AUTONOMOUS AGENTS AND MULTI-AGENT SYSTEMS (AASMA)

2018/2019

LAB 4 – INTRODUCTION TO REINFORCEMENT LEARNING

1. GOALS

1. Introduction to the computational framework of reinforcement learning
2. Application of learning algorithms and exploration strategies in a simplified Taxi domain
3. Analysis of the impact of different parameters and exploration strategies
4. Solving a challenging variant of the Loading Dock scenario

2. INTRODUCTION

Reinforcement learning (RL) is a discipline of machine learning that studies the changes occurring in behavior through trial-and-error interactions with a dynamic environment. One of the things that makes this framework so attractive is the fact that it assumes learning under uncertainty in a dynamic environment. The agent learns "on its own" with no examples of correct behavior being provided.

Formally, RL addresses the general problem of an agent faced with a sequential decision problem [1]. The RL model is depicted in Figure 1. By a process of trial-and-error, the agent's *decision making* component must learn a "good" mapping that assigns perceived environment *states* to *actions*. Such mapping determines how the agent acts and is commonly known as a *policy*. A *critic* situated in the environment provides a feedback signal at each step, know as the *reward*, indicating the adequacy of an action in a given state.

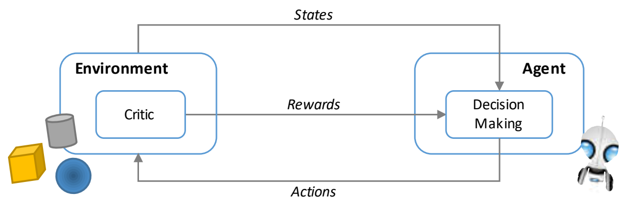

FIGURE 1: THE REINFORCEMENT LEARNING FRAMEWORK

3. LEARNING ALGORITHMS

In RL, the goal of the agent is to choose its actions so as to gather as much reward as possible during its lifespan, according to some cumulative measure of the received reward. The reward function *r(s, a)* implicitly encodes the *task* the agent must complete. Through a process of trial-and-error, the agent must learn an *optimal policy*, denoted as $\pi^*: S \rightarrow A$, usually by approximating an *action-value function*, denoted by $Q^*(s, a)$, for every state-action pair *s, a*. We will consider 2 classic RL algorithms: *Q*-learning and SARSA.

3.1 SARSA

On-policy control method which attempts to evaluate and improve the policy used to make decisions along time. We must estimate *Q(s,a) for the current behavior policy* and for all states and actions. SARSA considers transitions from state-action pairs to state-action pairs to update the action-value function according to:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)],\text{ where } \alpha \text{ is the learning rate.}$$

3.2 *Q*-LEARNING

Q-learning [2] is one of the best well-known algorithms within RL. It is an example of an *off-policy* control method because it directly approximates the optimal action-value function *independently of the agent's actions*. The action selection mechanism determines which state-action pairs are visited and updated but all that is required for correct convergence is that *all* pairs continue to be updated according to:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

4. Action selection strategies

One challenge arising in RL is the trade-off between *exploration* and *exploitation*: an agent must prefer actions tried in the past and found to be effective in producing reward (exploitation); however, to discover such actions, it has to try actions that it has not selected before (exploration). Efficient RL thus requires a balance between exploration and exploitation. In this class we consider two action selection mechanisms.

4.1 ε-greedy

An ε-greedy action selection strategy chooses actions with maximal estimated value – *i.e. greedy* action with probability 1–ε – but with probability ε selects a random available action. The parameter ε can be decreased through time to allow higher exploration rate at the beginning of learning and more exploitation in the end.

4.2 Soft-max

In soft-max, we vary the action probabilities as a graded function of the *Q* function. The greedy action is still given the highest selection probability, but all others are ranked and weighted according to their value estimates. Soft-max typically uses a Boltzmann distribution, choosing an action *a* in state *s* with probability:

$$\frac{e^{Q(s,a)/\tau}}{\sum_{b=1}^{n} e^{Q(s,b)/\tau}}$$

where τ is a positive parameter called the *temperature* which is usually decreased throughout time to cause a greater difference in selection probability in favor of the best actions by the end of learning.

5. Exercises

5.1 Taxi Domain

The provided *NetLogo* file implements a simplified multi-agent taxi domain. The agents are modeled as *taxis* that have to learn to pick-up several *passengers* scattered around the environment while avoiding obstacles. The agents can move in all cardinal directions and only observe their current position (x,y), thus ignoring the presence of other agents. Each agent has to pick-up a different passenger. When two taxis try to pick-up the same passenger, none succeeds. Environment is episodic, resetting the initial state once a task is completed.

- Complete the provided file and implement the SARSA and *Q*-learning algorithms' update rules
- Implement the two different action selection strategies, ε-greedy and soft-max
- Describe one drawback of the ε-greedy action selection mechanism
- In general, which exploration strategy achieves the better performance? Why?
- How does the computational complexity of the problem change with the number of agents?
- What can be done to improve coordination in multiagent scenarios?

5.2 New Loading Dock

Our agents in our LoadingDocks scenario face a new challenge: new boxes appear spontaneously in the ramp leading to a difficulty of placing specific intentions. As such, owners of these agents are eager to try RL. Please check the provided *Java* file that implements a new version of the LoadingDocks.

- Identify some of the problems of current implementation in the agent's learning
- Divise and implement new strategies to improve the behavior of the agents in the LoadingDocks

6. References

[1] Sutton, R. S., and Barto, A. G. *"Reinforcement Learning"*, MIT Press, Cambridge, MA, 1998. Available online at: **https://webdocs.cs.ualberta.ca/~sutton/book/the-book.html**

[2] Watkins, C. J., & Dayan, P., "Q-learning". *Machine learning*, 8(3-4), 279-292, 1992.