Released: 18th Mar Deadline: 7th Apr

Problem Set 5

INF368A - Reinforcement Learning
Spring 2023

IMPORTANT: Submission of this problem set is **compulsory** to be admitted to the final exam and graduate from the course.

Part I. Problems

In the first part (*Problems*), you are required to provide answers for all the *obligatory* questions, that is, questions marked by a star (*). The remaining questions are optional; however, you are invited to solve them because: (i) they are stepping-stones towards the obligatory questions; (ii) they are similar to the questions you will be asked at the oral exam.

1. Policy Gradient

Exercise1 Stochastic Policies

Consider an agent learning to make cookies. The underlying environment is described in the following way.

The agent starts in state `empty_bowl` where it has two actions: *wait*, which will leave the agent in the same state; and *prepare_dough* which deterministically transfer the agent to the `full_bowl` state. In state `full_bowl` the agent can choose between the following actions: *wait*, which will leave the agent in the same state; *add_sugar* which deterministically transfer the agent to the `sugared_bowl` state; or *add_salt* which deterministically transfer the agent to the `salted_bowl` state.

In state `sugared_bowl` the agent can choose between the following actions: *taste* which deterministically makes the robot get a sample of dough and analyze its content in state `sugar_sampling`; *cook* which deterministically transfer the agent to the

`baked_sugary_cookies` state; and *restart* which deterministically makes the robot discard the current dough and produce a new one, thus leading back to `full_bowl`. From the state `sugar_sampling` the agent can either choose the action *cook* which deterministically transfer the agent to the `baked_sugary_cookies` state; or the action *return_to_bowl*, which goes back to state `sugared_bowl`.

Symmetrically, in state `salted_bowl` the agent can choose between the following actions: *taste* which deterministically makes the robot get a sample of dough and analyze its content in state `salt_sampling`; *cook* which deterministically transfer the agent to the `baked_salty_cookies` state; and *restart* which deterministically makes the robot discard the current dough and produce a new one, thus leading back to `full_bowl`. From the state `salt_sampling` the agent can either choose the action *cook* which deterministically transfer the agent to the `baked_salty_cookies` state; or the action *return_to_bowl*, which goes back to state `salted_bowl`.

Both `baked_sugary_cookies` and `baked_salty_cookies` are terminal states, returning respectively a reward of $+10$ and -10.

1. Define the MDP underlying the above environment.
2. Draw out the backup diagram for this environment.
3. (*) Can you define an optimal deterministic policy for this environment? If so, write it out. If not, why not?
4. (*) Can you define an optimal stochastic policy for this environment? If so, write it out. If not, why not?

Suppose now that the agent has lost the ability to visually discriminate between salt and sugar (labels have been removed or the visual sensor of the agent has degraded). The environment is then re-defined as follows.

State `empty_bowl` and actions therein are unmodified. In state `full_bowl` the agent can now choose only between two actions: *wait*, as before; and *add_ingredient* which add a random ingredient between sugar and salt and deterministically transfer the agent to the `enriched_bowl` state. Notice that the addition of either sugar or salt is tracked by a latent environment variable σ unknown to the agent.

In state `enriched_bowl` the agent can choose between the following actions: *taste* which makes the robot get a sample of dough and analyze its content – the resulting state will be either `sugar_sampling` or `salt_sampling` deterministically according to σ; *cook* which transfers the agent to the `baked_sugary_cookies` or `baked_salty_cookies` state, again deterministically according to σ; and *restart* which deterministically makes the robot discard the current dough and produce a new one, thus leading back to `full_bowl`.

The remaining states `sugar_sampling`, `salted_bowl`, `baked_sugary_cookies` and `baked_salty_cookies` follow the same behaviour as before.

5. Define the MDP underlying the above environment.
6. Draw out the backup diagram for this environment.
7. (*) Can you define an optimal deterministic policy for this environment? If so, write it out. If not, why not?

8. (*) Can you define an optimal stochastic policy for this environment? If so, write it out. If not, why not?
9. (*) How does this setup differ from the previous one?
10. Is there any modification to the MDP that you would naturally suggest and which would improve your ability in defining a deterministic or stochastic policy?

Exercise2 Continuous Actions

In class we discussed how to define a function approximator for the policy $\pi(a|s)$ using neural networks. Assume we have a discrete and finite number of actions $\mathcal{A} = \{a_1, a_2, ..., a_n\}$. For instance, in chess, there is only a limited number of actions available.

1. Recall how we defined the neural networks so that the output would give a distribution over the actions.

Assume now we have a continuous number of actions $\mathcal{A} = \mathbb{R}$. For instance, an agent can move any place along a line.

2. Suggest a possible distribution function to model your policy.
3. Consider what sort of parameters or statistics you would need to work with the distribution you have chosen.
4. (*) How would you define your neural network to have a distribution over $\mathbb{R}$ in output?
5. What assumptions underlie your choice for the probability distribution?

Exercise3 MAB and Baselines

Recall the gradient bandit algorithm for MABs. We defined the update rule as:

$$H^{(t+1)}(a_i) = H^{(t)}(a_i) + \alpha(R^{(t)} - \hat{E}[R])(1 - p^{(t)}(a_i))$$

1. (*) Which term in this expression plays a role analogous to a baseline in a policy gradient algorithms?
2. (*) How does this term affect the learning process in the gradient bandit algorithm?

Exercise4 Actor-critic methods

In class we suggested the analogy that in an actor-critic method, the *critic* roughly performs a step similar to policy evaluation and the *actor* performs a step similar to policy improvement[1].

1. In which sense actor and critic may be assimilated to these two processes of reinforcement learning?

[1] Notice that in the *v1* version of the slides these terms are wrongly swapped. This has been corrected in *v1_1*.

Part II.
Report

In the second part (*Report*), you are required to provide a report of maximum 2 pages describing the experiments you have run and analyzed to answer the *obligatory* questions, that is, questions marked by a star (*). The remaining questions are optional; however, you are invited to solve them because: (i) they are stepping-stones towards the obligatory questions; (ii) they are similar to the questions you will be asked at the oral exam. You will be evaluated only on your report. Review the documents on `coding tips` and `writing tips` for advices on how to tackle this part.

Exercise5 Function Approximation and Policy Gradient

Consider the problem of monitoring a conveyor belt with a robot. The conveyor belt is transporting cookies, but because of random perturbations cookies are falling from the belt to the ground.

The task of the robot is to patrol the floor and retrieve the cookies as quickly as possible; however, if a cookie remains on the floor for longer than five second, it is considered discarded and disappears.

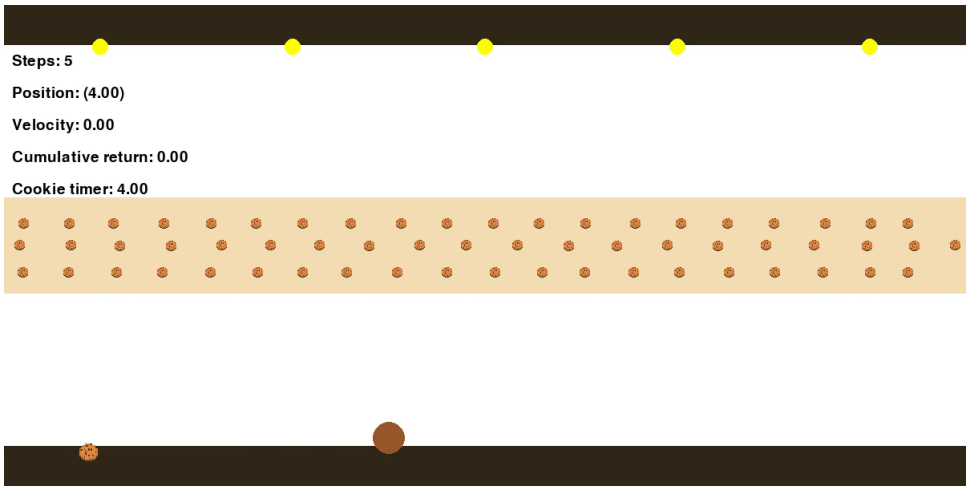
To make things more complicated, the robot is controlled by two thrusters that allow it to increase its velocity either rightward or leftward. In order not to crash the fallen cookies, the speed of the robot should not be too high; moreover, to prevent damage, the robot should also avoid hitting the walls of the room where it is patrolling.

We summarize below facts about the dynamics of the environment (see picture):

- *Objective:* the aim of the robot is to retrieve as many cookies as possible, while avoiding hitting the walls.
- *Dynamics:* Cookies are falling from the conveyor belt at random locations. A cookie remains on the floor until it is collected or a 5 seconds timer expires. As soon as a cookie is collected or disappears, another cookie immediately falls from the conveyor belt.
- *Representation:* the room and the conveyor belt are represented in one dimension. The conveyor belt stretches 10 meters. An observation of the environment provides the position and the velocity of the robot together with the location and the remaining timer of the cookie. All values are real numbers.
- *Start state:* the robot will start on the left side of the screen (position 4) with null speed (velocity 0).
- *Actions:* the robot can use its thrusters and it thus has three actions: $-5, 0, +5$ corresponding respectively to increasing its leftward velocity, doing nothing, and increasing its rightward velocity.
- *Transitions:* the velocity of the robot changes according to the action chosen and the friction of the floor which will slow the robot down proportionally to

friction $= -(\text{velocity}^2 \times 0.005)$. The overall change in position is computed as $\text{position}(t + \Delta) = \text{position}(t) + (\text{action}(t) + \text{friction}(t))\Delta$.

- *Reward:* the robot receives the following rewards:
  - $+1$ if it reaches or move beyond a cookie with velocity ≤ 4;
  - -1 if it reaches or move beyond a cookie with velocity > 4, thus crashing the cookie;
  - $-.5$ if it does not reach a cookie before the 5 seconds deadline;
  - $-\text{velocity}^2 \times 0.1$ if it hits one of the side walls.



1. Is this task most naturally described as episodic or continuing? Why?
2. Use a linear approximation to approximate $q_\pi(a, s)$ and solve the control problem using Q-learning. You may want to consider the use of feature engineering to pre-process the state.
3. Use a neural network to approximate $q_\pi(a, s)$ and solve the control problem using Q-learning. You may want to consider the use of feature engineering to pre-process the state.
4. (*) Explain the two solutions based on function approximation that you have implemented and comment the results you obtained.
5. Implement an actor-critic model to learn directly an optimal policy $\pi(a|s)$.
6. (*) Explain the actor-critic solution and compare the result with the value-based solutions above.
7. Consider again the solution based on neural approximation of Q-learning. Try to improve your solution adding a *replay buffer* or a *target network*. Discuss the result.