INF8953DE(Fall 2021) : Reinforcement Learning — Given on: Sep 30, 12:00 pm

Assignment #2 — Due on : Oct 26, 10:00 pm

- This is an individual assignment. However, you are allowed to discuss the problems with other students in the class. But you should write your own code and report.
- If you have any discussion with others, you should acknowledge the discussion in your answers by mentioning their name.
- For this assignment, you should submit a report that includes your answers for all the questions including graphs, analysis, explanation, figures on gradescope. This report should not include your code/jupyter notebook.
- Submit the actual Colab/Jupyter notebook (.pynb file) on **Moodle**.
- Be precise with your explanations in the report. Unnecessary verbosity will be penalized.
- You are free to use libraries with general utilities, such as matplotlib, numpy and scipy for python. However, you should implement the algorithms yourself.
- If you have questions regarding the assignment, you can ask for clarifications in Piazza. You should use the corresponding tag for this assignment.
- Instructions on how to submit the report in pdf on the Gradescope can be found here: https://www.youtube.com/watch?v=u-pK4GzpId0

Environment

Consider a variation of the `FrozenLake-v0` environment from the OpenAI gym library. The agent controls the movement of a character in a grid world. Some tiles of the frozen and walkable (`F`), while others are holes in the lake (`H`) that lead to the agent falling into the water. The states are denoted similarly to the GridWorld in assignment 1: $S = \{0, 1, 2, \ldots, 14, 15\}$ for a 4x4 lake. The agent starts on a starting tile (`S`), and its goal is to find the fastest walkable path to a goal tile (`G`).

The agent can move in the four cardinal directions, $A = \{$`left, down, right, up`$\}$, but the floor is slippery! Given a `slip_rate` of $0 \leq \xi < 1$, the agent will go in a random wrong direction with probability ξ.

The reward is -1 on all transitions, except for three cases that all result in the episode terminating: (1) The agent falling into a hole nets the agent a reward of -100, (2) The agent takes over 100 steps, all the ice melts and the agent gets a reward of -50, and (3) The agent reaches the goal state, rewarding it a reward of 0. The discount factor for this environment should be set to $\gamma = 0.99$. The environment is implemented for you, and can be found here https://github.com/micklethepickle/modified-frozen-lake

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

Table 1: Example 4x4 FrozenLake environment

1 Monte Carlo Methods

Consider in this section the 4x4 version of the FrozenLake environment, with a `slip_rate` of 0.1. Again, make sure to use a discount factor of $\gamma = 0.99$ for all your experiments. This environment can be instantiated with `env = FrozenLakeEnv(map_name="4x4", slip_rate=0.1)`.

1. Create a function `generate_episode` which takes as input a policy π ($\mathbb{N} \to \mathbb{R}^{|A|} : S \to p(a|S)$), the environment, and the boolean `render` which renders every step of the episode in text form (rendering the episode is as easy as calling `env.render()`). The output of this function should return the tuple `states`, `actions`, `rewards` containing the states, actions, and rewards of the generated episode following π.

2. Implement the first-visit Monte Carlo (for ϵ-soft policies) control algorithm to find the approximate optimal policy $\pi \approx \pi_*$. Let $\epsilon = 0.05$, run the algorithm for 2000 episodes, and repeat this experiment for 10 different runs. In the report, please include:

   (a) Plot the average undiscounted return across the 10 different runs with respect to the number of episodes (x-axis is the 2000 episodes, y-axis is the return for each episode).

   (b) Visualize an episode, by generating and rendeingr an episode using one of the 10 learned approximate π_* to verify its optimality.

   (c) What is the average undiscounted return for the last 100 episodes of the 10 runs (average over the 100 episodes, over 10 runs)? Why might this be smaller than you'd expect?

3. Consider a random behavior policy $b(a|s) = 0.25$ for every a and s. Use episodes generated with b to evaluate the value function $V^\pi(s)$ of the ϵ-greedy policy found in 2 **Specify which policy, since there are 10**. First use *ordinary importance sampling* Monte Carlo prediction. Then use *weighted importance sampling* Monte Carlo prediction. In both cases, run the algorithms for 2000 episodes across 10 runs. [1] Plot the average off policy Monte Carlo estimates of $V^\pi(s)$ for $s = 0$ against the number of episodes for both *ordinary importance sampling* and *weighted importance sampling*. In theses plots, include the standard deviation as a confidence band. For example, if `experiments` is an array containing `10runs` $\times$ `2000episodes` value estimates, then you may use the following function:

[1] For each individual run, plotting the the Monte Carlo estimates of $V^\pi(s)$ over the number of episodes should give a plot similar to Figure 5.4 in the Sutton Barto 2018 textbook.

```
def plot_many(experiments, label=None, color=None):
    mean_exp = np.mean(experiments, axis=0)
    std_exp = np.std(experiments, axis=0)
    plt.plot(mean_exp, color=color, label=label)
    plt.fill_between(range(len(experiments[0])), mean_exp + std_exp,
        mean_exp - std_exp, color=color, alpha=0.1)
```

Create the same figures for $s = 3, s = 7$, and $s = 14$. In the end, there should be 4 figures, with 2 curves in each. In the report, include and answer:

(a) The 4 figures containing 2 curves each. One for each of the asked states: $s = 0$, $s = 3$, $s = 7$, and $s = 14$.

(b) Which of the two methods do you expect to produce lower variance estimates, and why?

2 Prediction: Unifying Monte Carlo methods and Temporal Difference Learning

Consider in this section an easier version of the 4x4 FrozenLake environment with a `slip_rate` of 0 (no slipping). Use a discount factor of $\gamma = 0.99$. This environment can be instantiated with `env = FrozenLakeEnv(map_name="4x4-easy", slip_rate=0)`. We will be working with the same random policy used above: $\pi(a|s) = 0.25$ for all a and s.

S	F	F	F
F	H	F	F
F	F	F	F
H	F	F	G

Table 2: 4x4-easy FrozenLake environment

1. Implement the *Every visit Monte Carlo prediction* algorithm in order to estimate $V^\pi(s)$. Train the algorithm for 10000 episodes, and plot the learning curves for each s of $V^\pi(s)$ over the number of episodes. The result should be 1 figure, with 16 curves plotted inside it (one for each state, x-axis is the 10000 episodes, y-axis is the current estimate of $V^\pi(s)$). For the report:

   (a) Include the plot containing the learning curves for each s of $V^\pi(s)$

   (b) Visualize the final learned value function for each (put the values in a grid like Table 2). Intuitively, do these values make sense?

2. Implement the `TD(0)` prediction algorithm to estimate $V^\pi(s)$. Use a step size $\alpha = 0.01$. Train the algorithm for 10000 episodes as well, and plot the same figure as in the

previous question ($V^\pi(s)$ for each s over the number of episodes). Include this plot in the report.

3. Now, implement the *n-step* `TD` algorithm to estimate $V^\pi(s)$. Use a step size of $\alpha = 0.01$. This algorithm should take the additional hyper-parameter n to determine how much to bootstrap. Now set $n = 1$, and train the algorithm for 10000 episodes. Plot the the same figure as before ($V^\pi(s)$ for each s over the number of episodes), and compare it to `TD(0)` and *Every visit Monte Carlo Prediction*. For the report:

   (a) Include the plot containing the learning curves for each s of $V^\pi(s)$ for *n-step* `TD` with $n = 1$.

   (b) Compare this plot with the plots in questions 1 and 2 of this section (`TD(0)` and *Every visit MC* respectively). Which do you expect it to look similar to, and why? Do your results reflect that?

4. Using the same implementation of *n-step* `TD`, estimate $V^\pi(s)$ using $n = 100$ instead (still with $\alpha = 0.01$ and 10000 episodes). Again, plot the same figure as before ($V^\pi(s)$ for each s over the number of episodes). For the report:

   (a) Plot the same figure as in the previous three questions for $n = 100$.

   (b) Compare this to *Every visit MC*. Do the performances look similar? Why or why not? (hint: it does not.)

5. The intuition is that *n-step* `TD` should generalize both *Monte Carlo prediction* and `TD(0)`. We saw in the previous question that it does not seem to be equivalent to MC prediction. Modify your *n-step* `TD` algorithm such that when $n = 100$, it becomes equivalent to *Every visit Monte Carlo prediction*. Hint: This has to do with the step size α. In the report:

   (a) Write down the new formula for α such that it is equivalent to *Every visit Monte Carlo prediction*

   (b) Plot $V^\pi(s)$ for each s over the 10000 episodes for this new modified *n-step* `TD` algorithm.

   (c) Does this new plot visually match the performance of *Every Visit MC prediction*? Why or why not (hint: it should)?

3 Temporal Difference Control Methods

In the section consider the same version of the FrozenLake environment as in Question 1. Instantiate the environment with `env = FrozenLakeEnv(map_name="4x4", slip_rate=0.1)`, and use the discount factor $\gamma = 0.99$ in all experiments. In this question you need to implement a training procedure similar to the `generate_episode` function in Q1.1, but instead of running a fixed policy, you need to ensure that the agent is trained (i.e., value estimate is updated) throughout the learning phase.

1. Implement the SARSA control algorithm. Let the policy be an ϵ-greedy policy with $\epsilon = 0.01$. Use the step size $\alpha = 0.1$ to update value estimates. Run the algorithm 10 times with different random seeds, and for each seed run the algorithm for 2000 episodes. Plot the average undiscounted return across the 10 different seeds. Generate and render an episode using one of the 10 learned policies to verify its optimality.

2. Implement the Expected SARSA control algorithm. Let the behaviour policy be an ϵ-greedy policy with $\epsilon = 0.01$. Use the step size $\alpha = 0.2$ to update value estimates. Run the same experiments as in the previous subquestion and create an average undiscounted return vs. episode learning curve. Generate and render an episode using one of the 10 learned policies to verify its optimality.

3. Implement the Q-learning control algorithm. Let the behaviour policy be an ϵ-greedy policy with $\epsilon = 0.02$. Use the step size $\alpha = 0.1$ to update value estimates. Run the same experiments as in the previous subquestion and create an average undiscounted return vs. episode learning curve. Generate and render an episode using one of the 10 learned policies to verify its optimality.

4. Comparing the performance of MC methods in Q1 and that of TD methods, what do you observe (in terms of final performance, convergence rate, variance, etc)? Explain what you see.