Multi Agent System approach. A road to robot cooperation

February 2020

1 Introduction

2 Multi Agent Systems:an overview

3 Reinforcement Learning aproach for coordination

3.1 Theoretical formulation

Inside the machine learning methods we can find 3 different big types:

- Supervised Learning. Inferring a regression or classification through a labeled trainging data
- Unsupervised Learning. Inferring from datasets without labels.
- Reinforcement Learning. Study how an agent must behave in order to maximize the cumulative reward.

Thiss last type is what we will exploitf. It is the best scenario when we have an idea of which actions are "good" and "bad" and we want to maximize the goodness of our agent. Let's go through the theory and algorithms used in this paradigm.
In our framework we will have an agent that does the decision making and everything out of the agent is the environmnent. The environment is responsible to offer a response for each action the agent takes, providing a new environment state and the reward for the action the agent took (See figure 1). They interact continuouslly.
Lets state this in a more formal way. We have a set of states $\mathcal{S}$. Initially our agent starts at a determined state $s \subseteq \mathcal{S}$. For the sake of simplicity lets denote S_t the state the Agent is at the t timestep. For each state S_t we have a set of actions the agent can take $A(S_t)$. Once the agent has chosen an action the environment returns a new state S_{t+1} and a reward R_{t+1}.

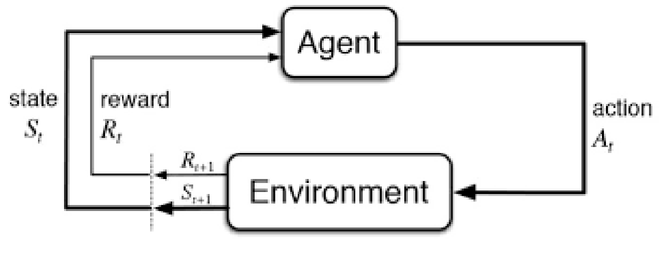

Figure 1: Agent-Environment diagram

The agent learns a policy $\pi(a|s)$ that at every state assigns a probability of each action available in this state of being chosen. This policy keeps changing over experience and our goal is to build a policy that maximizes the cumulated reward from the initial to the terminal state.
How can we maximize the total reward?
Lets denote $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots$ If our interactions go over infinity we will use the discounted return in order to make them finite.
$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + R_T = \sum_{i=0}^{\infty} \gamma^i R_{t+1+i}$ choosing a $\gamma < 1$. If the rewards are bounded we can easily see this series is convergent. We chose $\gamma < \delta < 1$ and we see that

$$\lim_{i\to\infty} \frac{\delta^i}{\gamma^i * R_{t+1+i}} = \lim_{i\to\infty} (\frac{\delta}{\gamma})^i * \frac{1}{R_{t+1+i}} = \infty$$

since it is the exponential of a number bigger than 1 multiplied by a bounded series. That means that from a point of the series the second series is bigger than the first, since the second series is the geometric series of δ and we know it is convergent, so it is the first one.
The parameter γ can also be used in non infinite tasks. In a sense it is a way of weighting the rewards. If γ is close to 1 we want each reward to count the same. If γ is close to 0 we want the first rewards to count more than the next ones in the task.

3.2 Markov Property

The election of our states will be very important. We want to choose as states some variables that give us enough information in order to sintesize all the information we need for our problem. For example, in a chess game, a good state variable is the position of all the pieces(the board itself) since it ofers all the information we need in order to know what action to choose next. It must be clear that there is some information lost in this representation. We don't know how the pieces arranged themselves to the positions that are in this moment.

A state signal that suceeds at storing all relevant information is said to have the Markov property.
We now define the Markov property in the Reinforcement Learning paradigm. We say that a state variable has the Markov propery if:

$$Pr(R_{t+1}, S_{t+1}|S_0, A_0, \dots, S_t, A_t) = Pr(R_{t+1}, S_{t+1}|S_t, A_t)$$

that is, the probability of going to the next state and geting a determined reward depends only in the previous state and action. That means that our state signal is really good since it is possible to sintesize all the past information into a state. This will simplify a lot lots of calculations.
A problem with the Markov property is called a Markov Decision Problem(MDP). And this are the types of problems I will be working on. If we know

$$p(s', r|s, a)$$

we have the problem completely specified. In this scenario we introduce value functions.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$$

The value function of a state given a policy π is the Expectation of the cumulated reward starting from this state following policy π. We also define:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A = a] = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A = a]$$

which is the Expectation starting from state s and taking firstly action a. At this point we can proceed in two ways for trying to estimate $v_\pi(s)$ and $q_\pi(s, a)$. First of all we can do it over experience. We can keep track of the results and then doing an average. We know, by the law of large numbers that it would converge to the true value. Otherwise, we can try to exploit the recursive nature of the equation:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s'}^{r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{1}$$

We get one equation of as many unkowns as states there are. We can do a similar reasoning for each state and get n equations and n unkowns. We can solve this system of equations using iterative methods since the number of states can be extremelly large in most problems. Solving, for each policy we get the value function of each state.
How can we keep improving our policy to make it the optimal?
We want to obtain the value functions of the optimal policy. But first of all we need a method for improving a policy. By improving a policy we try to find a different policy that yields a highest value functions for every state. That is the optimal policy π^* is the one that $v_{\pi^*}(s) \geq v_\pi(s) \forall s \forall \pi$.
For improving an existing policy we can choose the action that yields highest reward at each step. That is given a policy π, we chose our improved policy π' to be at a given state s, $\pi'(s) =$argmax $q_\pi(s,a)$.
That is, we choose the action with highest expected value for the given policy. We keep iterating for each state until the new policy becomes stable.(Insert image). We keep doing that until we arrive at the optimal policy. Now that we have a method for finding the optimal policy and the value functions at the optimal policy, we would think that we can solve every problem that we ran into. The problem with that though is that in lots of problems it becomes impractical or impossible to know beforehand all the transition values $p(s', r|s, a)$.
That makes this approach sometimes useless and we have to think of some variants.

3.3 Temporal Difference Learning

This methods blend between Monte Carlo methods(just averaging the samples) and DP methods(updating the values at each timestep). The idea is fairly simple, at each state we will update the value function of the state s,

$$V(S_t) = V(S_t) + \alpha * (G_t - V(S_t))$$

That is we update the value towards the true value being G_t an unbiased estimator of $V(S_t)$. In our case the update algorithm will be:

$$V(S_t) \leftarrow V(S_t) + \alpha(r + \gamma V(s') - V(s))$$

Being r and $V(s')$ the reward and new state following the policy from state s.This will be one of our common points in all the algorithms we implement.

3.3.1 Sarsa Algorithm

3.3.2 Q Learning Algorithm

3.3.3 Eligibility Traces