Multi-Agent Reinforcement Learning Overview, Applications, and Problem Proposal

Oliver Zanone
School of Electrical Engineering
Clemson University
Clemson, United States
rzanone@clemson.edu

***Abstract*—This paper presents an introduction to reinforcement learning and deep reinforcement learning, and their application to multi-agent problems. It explores the concept of fully decentralized learning and centralized learning/decentralized execution, and their applications to multi-agent systems and communication networks. It then presents an algorithm proposal based on the works of Zhang and Foerster that utilizes both ideas, and proposes a problem scenario involving two reinforcement networks working independently of one another. The problem involves a grid-like environment with limited sensory input and multiple agents attempting to cover an area while avoiding obstacles. The paper concludes with a reward system setup and a simulation example.**

***Index Terms*—reinforcement learning, deep learning, MDPs, multi-agent, communication networks**

I. Introduction

Reinforcement learning (RL) is a subset of machine learning in which an agent learns by interacting with the environment via trial and error. The agent creates a policy, or a distribution of actions to take, to maximize a reward. The overall goal then is to find the optimal set of actions given a certain environmental state such that the final reward is maximized [5] [3]. Before a reinforcement learning model can interact with the environment or decide what action is optimal to take, it must have some sense of what the environment is [5]. In classical reinforcement learning, this would require a feature engineer with domain knowledge to hand craft a feature map that would present the environment to the RL model [5]. This does not work for all scenarios, i.e. scenarios where domain knowledge is limited and features are hard to craft by hand. Some examples include vision and speech systems. Even if these features were hand-crafted the question would remain, are these systems able to generalize well? With recent achievements in deep learning, this is no longer an issue. Deep neural networks can be leveraged in conjunction with RL to enable automatic feature generation so that domain knowledge is either reduced or not required at all. The aim of this paper is to present an introduction to reinforcement learning and deep reinforcement learning, different areas reinforcement learning has been applied, and finally we present our own problem scenario.

II. Mathematical Background

This section looks to provide a mathematical background of the basic reinforcement learning problem. Reinforcement learning usually looks to solve sequential decision problems that can be modeled after a Markov decision process (MDP). Often times these MDPs focus on discrete time and finite spaces. The basic premise of an MDP is a decision maker chooses an action from an available set of actions. There are two consequences after choosing, firstly an immediate reward is recognized, and secondly, a probability distribution calculated for the subsequent system state. The decision maker then looks to find a set of actions called a policy such that the performance is optimized over a time horizon. The interested reader can see [6] to learn more about these processes.

In RL, the learning agent is the decision maker, as the algorithm learns which actions to choose in the MDP. Assuming a discrete MDP with finite states, at each time step $t \in T$ the agent observes the state $s_t \in S$ where S is a finite set of possible states. Then based on the policy $\pi(a_t, s_t)$ the agent chooses an action $a_t \in A$, A being the finite set of possible actions. The policy π gives a map of what action should be taken given the state for all action/state pairs. Once an action is taken, a scalar reward $r_t \in \mathbb{R}$ is given and the next state s_{t+1} is observed. The return at time t is modeled by:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

or the total discounted reward where $\gamma \in (0, 1]$ is the discount factor in which future rewards are discounted. This emphasizes finding a larger reward quicker. It also mathematically helps the objective function converge when the time horizon is infinite due to $\gamma < 1$. The goal of the agent then is to find a policy π that maximizes from every state s_t the expected discounted return $\mathbb{E}[R_t]$ or long-term performance using only immediate one-step performance (reward). There are several methods to find the optimal policy, but we will highlight the Q-learning. This uses an optimal state-action value function or Q-function. This function returns a scalar value of expected return using the current policy π for any state-action pair:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R_t | s_t = s, a_t = a]$$

The optimal Q-function then is:

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

and the optimal policy can be computed by choosing the largest Q-value for every state-action pair:

$$\pi^*(s) = \max_a Q^*(s, a)$$

Q-learning then is defined as an iterative process starting with an arbitrary Q-function. Upon observing transitions of states based on actions taken and rewards received, the Q-function can be updated [1]. The Q-function update is as follows:

$$\begin{aligned} Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + ... \\ ... + \alpha_k[r_{k+1} + \lambda \max_{a'} Q_k(s_t, a') - Q_k(s_t, a_t)] \end{aligned} \quad (1)$$

where α is the learning rate. Q-learning is a type of temporal difference learning and is one method of indirectly learning the optimal policy (value-based methods). Other methods of learning exist that fall into temporal difference learning such as SARSA but there are methods such as dynamic programming (value iteration and policy iteration), Monte Carlo (when all data is known), and then policy-based methods that directly update the policy [8].

The blanket term, deep reinforcement learning, is obtained when deep networks are used in any area of the RL problem such as state representation and/or to approximate any of the functions such as the value function, policy, and/or model. One example of deep learning improvement is in the aforementioned Q-learning. This has been furthered with the work titled Deep Q-Network (DQN) [4] which has achieved amazing results in Atari-style game play. For helpful tips on the implementation of DQN, the interested reader is directed towards [7].

This section has focused on a very simple background but there are many topics not covered here. It serves as a quick background/refresher to the reader.

III. RL for Multi-Agent Problems

Multi-agent reinforcement learning or MARL is simply reinforcement learning applied to multi-agent systems [3]. The primary difference is each agent's reward can be influenced by the actions of other agents so the joint action space of all agents must be considered [9]. There are several challenges when formulating problems, such as the dimensionality curse, where adding more agents exponentially grows possible state-action pairs. This problem also can exist in single agent RL, but is exacerbated with multiple agents. Note that independent learner techniques have been used successfully in some scenarios to remedy this scalability issue. In this, the agent only focuses on optimizing its own policy, ignoring that of other agents.

There are several proposed solutions and some superior to others dependent upon the problem definition. Does the problem require new problems to be solved online?

Often the MARL problem is categorized into task and goal categories. For the task, there is fully cooperative, fully competitive, and mixed. The target goal can be to obtain stability, adaptability, or a mix of both [1]. Another factor to consider are the learning goals of agents. The objective of agents are not necessarily aligned to one another, which leads to defining an equilibrium point for the system.

To extend the idea of single agent to multi agent, assume an environment of discrete time and finite action/state spaces. The states of the environment belonging to S are the same as in the single agent RL problem, but now there are $A_1...A_n$ action spaces corresponding to the number of agents. Likewise, there are $R_{t1}...R_{tn}$ return functions for each agent.

The simplest approach to this problem would be apply single agent learning to each agent, which can work depending on the problem definition. For problems which require more complexity, this approach does not work and requires multi-agent algorithms. The problem is fully cooperative when the rewards functions are the same, and competitive if they are direct opposites, i.e. $R_1 = -R_2$. The competitive problem is generally used when modeling zero-sum games, and the mixed problem is used for general-sum games.

One popular MARL learning scheme mainly used in cooperative settings is the category of centralized learning, decentralized execution. This is mathematically eloquent as it allows analysis using tools from single-agent RL to prove convergence and other properties [9]. This assumes that there is a central controller that can collect all of the information such as joint actions, rewards, observations, and design policies for agents. However, this is only realistic in simulation or game-like settings. For real-world scenarios or when a central controller does not make sense, fully decentralized learning must be applied. This relies on agent to agent communication to exchange information with one another through some sort of protocol. This type of learning scheme is referred to as decentralized networked agents [9]. Deep MARL has also been applied to the problem of creating the network protocol with some success, see [2]. The proceeding sections will focus on the cooperative setting using decentralized networked agents.

A. The Cooperative Decentralized Setting

This area of MARL comes from a standpoint of practicality. When deploying agents in real-world scenarios, often they involve collaboration, have no central controller, and communicate over a sparse network. This comes up in robot team navigation, smart grid operation, and multi-drone mission planning. This section will focus on the works of Zhang [10] and highlight some of their algorithms.

Zhang defines the decentralized multi-agent problem as a an environment where the state space and joint action space is globally known, but the rewards are only observable locally, and actions are executed locally. When the state space is large, it is assumed that the joint policy belongs to the parametric function class, or functions are approximated through parameters $\in \Theta$.

To fully understand these algorithms, first we must look at an update method that uses both the value function and direct policy updating to train referred to as the actor-critic update method. It can be thought of as a team, where the critic updates the action value function parameters, and then the

actor updates the policy parameters in the direction suggested by the critic [3].

An advantage function is used for the critic, which calculates the agent's temporal difference error or prediction error. In order to do this, it is required to estimate the the action-value function defined as Q_θ (a global estimate) of policy π_θ. Q_θ belongs to a set of parameterized functions, and each agent maintains its own parameter w^i to locally estimate the function $Q(.,.;w)$. This is the agent's local estimate of the Q_θ. This parameter w^i is information shared with neighboring agents over a defined network structure in order to reach a consensual estimate of Q_θ. This parameter sharing is the consensus update and is defined using a weight matrix $C_t = [c_t(i,j)]_{NxN}$, where $c_t(i,j)$ is the weight on the message transmitted from i to j at time t. The iterative update process is as follows:

$$\mu^i_{t+1} = (1 - \beta_{w,t}) \cdot \mu^i_t + \beta_{w,t} \cdot r^i_{t+1}$$

where μ^i_t tracks the long term return of agent i and β is a stepsize.

The parameters are updated with:

$$\tilde{w}^i_t = w^i_t + \beta_{w,t} \cdot \delta^i_t \cdot \nabla_w Q_t(w^i_t),$$
$$w^i_{t+1} = \sum_{j \in \mathcal{N}} c_t(i,j) \cdot \tilde{w}^j_t$$

where $Q_t(w) = Q(s_t, a_t; w)$ and the local TD-error δ^i_t is defined as:

$$\delta^i_t = r^i_{t+1} - \mu^i_t + Q_{t+1}(w^i_t) - Q_t(w^i_t)$$

Finally, the actor step can be implemented for each agent i to improve its policy using:

$$\theta^i_{t+1} = \theta^i_t + \beta_{\theta,t} \cdot A^i_t \cdot \psi^i_t$$

where $\beta_{\theta,t} > 0$ is the stepsize, A^i_t is the local advantage function, and ψ^i_t is a sample of the score function:

$$A^i_t = Q_t(w^i_t) - \sum_{a^i \in \mathcal{A}^i} \pi^i_{\theta^i_t}(s_t, a^i) \cdot Q(s_t, a^i, a^{-i}_t; w^i_t),$$

$$\psi^i_t = \nabla_{\theta^i} \log \pi^i_{\theta^i_t}(s_t, a^i_t)$$

where a^{-i}_t represents all actions except for a^i.

B. Algorithm

Therefore, the learning procedure can be summarized as follows. Initial parameters $\mu^i_0, w^i_0, \tilde{w}^i_0, \theta^i_0$ are defined for all $i \in \mathcal{N}$. The initial state s_0 of the Markov decision process and stepsizes $\beta_{w,t}, \beta_{\theta,t}$ are initialized to some starting state as well. Each agent i executes an initial action a^i_0 based on the policy $\pi^i_{\theta^i_0}(s_0, .)$ and observes the joint actions of its neighboring agents $a^1_0, \ldots, a^N_0$. Then for all agents, observe the next state s_{t+1} and reward r^i_{t+1}. Update the long term return or μ^i_{t+1} and select the next action a^i_{t+1} based off the new policy $\pi^i_{\theta^i_t}(s_{t+1}, .)$. Once looping through agents is complete, observe all joint actions a_{t+1}. Again loop through all agents updating the local TD-errors, weights, advantage functions, and score functions as part of the critic step or $\delta^i_t, \tilde{w}^i_t, A^i_t, \psi^i_t$. For the actor step, update the policy parameter θ^i_t. Now share $\tilde{w}^i_t$ with other agents according to some network structure $\mathcal{G}$. Finally, for all agents perform the consensus step w^i_{t+1}. Repeat this process until convergence.

For our research, we will focus this algorithm proposal. Zhang continues their work with another algorithm proposal, which is not reliant upon needing a^i_{t+1}, and also proves convergence of these algorithms. The interested reader is referred to [10] in order to learn more.

C. Communication Networks in MARL

This section focuses on how RL has been applied to communication networks looking at the works of Foerster [2]. The authors set up a problem scenario following centralized learning, decentralized execution. This is adequate for a communication network that doesn't involve change real-time, and that can be trained in a simulation. We assume there are multiple agents that have partial observability. At each time step, agents share a communication action denoted $m^i_t \in M$ that is observed by other agents but does not directly impact the environment or the reward function. In partially observable environment, agents then are encouraged to communicate, and therefore must develop and agree upon some sort of protocol to complete a task. A challenge arises when implementing this type of system as there are two steps for a reward to be realized. Firstly, the agent must send a useful message to another agent. Secondly, the agent that received the message must know how to properly interpret and then act upon the message. It is only then that the global return function will change. Therefore, the rewards that are realized may only happen sporadically.

Several algorithms are produced to solve this problem scenario, but the most notable is DIAL or differentiable inter-agent learning. This algorithm structure allows for something similar to feedback loops during their communication. This type of behaviour is closely related to how two humans would communicate where the listener acknowledges the speaker with constant non-verbal queues [2]. Although the paper does not go into algorithm details, it can be summarized. During the centralized learning step, agents are directly connected to one another, from one input network to the other's output network. Communication is still limited to discrete messages, but this messages can be sent as real-valued messages during the training process. This allows gradient propagation to happen throughout the whole network, which reduces the amount of trial and error learning. The authors propose a network they term a C-Net, which outputs two types of values, the Q-values for environment actions, and the real-valued vector messages to other agents. The authors continue to show very promising experimental results that shows how powerful learning a communication network can be. Source code for this algorithm was provided, but the complexity of it would require a considerable amount of time to fully comprehend.

IV. Problem Proposal

We would like to look at a problem scenario that utilizes concepts from both the decentralized learning scenario and the sparse communication network scenario. Therefore, we propose two reinforcement networks working independently of one another. We begin by setting up an MDP problem involving multiple agents. They communicate over a network $\mathcal{G}$ that is sparse and time varying. This network is learned by the DIAL network aforementioned, as training relies on centralized learning. Then, a fully decentralized MARL network is used to solve the MDP. This problem can be defined by the tuple $(\mathcal{S}, \mathcal{A}^i, P, R^i, \mathcal{G}_t)$ where $i \in \mathcal{N}$ and $\mathcal{N}$ is the number of total agents. We use the standard notation for state space, action space, state transition probability, and return. Similar to work by Zhang, we assume that the state space and joint action space is observable globally, but the reward only locally. Actions performed by agents are done so locally as well.

We look at a typical area coverage problem, a grid like environment where there are obstacles that must be avoided. Agents have limited sensory input, and can only see one grid space in the surrounding area. Agents are expected to learn to communicate their heading with one another, but nothing more. Agents are not aware of each other's initial positioning, or current positioning for $\forall t \geq 0$. They can only sense each other when the sensors are within range of the agents. A simulation was built involving this idea in Python and can be seen in Fig. 1.

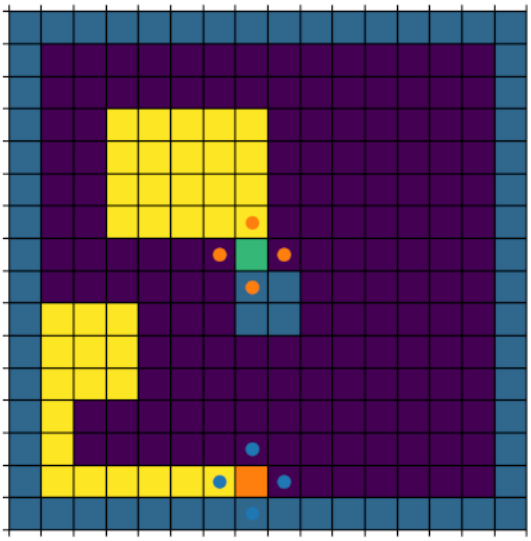

Fig. 1. Coverage simulation example with two agents. Their covered areas can be seen in yellow. Sensory input is shown by surrounding points.

The reward setup is outlined as can be seen in table I When the agents find each other, we offer a reward. However this happens only for the first occurrence. This behaviour is encouraged as the agents may now be spatially aware of one another via the communication network. At the end of the simulation, several additional rewards/penalties are appended based on how much overlap there was between the two agents' coverage areas.

TABLE I
Reward System

Action	Value
Moves to an unoccupied space	+1
Finds fellow agents	+1
Runs into wall/obstacle/agent	-0.5
Overlapping own coverage	-0.5
Does nothing	-0.1

V. Results

Most of this information was new to the author. Some preliminary testing was done implementing single agent basic algorithms such as Q-learning, but the full extent of what was proposed was not able to be implemented. This is very fascinating work, and more time would like to be dedicated to this type of research in the future. The final loss function can be seen in 2.

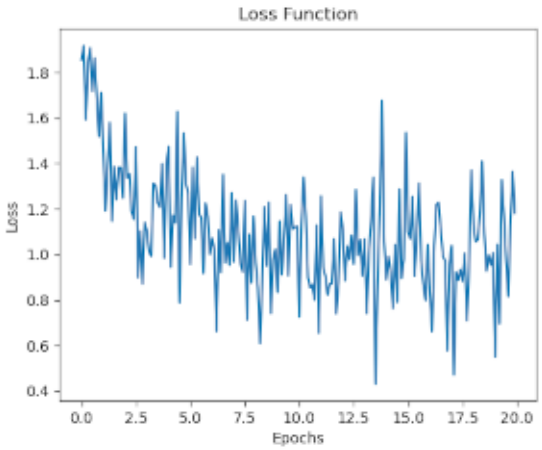

Fig. 2. Loss function when applying a single agent Q-learning algorithm.

VI. Conclusion

In conclusion, reinforcement learning (RL) is a powerful tool for machine learning that enables an agent to interact with an environment via trial and error to maximize a reward. By leveraging deep learning with RL, automatic feature generation can be achieved, allowing for domain knowledge to be reduced or not required at all. Multi-agent reinforcement learning (MARL) is an extension of this concept allowing for multiple agents to interact in the same environment. Recent research has proposed algorithms that leverage centralized learning and decentralized execution for communication networks, as well as decentralized learning for MARL. We proposed an algorithm utilizing both of these approaches to solve a coverage problem that involves sparse communication networks. More research is needed to fully understand the implications of this type of problem, and how it can be applied to real-world scenarios.

REFERENCES

[1] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. *Multi-agent Reinforcement Learning: An Overview*, pages 183–221. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[2] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[3] Yuxi Li. Deep reinforcement learning. *CoRR*, abs/1810.06339, 2018.

[4] 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.

[5] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. Deep reinforcement learning: An overview. In Yaxin Bi, Supriya Kapoor, and Rahul Bhatia, editors, *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016*, pages 426–440, Cham, 2018. Springer International Publishing.

[6] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[7] Melrose Roderick, James MacGlashan, and Stefanie Tellex. Implementing the deep q-network. *arXiv preprint arXiv:1711.07478*, 2017.

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

[9] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. *Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms*, pages 321–384. Springer International Publishing, Cham, 2021.

[10] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. In *International Conference on Machine Learning*, pages 5872–5881. PMLR, 2018.