Faculty of Mathematics
and Computer Science

Multiagent systems course (MAS 2022)

Overview of Multi-agent Reinforcement Learning in Game Playing

Ștefan Buciu

Department of Computer Science, Babeş-Bolyai University
1, M. Kogalniceanu Street, 400084, Cluj-Napoca, Romania

Abstract

Reinforcement Learning (RL) has been successfully applied in the field of AI game playing, with algorithms such as AlphaZero and MuZero achieving super human performance in board games, such as go. However, these algorithms are single agent based, meaning that there is only one agent that communicates with the environment. Extending the existing literature to multi-agent based algorithms has proven to be a challenging tasks. A couple of issues such as: nonstationarity, partial observaibility, and scalability have constantly been addressed by the scientific community, which has pivoted the start of new theoretical advancements.

Keywords:
reinforcement learning, rl, multi-agent rl, marl

1. Introduction

Intelligent game playing is a benchmark environment for AI Games, where algorithms are tested against simulated environments. Usually, when tackling problems, a good approach is to reduce the complexity and start small. Therefore, applying machine learning algorithms to games could prove useful in discovering their capabilities, limitations, and performances. This is especially true in the case of reinforcement learning, due to the fact that the game mechanics usually reward the player for taking a good action - representing the RL framework. Nowadays, a combination of ML the RL and DNN paradigms succeeded in producing agents that can overpower human champions in traditional board games such as chess and go, namely AlphaZero and MuZero which are the state-of-the-art model-based RL algorithms.

Due to the extraordinary empirical performance RL algorithms managed to achieve, researchers are now looking into ways of applying them to other fields such as autonomous driving. The academic community is also researching multi-agent reinforcement learning (MARL) and are trying to apply the existing expertise from the single-agent paradigm. However, this turns out to be a complex task, as the existing theory needs to be extended and generalized to accommodate for the multi-agent setting. Moreover, multiple challenges such as nonstationarity, scalability, and partial observability must be tackled in order to develop a good MARL theory that can produce algorithms on par of those of single-agent RL.

In this paper, we aim to give a brief overview of the basic theoretical foundation for MARL, the main challenges that MARL algorithms need to deal with, and present some algorithms and applications available in the literature.

2. Literature Review

2.1. Multi-Agent Framework

The theoretical basis of RL game playing it tightly coupled with Markovian theory, such as Markov decision process. We first give an overview over these processes and their generalizations, and afterwards we are going to have a look at MARL settings and challenges.

2.1.1. Markov Decision Process

In RL, one of the main components of an algorithm is the environment, thus it is extremely important how this environment is going to be modelled. In the case of single-agent RL, the most commonly known and used method of representation is the Markovian Decision Process (MDP). A MDP modelled environment has the property that given a state, we can move to the next state by choosing an available action. In an MDP, it is not important how we got to a certain state, but what we can do from there on. In a mathematical formalisation, an MDP is a tuple (S, A, P, R, γ), where $s \in S$ represents the **states**, $a \in A(s)$ represents the **actions** at a given state, $P(s'|s, a)$ represents the **probability** of transitioning to state s' from state s by taking action a, $R(s, a, s')$ represents the **reward** of the transition, and lastly, γ is called the **discount factor** and its responsible with balancing instantaneous versus future rewards [1].

2.1.2. Markov Game

Modelling multi-agent RL (MARL) environments using MDPs proves to be quite difficult, thus a generalization of MDPs is needed. Markov Games (MGs) are used to model MARL environments, and they can be formalized as a tuple $(N, S, \{A^i\}_{i\in N}, P, \{R^i\}_{i\in N}, \gamma$, where $N, N > 1$ represents the number of agents. The other components are similar in definition as in the case of MDPs, but due to the number of agents being greater than 1, we are dealing with multiple dimensions.

2.1.3. Partially-Observable Markov Decision Process

Another generalization of MDPs is partially-observable MDPs (POMDPs). The main difference in a POMDP is that some states could be visible or not, thus the name partially observable. Formally, POMDPs have a similar definition as in the case of MDPs, but with two additional stochastic processes: X_t and Y_t. The former is non observable and is responsible with the evolution of the state. Moreover, this process has a finite Markov chain representation. The latter is observable representing the observations made by agents.

$$q_{xy} \equiv P\{Y_t = y \big| X_t = x\} \tag{1}$$

In Equation 1, the relationship between X_t and Y_t is presented, and it represents the probability of making a certain observation $Y_t = y$ in a given state $X_t = s$ [1].

2.1.4. Decentralized Partially-Observable Markov Decision Process

Dec-POMDPs are similar with MGs, but with the additional partially-observable properties from POMDPS. The main aspect of this modelling approach is that the reward function R is shared by all agent [1, 8], making it applicable in cooperative settings.

2.1.5. *Nash Equilibrium*

The Nash equilibrium (NE) is a concept that comes from game theory and is highly important for MARL scenarios. In the case of MGs, NE represents a set of policies, one for each agent, such that all agents are not interesting in changing their own policy since it is the best response against all other agents' policies [4, 8].

2.2. *MARL Settings*

Based on how the environment is rewarding the agents, MARL algorithms can be divided into three categories [1, 8]:

- **Fully cooperative** - this setting is characteristic to scenarios where agents have a common goal and they are taking actions that maximize the long-term reward towards the collective's goal. A good example are the self-driving cars, which are communicating to avoid dangerous situations, like collisions.
- **Fully competitive** - in this setting, agents are competing against each other, striving to maximize their own rewards. Some good examples would be board games such as chess, or go. These games are also known as zero-sum games, because when one agent is gaining a reward, the other is punished for it (e.g., losing a piece in chess).
- **Mixed** - a combination of the previous settings makes a mixed one. In this case, agents collaborate and compete at the same time, and it usually characterizes team games. In such games, agents are helping their colleagues such that they are acting as a group towards their common goal. These kind of games are known as general-sum games.

2.3. *MARL challenges*

Even though single-agent algorithms have proven performance, and a strong theoretical basis, it turns out that trying to generalize the subject to multi-agent algorithms introduces new challenges that require new approaches and designs [1, 8]. These are summarized below.

2.3.1. *Nonstationarity*

In single-agent RL, the environment is stationary, meaning that only the agent's actions are modifying it. Moreover, in this setting, the RL algorithms have the Markov property, which is one of the reasons that they are performing so well. However, as soon as another agent is introduced, the environment becomes non-stationary, meaning that there are multiple sources that can modify it. Also, the Markov property is being lost in such cases due to the fact that now an agent would wrongly assume that the current state it is in depends only on the action it took from its previous state (i.e., another agent could have modified the state). Due to the fact that the policies that are based on Markov's property become obsolete in multiple-agent environments, the existing theoretical support is lost [8, 1].

There are a couple of approaches that try to deal with the nonstationarity challenge:

- **Independent learners (IL)**
  With this approach, algorithms basically ignore the nonstationarity issue and they model agents that are striving to optimize their own policy. These kind of algorithms usually fail to find an optimal policy, however under certain configurations they manage to attain a decent performance [8].
- **Joint action learners (JAL)**
  At the opposite end of the spectrum, JAL algorithms are taking into consideration the joint actions in computing the value function. This approach solves the issue, however the biggest downside is the increased space complexity. The joint action space has a complexity of $O(A^n)$ where A is the action space and n is the number of agents. With an exponential space complexity, these algorithms are difficult to scale. Moreover, another disadvantage lies in the need to ensure communication channels between agents, such that they know every actions available [1].
- **Lenient Multi-Agent Reinforcement Learning (LMRL)**
  LMRL is an improvement of the IL approach, but the agents have varying learning rates. By doing so, the agents do not necessarily try to improve their own policy, but to achieve the most optimal joint policy. In cooperative

games for example, the agent's action is graded by how beneficial it is to the entire group. The reward is shared and conditioned by the joint action, thus in the case that an agent took the optimal action for itself, but it hurts the group, then that agent is punished.

$$\begin{aligned} &\delta \leftarrow r - Q_i(a_i, s_i) \\ &Q_i(a_i, s_i) \leftarrow \begin{cases} Q_i(a_i, s_i) + \alpha\delta & \textbf{if } \delta \geq 0 \\ Q_i(a_i, s_i) & \textbf{else} \end{cases} \end{aligned} \tag{2}$$

In Equation 2, an heuristic is used which is able to create ”optimistic” agents, which have the property of ignoring negative rewards caused by the actions of others. The agents have a dynamic leniency factor which they show towards each other, which value is inversely proportional to the state-action pairs selected, and is continuously decreasing as the game progresses.

2.3.2. Scalability

This challenge is briefly touched on in the previous section. To briefly recap, if we choose the IL path, we are going to obtain poor results, and if we choose the JAL path, we are going to run into an exponential space complexity. Another solution would be to bring a decentralized approach to this issue. Agents are modelled into a network, and are able to communicate with the environment and to the neighbouring agents, resulting a loosely connection. This approach is called QD-learning, and is highly scalable and suitable to a wider range of applications. Similarly to the IL approach, agents are making local progress, but they are also informing their neighbours of their Q-values.

Another approach that tackles the scalability challenge is deep reinforcement learning (DRL). This is due to the fact that DRL is using neural networks, which are great at dimensionality reduction. Additionally, the combination of LMRL with DQN results in the lenient deep Q-network (LDQN) algorithm [1].

2.3.3. Partial observability

There are a lot of algorithms in the literature that are based on the full observability assumption. For example, in the case of games, if you are tackling the game of chess, you have all the information that you need available. But when applying the same algorithms to other applications, you might find out that the environment is partially observable. This is the case more frequently in real-world applications. For example, a vacuum cleaner agent only has information about its current position and the information it receives from sensors, but it does not know what it is happening in the entire room [1].

- **Centralized learning of decentralized policies**
  Usually, when dealing with real-world applications, algorithms are going to face both scalability and partial observability challenges. Self-driving cars are a prime example, such that, not every car will be able to know the full state of the city without a big overhead caused by communicating such information. Thus, it may not always be useful to know the full information when computing an optimal policy, however, it could be useful during training. This is what the centralized learning of decentralized policies paradigm is trying to address. Sometimes, during training, it is possible to have an environment simulator.
- **Communication between agents**
  In a cooperative setting with partial observability, communication between agents is often required to be able to perform a certain task, and the communication protocol could vary for each of them.

3. Discussion

3.1. Algorithms

In this subsection, we are going to have a look at some MARL algorithms that are addressing the main issues that we've discussed beforehand.

3.1.1. Counterfactual multi-agent policy gradient

The counterfactual multi-agent policy gradient (COMA) [2] addresses a couple of MARL issues, such as partial observability, communication constraints, and multi-agent credit assignment. COMA is modelled by using the stochastic games theory, and it is treating fully cooperative tasks by using the actor-critic and the centralised training of decentralized policies paradigms.

The main ideas of COMA are:

- **Centralized critic**
  This component is only used in training and it is not needed in execution. Its role is to condition the joint action space and on the available information. Moreover, the agents are taking actions only based on local observations.
- **Counterfactual baseline**
  This principle shapes the way agents receive rewards. An agent either receive a *global* reward, or a reward based on a *default action*. In literature, it is not strongly defined what makes a good default action, and additionally, this principle requires a simulator or an estimated reward function.
  COMA deals with these impediments by using the critic to compute an *advantage function* which is tailored for each agent. This function computes a reward estimate by fixing the actions of all the other agents and trying out all the available actions of the current agent.
- **Critic representation**
  COMA adopts a critic representation that allows for a batch computation of all Q-values for all agents in a single forward pass.

3.1.2. Parameter Sharing-Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) [3] is a single agent policy gradient algorithm. Its main idea is that it allows precise control over the policy improvement, with the help of the Kullback–Leibler divergence (as shown in Equation 3).

$$\begin{aligned} &\underset{\theta}{\text{Maximize}} \quad \mathbb{E}_{s\sim\rho_{\theta_k},a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A_{\theta_k}(s,a)\right] \\ &\text{subject to} \quad \mathbb{E}_{s\sim\rho_{\theta_k}}\left[D_{\text{KL}}(\pi_{\theta_k}(\cdot|s)||\pi_\theta(\cdot|s))\right] \leq \Delta_{\text{KL}} \end{aligned} \tag{3}$$

The TRPO algorithm can be extended to MAS by adopting parameter sharing (PS-TRPO). This allows for agents to better train their policies by using each other's experiences, allowing at the same time individual behaviour.

3.1.3. Hysteretic Q-learning

The Hysteretic Q-learning (HQ-Learning) [5] is applied in fully cooperative settings where teams of agents are working together to achieve a task. In this case, the actions taken need to benefit the team, thus a selfish action could result in a punishment for that particular agent. This represents the initial idea for *optimistic agents*, as shown in formula 2. The downside of this approach is that the agents could settle down for the local best policies by not managing to achieve a good coordination.

$$\begin{aligned} &\delta \leftarrow r - Q_i(a_i) \\ &Q_i(a_i) \leftarrow \begin{cases} Q_i(a_i) + \alpha\delta & \textbf{if } \delta \geq 0 \\ Q_i(a_i) + \beta\delta & \textbf{else} \end{cases} \end{aligned} \tag{4}$$

The HQ-Learning algorithm, as shown in Equation 4, employs two learning rates to mitigate the downsides of *optimistic agents*. α represents the increase rate, and β represents the decrease one. These rates are then added to the Q-values obtained by a joint action. HQ-Learning can also be applied to stochastic games, with the necessary adjustments being presented in Equation 5.

$$\begin{aligned} &\delta \leftarrow r + \gamma \max_{a'} Q_i(s', a') - Q_i(s, a_i) \\ &Q_i(s, a_i) \leftarrow \begin{cases} Q_i(s, a_i) + \alpha\delta & \textbf{if } \delta \geq 0 \\ Q_i(s, a_i) + \beta\delta & \textbf{else} \end{cases} \end{aligned} \tag{5}$$

3.1.4. Lenient Multi-Agent Deep Reinforcement Learning

Leniency is an approach that deals with relative generalization, a phenomena where agents are gravitating towards a sub-optimal joint policy. Relative generalization occurs when agents are being influenced by each other [7]. Leniency helps agents to converge to globally optimal solutions by using a leniency factor which decays over time, and initially, it helps guarding against sub-optimal actions selected by teammates. Equation 6 presents the leniency function which is responsible with lowering the Q-value of a state-action pair.

$$l(s_t, a_t) = 1 - e^{K * T_t(s_t, a_t)} \tag{6}$$

Lenient Multi-Agent Deep Reinforcement Learning (LDQN) [7] combines leniency with Deep Q-Networks (DQNs). The main idea of this approach is maintaining a dictionary during the training which maps a state-action pair to a temperature value. The temperature value is then used to ignore actions chosen from the replay memory, if they do not meet the leniency conditions.

3.1.5. Decentralized Hysteretic Deep Recurrent Q-Networks

The Decentralized Hysteretic Deep Recurrent Q-Networks (Dec-HDRQNs) [6] algorithm has been developed to be applied in non-stationary Dec-POMDP MARL applications. This algorithm combines HQ-Learning with DRQNs to be able to deal with non-stationarity and partial observability. HQ-Learning has been shown to be a robust approach for non-stationary application, and DRQNs useful for their representational power and memory-based decision making.

Omidshafiei et al. [6] obtained improved results, by adopting HQ-Learning to Dec-DRQN. The results for the combined approach can be seen in Figure 2, a drastic improvement over the results from Figure 1.

4. Conclusion

The theoretical basis for MARL is strongly linked to MDP. However, MDPs are well suited for single-RL applications, and extending this theoretical foundation has given a rise to a couple of issues which need to be addressed by MARL algorithms. Firstly, nonstationarity is the immediate product of introducing more than one agents in an environment. In such a setting, the current state is not necessarily dependant on the previous state and on a selected action,

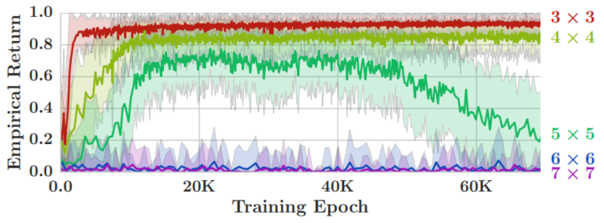

Fig. 1. Dec-DRQN [6]

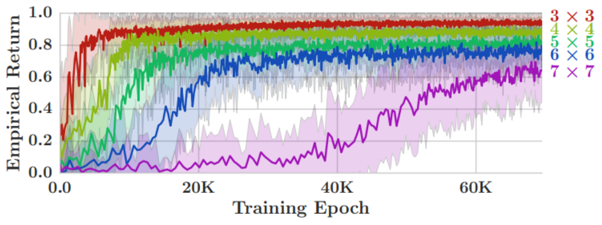

Fig. 2. Dec-HDRQN [6]

but on a joint-action. Secondly, having multiple agents causes scalability issues. This is partially due to the fact that algorithms are using joint-actions which results in exponential time complexities, however, the main obstacle is usually the communication and coordination aspect of MARL which usually results in task customized protocols. Lastly, the partial observability issues tend to be more frequent in MARL settings since observations could be localized to agent (i.e., an agent has some sensors). These local observations are unknown to other agents unless there is some sort of communication mechanisms. Additionally, the literature tends to apply MARL algorithms to more real-world applications, which are partially observable by nature.

References

[1] Canese, L., Cardarilli, G.C., Di Nunzio, L., Fazzolari, R., Giardino, D., Re, M., Spanò, S., 2021. Multi-agent reinforcement learning: A review of challenges and applications. Applied Sciences 11, 4948.

[2] Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S., 2018. Counterfactual multi-agent policy gradients, in: Proceedings of the AAAI conference on artificial intelligence.

[3] Gupta, J.K., Egorov, M., Kochenderfer, M., 2017. Cooperative multi-agent control using deep reinforcement learning, in: International conference on autonomous agents and multiagent systems, Springer. pp. 66–83.

[4] Lu, Y., Yan, K., 2020. Algorithms in multi-agent systems: A holistic perspective from reinforcement learning and game theory. arXiv preprint arXiv:2001.06487 .

[5] Matignon, L., Laurent, G.J., Le Fort-Piat, N., 2007. Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams, in: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE. pp. 64–69.

[6] Omidshafiei, S., Pazis, J., Amato, C., How, J.P., Vian, J., 2017. Deep decentralized multi-task multi-agent reinforcement learning under partial observability, in: International Conference on Machine Learning, PMLR. pp. 2681–2690.

[7] Palmer, G., Tuyls, K., Bloembergen, D., Savani, R., 2017. Lenient multi-agent deep reinforcement learning. arXiv preprint arXiv:1707.04402 .

[8] Zhang, K., Yang, Z., Başar, T., 2021. Multi-agent reinforcement learning: A selective overview of theories and algorithms. Handbook of Reinforcement Learning and Control , 321–384.