Reinforcement Learning Lab

Olivier Pietquin
SARSA and Q-Learning in discrete worlds

1 Introduction

Reinforcement learning is a general problem consisting in learning a control policy for a dynamical stochastic system through interaction. The main principle is to allow a learning agent to interact with the system and to reward it for good local actions. By constraining the agent to accumulate a maximum of rewards over time (and not to optimize the local rewards), algorithms should discover the optimal control which deals with rewards that are delayed in time. We have seen different algorithms of this kind and during this lab, we will implement two of them: SARSA and Q-Learning. We will also address the exploration-exploitation dilemma. This issue arises because during learning, the agent has to constantly choose between trying new actions to see their effects on the environment (*e*xplore) and use what it has learned so far to avoid sub-optimal actions (*e*xploit). We will try two simple exploration strategies: ϵ-greedy and softmax.

For the purpose of focusing on the algorithms, we will use standard environments provided by OpenAI Gym suite. OpenAI Gym provides controllable environments (`https://gym.openai.com/envs/`) for research in reinforcement learning. We will use a simple toy problem to illustrate reinforcement learning algorithms properties. Especially, we will try to solve the FrozenLake-v0 environment (`https://gym.openai.com/envs/FrozenLake-v0/`).

Exercise 1. Hands on OpenAI Gym (`https://gym.openai.com/docs/`)

To get used to the OpenAI Gym suite, we will first try to load an environment and apply random actions to it. To do so, open the `test_envs.py` file (lab materials) and run it to get used to basic functions. Identify the action selection command in the code.

The most important command is the `env.step(action)` one. It applies the selected action to the environment and returns an observation (next state), a reward, a flag that is set to $True$ if the episode has terminated and some info.

Try to use a different policy (for instance, a constant action) to understand the role of that command. Notice that the FrozenLake-v0 environment is non-deterministic and you can't compute the transition probabilities easily. This is why we will use reinforcement learning.

Exercise 2. SARSA

To solve the FrozenLake-v0 problem with reinforcement learning, we will first use the SARSA algorithm. It is based on the online update of the so-called Q-function for the current policy defined as:

$$Q^\pi(s,a) = \mathbb{E}^\pi \left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t) | s = s_0, a = a_0 \right],$$

where $\gamma \in [0,1]$ is the discount factor, and H the horizon of the episode.

The SARSA algorithm updates a tabular estimate of the Q-function using the following update rule:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \left(r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \right),$$

where $\alpha \in (0,1]$ is the learning rate, and r_t is the reward received by the agent at time step t.

Most of the time, the SARSA algorithm is implemented with an ϵ-greedy exploration strategy. This strategy consists in selecting the best action learned so far with probability $(1-\epsilon)$ and to select a random

action with probability ϵ.

To do:

- Define a q_table of the correct size to host the Q-function estimate.
- Implement the SARSA algorithm with ϵ-greedy exploration (start with $\epsilon = .5$).
- As a measure of performance, count the number of successful trials on a sliding window of 100 episodes.

Note: OpenAI considers the task is solved if you reach 76% of success over the last 100 episodes.

Exercise 3. Q-Learning

Another reinforcement learning algorithm is the so called Q-Learning algorithm. The fundamental difference with SARSA is that it is an off-policy algorithm. This means that it doesn't estimate the Q-function of its current policy but it estimates the value of another policy which is the optimal one. To do so, it uses the following update rule:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \left(r_t + \gamma \max_b Q_t(s_{t+1}, b) - Q_t(s_t, a_t) \right).$$

To do:

- In this exercise, you will replace the SARSA update rule by the Q-learning one and analyze the differences in performances.

Exercise 4. Softmax exploration

Now we will replace ϵ-greedy exploration by a softmax one assigning a probability for an action to be performed according to the following rule:

$$P(a_i|s) = \frac{e^{\frac{1}{\tau}Q(s,a_i)}}{\sum_j e^{\frac{1}{\tau}Q(s,a_j)}}.$$

This will allow to assign a selection probability proportionally to the quality of the action according to the current estimate of the Q-funcion.

Exercise 5. Schedule exploration

Exploration should not remain permanently. Indeed, as learning improves, the agent should be more and more confident about its estimate. While softmax explicitly accounts for Q-function values, ϵ-greedy doesn't. Therefore one should decay the value of ϵ with time.

To do:

- Schedule the decay of ϵ with time.
- To assess the quality of scheduling, measure the difference of performance between the greedy policy (learned one) and the behaviour policy (exploration policy).

Exercise 6. Proper evaluation

As the environment and the exploration policies are stochastic the evaluation of the performance we did so far is not ideal to analyze the quality of the algorithms in average. Thus, a proper evaluation would require running the policy many times to estimate the average performance.

To do:

- Implement an inner loop that evaluates the average performance of both the greedy and the behavior policies of SARSA and Q-Learning.
- Conclude on the circumstances under which one algorithm would be better than the other.