University of Rome "La Sapienza"
Department of Ingegneria Informatica, Automatica e Gestionale

Reinforcement Learning

Assignment 1

Value Iteration, Policy Iteration & iLQR

Gianmarco Scarano

Matricola:
2047315

November 8, 2023

Contents

Chapter 1

Theory

1.1 Iterations for Value Iteration

If we want to get the minimum number of iterations of Value Iteration in order to have an ϵ-error on the quality of the policy, we can simply reason about this formula here:

$$\frac{2\gamma i}{(1-\gamma)} \cdot \|Q_0 - Q^*\| \leq \epsilon$$

We know from theory that Q^* and Q_0 are in range $[0, \frac{1}{1-\gamma}]$ since under the assumption that $R(s, a) \in [0, 1]$, the maximum possible value of Q evolves into a geometric series.

Due to the fact that the difference between $Q_0 - Q^*$ is exactly $\frac{1}{1-\gamma}$ since we are doing an infinity norm, we can just leave Q^*.

$$\frac{2\gamma i}{(1-\gamma)} \cdot \|Q^*\| \leq \epsilon$$

Now, as in Analysis demonstrations, one additionally adds and subtract 1 (basically doing nothing to the formula), but this allows us to rewrite our formula in the following form:

$$\frac{2(1-(1-\gamma))^i}{1-\gamma} \cdot \|Q^*\| \leq \epsilon$$

We can get rid now of Q^*, since we know that it is equal to $\frac{1}{1-\gamma}$, so we'll get:

$$\frac{2(1-(1-\gamma))^i}{(1-\gamma)^2} \leq \epsilon$$

Always from Analysis theory, one knows that the following inequality holds: $1+x \leq e^x, \forall x \in \mathbb{R}$, leading us to the next step of our proof:

$$\frac{2 \cdot e^{-(1-\gamma) \cdot i}}{(1-\gamma)^2} \leq \epsilon$$

Now we simply divide by 2 and multiply by $(1-\gamma)^2$, starting isolating the i term as follows:

$$e^{-(1-\gamma)\cdot i} \leq \frac{\epsilon\cdot(1-\gamma)^2}{2}$$

Exploiting logarithm properties, this now becomes:

$$-i\cdot(1-\gamma) \leq -log(\frac{2}{\epsilon\cdot(1-\gamma)^2})$$

Now, we just divide by $-\frac{1}{(1-\gamma)}$, thus changing the inequality sign and getting that the final result is:

$$i \geq \frac{log(\frac{2}{\epsilon\cdot(1-\gamma)^2})}{1-\gamma}$$

1.2 Value Iteration Exercise

From theory, one knows that the Value Iteration algorithm is explained as follows (just for V):

Algorithm 1 Value Iteration

1: Initialize $V_0^*(s) = 0 \; \forall s \in S$
2: For i = 1, . . . , H
3: For all states s in S:
4: $V_{i+1}^*(s) = \max_a \sum_{s'} T(s,a,s^{'})[R(s,a,s^{'}) + \gamma V_i^*(s')]$

So, simply apply this definition on our problem.

$V_{k+1}^*(s_6) = [(0.3*(0+0.9*0)) + (0.7*(0+0.9*5))]$
$V_{k+1}^*(s_6) = [(0.3*(0+0)) + (0.7*(0+4.5))]$
$V_{k+1}^*(s_6) = [(0.3*0) + (0.7*4.5)]$
$V_{k+1}^*(s_6) = [0+3.15] = 3.15$

Finally, $V_{k+1}^*(s_6) = 3.15$.

Chapter 2

Practice

2.1 Policy Iteration

Really simply, I just edited the `student.py` file in order to fill in the function `reward_function()`, `check_feasibility()` and `transition_probabilities()`, due to the fact that the whole code for Policy Iteration Algorithm was already implemented by TAs.

With this being said, let's analyze the functions:

- ***reward_function()***:

  I checked if the values in state `s` were equal to the final state `[env_size-1,env_size-1]`. If yes, then return reward = 1, else 0.

- ***check_feasibility()***:

  As we did in Value Iteration (during Practicals) we check how feasible is our new state by considering if we have exceeded the boundaries of our World (`if s_prime[0 (for 1 too)] >= env_size` as well as `if (s_prime < 0).any()`). I have not decided to take into consideration obstacles since they are still a feasible path in our grid world.

- ***transition_probabilities()***:

  Also here, as we did in practicals, we have to compute the table for the `prob_next_state` array. First, we check the feasibility of the next state (as well as next state with action ± 1) with the previous explained function, then we assign a certain probability to each state (in this case the exercise provided 1/3 as value). I've put += 1/3 as probability for the next state, since if we are dealing with a boundary state, we should sum up the probability of staying where we are.

2.2 iLQR

In the iLQR problem, I populated the following functions: `backward()`, `forward()` and `pendulum_dyn()`.

- ***backward()***:

  I populated the `kt`, `pt` matrices according to the formulas given in the Assignment presentation PDF. Then, `Kt` and `Pt` have been updated using the standard LQR-LTV pseudo-code from the slides.

- ***forward()***:

  I update the control using the formula given in the Assignment presentation PDF:

  ```
  control = k_seq[t] + K_seq[t] @ (x_seq_hat[t] - x_seq[t]).
  ```

  The only difficult part was understanding x_t^i and x_t^{i-1}, but then I understood that `x_seq` was referring to the previous iteration and `x_seq_hat` to our new actual iteration. The time-step is the same for both arrays.

- ***pendulum_dyn()***:

  I easily plugged in the Assignment presentation PDF formulas into the function, as the Pendulum parameters were already given.

Chapter 3

Collaborations

I discussed this assignment with the following students:

- Giancarlo Tedesco
- Emanuele Rucci
- Mostafa Mozafari