Tópicos especiais em Inteligência Computacional II: Aprendizado por Reforço
Exercício 1: Programação Dinâmica

Wouter Caarls
wouter@ele.puc-rio.br

18 de fevereiro de 2020

Reinforcement learning deals with a special type of system in which the way a system's *state* is reached – the history of states and actions that precedes it – is irrelevant, i.e. the state itself summarizes past sensations compactly. These systems are said to have the *Markov property*. In robots, this often means that the state should include the joint positions as well as their velocities. **S&B:** 3.5

In reinforcement learning the system dynamics are unknown, and the problem has to be solved from experience gathered in the real world. This exercise doesn't yet deal with the full reinforcement learning problem, but instead focuses on deciding optimal actions in Markov systems with known dynamics: the problem of solving *Markov Decision Processes* (MDPs). MDPs with unknown dynamics are the subject of the next exercise. **S&B:** 3.6

An important concept in MDPs are *delayed rewards*: an MDP may perhaps only give a reward many state transitions (time steps) after the responsible action was taken. For example, scoring a goal seconds after then ball has been kicked. The agent then has to propagate this information to earlier states and actions. It does this by not only keeping track of the expected immediate reward of an action, but also the cumulative reward of following the same policy afterwards. To avoid infinities, this *return* is discounted with a *discount factor* γ the further it lies in the future.

The return is tracked through the *value function* V, indexed with the state s and policy π, as $V^\pi(s)$. We will now look at a set of methods for calculating and optimizing V, known as *dynamic programming* techniques. **S&B:** 3.7

1 The grid world

Our toy problem for this chapter is a simple grid world. This is a 5x5 grid in which the agent can move according to the four cardinal directions. Movement to a square outside the grid is prohibited, instead resulting in a reward of -1. Positive rewards are awarded for reaching certain points A (+10) and B (+5), after which the agent is placed at specific successor points A' and B'. **S&B:** Example 3.8

The goal is to find the optimal action for each state. An optimal policy π^* will be *greedy* with respect to the optimal value function V^* , i.e. in each state s it will take the action that leads to the state s' with the highest $V^*(s')$. Our first solution method, called *policy iteration*, uses this property to successively approximate V^*. It interleaves *policy evaluation*, which calculates V^π, and *policy improvement*, which improves π by making it greedy with respect to V.

The policy iteration algorithm in the book is given for stochastic transition functions. The grid world uses deterministic transitions, which simplifies it somewhat. Most importantly, line 8 of Figure 4.3 can be rewritten as: **S&B:** Figure 4.3

$$V(s) \leftarrow \mathcal{R}(s, \pi(s)) + \gamma V(\mathcal{T}(s, \pi(s)))$$

where $\mathcal{R} : S \times A \to \mathbb{R}$ is the reward r for taking action $a \in A$ in state $s \in S$, and $\mathcal{T} : S \times A \to S$ is the successor state s'. In the Matlab function `gridworld`, these have been combined in a single function `[`r`, `s'`] = gridworld(`s`, `a`)`. Similarly, line 15 can be rewritten as:

$$\pi(s) \leftarrow \underset{a}{\operatorname{argmax}} \left(\mathcal{R}(s, a) + \gamma V(\mathcal{T}(s, a))\right)$$

Exercise 1.1 Implement policy evaluation for the grid world. Use θ of 10^{-5} as the stopping criterion and $\gamma = 0.9$. **S&B:** 4.1

Exercise 1.2 Implement the policy improvement step, and run the algorithm. Verify that the results are the same as Figure 3.8 in the book (use `plotv`). How many observations did it take to converge? **S&B:** 4.2

The value of γ is a measure of how much into the future the algorithm looks. Lower values require fewer iterations to converge, but make it short-sighted.

Exercise 1.3 Use a value of $\gamma = 0.8$. What changes in the resulting policy? How about the required number of observations? **S&B:** 3.3

Policy iteration has the disadvantage that it uses multiple iterations in the policy evaluation step, slowing down the convergence. It has been proven that is not necessary to wait for full convergence during policy evaluation. Instead, it is possible to alternate a predetermined number of policy evaluation sweeps

with policy improvement steps. The special case of one policy evaluation sweep, called *value iteration* is particularly easy to implement, because the policy evaluation and policy improvement steps can be coalesced into a single update.

Exercise 1.4 Implement value iteration. Since `plotv` already computes π from V, you can skip the last two lines of Figure 4.5 in the book. Verify that the results are the same as using policy iteration. How many observations did it take this time? **S&B:** 4.4

It is not necessary to sweep through the state space deterministically. This will help us later on, because in many real-world cases it is not even possible to do this (imagine successively putting your robot in all possible configurations, including joint speeds!). While real RL uses actual real-world trajectories, for now we will do this asynchronous updating randomly.

Exercise 1.5 Implement asynchronous value iteration by modifying your value iteration loop to have a fixed number of 250 iterations, and choosing s randomly for each iteration (use `randi`). Verify that only 1000 observations are made and compare the resulting V with the V^* you found in the earlier exercises. Also compare the policies, and explain the results. **S&B:** 4.5

Send your report in pre-executed ipynb format to `wouter@ele.puc-rio.br`.