RL - Reinforcement Learning

Exercises on Model-based methods

Prof. Riccardo Berta

2025.10.03

1 Frozen Lake

Apply and compare SARSA(lambda), Q(lambda), DynaQ and Trajectory sampling agents on the Frozen Lake (FL) environment using a discount factor gamma=0.99. For the comparison, you can plot only states (0, 6, 10). Get also the optimal state-value functions using Dynamic Programming.

Remember: FL is a simple grid-world environment. It has 16 states and 4 actions. The goal of the agent is to go from a start location to a goal location while avoiding falling into holes. All transitions landing on the goal state provide a +1 reward, while every other transition in the entire grid world provides no reward. The challenge is that the surface of the lake is frozen, and therefore slippery. So actions have stochastic effects, and the agent moves only a third of the time as intended. The other two-thirds is split evenly in orthogonal directions.

FL is a more challenging environment than Slippery Walk environment. Therefore, one of the most important changes you need to make is to increase the number of episodes the agent interacts with the environment. While in the Slippery Walk environment we allow the agent to interact for only 3.000 episodes, in the FL environment, let your agent gather experience for 10.000 episodes.

1 - Import the FrozenLake environment and extract it MPD dynamics.

```
# YOUR CODE HERE

# You can get the environment from Gymnasium 'FrozenLake-v1';
# In order to visually plot the environment you can import it
# using render_mode="rgb_array"
# You can also import the actions LEFT, DOWN, RIGHT, UP from gym
# Finally, you can import the transition probabilities P from gym
```

2- Create a random policy as a baseline

```
# YOUR CODE HERE

# You have to create a function that get in input a state and provide a random action
```

3 - Show the policy in action by rendering the environment several times after different decisions from the random policy:

```
# YOUR CODE HERE

# You have to use a while loop in order to provide the current state to
# the policy and then make a step in the environment using the action
# provided by the policy.
# Try to create a function (to be called "show_policy")
# to be reused with other policies later.
# Hints: env.render() provides you an array representing
# an image of the environment; plt.imshow() can be used to visualize
# the image on the screen.
```

4 - Solve the environment using Dynamic Programming:

5 - Print the values of the optimal state-value function

```
# YOUR CODE HERE

# Copy the print_state_value_function function from the lecture notes and use it to
# print the state-value function for the best policy
```

6 - Implement the SARSA(lambda) algorithm and use it against the FL environment

```
# YOUR CODE HERE

# Copy the decay_schedule function from the lecture notes
```

```
# YOUR CODE HERE

# Copy the decay_discounts function from the lecture notes
```

```
# YOUR CODE HERE

# Copy the select_action function from the lecture notes
```

```
# YOUR CODE HERE
# Copy the sarsa_lambda function from the lecture notes and modify it in order to ask
↪  the environment the number
# of states and actions following the Gymnasium interface (.n), and extract the id of
↪  the starting state from the first
# output of the reset() function.
```

```
# YOUR CODE HERE

# Apply sarsa(lambda) for 10.000 episodes ans gamma = 0.99
```

7 - Plot the state-value functions for states (0, 6, 10) for SARSA(lambda) and compare with the optimal values. Than observe how the agent struggle to estimate the optimal state-value function.

```
# YOUR CODE HERE

# import matplotlib.pyplot as plt and plot the V_track_sarsa_lambda for each episode
↪  for states 0, 6 and 10
# draw a horizontal line for the optimal_V for states 0, 6 and 10
```

8 - Implement the Q(lambda) algorithm and use it against the FL environment

```
# YOUR CODE HERE

# Copy the q_lambda function from the lecture notes and modify it in order to ask the
↪  environment the number
# of states and actions following the Gymnasium interface (.n) and extract the id of
↪  the starting state from the first
# output of the reset() function.
```

```
# YOUR CODE HERE

# Apply q(lambda) for 10.000 episodes ans gamma = 0.99
```

9 - Plot the state-value functions for states (0, 6, 10) for Q(lambda) and compare with the optimal values. Than observe as Q(lambda) agent is moving the estimates of the optimal state-value

function toward the true values, unlike SARSA(lambda). Now, to be clear, this is a matter of the number of steps, also SARSA(lambda) would converge to the true values if given more episodes.

```
# YOUR CODE HERE

# import matplotlib.pyplot as plt and plot the V_track_q_lambda for each episode for
↪  states 0, 6 and 10
# draw a horizontal line for the optimal_V for states 0, 6 and 10
```

10 - Implement the DynaQ algorithm and use it against the FL environment.

```
# YOUR CODE HERE

# Copy the dyna_q function from the lecture notes and modify it in order to ask the
↪  environment the number
# of states and actions following the Gymnasium interface (.n) and extract the id of
↪  the starting state from the first
# output of the reset() function.
```

```
# YOUR CODE HERE

# Apply DynaQ for 10.000 episodes and gamma = 0.99
```

11 - Plot the state-value functions for states (0, 6, 10) for DynaQ and compare with the optimal values.

```
# YOUR CODE HERE

# import matplotlib.pyplot as plt and plot the V_track_dq for each episode for states
↪  0, 6 and 10
# draw a horizontal line for the optimal_V for states 0, 6 and 10
```

12 - Implement the Trajectory Sampling algorithm and use it against the FL environment.

```
# YOUR CODE HERE

# Copy the trajectory_sampling function from the lecture notes and modify it in order
↪  to ask the environment the number
```

```
# of states and actions following the Gymnasium interface (.n) and extract the id of
 ↪  the starting state from the first
# output of the reset() function.
```

```
# YOUR CODE HERE

# Apply trajectory_sampling for 10.000 episodes ans gamma = 0.99
```

13 - Plot the state-value functions for states (0, 6, 10) for Trajectory Sampling and compare with the optimal values.

```
# YOUR CODE HERE

# import matplotlib.pyplot as plt and plot the V_track_ts for each episode for states
 ↪  0, 6 and 10
# draw a horizontal line for the optimal_V for states 0, 6 and 10
```

14 - Plot the state-value function mean absolute error for all the methods.

```
# YOUR CODE HERE

# Copy the moving average function from the lecture notes
```

```
# YOUR CODE HERE

# plot the moving average of the absolute error between the optimal_v
# and the v for each algorithm
```

15 - Show the optimal policy:

```
# YOUR CODE HERE

# You can reuse the "show_policy" function written before,
# in order to show the optimal policy
```