Tópicos especiais em Inteligência Computacional II: Aprendizado por Reforço Exercício 2: SARSA

Wouter Caarls
wouter@ele.puc-rio.br

February 20, 2020

The goal of this exercise is to swing up a pendulum, even though the torque the motor can apply to its joint is not enough to do this in one go. This is called the "underactuated pendulum swing-up" problem. By answering the theoretical questions and implementing their solutions, you will construct a temporal difference reinforcement learning solution to this problem using the tabular SARSA algorithm. Send your report in pre-executed ipynb format to `wouter@ele.puc-rio.br`.

The Python code for this exercise (`ex2.py`) contains three main functions, see Table 1.

Table 1: Main source files

`ex2.train`	Main function. Run it to learn your controller.
`ex2.check`	Testing harness. Run it to verify your code.
`ex2.test`	Visualization of the finished controller.

1 Understanding the code

Read `ex2.train`. Note how the function takes an object defining the learning algorithm. This is the class you have to write in this exercise. Compare the steps to figure 6.9 from the text.

S&B: 6.4

Exercise 1.1 How many simulation steps are executed in a trial?

Now use the basic class definition given in `ex2_sol.py` and run `check(SARSA)`; this will report any basic errors in your code.

Exercise 1.2 What does it report? Why is this value not correct? Think about what it means in terms of the learning algorithm.

2 Setting the learning parameters

Look at the `__init__` function and set the random action rate ϵ to 0.05, and the learning rate α to 0.2.

Exercise 2.1 Learning is faster with higher learning rates. Why would we want to keep it low anyway?

Exercise 2.2 The simulation automatically wraps the pendulum angle to the interval $[-\pi, \pi]$. Set the position discretization such that there is at least one state for every $\pi/15$ radians.

Exercise 2.3 Assuming that the velocity stays in the interval $[-12\pi, 12\pi]$ rad$\cdot s^{-1}$, set the velocity discretization such that there is at least one state for every $4\pi/5$ rad$\cdot s^{-1}$.

Set the action discretization to 3 actions.

Exercise 2.4 Set the amount of trials to 1000. Why would we need so many trials? Assuming that state-action pairs are visited uniformly, how many updates per pair can be performed in 1000 trials?

Run `check(SARSA)` to make sure that you didn't make any obvious mistakes.

3 Initialization

The initial values in your Q table can be very important for the exploration behavior, and there are therefore many ways of initializing them. This is done in the `init_Q` function.

S&B: 2.7

Exercise 3.1 Pick a method, and give a short argumentation for your choice.

Exercise 3.2 Implement your choice. The Q table should be of size NxMxO, where N is the number of position states, M is the number of velocity states, and O is the number of actions.

Run `check(SARSA)` to find obvious mistakes.

4 Discretization

Before, you determined the amount of position and velocity states that your Q table can hold, and the amount of actions the agent can choose from. The state discretization is done in the `discretize_state` function.

Exercise 4.1 Use the provided `discretize` function to implement the position discretization. The resulting state must be in the range $[0, \texttt{self.pos\_states} - 1]$. This means that 0 (the "up" direction) will be in the middle of the range.

Exercise 4.2 Implement the velocity discretization. Even though we assume that the values will not exceed the range $[-12\pi, 12\pi]$, they must be clipped to that range to avoid errors. The resulting state must be in the range $[0, \texttt{self.vel\_states} - 1]$. This means that zero motion will be in the middle of the range.

Exercise 4.3 What would happen if we clip the velocity range too soon, say at $[-5\pi, 5\pi]$?

Exercise 4.4 If we would double the resolution of each state dimension in the state-action space, how many elements would Q have? By which factor would doubling the resolution increase the number of elements if there were 10 state dimensions?

Now you need to specify how the actions are turned into voltages, in the `take_action` function.

Exercise 4.5 The allowable voltage is in the range $[-\texttt{self.maxvoltage}, \texttt{self.maxvoltage}]$. Distribute the actions uniformly over this range. This means that zero voltage will be in the middle of the range.

Run `check(SARSA)`, and look at the plots of continuous vs. discretized position. Are they what you would expect?

5 Reward and termination

Now you should determine the reward function, which is implemented in `observe_reward`.

Exercise 5.1 What is the simplest reward function that you can devise, given that we want the system to balance the pendulum at the top?

Exercise 5.2 Implement `observe_reward`.

Exercise 5.3 What additional rewards would you give if the goal is to minimize the required energy?

Run `check(SARSA)`, and verify in the lower left plot that you have indeed implemented the reward function you wanted.

You also need to specify when a trial is finished. While we could learn to continually balance the pendulum, in this exercise we will only learn to swing up into a balanced state. The trial can therefore be ended when that state is reached.

Exercise 5.4 Implement `is_terminal`.

Run `check(SARSA)`, and verify that your termination criterion is correct.

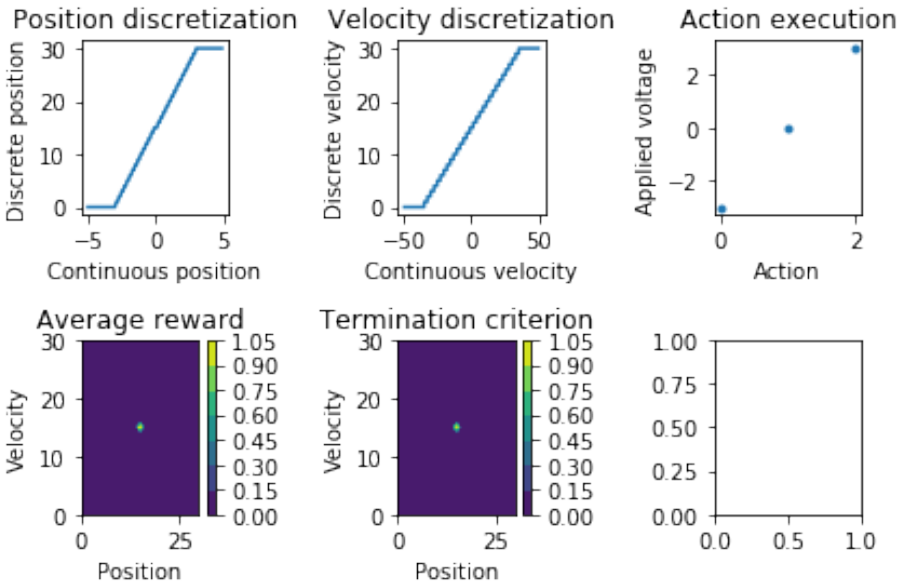

Figure 1: Output of `check(SARSA)` after completing Q1-Q6.

6 The policy and learning update

The next step is to implement the action selection algorithm in `execute_policy`.

Exercise 6.1 Implement ϵ-greedy action selection. Take particular care when there is more than one maximum value, in which case you should select among those randomly. Hint: use the `find` and `randi` functions.

Exercise 6.2 Finally, implement the SARSA update rule in `update_Q`.

Run `check(SARSA)` a final time to check for errors. The result should be similar to Figure 2.

7 Make it work

It is time to see how your learning algorithm behaves! A successful run looks somewhat like Figure 2.

Exercise 7.1 Run `ex2` and check your result against Figure 2.

Exercise 7.2 What happens when you set the discount rate to 0.5? Explain.

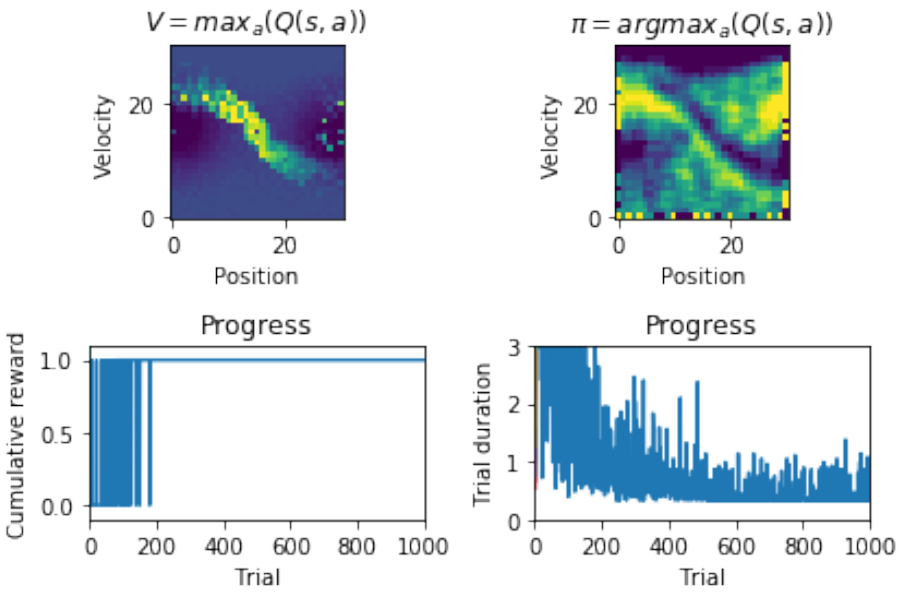

Figure 2: Output of a successful run of `train(sarsa)`.

Exercise 7.3 Set the discount rate to 0.9999. The value function in the upper-left graph diminishes further away along the path from the goal state, even though there is practically no discounting. Give at least one reason for this.

8 Speed it up

While learning in this simple system is quite fast, we would still like to speed it up using eligibility traces. We will use a matrix `e` the same size as `Q` to record eligibilities. This matrix is already initialized to 0 automatically at the start of every episode.

S&B: 7.5

Exercise 8.1 Create a new parameter, `self.lambda` in `__init__` to set the trace decay rate λ. Set it to 0.5.

Exercise 8.2 Implement the SARSA(λ) update rule in `update_Q`. Does it speed up the learning?

Exercise 8.3 Run your experiment again with $\lambda = 0.9$ and explain the result.