OUSMANE THIONGANE

SAPIENZA
UNIVERSITÀ DI ROMA

Reinforcement Learning

Assignment 2 – Sarsa-λ, Linear Time-Varying LQR and Q-Learning Temporal Differences

Matricula: 2184503

November 8th, 2024

ASSIGNMENT 2 –SARSA-λ, LINEAR TIME-VARYING LQR AND Q-LEARNING TEMPORAL DIFFERENCES

I – Theory

1. Let's consider the following Q-table:

$$Q(s,a) = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} Q(1,1) & Q(1,2) \\ Q(2,1) & Q(2,2) \end{pmatrix}$$

With $\alpha = 0.2, \gamma = 0.7$, and after an experience, we got $(s,a,r,s') = (2,2,5,1)$.

Let's compute the update of both Q-Learning and SARSA (for the latter, we will consider that $a' = \pi_\epsilon(s') = 1$):

As we have seen in class, the Q-Learning algorithm is defined as follows:

Algorithm Q-Learning

Initialize $Q(s,a), \forall s \in S, a \in A(s)$, arbitrarily and $Q(\textit{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize S

Repeat (for each step of episode):

Choose A from S using policy derived from Q (e.g., ϵ-greedy)

Take action A, observe R, S'

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a\big(Q(S',a)\big) - Q(S,A)\right]$$

$S \leftarrow S'$;

until S is terminal

Let's execute the Q-Learning algorithm:

The next state is $s' = 1$. We are taking the maximum of the two actions available in this state, this gives us:

$$\max_a\big(Q(S',a)\big) = \max_a\big(Q(1,1), Q(1,2)\big) = Q(1,2) = 2$$

By replacing the values in the update formula, we finally got:

$$Q(2,2) = Q(2,2) + \alpha[R + \gamma Q(1,2) - Q(2,2)]$$

$$\Rightarrow Q(2,2) = 4 + 0{,}2 \times (5 + 0{,}7 \times 2 - 4)$$

$$\Rightarrow Q(2,2) = 4{,}48$$

Then, the SARSA algorithm is defined as follows:

Algorithm SARSA

Initialize $Q(s,a), \forall s \in S, a \in A(s)$, arbitrarily and $Q(\textit{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize S

Choose A from S using policy derived from Q (e.g., ϵ-greedy)

Repeat (for each step of episode):

Take action A, observe R, S'

Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)

$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$

$S \leftarrow S'; A \leftarrow A';$

until S is terminal

As stated in the assumptions, we are taking $a' = \pi_\epsilon(s') = 1$, this gives:

$$Q(2,2) = Q(2,2) + \alpha[R + \gamma Q(1,1) - Q(2,2)]$$

$$\Rightarrow Q(2,2) = 4 + 0{,}2 \times (5 + 0{,}7 \times 1 - 4)$$

$$\boxed{\Rightarrow Q(2,2) = 4{,}34}$$

Note: In this case, the Q-Learning algorithm produces a higher value than the SARSA.

2. In this exercise, we will the complete Linear Quadratic approximation of a generic non-linear system in a Linear Time-Varying LQR for trajectory following. For this exercise, we got:

- A non-linear transition function $f(x,u)$ such that $x_{t+1} = f(x_t, u_t)$
- A non-quadratic cost function $c(x,u)$, assumed for simplicity to satisfy:
$$c(x,u) = c_x(x) + c_u(u)$$
- A nominal trajectory $\{(x_t^*, u_t^*)\}$, with $t = 0, \dots, H$.

Let's derive the general form of the matrices A, B, Q and R that linearize the system around a generic trajectory point (x_t^*, u_t^*):

By using the first order of Taylor expansion on the transition function we got:

$$x_{t+1} \approx f(x_t^*, u_t^*) + \nabla_x f(x_t^*, u_t^*)(x_t - x_t^*) + \nabla_u f(x_t^*, u_t^*)(u_t - u_t^*)$$

$$\Rightarrow x_{t+1} \approx f(x_t^*, u_t^*) + \nabla_x f(x_t^*, u_t^*)(x_t - x_t^*) + \nabla_u f(x_t^*, u_t^*)(u_t - u_t^*)$$

$$\Rightarrow x_{t+1} - x_{t+1}^* \approx f(x_t^*, u_t^*) - x_{t+1}^* + \nabla_x f(x_t^*, u_t^*)(x_t - x_t^*) + \nabla_u f(x_t^*, u_t^*)(u_t - u_t^*)$$

By using the following variables substitutions,

$$z_t = x_t - x_t^* \text{ and } v_t = u_t - u_t^*$$

We can rewrite the previous equation:

$$z_{t+1} \approx \nabla_x f(x_t^*, u_t^*) z_t + \nabla_u f(x_t^*, u_t^*) v_t + f(x_t^*, u_t^*) - x_{t+1}^*$$

That gives us:

$$z_{t+1} = A z_t + B v_t + \Delta_t$$

With:

$$A = \nabla_x f(x_t^*, u_t^*) \text{ and } B = \nabla_u f(x_t^*, u_t^*)$$

$$\Delta_t = f(x_t^*, u_t^*) - x_{t+1}^*$$

Note: Since we have a nominal trajectory with both states and controls, it might be the case that a certain nominal state x_{t+1}^* does not coincide with the forward dynamics $f(x_t^*, u_t^*)$ of the previous nominal state x^* and nominal control u^*(i.e. the trajectory might be "unfeasible"), that's why we have the additional term Δ_t in the Taylor expansion for the dynamics.

For the cost function, we use the second-order Taylor expansion:

$$\begin{aligned} c(x,u) \approx c(x^*,u^*) &+ \nabla_x c(x^*,u^*)^T (x - x^*) \\ &+ \nabla_u c(x^*,u^*)^T (u - u^*) \\ &+ \frac{1}{2}(x - x^*)^T \nabla_x^2 c(x^*,u^*)(x - x^*) \\ &+ \frac{1}{2}(u - u^*)^T \nabla_u^2 c(x^*,u^*)(u - u^*) \\ &+ (x - x^*)^T \nabla_{x,u}^2 c(x,u)(u - u^*) \end{aligned}$$

Then, by replacing the cost function by its expression $c(x,u) = c_x(x) + c_u(u)$, we can rewrite and eliminate some terms ($e.g. \nabla_{x,u}^2 c(x,u) = 0$):

$$\begin{aligned} c(x,u) \approx c(x^*,u^*) &+ \nabla_x c_x(x^*)^T (x - x^*) \\ &+ \nabla_u c_u(u^*)^T (u - u^*) \\ &+ \frac{1}{2}(x - x^*)^T \nabla_x^2 c_x(x^*)(x - x^*) \\ &+ \frac{1}{2}(u - u^*)^T \nabla_u^2 c_u(u^*)(u - u^*) \end{aligned}$$

Then, by neglecting the lower-order terms, we obtain:

$$c(x,u) \approx c(x^*,u^*) + \frac{1}{2}(x - x^*)^T \nabla_x^2 c_x(x^*)(x - x^*) + \frac{1}{2}(u - u^*)^T \nabla_u^2 c_u(u^*)(u - u^*)$$

And by using the same variables z and v than before, that finally gives us:

$$c(z,v) = c(x,u) - c(x^*,u^*) = z^T Q z + z^T R z$$

With:

$$Q = \frac{1}{2}\nabla_x^2 c_x(x^*) \text{ and } R = \frac{1}{2}\nabla_u^2 c_u(u^*)$$

II – Practice

1. Let's implement the SARSA-λ algorithm on the Taxi environment:

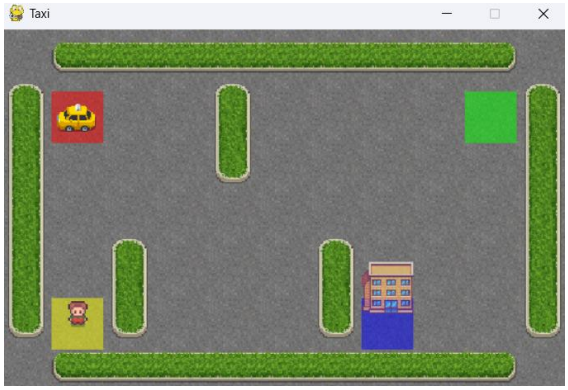

Figure 1 – Taxi environment display

Firstly, we implemented the ϵ-greedy strategy the same way than the previous times:

$$\begin{cases} \text{The agent will randomly choose an action} & \text{with probability } \epsilon \\ \text{The agent will choose the current best action} & \text{with probability } 1\text{-}\epsilon \end{cases}$$

We then complete the SARSA-λ code by following the logic of the algorithm:

Algorithm SARSA-λ

Initialize $Q(s,a)$ arbitrarily, $\forall s \in S, a \in A(s)$

Repeat (for each episode):

$\boldsymbol{E(s,a) = 0, \forall s \in S, a \in A(s)}$

Initialize S, A

Repeat (for each step of episode):

Take action A, observe R, S'

Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)

$\boldsymbol{\delta \leftarrow R + \gamma Q(S',A') - Q(S,A)}$

$\boldsymbol{E(S,A) \leftarrow E(S,A) + 1}$

$\forall s \in S, a \in A(s)$:

$\boldsymbol{Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)}$

$\boldsymbol{E(s,a) \leftarrow \gamma \lambda E(s,a)}$

$S \leftarrow S'; A \leftarrow A';$

until S is terminal

Note: The parts of the algorithm that have been implemented are specified in bold.

```
TRAINING FINISHED
        Total Reward ep 0: 11
        Total Reward ep 1: 5
        Total Reward ep 2: 7
        Total Reward ep 3: 8
        Total Reward ep 4: -1
        Total Reward ep 5: 5
        Total Reward ep 6: 8
        Total Reward ep 7: 7
        Total Reward ep 8: 2
        Total Reward ep 9: 8
Mean reward over 10 episodes: 6
```

Figure 2 – SARSA–λ output

In the end, the agent generally succeeds in accomplishing his task, i.e. taking customers to the hotel.

2. Let's implement the Q-Learning TD(λ) algorithm with linear approximation on the Mountain Car environment. We will use the RBF representation for the states and the actions:

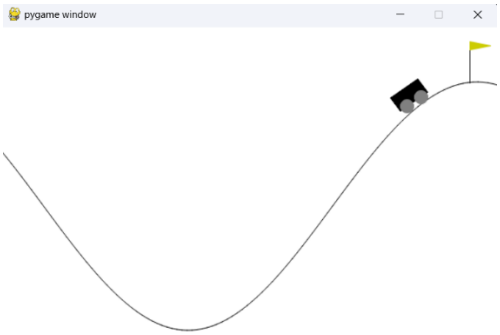

Figure 3 – The Mountain Car environment

`RBFFeatureEncoder:`

The `FeatureUnion()` method from `sklearn.preprocessing` allows us to combine several RBF Samplers in parallel with different γ values to experiment different scenarios.

The `StandardScaler()` function as well from `sklearn.preprocessing` allows us to scale our data before passing it to the transform operator of the encoder, like that, we will pass from a state representation to a feature representation. Since we are performing 10000 Mountain Car experiments (with `variable observation_examples`), and we have 4 RBF Samplers each with a number of components of 500, we deduce the size of the observed features which is $(10000, 4 \times 500) = (100000,\ 2000)$. Here, the number of features is the 2nd component of the vector, so 2000 (we can get it by returning `self.features_obs.shape[1]`).

TDLambda LVFA:

To complete the transition update of the TD(λ), we are using the update of the backward-view incremental prediction algorithm:

$$\delta_t = r_t + \gamma Q(s_{t+1}, ., w_t) - Q(st_t, a_t, w_t)$$

$$e_t = \gamma\lambda e_{t-1} + \nabla_w Q(s_t, a_t, w_t)$$

$$w_{t+1} = w_t - \alpha\delta_t e_t$$

With:

- e_t: Eligibility traces
- w_t: Weight

However, we should be aware that the rewards of the Mountain Car environment are negative, so the new update formula will become:

$$w_{t+1} = w_t + \alpha\delta_t e_t$$

```
ep | eval | epsilon | alpha
0 -200.0 0.2985 0.01
20 -123.7 0.27002622836197326 0.01
40 -121.0 0.24426855612526807 0.01
60 -134.3 0.22096789587246615 0.01
80 -143.3 0.2 0.01
100 -104.1 0.2 0.01
120 -120.4 0.2 0.01
140 -104.2 0.2 0.01
160 -127.3 0.2 0.01
180 -113.1 0.2 0.01
```

Figure 4 – Q-Learning TD(λ) output

We can see that over the 200 episodes, the average reward increases. In simulation, the agent can manage the momentum of the car to reach the flag.

III – Resources

1. The algorithms of the 8th Chapter: “Q-Learning, SARSA”

2. The slides of the 6th Chapter: “LQR, iLQR, MPC”

3. Taylor’s theorem:
https://en.wikipedia.org/wiki/Taylor%27s_theorem

4. Introduction to Reinforcement Learning Fall 2024 – From LQR to Nonlinear Control:
https://lucasjanson.fas.harvard.edu/courses/6.pdf

5. The Gymnasium Taxi environment:
https://gymnasium.farama.org/environments/toy_text/taxi/

6. The repository of the Q-Learning practical (for the epsilon-greedy strategy):
https://github.com/KRLGroup/RL_2024/blob/main/p04_q_learning/Qlearning_sol.ipynb

7. The SARSA-λ algorithm of the 9th Chapter: “n-step bootstraping”

8. The Mountain Car environment:
https://gymnasium.farama.org/environments/classic_control/mountain_car/

9. The RBF Sampler documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.kernel_approximation.RBFSampler.html

10. Guide to Deep Reinforcement Learning: Key Concepts & Use Cases:
https://blog.mlq.ai/guide-to-deep-reinforcement-learning/

11. The slides of the 10th Chapter: “Linear Value Approximation” (Incremental Predictions Algorithms – Backward View)

12. Mountain Car Q-Learning repository:
https://github.com/lazyprogrammer/machine_learning_examples/blob/master/rl2/mountaincar/q_learning.py

13. Q-Learning with Linear Value Function Approximation repository:
https://github.com/dennybritz/reinforcement-learning/blob/master/FA/Q-Learning%20with%20Value%20Function%20Approximation%20Solution.ipynb