CS698R: Deep Reinforcement Learning **(Submission Deadline: 11/09/21, 2100 Hrs)**

Assignment #1

Instructor: Ashutosh Modi
Submission Link: https://forms.gle/VyBNtoQ8fpwJvyQC6

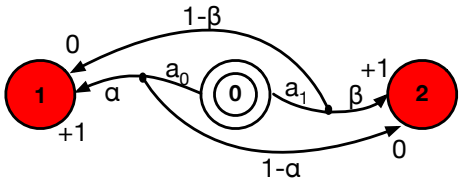
Read all the instructions below carefully before you start working on the assignment.

- The purpose of this course is that you learn RL and the best way to do that is by implementation and experimentation.
- The assignment requires your to implement some algorithms and you are required report your findings after experimenting with those algorithms.
- You are required to submit a report which would include the graphs/plots of the experiments you run and your findings.
- Solutions to the assignment need to be typeset in the LaTeX. Template for the report (Solutions-Assignment-1.tex file) is provided. DO NOT change the format of the template, use it as is provided to you without modifying it. In case you need any additional latex package, you can add it. You are required to compile the tex file and submit pdf version of the report.
- Implement the code in Google Colab Notebook and include link to your notebook in your report. Provide a shorter tinyurl link instead of the entire link. The code should be very well documented. There are marks for that.
- You are expected to implement algorithms on your own and not copy it from other sources/class mates. Of course, you can refer to lecture slides.
- If you use any reference or material (including code), please cite the source, else it will be considered plagiarism. But referring to other sources that directly solve the problems given in the assignment is not allowed. There is a limit to which you can refer to outside material.
- This is an individual assignment.
- In case your solution is found to have an overlap with solution by someone else (including external sources), all the parties involved will get zero in this and all future assignments plus further more penalties in the overall grade. We will check not just for lexical but also semantic overlap. Same applies for the code as well. **Even an iota of cheating would NOT be tolerated.** If you cheat one line or cheat one page the penalty would be same.
- Be a smart agent, think long term, if you cheat we will discover it somehow, the price you would be paying is not worth it.
- In case you are struggling with the assignment, seek help from TAs. Cheating is not an option! I respect honesty and would be lenient if you are not able to solve some questions due to difficulty in understanding. Remember we are there to help you out, seek help if something is difficult to understand.
- The deadline for the submission is given above. Submit at least 30 minutes before the deadline, lot can happen at the last moment, your internet can fail, there can be a power failure, you can be abducted by aliens, etc.
- You have to submit your assignment via following Google Form (link above)
- The form would close after the deadline and we will not accept any solution. No reason what-so-ever would be accepted for not being able to submit before the deadline.
- Since the assignment involves experimentation, reporting your results and observations, there is a lot of scope for creativity and innovation and presenting new perspectives. Such efforts would be highly appreciated and accordingly well rewarded. Be an exploratory agent!

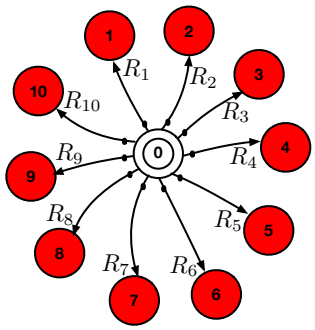
- In your plots, have a clear legend and clear lines, etc. Of course you would generating the plots in your code but you must also put these plots in your report. Generate high resolution pdf/svg version of the plots so that it doesn't pixilate on zooming.
- For implementing a new environment in OpenAI gym, you can refer to this tutorial: https://github.com/openai/gym/blob/master/docs/creating-environments.md. Of course, you are encouraged to refer to other tutorials as well but cite your sources. Also check out the `env` class: https://github.com/openai/gym/blob/master/gym/core.py. You are not required to render the environment on the screen/terminal. Do remember to set the seed, this will be useful for reproducing your experiments. And we will use the seed provided by you to verify your results. Also for each instance of the environment that you create use a different seed and save these seeds, these will be useful for reproducing the results. Do not set the seed to 42 :P
- For all experiments, report about the seed used in the code documentation and also in your report, write about the seed used.
- In your report write about all things that are not obvious from the code e.g., if you have made any assumptions, references/sources, running time, etc.

Problem 1: Multi-armed Bandits
(10+10+60+20+20+20+20+20+20=200 points)

In lecture 6 and 7 we learnt about Multi-armed bandit. In this problem, you will be implementing Multi-armed bandit with different strategies. The aim is to compare the performance of different strategies.
In particular, you will be implementing 2-armed Bernoulli Bandit and 10-armed Gaussian Bandit as described in slides 9-12 in Lecture 7. For more details about the setting please refer to the lecture.



(a) Two-armed Bernoulli Bandit



(b) Ten-armed Gaussian Bandit

2-armed Bernoulli Bandit: The mathematical formulation is as follows:
$R_0 \sim Bernoulli(\alpha)$
$R_1 \sim Bernoulli(\beta)$

10-armed Gaussian Bandit: The mathematical formulation is as follows:
$q_*(k) \sim \mathcal{N}(\mu = 0, \sigma^2 = 1)$
$R_k \sim \mathcal{N}(\mu = q_*(k), \sigma^2 = 1)$

1. In OpenAI Gym create the environment for 2-armed Bernoulli Bandit. The environment should take α and β as input parameters and simulate 2-armed bandit accordingly. Once you have implemented the environment, run it using different values of α and β to make sure it is executing as expected. For, example, you can try with $(\alpha, \beta) = (0, 0), (1, 0), (0, 1), (1, 1), (0.5, 0.5)$, etc. Report about your test cases and how they point towards the correct implementation. You can also report about your general observations.
2. Similarly, in OpenAI Gym create the environment for 10-armed Gaussian Bandit. Make sure it is executing as expected by creating certain test cases, e.g., by playing with σ. Report about your test cases and how they point towards the correct implementation. You can also report about your general observations.

3. Bandit Agents:
   (a) Create a function that implements the **Pure Exploitation (Greedy) strategy** (the function should have same signature as shown in the lecture). Run this for 2-armed Bernoulli Bandit to generate a table of actions and rewards (as in the lecture) and manually verify that the strategy is working as expected.
   (b) Create a function that implements the **Pure Exploration strategy** (the function should have same signature as shown in the lecture). Run this for for 2-armed Bernoulli Bandit to generate a table of actions and rewards (as in the lecture) and manually verify that the strategy is working as expected.
   (c) Create a function that implements the **ϵ-Greedy strategy** (the function should have same signature as shown in the lecture). Run this for 2-armed Bernoulli Bandit with different values of ϵ, ranging from small to large values. Verify that your implementation is working.
   (d) Create a function that implements the **decaying ϵ-Greedy strategy** (the function should have same signature as shown in the lecture). You can try two different versions of decay: linear and exponential. Start with the value of 1.0 for epsilon and decay it linearly/exponentially with pre-defined rate to 0.0. You can try with different rates of decays. The type of decay and the final decay rate are input parameters to the function, include these in the function definition.
   (e) Create a function that implements the **Softmax strategy** (the function should have same signature as shown in the lecture). You would have to play with the initial temperature parameter. For example, you start with initial temperature of 100 and decay it linearly to 0.01 or you start with initial temperature of ∞ and decay it linearly to 0.005, etc.
   (f) Create a function that implements the **UCB strategy** (the function should have same signature as shown in the lecture). You would have to play with the c parameter. For example, you can try $c = 0.2, c = 0.5$, etc.
4. Create 50 different 2-armed Bernoulli Bandit environments by sampling different values of α and β from a standard uniform distribution ($\mathcal{U}(0, 1)$). Run each of the agents above (6 in total) for 1000 time-steps (this is one run) for each instance of the environment. At each time step record the received reward. For a given agent, at each time step, average out the rewards over 50 instances of the environment. Draw a plot (e.g., using Matplotlib) of average rewards received vs. time step. Do this for all agents and plot it in the same plot. Now you can compare different agents (i.e., different strategies). Since different agents have different hyper-parameters, play with different settings and you can generate multiple different plots so as to aid comparison. Analyze the plots, write about your key observations, e.g., what strategy works better, what settings for a strategy works better, what your findings about the agents, etc.
5. Repeat the same thing as above for 10-armed Gaussian Bandit.
6. In the lecture we defined *Regret* as $\sum_{e=1}^{E} \mathbb{E}[v_* - q_*(A_e)]$, i.e. episode wise sum of the expected value of the difference between optimal action value (v_*) and true value (q_*) of taking an action A_e. For the 2-armed Bernoulli Bandit plot the regret vs episodes for each of the agent. Use the same setting of 50 environment instances as in the previous part. What do you observe? How do regrets evolve for different methods? Describe your observations in details.
7. Repeat the same thing as above for 10-armed Gaussian Bandit.
8. In lecture 7, slide 13, we saw the plot of % Optimal Action vs episodes. Repeat the same experiment using the setting of 50 instances of environment described above for 2-armed Bernoulli Bandit. What are your observations for different agents?
9. Repeat the same thing as above for 10-armed Gaussian Bandit.

Problem 2: Monte Carlo Estimates and TD Learning

(10+10+40+40+20+20+20+20+20+20+10+20+20+20+10=300 points)

In lecture 8, we saw the Random Walk Environment (RWE). Figure 2 is the RWE for reference, please refer to lecture for more details. In nutshell, there are 5 non-terminal states, two terminal states. An action leading

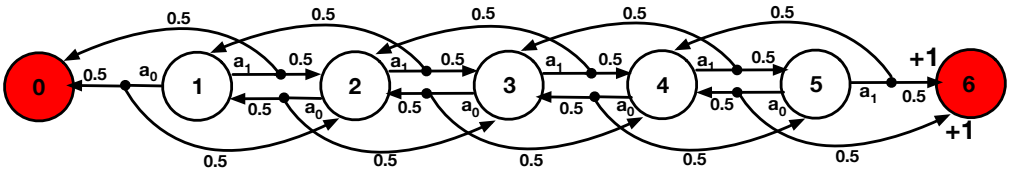


Figure 2: Random Walk Environment

to state 6 gives a reward of +1 and rest all actions give a reward of 0. The environment is purely random, no matter what action you take there is 50% chance that it will be executed as intended and rest 50% times it goes in opposite direction.

In lecture 8, we learnt about Monte Carlo method and TD learning method for estimating the value of a state:

Monte Carlo Estimate: $V_{e+1}(s) = V_e(s) + \alpha(e)\left[G_e - V_e(s)\right]$

TD Learning Estimate: $V_{t+1}(S_t) = V_t(S_t) + \alpha(t)\left[R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)\right]$

In this problem, you are required to calculate Monte Carlo and TD estimates for the RWE.

1. Implement a function that would simulate and generate a trajectory for RWE for a given policy π and maximum number of steps. The function definition would be like this:

   `def generateTrajectory(env, π, maxSteps)`

   The function returns a list of experience tuples. Here, `maxSteps` parameter is used to terminate the episode if it exceeds maxSteps count. In such a case, the partial trajectory is discarded and and empty list is returned. Test the function using suitable test cases and make sure it is working.

2. Implement a function that would decay the step size parameter (α). The function definition would be like this:

   `def decayAlpha(initialValue, finalValue, maxSteps, decayType)`

   Here `decayType` can be linear or exponential. `maxSteps` is the maximum number of steps the step parameter should decay for. `initialValue` and `finalValue` are initial and final values of the step size parameter. The function should return a list of step size parameter values. Test the function by trying out different parameter settings. Plot value of *alpha* vs time step both for linear and exponential decays.

3. As explained in the lecture, implement `MonteCarloPrediction` algorithm. Use the same function definition as described in the slides. Make use of the functions implemented in above two parts. Note `MonteCarloPrediction` should work for both FVMC and EVMC settings. Test the algorithm for RWE using some pre-defined test cases and see the algorithm produces the desired results. Report your test cases and observations.

4. As explained in the lecture, implement `TemporalDifferencePrediction` algorithm. Use the same function definition as described in the slides. Test the algorithm for RWE using some pre-defined test cases and see the algorithm produces the desired results. Report your test cases and observations.

5. Plot the MC-FVMC estimate of each non-terminal state of RWE as it progress through different episodes. In the same plot also plot the true estimate. The plot will be similar to shown in slide 74, lecture 8. Take maximum of 500 episodes. The step size parameter (α) starts from 0.5 and decreases exponentially to 0.01 till 250 episodes and after that it is constant. Analyze the plots for each state and report your observations, findings and possible reasons for the observed behavior.

6. Plot the MC-EVMC estimate of each non-terminal state of RWE as it progress through different episodes. In the same plot also plot the true estimate. Take maximum of 500 episodes. The step size parameter (α) starts from 0.5 and decreases exponentially to 0.01 till 250 episodes and after that it is constant. Analyze the plots for each state and report your observations, findings and possible reasons for the observed behavior. How does EVMC fair against FVMC?

7. Plot the TD estimate of each non-terminal state of RWE as it progress through different episodes. In the same plot also plot the true estimate. The plot will be similar to shown in slide 75, lecture 8. Take maximum of 500 episodes. The step size parameter (α) starts from 0.5 and decreases exponentially to 0.01 till 250 episodes and after that it is constant. Analyze the plots for each state and report your observations, findings and possible reasons for the observed behavior.

8. Plot the MC-FVMC estimate of each non-terminal state of RWE as it progress through different episodes. But this time, the x-axis (episodes) should be log-scale. In the same plot also plot the true estimate. The plot will be similar to shown in slide 76, lecture 8. Take maximum of 500 episodes. The step size parameter (α) starts from 0.5 and decreases exponentially to 0.01 till 250 episodes and after that it is constant. This plot will help to zoom in and observe the behavior of the estimates in the initial stages. Analyze the plots for each state and report your observations, findings and possible reasons for the observed behavior.

9. Plot the MC-EVMC estimate of each non-terminal state of RWE as it progress through different episodes. But this time, the x-axis (episodes) should be log-scale. In the same plot also plot the true estimate. Take maximum of 500 episodes. The step size parameter (α) starts from 0.5 and decreases exponentially to 0.01 till 250 episodes and after that it is constant. This plot will help to zoom in and observe the behavior of the estimates in the initial stages. Analyze the plots for each state and report your observations, findings and possible reasons for the observed behavior. How does EVMC fair against FVMC?

10. Plot the TD estimate of each non-terminal state of RWE as it progress through different episodes. But this time, the x-axis (episodes) should be log-scale. In the same plot also plot the true estimate. The plot will be similar to shown in slide 77, lecture 8. Take maximum of 500 episodes. The step size parameter (α) starts from 0.5 and decreases exponentially to 0.01 till 250 episodes and after that it is constant. This plot will help to zoom in and observe the behavior of the estimates in the initial stages. Analyze the plots for each state and report your observations, findings and possible reasons for the observed behavior.

11. Based on the plots, compare MC-FVMC, MC-EVMC and TD approaches and report your observations.

12. Plot the MC-FVMC Target value (G_t) for any one non-terminal state of RWE as it progress through different episodes. Use the same setting as above. In the same plot also include the optimal value of the state. The plot will be similar to discussed in slide 78 of lecture 8. What do you observe and what are the reasons for what you observe? Explain and Report.

13. Plot the MC-EVMC Target value (G_t) for any one non-terminal state of RWE (use the same state as above) as it progress through different episodes. Use the same setting as above. In the same plot also include the optimal value of the state. What do you observe and what are the reasons for what you observe? Explain and Report.

14. Plot the TD Target value (G_t estimate) for any one non-terminal state of RWE (use the same state as above) as it progress through different episodes. Use the same setting as above. In the same plot also include the optimal value of the state. The plot will be similar to discussed in slide 79 of lecture 8. What do you observe and what are the reasons for what you observe? Explain and Report.

15. Based on the plots, compare MC-FVMC, MC-EVMC and TD targets and report your observations.