Reinforcement Learning Lab 2

Olivier Pietquin - Pierre Richemond

Policy gradient in continuous state spaces

June 29, 2018

Introduction

In a previous lab, we have dealt with reinforcement learning in discrete state and action spaces. To do so, we used methods based on value function and, especially, Q-function estimation. The Q-function was stored in a table and updated with on- or off- policy algorithms (namely SARSA and Q-Learning).

Yet, these methods do not scale to large state spaces and especially not to the case of continuous state spaces. To address these issues one can either extend value-based methods making use of value-function approximation or directly search in policy spaces. In this lab, we will explore the second solution.

More specifically, we will search in a family of parameterized policies $\pi_\theta(s, a)$ using a policy gradient method. This method performs gradient ascent in the policy space so that the total return is maximized. We will restrict our work to episodic tasks, *i.e.* tasks that have a starting states and last for a finite and fixed number of steps H, called horizon.

More formally, we define an optimization criterion that we want to maximize:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{H} r(s_t, a_t)\right],$$

where $\mathbb{E}_{\pi_\theta}$ means $a \sim \pi_\theta(s, .)$ and H is the horizon of the episode. In other words, we want to maximize the value of the starting state: $V^{\pi_\theta}(s)$. The policy gradient theorem tells us that:

$$\nabla_\theta J(\theta) = \nabla_\theta V^{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)\right],$$

With

$$Q^\pi(s, a) = \mathbb{E}^\pi\left[\sum_{t=1}^{H} r(s_t, a_t) | s = s_1, a = a_1\right].$$

Policy Gradient theorem is extremely powerful because it says one doesn't need to know the dynamics of the system to compute the gradient if one can compute the Q-function of the current policy. By applying the policy and observing the one-step transitions is enough. Using a stochastic gradient ascent and replacing $Q^{\pi_\theta}(s_t, a_t)$ by a Monte Carlo estimate $R_t = \sum_{t'=t}^{H} r(s_{t'}, a_{t'})$ over one single trajectory, we end up with a special case of the REINFORCE algorithm (see Algorithm 1).

Notice that, by replacing the Q-function by a Monte-Carlo estimate, we get rid of the Markov assumption and this algorithm is expected to work even in non-Markovian systems.

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. Especially, we will try to solve the CartPole-v0 environment (`https://gym.openai.com/envs/CartPole-v0/`) which offers a continuous state space and discrete action space. The Cart Pole task consists in maintaining a pole in a vertical position by moving a cart on which the pole is attached with a joint. No friction is considered. The task is supposed to be solved if the pole stays up-right (within 15 degrees) for 195 steps in average over 100 episodes while keeping the cart position within reasonable bounds. The state is given by $\{x, \frac{\partial x}{\partial t}, \omega, \frac{\partial \omega}{\partial t}\}$ where x is the position of the cart and ω is the angle between the pole and vertical position. There are only two possible actions: $\{$LEFT,RIGHT$\}$.

Algorithm 1 REINFORCE with PG theorem Algorithm

```
Initialize $\theta^0$ as random
Initialize step-size $\alpha_0$
n = 0
while no convergence do
    Generate rollout $h_n = \{s_1^n, a_1^n, r_1^n, \dots, s_H^n, a_H^n, r_H^n\} \sim \pi_{\theta^n}$
    $PG_\theta = 0$
    for $t = 1$ to $H$ do
        $R_t = \sum_{t'=t}^{H} r_{t'}^n$
        $PG_\theta$ += $\nabla_\theta \log \pi_{\theta^n}(s_t, a_t) R_t$
    end for
    n++
    $\theta^n \leftarrow \theta^{n-1} + \alpha_n PG_\theta$
    update $\alpha_n$ (if step-size scheduling)
end while
return $\theta^n$
```

Exercise 1. Hands on Cart Pole

As for the previous lab, open the `test_envs.py` file (lab materials) and run it to get used to basic functions on Cart Pole. Identify the action selection command in the code. You can notice it's not different from previously. Some more functions are available for multidimensional continuous states.

The most important command is still 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. Although the Cart Pole has easy dynamics that can be computed analytically, we will solve this problem with Policy Gradient based Reinforcement learning.

Exercise 2. Implement a sigmoid policy

As the number of actions is 2, one can see the problem of controlling the Cart Pole as a binary classification problem. Binary classification can be easily solved thanks to logistic regression which transforms the classification problem into a regression problem using the sigmoid function defined by:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

To do:

- Build a policy that select the RIGHT action with probability $\sigma(\theta^\top s)$, where θ is the parameter vector and s is the 4-dimension state vector.
- As a measure of performance, count the average return (sum or rewards) over 100 episodes.

Exercise 3. Compute $\nabla_\theta \log \pi_\theta(s, a)$

Verify that, for a sigmoid policy:

- $\nabla_\theta \log \pi_\theta(s, \text{RIGHT}) = \pi_\theta(s, \text{LEFT}) \times s$
- $\nabla_\theta \log \pi_\theta(s, \text{LEFT}) = -\pi_\theta(s, \text{RIGHT}) \times s$

Exercise 4. Implement REINFORCE

To implement REINFORCE, you will need to follow these steps:

- Run rollouts with current policy $\pi_\theta(s, a)$ with fixed horizon H. Since the goal is to attain an average return of 195, horizon should be larger than 195 steps (say 250 for instance).
- Compute policy gradient $= \sum_{t=1}^{H} \nabla_\theta \log \pi_\theta(s_t, a_{,t}) R_t$.

- Update parameters with gradient ascent.
- Verify if the new policy meets success requirement (average return >194).

Exercise 5. BONUS: Reduce gradient variance

As the policy gradient comes from the maximization of a cumulative function ($\sum_t r_t$), it is possible to subtract a constant (called baseline) to the returns without changing the result of the gradient ascent (gradient estimate would still be unbiased). Yet, this may help reducing the variance of the gradient estimate if the baseline is well chosen. A good candidate is the average return since by subtracting this, we would measure how well the current policy performs in comparison to average instead of pure performance (more noisy).

To do:

- Use Monte Carlo estimate of average return to reduce variance
- Use a parametric Value function $V_\omega(s)$ to estimate average returns and use it as a baseline. Notice that this will bring the Markov property back.

Exercise 6. BONUS: Use different policy parameterizations

We have used a very simple parameterization of the policy taking advantage of the fact that only two actions were available. We should try more generic parameterized policies to address other problems.

To do:

- Implement a softmax policy: $\pi_\theta(s,a) = \frac{e^{\theta^\top \phi(s,a)}}{\sum_b e^{\theta^\top \phi(s,b)}}$.
- Implement a neural network with either sigmoid or softmax on top of the last layer. Then use standard back-propagation to update deeper layers parameters.