multi-agent reinforcement learning extends the single agent reinforcement learning to settings with multiple agents. This immediately creates a computational problem when dealing with large data sets. In addition, multi-agent reinforcement learning has to deal with other challenges such as non-stationarity and partially observability.

Non-stationarity refers to how in multi-agent problems an agent observes the actions and outcomes of other agents, and the learning process happens simultaneously among all the agents. In addition in real applications there is a need to include heterogeneity between agents when it comes to modeling the goals of the system. Given the complexity of non-stationarity, traditional methods can perform poorly. Recently, different authors (e.g. [3, 1]) have proposed different MADRL methods that attempt to address the issue of non-stationary.

As for partially observability, MADRL offers the flexibility inherited from deep learning methods for modeling missing data problems. Successful examples of deep learning that directly model partially observable frameworks include [9, 8].

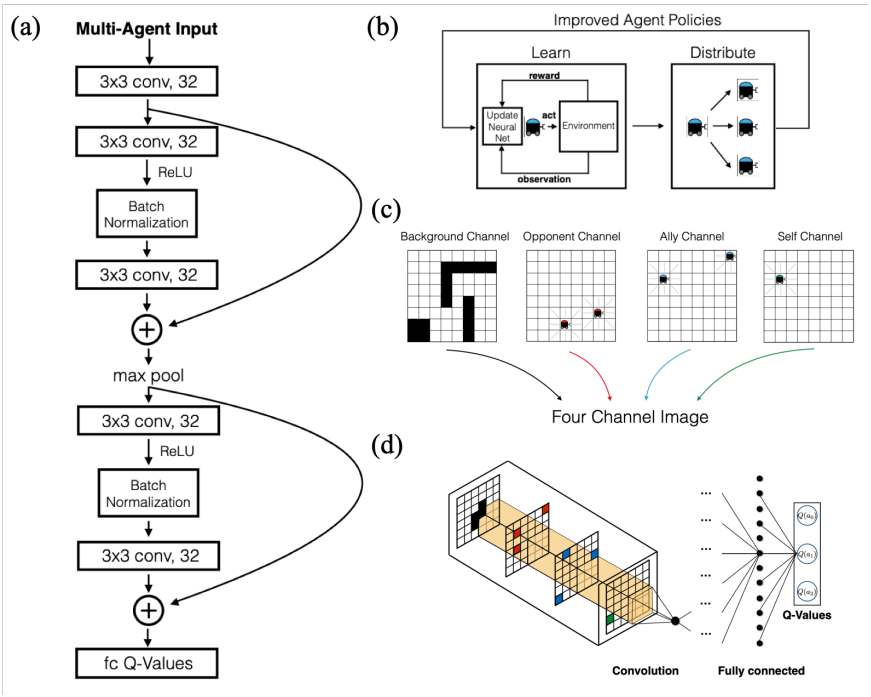

Figure 3: MADRL framework showing (a) an example of detail DQN structure (b) Improved Agent policies (c) input states expressed by four channel images (d) structure of a DQN [4]

3 Methods

3.1 Single-agent Deep Reinforcement Learning method

This section reviews the fundamentals of single-agent deep reinforcement learning methods. If the system is totally observable, we will use deep Q-learning network (DQN) and when the system is partially observable, which means there is a hidden state, we will use deep recurrent Q-learning network (DRQN).

3.1.1 Deep Q-networks (DQN)

DQN is mostly used in deep reinforcement learning, in which a DNN is used to learn the Q function directly. At each discrete time t, an agent observes state s_t and takes an action a_t. The DNN will produce a $Q(s, a)$ for every possible action a To update $Q(s, a)$ for every possible action a, we obtain an ending state s' and then compute $Q(s', a)$ for every possible action a. Then the loss can be calculated as the following:

$$L(s, a) = (R(s, a, s') + \gamma \max_a Q(s', a) - Q(s, a))^2 \tag{14}$$

For every $Q(s, a)$ there is an associated loss to be used for updating weights of DNN according to losses. However, since the same network is used for both estimation and update, over-estimation can occur. To overcome this challenge, we use different networks to do estimation and updating and making them same every N steps [2]. Then the loss function becomes:

$$L(s, a) = (R(s, a, s') + \gamma \max_a Q(s', a|\theta^-) - Q(s, a|\theta))^2 \tag{15}$$

where θ^- is the parameter of target network, and will be update to current network every N steps.

Also, for a 2D environment, each frame can be treated as an image, and as a convolution neural network (CNN). The latter have been found to be successful at dealing with image data.

On another note, deep learning always assume that data are independent, while in reinforcement learning the data are always highly correlated. Correlated data leads to a non-stationary distribution and high variance [14]. To avoid this, experience replay has been used. Experience replay consists of picking trained data again and again randomly. Consequently, the data become less correlated and the performance improves.

3.1.2 Deep Recurrent Q-Networks (DRQN)

Recurrent neural network (RNN) is used for partially observable system. In the network, we have a hidden state h_t in addition to the input states and output states. The hidden state will update itself by an RNN cell which is same for all time steps, hence, the name recurrent.

However, when time steps become large, the whole network will contain too many RNN cells and training become cumbersome. Specifically, the RNN cells which are far away from

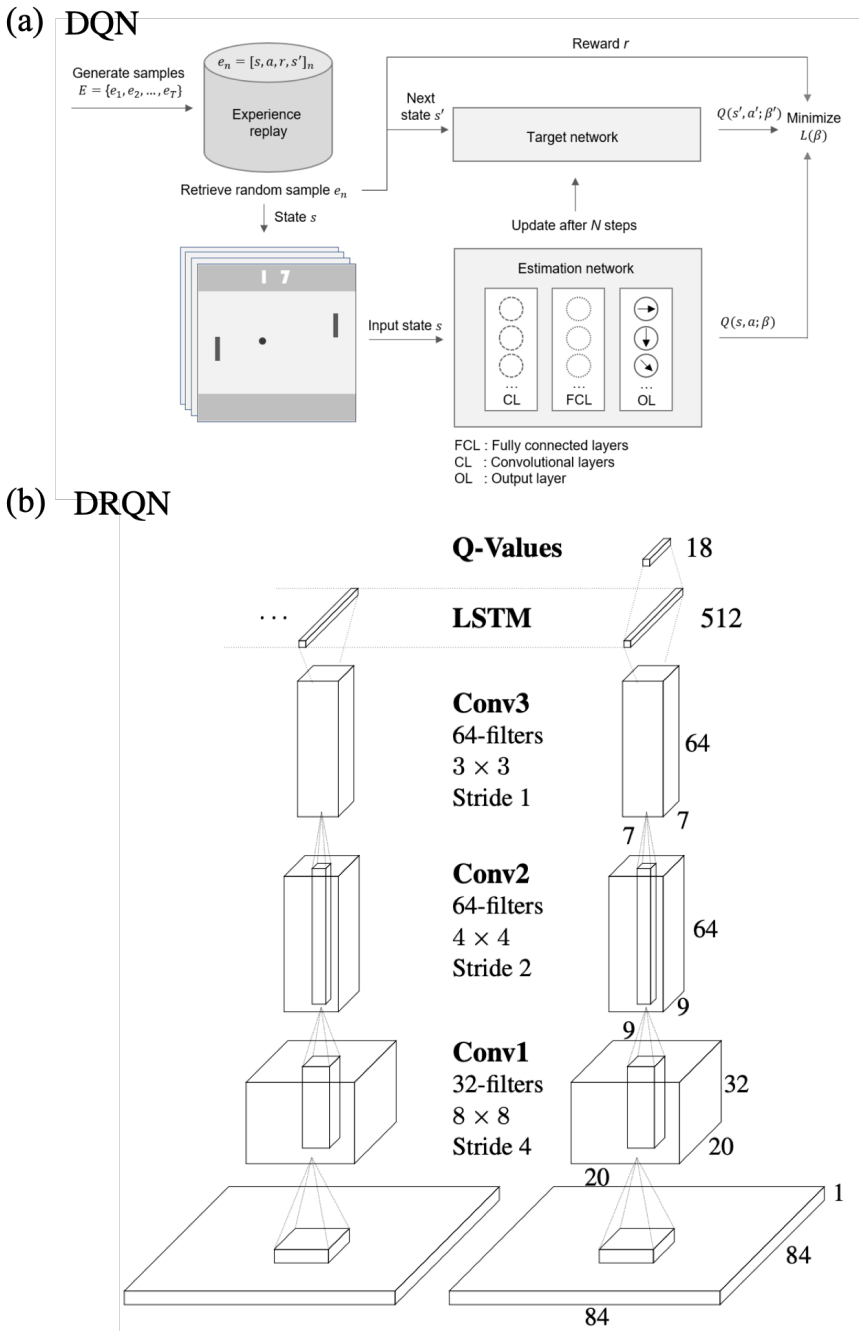

Figure 4: (a) Deep Q-Network framework [16]. (b) An example DRQN architecture in which a single-channel image input convolves, passes through a LSTM layer and a fully connected layer to output a Q-value [17].

the present will be ignored. This new approach is called long short term memory (LSTM), which has a forget gate. Figure 4 (b) shows an example DRQN architecture that incorporated

the LSTM.

For partially observable systems in reinforcement learning, RNN is also used. Just like DQN, RNN can be used to estimate Q-functions, in which the inputs are observations of state s and output are still $Q(s, a)$ for every possible action a and state s. In the hidden state of RNN, the state s will be learned automatically [7].

3.2 Multi-agent Deep Reinforcement Learning method

3.2.1 Independent Q-learning Deep Network (IQDN)

In independent Q-learning, the agents each learn their own network parameters, treating the other agents as part of the environment [25]. The independent Q-learning Deep Network (IQDN) allows a flexible framework that merges independent Q-learning with deep learning. As in [24] the resulting modeling setting can be extended to cooperative multi-agent contexts, and consequently to MADRL as well.

We refer the reader to Fig.5(a) for a pictorial representation of IQDN. In Fig.5(a) observations at different time steps enter the network of two agents and pass through the low-level linear layer to the recurrent layer, producing individual Q-values.

A potential drawback of IQDN framework has to do with convergence problems. This issue arises since one agents learning makes the environment appear non-stationary to other agents. However, empirical evidence suggests that IQDN can be successful in real applications, see [28].

3.2.2 Counterfactual multi-agent learning (COMA)

In counterfactual multi-agent (COMA) learning there are a centralised critic that learns a Q function for the central state according to all agents actions, a counterfactual baseline, and a critic representation that allows efficient evaluation of the baseline [6]. In this framework, each agent a evaluates an advantage function, which consists of comparing the Q-value of the central state and all the agents' actions against a counterfactual baseline. The latter is calculated as the expected Q-value with respect to the action of agent a while keeping the actions of all the other agents fixed. Fig.5(b) shows how the information flows between the decentralized actors, the environment and centralized critic in COMA. Formally, the setting of COMA can be described as a stochastic game. In detail, there exists a central $s \in \mathbf{S}$, and every agent $a \in \mathbf{A} := \{1, \ldots, n\}$ observes a noisy version O_a of s. In addition, at each time instance, agents simultaneously choose an action from the set of actions $\mathbf{U}$. Each agent a also has an action history, τ^a, which together with his action u^a, gives rise to the stochastic policy $\pi^a(u^a|\tau^a) \in [0, 1]$ corresponding to agent a. Based on the actions of all the agents, $\mathbf{u} \in \mathbf{U}$, agents obtain a shared reward $r(s, \mathbf{u})$. As usual in reinforcement learning, the goal is to learn the $Q(s, \mathbf{u}$ function, which computes the highest possible aggregated return over time with a discounted factor $\gamma \in [0, 1]$ and based on an optimal policy.

The naive approach for learning the Q-function would be based on following a gradient based on temporal difference error estimator from the critic corresponding to each agent.

However, this fails to capture how a particular agent's action impacts the global reward. To address this issue, in COMA each agent computes the advantage function:

$$A^a(s, \mathbf{u}) = Q(s, \mathbf{u}) - \sum_{u'^a} \pi^a(u'^a|\tau^a) Q(s, (\mathbf{u}^{-a}, u^a)). \tag{16}$$

Notably, Lemma 1 in [6] shows that by working with the advantage functions, COMA converges to a locally optimal policy.

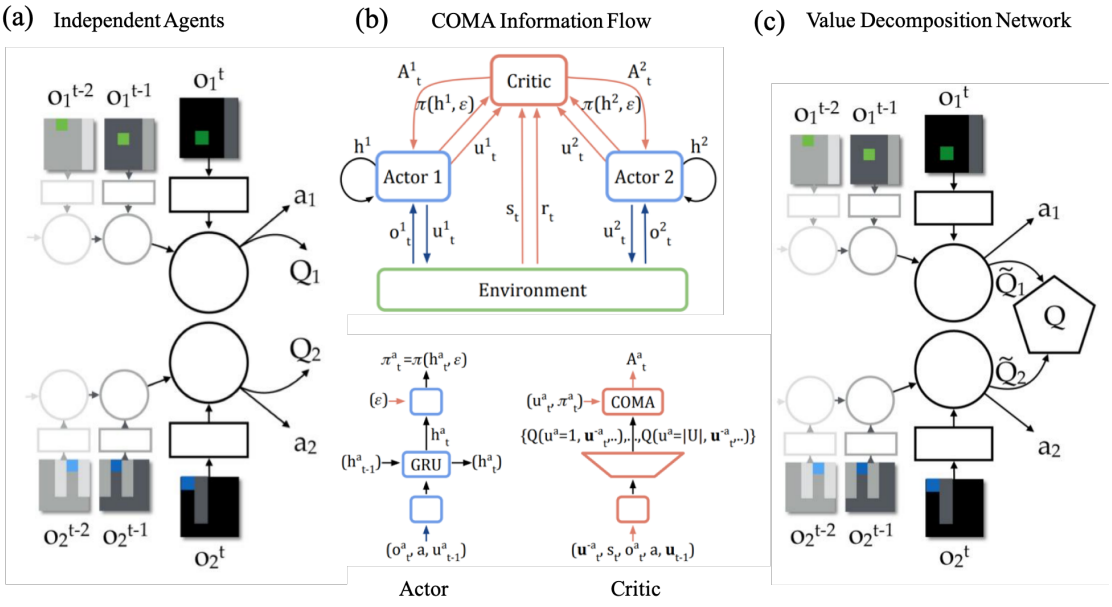

Figure 5: (a) Independent agents in a multi-agent Q-network [22]. (b) The diagram shows the information flow in COMA, between the decentralised actors, the environment and the centralised critic. During centralised learning, the red arrows and blocks are activated. The architectures of both the actor and critic are shown in the bottom [6]. (c) Value decomposition network architecture with a joint action value function Q [22]

3.2.3 VDN

Recently, Sunehag et *al.* [22] proposed a novel multi-agent learning approach based on team-reward. Thus, once again, different agents must jointly optimize a reward collected over time. This problem can be studied from two possible perspectives. The first is with a centralized approach, consisting on reducing the problem to a single-agent reinforcement learning task. However, as argued in [22], this can fail in practice. A second approach is to train independent learners to optimize for the team reward. Unfortunately, this can be problematic since each agent can only partially observed the environment.

To alleviate the aforementioned issues, [22] considered a learned additive value-decomposition approach over individual agents. The main idea consists of writing the Q-function as the sum of the value functions of different agents

$$Q((h^1,\dots,h^d),(a^1,\dots,a^d)) \approx \sum_{i=1}^{d} \tilde{Q}_i(h^i,a^i),$$

where the learning the functions $\tilde{Q}_i$, corresponding to each agent, can be done by back propagating gradients from the Q-learning using the joint reward.

Figure 5 (c) shows a pictorial comparison between training independent learners and the framework proposed by [22].

4 Applications

The field of MADRL has seen numerous real-world applications. This section reviews some representative applications of MADRL. It won't delve too much into the algorithm of the individual problems.

The first example comes from [11], where MADRL is applied to a multi-object tracker which aims to track the trajectory of objects in videos, such as tracking people on the street. Usually, videos are separated into several frames and a CNN based detector is used to do detection on each frame. In detail, the detector searches for a whole frame every time. However, the trajectories are always predictable, for example, people often walk along the road instead of standing still. Hence, MADRL is used to get a estimated trajectory of different objects, and so the detector can only detect near the estimated trajectory. In this work, IQL is applied, because people's action are always independent from one person to another. Therefore it is natural to use a CNN for detection and then a DQN for action estimation. The performance improves both in speed and accuracy. The algorithm pipeline and MADRL framwork are outlined in Fig.6 (a).

The second example is a complex dynamic power allocation in the wireless network problem [15]. The objective is to maximize a weighted sum-rate utility function which is extremely complex:

$$\begin{aligned} \max_{p} \quad & \sum_{i=1}^{n} w_i^{(t)} C_i^{(t)}(p) \\ \text{subject to} \quad & 0 \le p_i \le P_{\max}, \end{aligned} \tag{17}$$

where $w_i^{(t)}$ is a non-negative weight of link i at time t, $P_{\max}$ is the maximum power spectral density, and $C_i^t(p)$ measures the efficiency of link i at time t, see [15] for more details.

Originally, weighted minimum mean square error (WMMSE) algorithm are used to solve (17, but the algorithm is complex and time-consuming. As an alternative, MADRL are applied. Because this time agents are not independent, two DQNs are trained. The first one is coped by each agent to get their own Q-function, and the other one is a centralized DQN

trained by using the experiences gathered from all agents. The performance is better than existing state-of-the-art centralized algorithms.

One of the common applications of MADRL is Game AI. In Game AI, communication and cooperation are considered to be a non-trivial problem.

Many researchers have tried to make agents cooperate to each other and even learn human with human strategies by using MADRL [16]. For example, in a tank game, two players are expected to cooperate to protect the base. However, the reward is given when a single player beats an enemy so they do not protect base by themselves. As a result, human strategies are used to generate a goal map and goal tasks, and then A3C is used to train the controller. The performance is boosted three times. In addition, in a pong game [8], two agents compete with each other when it is a zero-sum game, however, when the reward is changed and it becomes a negative-sum game, two agents learned to cooperate with each other by not getting goals or even not to start a game. Consequently, a drop of expected Q values can be seen when they start a game.

Others focus on making agents communicate to each other so that they can finish a task as a team. For example, reinforced inter-agent learning [7] has been proposed, which combines independent Q-learning for action and communication selection. The network is divided into two Q function: Qu and Qm for the environment and communication. Two modifications were found to be important for performance: experience replay and feed in the actions u and m taken by each agent as inputs on the next time-step.

5 Challenges and Future Work

Though powerful in addressing high dimensional multi-agent problems, MADRL still encompasses a few challenges, including non-stationarity that arises from the *heterogenity* of agents, partial observability, extremely noisy observation with weak correlation to true environment state, and agent communications [16].

To deal with the non-stationarity issue that arises from concurrent learning of the agents, [20] proposes a lenient-DQN model(LDQN) 7 which incorporates a scheduled replay strategy. To adjust policy updates sampled from memory of experience replay, LDQN applies leniency with decaying temperature values, as shown in Fig.7 (a). LDQN shows enhanced performance in converging to optimal policies in a stochastic reward environment. In terms of addressing partial observaility, deep policy inference Q-network and deep recurrent policy inference Q-network learn by adapting the attention of the neural network to policy features and their own Q-values during training process, and demonstrate better performance compared to baseline DQN [16].

In the area of MADRL, there still remain many open research areas including off-policy, safety, heterogenity, and optimization [18].

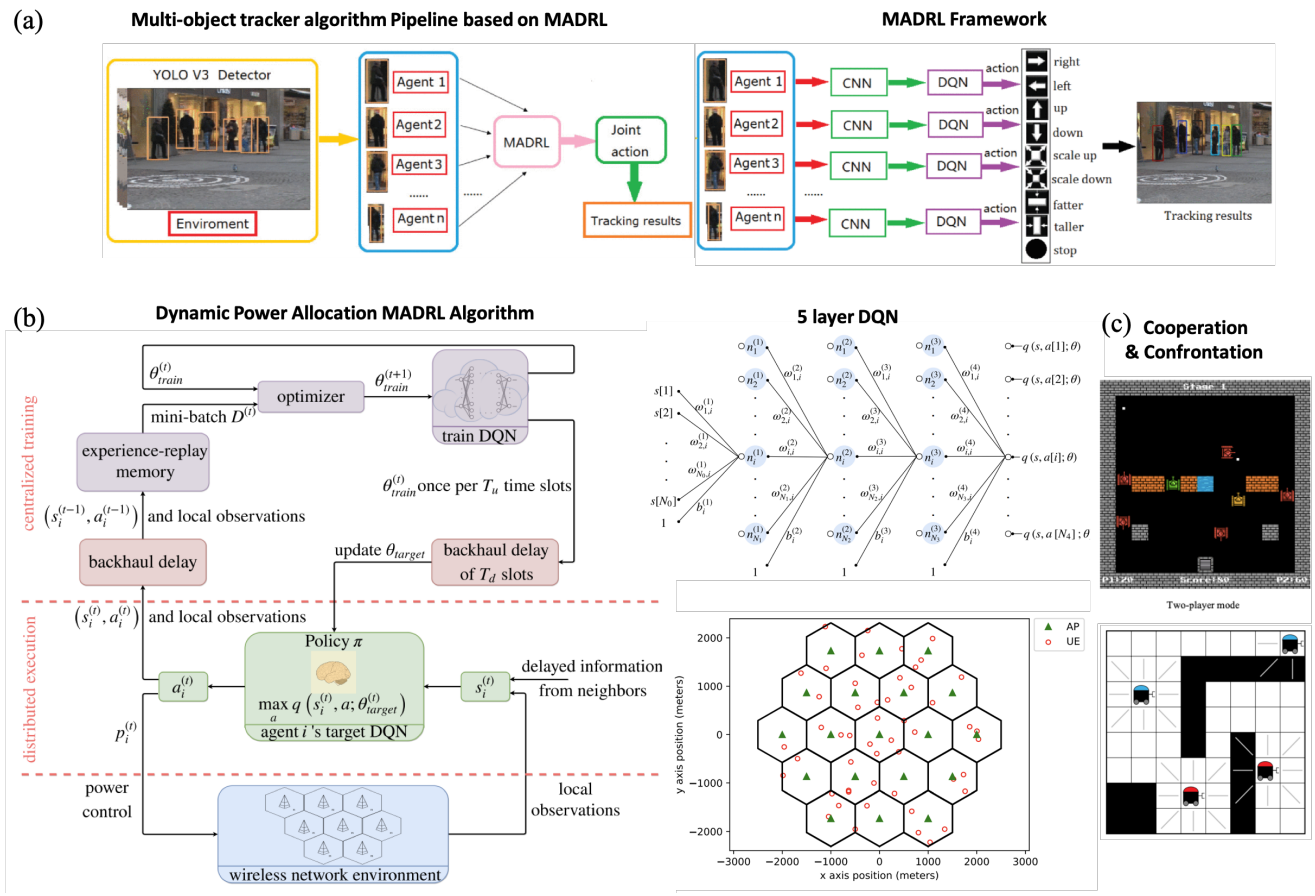

Figure 6: Representative applications of MADRL. (a) Multi-object tracker algorithm based on MADRL [11] (b) Dynamic Power Allocation MADRL Algorithm [15] (c) application in game AI [16, 4]

6 Concluding remarks

Recently there has been a surge of interest in multi-agent deep reinforcement learning combing the existing knowledge of multi-agent learning, reinforcement learning (RL) and the advantages of deep neural network. This paper gives a summary of evolution of multi-agent deep reinforcement learning and a survey of multiple approaches related to MADRL. It also picked a several representative applications to show how MADRL enables us to solve very complex problems involving multi-agent. The goal of this review paper is to provide insight of the advantages, potential of MADRL methods that could pave the way to more robust methods for multi-agent learning.

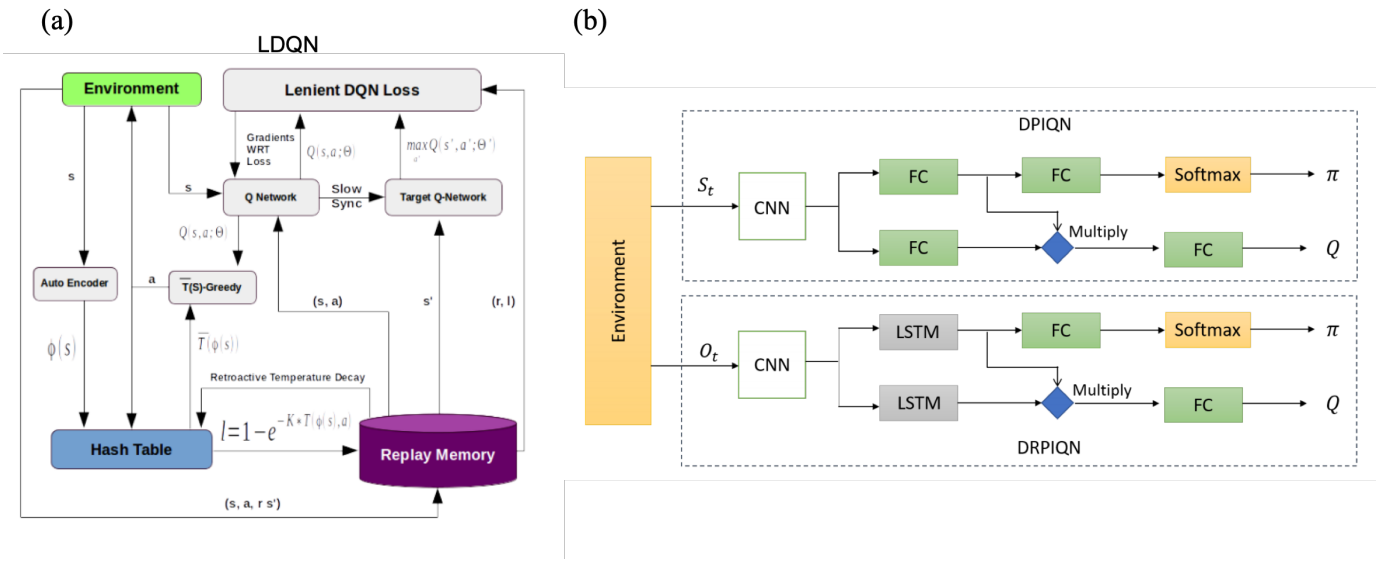

Figure 7: (a) Lenient-DQN architecture. (b) Deep policy inference Q-network (DPIQN) and deep recurrent policy inference Q-network (DRPIQ).

7 Acknowledgements

We would like to thank Dr. Ankur Mehta and Alexie Pogue for their instruction and guidance with this final project.

References

[1] Sherief Abdallah and Michael Kaisers. "Addressing environment non-stationarity by repeating Q-learning updates". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1582–1612.

[2] Kai Arulkumaran et al. "A Brief Survey of Deep Reinforcement Learning". In: *IEEE Signal Processing Magazine* 34.6 (Nov. 2017). arXiv: 1708.05866, pp. 26–38. ISSN: 1053-5888. DOI: 10.1109/MSP.2017.2743240. URL: http://arxiv.org/abs/1708.05866.

[3] Alvaro Ovalle Castaneda. "Deep reinforcement learning variants of multi-agent learning algorithms". In: *Master's thesis, School of Informatics, University of Edinburgh* (2016).

[4] Maxim Egorov. "Multi-agent Deep Reinforcement Learning". In: *CS231n: Convolutional Neural Networks for Visual Recognition* (2016).

[5] ETS Asset Management Factory. *What is the difference between deep learning and machine learning*. Aug. 1, 2019. URL: https://quantdare.com/what-is-the-difference-between-deep-learning-and-machine-learning/.

[6] Jakob Foerster et al. *Counterfactual Multi-Agent Policy Gradients*. 2017. arXiv: 1705.08926 [cs.AI].

[7] Jakob Foerster et al. "Learning to Communicate with Deep Multi-Agent Reinforcement Learning". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016, pp. 2137–2145. URL: http://papers.nips.cc/paper/6042-learning-to-communicate-with-deep-multi-agent-reinforcement-learning.pdf.

[8] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. "Cooperative multi-agent control using deep reinforcement learning". In: *International Conference on Autonomous Agents and Multiagent Systems*. Springer. 2017, pp. 66–83.

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

[10] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[11] Mingxin Jiang et al. "Multi-Agent Deep Reinforcement Learning for Multi-Object Tracker". In: *IEEE Access* 7 (2019), pp. 32400–32407. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2901300.

[12] Yuxi Li. "Deep Reinforcement Learning: An Overview". In: *arXiv:1701.07274 [cs]* (Nov. 2018). arXiv: 1701.07274. URL: http://arxiv.org/abs/1701.07274.

[13] Ryan Lowe et al. "Multi-agent Actor-critic for Mixed Cooperative-competitive Environments". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6382–6393. ISBN: 978-1-5108-6096-4. URL: http://dl.acm.org/citation.cfm?id=3295222.3295385.

[14] Hossam Mossalam et al. "Multi-Objective Deep Reinforcement Learning". In: *arXiv:1610.02707 [cs]* (Oct. 2016). arXiv: 1610.02707. URL: http://arxiv.org/abs/1610.02707.

[15] Yasar Sinan Nasir and Dongning Guo. "Multi-Agent Deep Reinforcement Learning for Dynamic Power Allocation in Wireless Networks". In: *IEEE Journal on Selected Areas in Communications* 37.10 (Oct. 2019), pp. 2239–2250. ISSN: 0733-8716, 1558-0008. DOI: 10.1109/JSAC.2019.2933973.

[16] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. "Deep reinforcement learning for multi-agent systems: a review of challenges, solutions and applications". In: *arXiv preprint arXiv:1812.11794* (2018).

[17] Thanh Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. "Multi-Agent Deep Reinforcement Learning with Human Strategies". In: *CoRR* abs/1806.04562 (2018). arXiv: 1806.04562. URL: http://arxiv.org/abs/1806.04562.

[18] Afshin OroojlooyJadid and Davood Hajinezhad. *A Review of Cooperative Multi-Agent Deep Reinforcement Learning*. 2019. arXiv: 1908.03963 [cs.LG].

[19] Afshin OroojlooyJadid and Davood Hajinezhad. "A Review of Cooperative Multi-Agent Deep Reinforcement Learning". In: *arXiv:1908.03963 [cs, math, stat]* (Sept. 2019). arXiv: 1908.03963. URL: http://arxiv.org/abs/1908.03963.

[20] Gregory Palmer et al. "Lenient Multi-Agent Deep Reinforcement Learning". In: *CoRR* abs/1707.04402 (2017). arXiv: 1707.04402. URL: http://arxiv.org/abs/1707.04402.

[21] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. DOI: 10.1038/nature16961.

[22] Peter Sunehag 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*. AAMAS '18. event-place: Stockholm, Sweden. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 2085–2087. URL: http://dl.acm.org/citation.cfm?id=3237383.3238080.

[23] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[24] Ardi Tampuu et al. "Multiagent cooperation and competition with deep reinforcement learning". In: *PloS one* 12.4 (2017), e0172395.

[25] Ming Tan. "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents". In: *In Proceedings of the Tenth International Conference on Machine Learning*. Morgan Kaufmann, 1993, pp. 330–337.

[26] Ming Tan. "Readings in Agents". In: ed. by Michael N. Huhns and Munindar P. Singh. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998. Chap. Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents, pp. 487–494. ISBN: 1-55860-495-2. URL: http://dl.acm.org/citation.cfm?id=284860.284934.

[27] Zhaoyang Yang et al. "Multi-Task Deep Reinforcement Learning for Continuous Action Control". In: *IJCAI*. 2017. DOI: 10.24963/ijcai.2017/461.

[28] Erik Zawadzki, Asher Lipson, and Kevin Leyton-Brown. "Empirically evaluating multiagent learning algorithms". In: *arXiv preprint arXiv:1401.8074* (2014).