Reinforcement Learning Lab

Lesson 5: Dyna-Q

Gabriele Roncolato and Alberto Castellini

University of Verona
email: gabriele.roncolato@univr.it

Academic Year 2024-25

Environment Setup

The first step for the setup of the laboratory environment is to update the repository and load the miniconda environment.

Safe Procedure

Always back up the previous lessons' solutions before executing the repository update.

- Update the repository of the lab:

```
cd RL-Lab
git stash
git pull
git stash pop
```

- Activate the *miniconda* environment:

```
conda activate rl-lab
```

Today Assignment

In today's lesson, we implement the Dyna-Q and Dyna-Q+ algorithms in Python. In particular, the file to complete is:

```
RL-Lab/lessons/lesson_5_code.py
```

Inside the file, two functions are partially implemented. The objective of this lesson is to complete them.

- **def dynaQ()**
- **def dynaQplus()**

Expected results can be found in:

```
RL-Lab/results/lesson_5_results.txt
```

Algorithm: Dyna-Q

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:

(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon$-greedy(S, Q)
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Loop repeat n times:

$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Figure: Pseudocode for Dyna-Q, from the Sutton and Barto book *Reinforcement Learning: An Introduction*

Algorithm: Dyna-Q+

Prioritized sweeping for a deterministic environment

Initialize $Q(s,a)$, $Model(s,a)$, for all s,a, and $PQueue$ to empty

Loop forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow policy(S,Q)$

(c) Take action A; observe resultant reward, R, and state, S'

(d) $Model(S,A) \leftarrow R, S'$

(e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$.

(f) if $P > \theta$, then insert S,A into $PQueue$ with priority P

(g) Loop repeat n times, while $PQueue$ is not empty:

$S,A \leftarrow first(PQueue)$

$R,S' \leftarrow Model(S,A)$

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

Loop for all $\bar{S},\bar{A}$ predicted to lead to S:

$\bar{R} \leftarrow$ predicted reward for $\bar{S},\bar{A},S$

$P \leftarrow |\bar{R} + \gamma \max_a Q(S,a) - Q(\bar{S},\bar{A})|$.

if $P > \theta$ then insert $\bar{S},\bar{A}$ into $PQueue$ with priority P

Figure: Pseudocode for Dyna-Q+, from the Sutton and Barto book *Reinforcement Learning: An Introduction*

Assignment Notes

Today's assignment is based on the *DangerousGridWorld* environment and makes use of the epsilon_greedy() function (provided).

Hint (Code)

The solutions of the previous lessons can be used to complete today's assignment. In particular, the update rule for the Q-table can be adapted from lesson 4 (Temporal difference methods) while the general structure follows lesson 3's solution (MC RL methods).

Results Disclaimer

Given the (high) stochasticity of the method, the obtained results may differ.

Hint (NumPy)

Numpy provides useful functions to simplify the assignment. In particular, numpy.random.choice() and numpy.where() are used in the suggested solution. More details can be found on the official website (here).