Multi-Agent Reinforcement Learning: A Report on Challenges and Approaches

Sanyam Kapoor

sanyam@nyu.edu

July 26, 2018

Abstract

Reinforcement Learning (RL) is a learning paradigm concerned with learning to control a system so as to maximize an objective over the long term. This approach to learning has received immense interest in recent times and success manifests itself in the form of human-level performance on games like *Go*. While RL is emerging as a practical component in real-life systems, most successes have been in Single Agent domains. This report will instead specifically focus on challenges that are unique to Multi-Agent Systems interacting in mixed cooperative and competitive environments. The report concludes with advances in the paradigm of training Multi-Agent Systems called *Decentralized Actor, Centralized Critic*, based on an extension of MDPs called *Decentralized Partially Observable MDP*s, which has seen a renewed interest lately.

Contents

List of Figures

3

1 Introduction

1.1 The Reinforcement Learning Problem

Reinforcement Learning (RL) refers to both the learning problem and sub-field of machine learning. The learning problem is to control a system so as to maximize a numerical value which represents a long-term objective. The learner is known as the *agent* and everything outside the agent is known as the *environment*. The *agent* selects *actions* and the *environment* responds by presenting a *reward* and a new *state*. A canonical view of this feedback loop is shown in Figure 1.
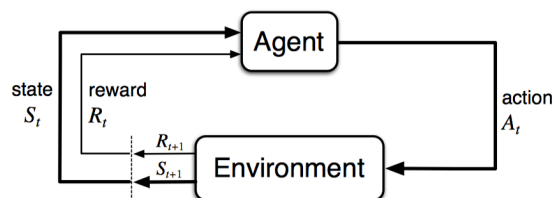

Figure 1: The agent-environment feedback loop [Sutton and Barto, 1998]

Reinforcement Learning, in a sense, is the most general formulation of the learning problem. Unlike Supervised Learning, the feedback is partial and in many cases the rewards are delayed. It also differs from Unsupervised Learning because the aim is not to find hidden structure in unlabeled data but to solely maximize the reward signal. This importance of the reward signal is embodied by an informal idea known as the *reward hypothesis* [Sutton and Barto, 1998].

Hypothesis 1 (**The Reward Hypothesis**). *That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).*

The Reward Hypothesis is a matter of ongoing discussion and has not yet received a convincing consensus[1]. An RL researcher is conventionally expected to come up with a good reward function and subsequently provide a robust RL algorithm to generalize to unseen trajectories of the feedback loop seen in Figure 1. The process of designing rewards for the problem is known as *reward shaping*. It has had apparent criticism because deciding the right reward is a crucial and delicate matter. Exhaustively modeling dependencies in the environment can become messy very quickly. This limitation has kept the reward functions in the literature

---

[1]Some perspectives can be read at `http://incompleteideas.net/rlai.cs.ualberta.ca/RLAI/rewardhypothesis.html`

to be relatively simple and easy to interpret. For instance, in a zero-sum game the rewards are sparse (+1 for a win and −1 for a loss). In other cases, where time is critical to achieve the objective, a reward of −1 is given until the agent reaches the objective.

In the following discussion, this hypothesis is assumed to be true. This assumption is motivated by the promise shown in recent works which use fairly simple and interpretable reward structures. The reader is however invited to read a more formal discussion around *reward shaping* in [Ng et al., 1999].

1.2 A Note on Evolutionary Computation

It is worth acknowledging *Evolutionary Computation* (EC) as an alternative formulation of the problem of learning autonomous agents. This is a family of techniques in which abstract *Darwinian* models of evolution are applied to refine populations of candidate solutions to a given problem. The refinement happens when the fitness function is used to modify the current population with a breed of new individuals via *Genetic Algorithms* (GA) and *Evolution Strategies* (ES). Interested readers are invited to read [DeJong and Jong, 2002].

EC is different from RL in that there is no explicit interaction between the environment and the agent (or an individual in EC). The interaction, if any, happens implicitly via the fitness function leading to selection, mutation or breeding of individuals. As a result, these methods ignore much of the structure that is available in the Reinforcement Learning setting - they don't observe the trajectory of states & actions that an agent goes through to achieve the objective. The set of trajectories can provide a rich signal to be exploited and potentially generalize unseen environments better.

Subsequently, a detailed discussion on EC, the comparison of EC and RL algorithms and the idea of hybrid EC-RL based approaches are beyond the scope of this work. Recent work by [Salimans et al., 2017] discovers that *Evolution Strategies* represent simple hill- climbing in a high-dimensional space based only on finite differences and might be of interest to relevant audience.

1.3 Motivating the Multi-Agent Setting

The grand vision of Artifical Intelligence since its inception has been to build autonomous agents that can interact with the environment and amongst each other. RL formulation of the learning problem for single agents comes the closest to this vision. Most successes in RL have been in Single Agent domains where the

environment stays largely stationary. It is also promising to see RL being used in large scale systems such as data center cooling [dee,]. Nevertheless, progress in Multi-Agent RL Systems is due.

A number of complex problems in today's society can be modeled as a Multi-Agent Learning problems. A few examples include Multi-robot control [Matignon et al., 2012], analysis of social dilemmas [Leibo et al., 2017], managing air traffic flow [Agogino and Tumer, 2012] and energy distribution [Pipattanasomporn et al., 2009]. Traditional RL algorithms are poorly suited for such problems as we will discuss in forthcoming sections.

The *StarCraft II Learning Environment* [Vinyals et al., 2017] has emerged to be a popular testbed for Multi-Agent RL algorithms because it allows for granular control over the objectives and constraints in the environment map. While being a formidable environment to be mastered given the enormous complexity in the interactions, my focus recently has been on a simpler and interpretable environment called Pommerman [pom,]. This environment derives from the Bomberman game where four agents compete each other to be the last one standing (or in a team the last team standing).

The rest of the report organized as follows - Section 2 discusses the key theoretical underpinnings behind Reinforcement Learning. Section 3 expands on the theory to provide a review of approximate approaches to RL problems. Section 4 discusses the unique challenges that a Multi-Agent environment faces. Section 5 gives a non-exhaustive review of some of the recent approaches tackling the multi-agent problem. Section 6 presents research goals and expectations from the future.

2   Formal Background

This section highlights the key definitions and theoretical underpinnings of modern Reinforcement Learning algorithms.

2.1   Markov Decision Processes (MDPs)

Markov Decision Processes (MDPs) allow a mathematically idealized formulation of the reinforcement learning problem and have their origins in dynamical systems. The agent and environment interact at discrete time steps t. At each time step the agent receives a *state* S_t and decides to take an action A_t. The environment responds by giving a reward R_{t+1} and a new state S_{t+1}. Therefore, this repeated

process gives rise to a sequence known as the *trajectory*

$$S_0, A_0, R_1, S_1, A_1, \ldots \tag{1}$$

The *dynamics* of the environment are defined by a probability distribution which defines the probability that taking an action A_{t-1} in state S_{t-1} will give a reward R_t and move the agent to state S_t. This part of the system is outside the control of agent(s). The exclusive dependence on just the current state and not the complete history makes this process Markovian and is crucial to various theoretical properties. We now see a formal definition of MDPs.

Definition 1 (**Markov Decision Process**). *A Markov decision process is a tuple* $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$ *such that*

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{2}$$

where $S_t \in \mathcal{S}$ *(state space),* $A_t \in A$ *(action space),* $R_t \in \mathcal{R}$ *(reward space) and* p *defines the dynamics of the process.*
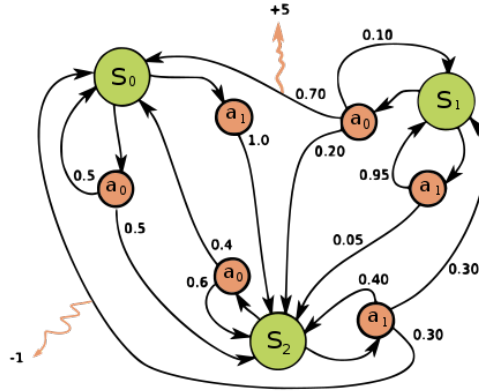

Figure 2: Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows). *by Waldoalvarez distributed under a CC-BY 4.0 license*

Before we go on to define the objective of a Reinforcement Learning agent, we define the notion of returns. *Returns* are one way of capturing the "long-term" objective of an RL agent.

Definition 2 (**Discounted Returns**). *Discounted Return is defined as the total sum of rewards following a time step t until the end of the sequence of rewards discounted by a factor γ at each time step*

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{3}$$
$$G_t = R_{t+1} + \gamma G_{t+1}$$

where $R_i \in \mathcal{R} \; \forall i$ and $\gamma \in [0, 1]$.

The notion of discounted returns is important for it being a good objective for the "long-term" because it controls the degree of importance of future rewards for the current time step. An agent which tries to maximize the objective with $\gamma = 0$ is called "myopic" and $\gamma = 1$ gives us the undiscounted returns, being more sensitive to changes in rewards from the future.

It is important to note that $\gamma < 1$ for *continuing* tasks because the *returns* must not diverge. We will use *returns* and *discounted returns* interchangeably from now onwards.

2.2 Value Functions

We are now ready to define two imperative value functions which serve as the objective of almost all reinforcement learning algorithms.

Definition 3 (**Policy**). *A policy is defined as the probability distribution of actions at a given states.*

$$\pi(A_t = a \mid S_t = s) \; \forall S_t \in \mathcal{S} \tag{4}$$

where $A_t \in \mathcal{A}(s)$ is the state specific action space.

As with any probability distribution, $\sum_a \pi(A_t = a \mid S_t = s) = 1$. When the agent follows a *policy*, it gives rise to a trajectory as seen in Equation 1.

Definition 4 (**State Value Function**). *Value function of a state s under policy π is defined as the expected return when starting in state s and following a policy π to take actions*

$$V^\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s] \; \forall s \in \mathcal{S} \tag{5}$$

Definition 5 (**Action Value Function**). *Value function of a state s and action a under policy π is defined as the expected return when starting in state s, taking action a and following a policy π to take actions further.*

$$Q^\pi(s, a) = \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a] \; \forall s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{6}$$

2.3 Bellman Equations

The *value functions* defined above take a nice recursive form which are defined by the *Bellman* Equations [Bellman, 1957] from Dynamic Programming literature.

Definition 6. *The Bellman Expectation Equation for $V^\pi(s)$ is given by*

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}_\pi\left[G_t \mid S_t = s\right] && (7)\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right]\\
&= \sum_a \pi(a|s)\left[\sum_{s',r} p(s',r|s,a)\left[r + \gamma\mathbb{E}\left[G_{t+1}|S_{t+1} = s'\right]\right]\right]\\
&= \sum_a \pi(a|s)\left[\sum_{s',r} p(s',r|s,a)\left[r + \gamma V^\pi(s')\right]\right]\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma V^\pi(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

The above expectation considers all possibilities of actions by the policy and the induced states by those actions defined by the environment dynamics. Similarly, we have

Definition 7. *The Bellman Expectation Equation for $Q^\pi(s,a)$ is given by*

$$
\begin{aligned}
Q^\pi(s,a) &= \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] && (8)\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma Q^\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]
\end{aligned}
$$

It turns out, that for a *finite* MDP, both the Bellman equations above have a unique solution that can be solved by a system of linear equations defined recursively in definitions 6 and 7. A concise matrix solution is given in Equation 9.

$$
\mathbf{V}^\pi = \mathcal{R}^\pi + \gamma\mathcal{P}^\pi\mathbf{V}^\pi \tag{9}
$$

$$
\implies \mathbf{V}^\pi = (\mathbf{I} - \gamma\mathcal{P}^\pi)^{-1}\mathcal{R}^\pi \tag{10}
$$

where $\mathbf{V}^\pi$ represents the value vector for each node in a Markov decision process under a policy π and $\mathcal{P}^\pi$ represents the state transition matrix. It should be noted that the runtime of this form is prohibitive in practice ($O(n^3)$) and we will see practical solutions in §3. This also forms the solution for action value because of the following relation

$$
V^\pi(s) = \sum_a \pi(a|s)Q^\pi(s,a) \tag{11}
$$

Hence, the objective of an RL agent is to maximize these expected value objectives. The next two optimality equations form the basis of solving the control problem in Reinforcement Learning as we discuss in the further sections.

Definition 8. *The Bellman Optimality Equation for $V^\star(s)$ is given by*

$$
\begin{aligned}
V^\star(s) &= \max_\pi V^\pi(s) \\
&= \max_a Q^\star(s, a) \\
&= \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V^\star(s')\right]
\end{aligned}
\tag{12}
$$

Definition 9. *The Bellman Optimality Equation for $Q^\star(s, a)$ is given by*

$$
\begin{aligned}
Q^\star(s, a) &= \max_\pi Q^\pi(s, a) \\
&= \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} Q^\star(s', a')\right]
\end{aligned}
\tag{13}
$$

An extensive discussion on MDPs is presented in [Dimitri, 2017].

3   Reinforcement Learning and Control

As seen previously in matrix form, finite MDPs have a unique solution for the *state-value* and *action-value* functions. However, the runtime is prohibitive and this section discusses the general frameworks for practical solutions.

The two classic iterative approaches to solve the Dynamic Programming problems are known as the *Value Iteration* and *Policy Iteration*. These approaches are only possible when the environment dynamics are known to the agent and in most modern problems of relevance, that is not the case. Even when the environment dynamics are known, if the state or action space grows very large, these tabular methods will not be feasible. Hence, in the interest of brevity and space, this approach has been omitted from discussion. Interested readers can refer [Sutton and Barto, 1998].

In the absence of environment dynamics or a large state-action space, function approximators are imperative. Most Reinforcement Learning techniques today can be broadly classified into either approximation of the table of state and action values, learning a policy distribution for each state or a mixture of the two. Since, most of these methods require the calculation of expectation over the trajectories induced by a policy, Monte Carlo (MC) methods are used to approximate the expectation values.

10

3.1 Q-Learning

Q-Learning [Watkins and Dayan, 1992] has been one of the most influential methods in Reinforcement Learning. The objective here is to learn the *action-value* function $Q^\pi(s, a)$ for policy π by minimizing the expected loss $\mathcal{L}(\theta)$.

$$\mathcal{L}(\theta) = \mathbb{E}_\pi \left[(Q_\theta(s, a) - y)^2 \right] \tag{14}$$

where $y = r + \gamma \max_{a'} Q_{\theta'}(s', a')$. y represents the Q-Learning target value. Since, the expectation is not directly computable, it is approximated by sampling a large number of trajectories following a suitable coverage policy for exploration like ϵ-greedy or Boltzmann exploration [Cesa-Bianchi et al., 2017].

Q-Learning in its vanilla form tends to be highly unstable especially in the form where Deep Neural Networks are used for function approximation. Deep Q-Learning [Mnih et al., 2015] was proposed to overcome this problem by introducing the notion of target network whose parameters are kept constant for a finite number of training steps and is used to generate the values for y. The other technique used to stabilize the performance and overcome the problem of catastrophic forgetting is to use an Experience Replay Buffer from which transitions are sampled at random. This also helps break correlation between the sequential transitions from trajectories generated by the policy π.

3.2 Policy Gradient Methods

Policy Gradient methods differ from Q-Learning in the sense that they explicitly learn a stochastic policy distribution π_θ parametrized by θ. One natural choice for the objective here is to maximize the expected return over the trajectories induced by the policy π_θ. If we denote the reward of a trajectory τ generated by policy $\pi_\theta(\tau)$ as $r(\tau)$

$$J(\theta) = \mathbb{E}_{\pi_\theta} [r(\tau)] = \int \pi_\theta(\tau) r(\tau) d\tau \tag{15}$$

$$\nabla J(\theta) = \int \nabla \pi_\theta(\tau) r(\tau) d\tau \tag{16}$$

$$= \int \pi_\theta(\tau) \nabla \log \pi_\theta(\tau) r(\tau) d\tau$$

$$= \mathbb{E}_{\pi_\theta} [\nabla \log \pi_\theta(\tau) r(\tau)]$$

11

Now, the probability of generating the trajectory is

$$\pi_\theta(\tau) = \mathcal{P}(s_0) \prod_{t=1}^{T} \pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t) \tag{17}$$

where $\mathcal{P}$ refers to the ergodic distribution of the MDP. In simple terms, it is the probability of the agent being in some initial state. Taking the log and the gradient reveals a surprisingly beautiful result

$$\log \pi_\theta(\tau) = \log \mathcal{P}(s_0) + \sum_{i=1}^{T} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t) \tag{18}$$

$$\nabla \log \pi_\theta(\tau) = \sum_{i=1}^{T} \nabla \log \pi_\theta(a_t|s_t)$$

$$\implies \nabla J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla \left(\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t)\right) r(\tau)\right]$$

The gradient comes out to be independent of the environment dynamics and the ergodic distribution. This means we can now just run Monte-Carlo simulations and approximate the gradient to find the best parameters $\theta^\star$ and the computed gradient is an unbiased estimator of the true gradient.

It should be further observed that the $r(\tau)$ term in the gradient remains effectively uninfluenced during the gradient operation. Replacing $r(\tau)$ by G_t (discounted returns from Definition 2) gives us the REINFORCE Algorithm [Williams, 1992]. This replacement is possible because rewards from the past cannot influence the rewards in the future.

3.3 Actor-Critic Methods

The objective used in Policy Gradient Methods leads to very high variance models and part of the problem is aggravated by the scale of rewards. If observed closely, the policy gradient is equivalent to a Maximum Likelihood Estimation (MLE). Data overwhelms the prior. Any erratic trajectories which produce unusual rewards would cause an unexpected change in the resulting distribution. To mitigate this problem, an idea that helps reduce the variance is to instead maximize an objective which keeps track of the relative reward difference. This leads to an algorithm called REINFORCE with a Baseline by introducing a term in the gradient which does not induce additional bias. Hence, the gradient becomes

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla \left(\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) \right) (G_t - b) \right] \qquad (19)$$

where b is the introduced baseline.

$$\mathbb{E}_{\pi_\theta} \left[\nabla \left(\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) \right) b \right] = \int \sum_{t=1}^{T} \pi_\theta(a_t|s_t) \nabla \log \pi_\theta(a_t|s_t) b \, d\tau \qquad (20)$$

$$= \int \nabla \sum_{t=1}^{T} \pi_\theta(a_t|s_t) b \, d\tau$$

$$= \int \nabla \pi_\theta(\tau) b \, d\tau$$

$$= b \nabla \int \pi_\theta(\tau) d\tau$$

$$= b \nabla 1 = 0$$

The above calculations show that the addition of a baseline keeps the gradient estimate unbiased while decreasing the variance. Often, choosing the right baseline is a challenge in itself and modern methods resolve to using another parametric value function $V^\omega(s)$ as the baseline which is commonly known as the "critic". A problem with the objective dicussed in 15 is that it might not be fully differentiable and hence we introduce a differentiable surrogate in the form of $Q(s, a)$, the action-value function. An extended treatment of Policy Gradient methods can be found in [Sutton and Barto, 1998]. Recent methods like A3C [Mnih et al., 2016] and PPO family of algorithms [Schulman et al., 2017] have been proposed to improve the stability of learned policies. The difference value of the the objective and the baseline is also known as the *advantage estimate*.

3.4 Deterministic Policy Gradients

Instead of representing policies by a parametric probabilistic distribution $\pi_\theta(a|s)$ that stochastically selects actions a, this approach considers determinstic policies of the form $a = \mu_\theta(s)$. It turns out that the Determinstic Policy Gradient is a limiting case of Stochastic Policy Gradients when the variance approaches zero [Silver et al., 2014]. This approach has been shown to outperform Stochastic Policy Gradients discussed earlier in high dimensional action spaces. Following a similar approach as for Policy Gradients, the gradient expression is

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}\left[\nabla_\theta \mu_\theta(s)\nabla_a Q^\mu(s,a)\Big|_{a=\mu_\theta(s)}\right] \tag{21}$$

This result is surprising as well because this also does not depend on the environment dynamics. The policy is decided by solving the following maximization problem.

$$\mu^{k+1}(s) = \underset{a}{argmax}\ Q^{\mu^k}(s,a) \tag{22}$$

However, this is computationally hard to do at each decision step. Instead, multiple gradients are averaged from multiple trajectories to reduce variance in the estimate. This work has since been extended by [Lillicrap et al., 2015] to be more stable by using the idea of Experience Replay Buffer and Target Network updates to keep the target network constant for few gradient steps.

4   Challenges in Multi-Agent Environments

Section 2 and 3 provide a concise overview of modern techniques. However, these have primarily been developed for the Single-Agent case. Before we see extensions of these techniques to a Multi-Agent environment, it is imporant to understand the various ways in which Multi-Agent Environments fall beyond the direct scope of the control algorithms discussed above. Here is a non-exhaustive list of challenges.

4.1   Joint Action Space

In its most general form, the MDPs can be extended as a framework for Multi-Agent systems as a Markov Game [Littman, 1994]. The state transitions are controlled by the current state and one action from each agent. For an environment with n agents

$$\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times \ldots \times \mathcal{A}_n \rightarrow PD(\mathcal{S}) \tag{23}$$

$$R_i : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times \ldots \times \mathcal{A}_n \rightarrow \mathcal{R} \tag{24}$$

$$\pi_{\theta_i} : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times \ldots \times \mathcal{A}_n \rightarrow [0,1] \tag{25}$$

14

where $PD(S)$ represents the probability distribution over the resultant state space. As is apparent, all the routines above are now exponentially dependent on the *action space*. Any decision made by the policy becomes immediately affected by joint actions taken by all the actors and considering all of them becomes imperative to maximize the agent reward R_i. This is now *computationally hard*. A concerning result by [Lowe et al., 2017] shows that for a simple setting of binary actions, the probability of taking a gradient step in the correct direction decreases exponentially with the number of agents. Formally

$$Pr\left[\langle \hat{\nabla} J, \nabla J \rangle > 0\right] \propto 0.5^N \tag{26}$$

where the agent's policy is initialzed to an uninformed policy s.t. $\pi(a = 1|s) = 0.5$, N is the number of agents and $\hat{\nabla} J$ is the gradient estimate from a single sample.

4.2 Game-Theoretic Effects

As we've noted earlier, every MDP has at least one optimal policy and of the given optimal policies at least one is stationary and deterministic. However, for many Markov games, as defined in Section 4.1, there is no deterministic optimal policy that is *undominated* because it critically depends on the the behavior of the opponent. The need for stochasticity arises from the agent's uncertainty in its opponent's moves. An illustration for the point can be read in Box 4.2.

| | | Agent | | |
		rock	paper	scissors
	rock	0	1	-1
Opponent	paper	-1	0	1
	scissors	1	-1	0

Figure 3: The matrix-game for "Rock, Paper, Scissors" [Littman, 1994]

While Single-Agent systems have a relatively strong theoretical foundation, a thorough understanding of the learning problem in *multi-agent* settings is still an open problem. The transient nature of dynamics in such a setting makes the problem harder to analyze. [Shoham et al., 2007] strongly cautions to rely not too strongly on requirements such as convergence to a *Nash Equilibrium* when evaluating learning algorithms in a multi-agent setting. Instead, *Evolutionary Game Theory* is emerging as the preferred framework rather than classical game theory and is surveyed in detail by [Bloembergen et al., 2015].

> **Box 4.2: Rock, Paper, Scissors**
>
> A mathematically convenient way to illustrate stochasticity in Multi-Agent systems is to consider a zero-sum Markov game called "Rock, Paper, Scissors" with only one state, also called a Matrix game. The objective is to maximize the expected reward. $R_{o,a}$ represents the reward when agents takes an action a and the opponent takes the action o. The game is shown in Figure 3.
>
> The linear constraints on the problem for expected rewards for a policy π and total value pay-off V
>
> $$\pi_{\text{paper}} - \pi_{\text{scissors}} \geq V \tag{27}$$
> $$-\pi_{\text{rock}} + \pi_{\text{scissors}} \geq V \tag{28}$$
> $$\pi_{\text{rock}} - \pi_{\text{paper}} \geq V \tag{29}$$
> $$\pi_{\text{rock}} + \pi_{\text{paper}} + \pi_{\text{scissors}} = 1 \tag{30}$$
>
> Linear Programming gives a solution for this as $\pi = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ and $V = 0$. Hence, the optimal policy for the game is using the random strategy at best! This can be concisely written in the *maximin* formulation as
>
> $$V = \max_{\pi} \min_{o} \sum_{a} R_{o,a} \pi_a \tag{31}$$

4.3 Credit Assignment and Lazy Agent Problem

The *Credit Assignment Problem* concerns with how the success of an overall system can be attributed to the various contributions of a systems components [Minsky, 1961]. In a Reinforcement Learning setup, it is already hard to attribute an outcome to a particular action in history. With the extension to Multi-Agent settings, this problem increases in complexity mutlifold. It is extremely doubtful that the signal presented by an outcome (e.g. a win or a loss) contains enough information to make this inference. The naive approach of equally dividing the outcome reward to each of the agents seldom makes sense. Interested audience can find a detailed discussion in [Sutton, 1984] and behavioral analysis with different reward functions in [Balch, 1997].

Another phenomena which arises due to partial observability is called the "Lazy-Agent Problem" [Sunehag et al., 2017] which particularly can occur in Cooperative environments. Learning can fail when one of the agent becomes inactive

because when one agent learns a useful policy, the second agent can be discouraged from exploration so as to not affect the first agent's performance.

4.4 Non-Markovian Nature of Environments

The Markov assumption is crucial in the current formulation of RL algorithms. It provides a mathematically clean framework to approach the learning problem. However, this assumption can be easily violated in the simplest of scenarios. Consider the simple case of an agent learning to find the shortest path from *A* to *B*. In the unconstrained form, this problem is *Markovian* however, with a simple addition of the constraint that no intermediate states can be revisited makes it non-*Markovian*.

This problem was addressed using Recurrent Networks in [Schmidhuber, 1991] to allow the state representation over sequences of state history. Modern treatment of this problem has followed the same approach to build a hidden representation of state sequence with Gated Neural Networks or Convolutional Neural Networks.

5 Decentralized Actor, Centralized Critic

In this section, we will take a look at recent approaches to solve the learning problem in Multi Agent settings. We will specifically focus on an appealing paradigm of training Reinforcement Learning Systems for Multiple Agents known as *Decentralized Actor, Centralized Critic*. The core idea behind this paradigm is summarized in Figure 4.

As seen in Section 4, the action space in a Multi-Agent system grows exponentially with the number of agents. In many cases, learning becomes impossible because of partial observability and communication constraints. This necessitates the need of a *decentralized policies* which only depend on local observations of the agents. Such a formulation naturally attenuates the problem of exponentially growing joint action spaces.

These techniques are augmented in a laboratory setting via a centralized critic which provides an indirect observation (possibly partial) of the complete global state to each of the actors. This helps work around the constraint of inter-agent communications. The primary theoretical foundation driving work in this region is by an extension of MDPs known as the Decentralized Partially Observable Markov Decison Processes (Dec-POMDPs). This framework uses a generalized notion of states in the form of *observations*. The core idea is that an agent may

Figure 4: Overview of Multi-Agent Decentralized Actor, Centralized Critic approach [Lowe et al., 2017]

only be able to get information over a restricted *horizon*. A formal definition can be found in [Oliehoek and Amato, 2016] with a much detailed discussion.

[Kraemer and Banerjee, 2016] present an approach where the agents are allowed to rehearse with information that will not be available during policy execution and also present weak convergence guarantees. Following suit, end-to-end Deep architectures have been proposed recently.

One recent approach introduces a clever approach to estimate the *advantage estimate* by using a counterfactual baseline for policy gradients [Foerster et al., 2017] as

$$A^a(s, \mathbf{u}) = Q(s, \mathbf{u}) - \sum_{u'^a} \pi^a(u'^a | \tau^a) Q(s, (\mathbf{u}^{-a}, u'^a)) \tag{32}$$

where the advantage estimate is computed for each agent and the baseline marginalizes out the actions (u) of an agent a. This allows the centralized critic to reason about the counterfactuals in which only a's actions change. The interactions between the environment and the actors are shown in Figure 5. h represents the hidden state of the actors which are existent for the Gated Neural Networks to account for the Non-Markovian nature of the environments. The training of this network happens via the standard Actor-Critic approach as seen in Section 3.3.

Another approach applies the same paradigm to Q-Learning by proposing a new objective for the supervised loss. The key idea is named QMIX [Rashid et al., 2018] which exploits a linear decomposition of the joint value function across
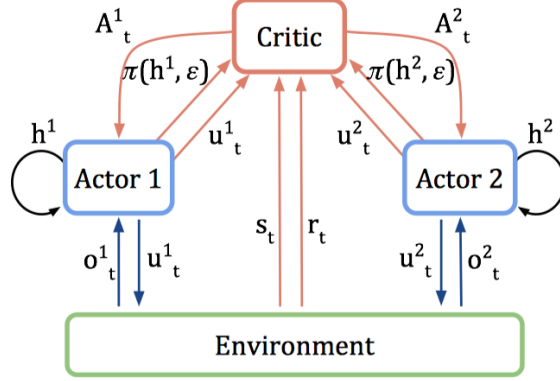
18

Figure 5: Information Flow in COMA [Foerster et al., 2017]

agents by maintaining monotonicity in the local and global maximum value functions. The information flow is depicted via Figure 6.

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0 \text{ (for all agent's Q-Networks)} \tag{33}$$

$$\mathcal{L}(\theta) = \sum_i \left[y_i^{tot} - Q_{tot}(\tau, \mathbf{u}, s; \theta) \right] \tag{34}$$

$$y_i^{tot} = r + \gamma \max_{\mathbf{u'}} Q_{tot}(\tau', \mathbf{u'}, s'; \theta^-) \tag{35}$$

The "Mixing Network" component of the QMIX architecture in Figure 6 is the one that enforces the monotonicity by taking absolute values of the weights generate by an auxiliary hyper-network. The end-to-end training of this network happens via the standard (Deep) Q-Learning framework as seen in Section 3.1.

By the transient nature of the environment which is critically dependent on the behavior policies of other interacting agents in the environment, the training becomes highly unstable. [Lowe et al., 2017] propose to use an ensemble of policies chosen from a pool at random at the start of each trajectory so that the agents are robust to changes in the environment and uses Deterministic Policy Gradients as the choice of training algorithm as discussed in Section 3.4.

5.1 Experiments with Pommerman

A large part of my work currently is about adapting the knowledge above to the novel environment of Pommerman [pom,]. The particular variant of the game I
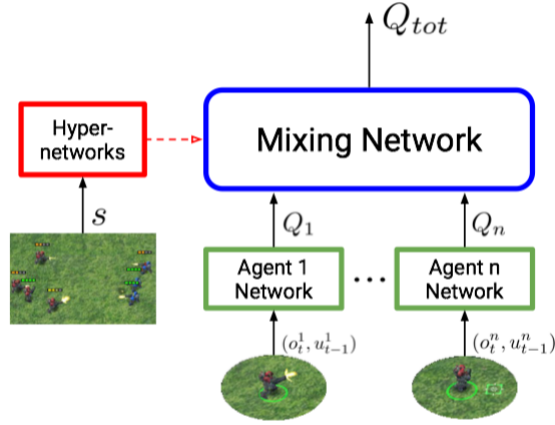
Figure 6: Information Flow in QMIX [Rashid et al., 2018]

am interested in solving is the 2v2 variant where two autonomous agents team up against another team of 2. The paradigm of Multi-Agent training seen above suits exceptionally well to this environment.

My implementation is inspired by the QMIX architecture seen in Figure 6 where the agents are being trained against a team of rule-based agents. An extensive behavorial analysis of the algorithm and tests for robustness are still in progress. It is also to be seen whether these agents can be paired up with novel teammates.

6   Future Work

A large set of problems still stay open in both the theoretical and applied aspects of Reinforcement Learning Systems for Multi-Agent Systems. My immediate next steps to empirically improve the performance described above will be to incorporate the idea of "competitive self-play" [Bansal et al., 2017] and "kickstarting" agents from a pool of pre-trained agents [Schmitt et al., 2018]. I expect these approaches to perform better than the current approach of training agents from scratch. I am also optimistic about *Dec-POMDP*s to be the foundational theoretical framework behind Multi-Agent Reinforcement Learning and will be exploring this topic further.

Acknowlegements

I would like to thank Joan Bruna, Roberta Raileanu and Cinjon Resnick for insightful discussions and directions along the way.

References

[dee,] Deepmind ai reduces google data centre cooling bill by 40 `https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/`.

[pom,] Pommerman: AI research into multi-agent learning. `https://www.pommerman.com/`.

[Agogino and Tumer, 2012] Agogino, A. K. and Tumer, K. (2012). A multiagent approach to managing air traffic flow. *Autonomous Agents and Multi-Agent Systems*, 24(1):1–25.

[Balch, 1997] Balch, T. (1997). Learning roles: Behavioral diversity in robot teams. pages 7–12. AAAI.

[Bansal et al., 2017] Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., and Mordatch, I. (2017). Emergent Complexity via Multi-Agent Competition. *ArXiv e-prints*.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.

[Bloembergen et al., 2015] Bloembergen, D., Tuyls, K., Hennes, D., and Kaisers, M. (2015). Evolutionary dynamics of multi-agent learning: A survey. *J. Artif. Int. Res.*, 53(1):659–697.

[Cesa-Bianchi et al., 2017] Cesa-Bianchi, N., Gentile, C., Lugosi, G., and Neu, G. (2017). Boltzmann Exploration Done Right. *ArXiv e-prints*.

[DeJong and Jong, 2002] DeJong, K. A. and Jong, K. A. D. (2002). *Evolutionary Computation*. The MIT Press.

[Dimitri, 2017] Dimitri, P. B. (2017). *DYNAMIC PROGRAMMING AND OPTIMAL CONTROL*. Athena Scientific.

[Foerster et al., 2017] Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2017). Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926.*

[Kraemer and Banerjee, 2016] Kraemer, L. and Banerjee, B. (2016). Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94.

[Leibo et al., 2017] Leibo, J. Z., Zambaldi, V., Lanctot, M., Marecki, J., and Graepel, T. (2017). Multi-agent Reinforcement Learning in Sequential Social Dilemmas. *ArXiv e-prints*.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *ArXiv e-prints*.

[Littman, 1994] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, pages 157–163, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Lowe et al., 2017] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *ArXiv e-prints*.

[Matignon et al., 2012] Matignon, L., Jeanpierre, L., Mouaddib, A.-I., et al. (2012). Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes.

[Minsky, 1961] Minsky, M. (1961). Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA. PMLR.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

[Ng et al., 1999] Ng, A. Y., Harada, D., and Russell, S. J. (1999). 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. Morgan Kaufmann Publishers Inc.

[Oliehoek and Amato, 2016] Oliehoek, F. A. and Amato, C. (2016). *A Concise Introduction to Decentralized POMDPs*. Springer Publishing Company, Incorporated, 1st edition.

[Pipattanasomporn et al., 2009] Pipattanasomporn, M., Feroze, H., and Rahman, S. (2009). Multi-agent systems in a distributed smart grid: Design and implementation. In *2009 IEEE/PES Power Systems Conference and Exposition*, pages 1–8.

[Rashid et al., 2018] Rashid, T., Samvelyan, M., Schroeder de Witt, C., Farquhar, G., Foerster, J., and Whiteson, S. (2018). QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. *ArXiv e-prints*.

[Salimans et al., 2017] Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *ArXiv e-prints*.

[Schmidhuber, 1991] Schmidhuber, J. (1991). Reinforcement learning in markovian and non-markovian environments. In *Advances in neural information processing systems*, pages 500–506.

[Schmitt et al., 2018] Schmitt, S., Hudson, J. J., Zidek, A., Osindero, S., Doersch, C., Czarnecki, W. M., Leibo, J. Z., Kuttler, H., Zisserman, A., Simonyan, K., and Eslami, S. M. A. (2018). Kickstarting Deep Reinforcement Learning. *ArXiv e-prints*.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *ArXiv e-prints*.

[Shoham et al., 2007] Shoham, Y., Powers, R., and Grenager, T. (2007). If multi-agent learning is the answer, what is the question? *Artif. Intell.*, 171(7):365–377.

[Silver et al., 2014] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China. PMLR.

[Sunehag et al., 2017] Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. (2017). Value-Decomposition Networks For Cooperative Multi-Agent Learning. *ArXiv e-prints*.

[Sutton, 1984] Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis. AAI8410337.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

[Vinyals et al., 2017] Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., et al. (2017). Starcraft ii: a new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*.

[Watkins and Dayan, 1992] Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer.