Homework 2

6501 Reinforcement Learning (Spring 2024)

Submission deadline: 11:59pm, March 17

Rules

- Collaboration is allowed but must be stated in precision per sub-problem. For example, comment in the sub-problem which idea is a result of a discussion with whom.
- Leveraging large language models is allowed but must be stated. Describe how you use them or provide precise prompts that you use.
- Please put all proofs and figures for experiments in a single pdf file. Submit the pdf to the Question 1 in Gradescope. Put all codes in a single .py file **(not .ipynb file)**, and submit it to Question 2 in Gradescope.
- For theoretical problems, rigorous proofs are required.

1 Approximate Bellman Optimality Implies Approximate Value Function

Consider an MDP with state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $R : \mathcal{S} \times \mathcal{A} \to [0, 1]$, transition $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, and discount factor $\gamma \in [0, 1)$.

Suppose that $\hat{V} : \mathcal{S} \to \mathbb{R}$ is a function that satisfies the approximate Bellman optimality equation:

$$\forall s \in \mathcal{S}, \qquad \left| \hat{V}(s) - \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) \hat{V}(s') \right) \right| \leq \epsilon. \tag{1}$$

for some $\epsilon > 0$.

For simplicity, below we use the Bellman optimality operator notation $\mathcal{T}V$ defined as

$$\mathcal{T}V(s) := \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V(s') \right)$$

for any V. Therefore, Eq. (1) can also be written as

$$\forall s \in \mathcal{S}, \qquad \left| \hat{V}(s) - \mathcal{T}\hat{V}(s) \right| \leq \epsilon. \tag{2}$$

(a) (6%) Prove that for all $s \in \mathcal{S}$,

$$\left| \hat{V}(s) - V^\star(s) \right| \leq \left| \mathcal{T}\hat{V}(s) - \mathcal{T}V^\star(s) \right| + \epsilon.$$

(b) (8%) Prove that for all $s \in \mathcal{S}$,

$$\left| \mathcal{T}\hat{V}(s) - \mathcal{T}V^\star(s) \right| \leq \gamma \max_{s' \in \mathcal{S}} \left| \hat{V}(s') - V^\star(s') \right|.$$

(c) (4%) Combine (a) and (b) to show

$$\forall s \in \mathcal{S}, \qquad \left| \hat{V}(s) - V^\star(s) \right| \leq \frac{\epsilon}{1 - \gamma}.$$

2 Approximate Bellman Optimality Implies Near-Optimal Policy

Consider the same condition as in Problem 1: Suppose that $\hat{V}$ satisfies Eq. (1). Define the greedy policy with respect to $\hat{V}$ as

$$\hat{\pi}(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \hat{V}(s') \right).$$

(a) (6%) Prove that $\hat{V}$ satisfies approximate Bellman equations with respect to $\hat{\pi}$. That is,

$$\forall s \in \mathcal{S}, \qquad \left| \hat{V}(s) - \left(R(s, \hat{\pi}(s)) + \gamma \sum_{s'} P(s'|s, \hat{\pi}(s)) \hat{V}(s') \right) \right| \leq \epsilon.$$

(b) (6%) Prove that

$$\forall s \in \mathcal{S}, \qquad \left| \hat{V}(s) - V^{\hat{\pi}}(s) \right| \leq \frac{\epsilon}{1-\gamma}.$$

(c) (4%) Prove that

$$\forall s \in \mathcal{S}, \qquad V^{\star}(s) - V^{\hat{\pi}}(s) \leq \frac{2\epsilon}{1-\gamma}.$$

3 Implementation of Policy-Based Contextual Bandit Algorithms

In this problem, we will implement a policy-based contextual bandit algorithm. This setting is actually not covered in the class — we studied exponential weight and projected gradient descent under the setting without contexts. The goal of this problem is to show how to handle contexts for this class of methods. The algorithm to be implemented is closely related to the Proximal Policy Optimization (PPO) algorithm for general MDPs, which we will / plan to implement in future Homework. Therefore, this problem serves as a primer for it.

Recall that the exponential weight algorithm for finite-arm setting is the following:

$$p_{t+1}(a) = \frac{p_t(a) \exp\left(\eta \hat{r}_t(a)\right)}{\sum_{a'} p_t(a') \exp\left(\eta \hat{r}_t(a')\right)} \tag{3}$$

where $\hat{r}_t(a)$ is an unbiased reward estimator of the following form:

$$\hat{r}_t(a) = \frac{(r_t(a) - b_t - c_t(a)) \mathbb{I}\{a_t = a\}}{p_t(a)} + c_t(a)$$

with b_t being an action-independent baseline (see Page 21 here), and $c_t(a)$ an action-dependent baseline (we call it doubly robust estimator in the class; see Page 16 here). In multi-armed bandits, we have shown that by setting $b_t = 1$ and $c_t(a) = 0$ we can get $\tilde{O}(\sqrt{AT})$ regret.

We have shown in the class that Eq. (3) is equivalent to KL-divergence regularized policy update:

$$p_{t+1} = \operatorname*{argmin}_{p \in \Delta(\mathcal{A})} \left\{ \langle p, \hat{r}_t \rangle - \frac{1}{\eta} \mathrm{KL}(p, p_t) \right\} = \operatorname*{argmin}_{p \in \Delta(\mathcal{A})} \left\{ \sum_a p(a) \hat{r}_t(a) - \frac{1}{\eta} \sum_a p(a) \ln \frac{p(a)}{p_t(a)} \right\} \tag{4}$$

where $\Delta(\mathcal{A})$ is the probability space over the action set $\mathcal{A}$. Below, we extend the algorithm Eq. (4) to allow context inputs (Section 3.1) and allow function approximation (Section 3.2).

3.1 Handling Contexts

When there are contexts, the policy becomes a mapping from context to action distribution. We use $p(a|x)$ to denote such distribution, where $x \in \mathcal{X}$ in the context and $a \in \mathcal{A}$ is the action.

Suppose that the context is i.i.d. drawn from some distribution $\mathcal{D}$. Then we modify Eq. (4) as the following:

$$\begin{aligned} p_{t+1} &= \operatorname*{argmin}_{p:\mathcal{X}\to\Delta(\mathcal{A})} \left\{ \mathbb{E}_{x\sim\mathcal{D}} \left[\langle p(\cdot|x), \hat{r}_t(x,\cdot)\rangle - \frac{1}{\eta}\mathrm{KL}(p(\cdot|x), p_t(\cdot|x)) \right] \right\} \\ &= \operatorname*{argmin}_{p:\mathcal{X}\to\Delta(\mathcal{A})} \left\{ \mathbb{E}_{x\sim\mathcal{D}} \left[\sum_a p(a|x)\hat{r}_t(x,a) - \frac{1}{\eta}\sum_a p(a|x) \ln \frac{p(a|x)}{p_t(a|x)} \right] \right\}, \end{aligned} \tag{5}$$

where $\hat{r}_t(x,a)$ is an unbiased reward estimator for action a when context x is presented (will be defined later).

Since $\mathcal{D}$ is unknown, in each iteration of the policy update, we will draw several i.i.d. samples to approximate $\mathcal{D}$. Let $x_{t,1}, x_{t,2}, \ldots, x_{t,N}$ be i.i.d. context samples drawn in iteration t. Then we approximate Eq. (5) with the following:

$$p_{t+1} = \operatorname*{argmin}_{p:\mathcal{X}\to\Delta(\mathcal{A})} \left\{ \frac{1}{n}\sum_{n=1}^{N} \left[\sum_a p(a|x_{t,n})\hat{r}_t(x_{t,n},a) - \frac{1}{\eta}\sum_a p(a|x_{t,n}) \ln \frac{p(a|x_{t,n})}{p_t(a|x_{t,n})} \right] \right\}. \tag{6}$$

To construct the reward estimator $\hat{r}_t(x,a)$, we again use the idea of importance sampling / inverse propensity weighting. Suppose that in iteration t, we use policy p'_t to choose actions, i.e., when seeing $x_{t,n}$, we sample $a_{t,n} \sim p'_t(\cdot|x_{t,n})$, and obtain reward $r_{t,n}$ with $\mathbb{E}[r_{t,n}] = R(x_{t,n}, a_{t,n})$. Then we can construct the following unbiased reward estimator:

$$\hat{r}_t(x_{t,n}, a) = \frac{r_{t,n}\mathbb{I}\{a_{t,n} = a\}}{p'_t(a|x_{t,n})}. \tag{7}$$

Note that p'_t can be same or different from p_t (e.g., adding extra ϵ-greedy exploration as in Page 16 here). It is again possible to add baselines: let $b_t(x)$ be a function of x, and $c_t(x,a)$ be a function of (x,a). Then

$$\hat{r}_t(x_{t,n}, a) = \frac{(r_{t,n} - b_t(x_{t,n}) - c_t(x_{t,n},a))\mathbb{I}\{a_{t,n} = a\}}{p'_t(a|x_{t,n})} + c_t(x_{t,n}, a) \tag{8}$$

also serves as an unbiased reward estimator.

Combining the update rule Eq. (6) and the reward estimator Eq. (7) or Eq. (8), we get a concrete algorithm that allows us to update policies. To make the algorithm practical, we restrict our search of policies in a parameterized set of policies (discussed in the next subsection).

3.2 Using Function Approximation

We will use function approximation to represent the policy. Let $p_\theta : \mathcal{X} \to \Delta(\mathcal{A})$ be a neural network with weights θ, which takes the context as input, and output a distribution over actions. In iteration t of policy updates, we let $p_t = p_{\theta_t}$. To perform Eq. (6), we simply run gradient descent on θ to minimize the objective on the right-hand side of Eq. (6). The update of θ in iteration t can be more concretely written out as Algorithm 1.

We emphasize again that the whole Algorithm 1 is only doing a single update represented in Eq. (6). The m-for-loop is using multiple gradient steps to approximate the argmin in Eq. (6). This is also the reason why the p_{θ_t} in Algorithm 1 should be kept fixed over all M steps, instead of changing over m.

3.3 Combining Everything

According to the discussions in the previous two subsections, our contextual-bandit algorithm is designed as in Algorithm 2.

Algorithm 1 The t-th iteration of Eq. (6)

(Suppose that you have already used p'_t to collect data $\{(x_{t,n}, a_{t,n}, r_{t,n})\}_{n=1}^N$ and constructed $\{\hat{r}_t(x_{t,n}, a)\}_{n=1}^N$ as in Eq. (7) or Eq. (8))

$\theta \leftarrow \theta_t$

for $m = 1, \ldots, M$ **do**

$$\theta \leftarrow \theta - \lambda \nabla_\theta \left\{ \frac{1}{N} \sum_{n=1}^N \left[\sum_a p_\theta(a|x_{t,n}) \hat{r}_t(x_{t,n}, a) - \frac{1}{\eta} \sum_a p_\theta(a|x_{t,n}) \ln \frac{p_\theta(a|x_{t,n})}{p_{\theta_t}(a|x_{t,n})} \right] \right\}. \tag{9}$$

$\theta_{t+1} \leftarrow \theta$

Algorithm 2 Policy-Based CB Algorithm

Randomly initialize a policy network where the input is the context and output is an action distribution.

Let θ_1 be the initial weights for the policy network.

for $t = 1, \ldots, T$ **do**

  for $n = 1, \ldots, N$ **do**

    Receive context $x_{t,n}$.

    Sample action $a_{t,n} \sim p_{\theta_t}(\cdot|x_{t,n})$ (may also use some other policy p'_t).

    Receive reward $r_{t,n}$.

    Define for all $a \in \mathcal{A}$

$$\hat{r}_t(x_{t,n}, a) = \frac{r_{t,n} \mathbb{I}\{a_{t,n} = a\}}{p_{\theta_t}(a|x_{t,n})}. \tag{10}$$

    (may use baselines as in Eq. (8))

  $\theta \leftarrow \theta_t$

  for $m = 1, \ldots, M$ **do**

$$\theta \leftarrow \theta - \lambda \nabla_\theta \left\{ \frac{1}{n} \sum_{n=1}^N \left[\sum_a p_\theta(a|x_{t,n}) \hat{r}_t(x_{t,n}, a) - \frac{1}{\eta} \sum_a p_\theta(a|x_{t,n}) \ln \frac{p_\theta(a|x_{t,n})}{p_{\theta_t}(a|x_{t,n})} \right] \right\}. \tag{11}$$

  $\theta_{t+1} \leftarrow \theta$

3.4 Some Practical Considerations

Below, we assume that in iteration t, we simply use $p'_t = p_{\theta_t}$ to choose actions.

3.4.1 The regularization term

In Eq. (11), the first summation actually can be written as the following

$$\sum_a p_\theta(a|x_{t,n}) \hat{r}_t(x_{t,n}, a) = \frac{p_\theta(a_{t,n}|x_{t,n})}{p_{\theta_t}(a_{t,n}|x_{t,n})} r_{t,n} \tag{12}$$

by the definition of $\hat{r}_t(x_{t,n}, a)$. This only involves $a_{t,n}$ but not other a. The second summation $\sum_a p_\theta(a|x_{t,n}) \ln \frac{p_\theta(a|x_{t,n})}{p_{\theta_t}(a|x_{t,n})}$, however, requires a summation over all actions, making the implementation slightly more complicated. One straightforward way to make it simpler is to use the reversed KL-divergence $\sum_a p_{\theta_t}(a|x_{t,n}) \ln \frac{p_{\theta_t}(a|x_{t,n})}{p_\theta(a|x_{t,n})}$ to do the distance regularization (i.e., changing the roles of θ, θ_t in the KL-divergence regularization term). Then further observe that since

$a_{t,n}$ is drawn from p_{θ_t}, we have

$$\mathbb{E}\left[\ln \frac{p_{\theta_t}(a_{t,n}|x_{t,n})}{p_\theta(a_{t,n}|x_{t,n})}\right] = \sum_a p_{\theta_t}(a|x_{t,n}) \ln \frac{p_{\theta_t}(a|x_{t,n})}{p_\theta(a|x_{t,n})},$$

with which we can replace the regularization term in Eq. (11) by

$$-\frac{1}{\eta} \ln \frac{p_{\theta_t}(a_{t,n}|x_{t,n})}{p_\theta(a_{t,n}|x_{t,n})} \tag{13}$$

without changing its expectation. This way, this term also just involves $a_{t,n}$ but not all a, which is simpler to implement.

Actually, this is not the only way to construct an unbiased estimator for $\mathrm{KL}(p_\theta(\cdot|x_{t,n}), p_{\theta_t}(\cdot|x_{t,n}))$ or $\mathrm{KL}(p_{\theta_t}(\cdot|x_{t,n}), p_\theta(\cdot|x_{t,n}))$. According to an anslysis in John Schulman's article, the following two equalities are true. Define $u = \frac{p_\theta(a_{t,n}|x_{t,n})}{p_{\theta_t}(a_{t,n}|x_{t,n})}$:

$$\mathbb{E}\left[u \ln u - (u-1)\right] = \mathrm{KL}(p_\theta(\cdot|x_{t,n}), p_{\theta_t}(\cdot|x_{t,n})), \tag{14}$$

$$\mathbb{E}\left[u - 1 - \ln u\right] = \mathrm{KL}(p_{\theta_t}(\cdot|x_{t,n}), p_\theta(\cdot|x_{t,n})). \tag{15}$$

Therefore, one can choose either one of them in the implementation of Eq. (11). As stated in John Schulman's article, there are other considerations why we want to use these alternatives besides the fact that they are simpler to implement.

3.4.2 The reward-estimator term

The reward estimator term $\frac{p_\theta(a_{t,n}|x_{t,n})}{p_{\theta_t}(a_{t,n}|x_{t,n})} r_{t,n}$ in Eq. (12) could have a large scale due to the division of $p_{\theta_t}(a_{t,n}|x_{t,n}) < 1$. In exponential weight algorithm, one way to control it is to add a small uniform exploration probability ϵ and so this reward-estimation term would become $\frac{p_\theta(a_{t,n}|x_{t,n})}{(1-\epsilon)p_{\theta_t}(a_{t,n}|x_{t,n}) + \frac{\epsilon}{A}}$ (Page 16 here). In fact, by just adding a small number in the denominator without actual performing extra exploration is also fine (this will make the estimator slightly biased though). We will introduce some more common clipping trick when we introduce PPO later in the course.

3.5 Tasks

For simplicity, we will use the digit dataset in Homework 1 again. The goal of this homework problem is not to achieve improved performance compared to Homework 1, but more on getting familiar with this different style of algorithm.

The starter code is provided in `https://bahh723.github.io/course/HW/HW2.py`. The N and M in Algorithm 2 is also denoted as N and M in the starter code. You mainly will implement Eq. (10) and Eq. (11), which are both in the `update_batch` function. The number of lines you need to code is $\lesssim 10$. The performance of this algorithm might be slightly worse than what's achieved in Homework 1, but still try to tune the parameters to achieve an average reward $\gtrsim 0.6$.

(a) (25%) Read and understand the starter code. Implement the TODO's based on the instructions above. You can make any modifications to the starter code, but try to place all codes in a single file.

There is an implementation detail to notice: Observe that in Eq. (11), the values of $p_{\theta_t}(a|x_{t,n})$ (which appears both in the reward estimator term and the KL regularization term) need to be viewed as a constant during the m-for-loop. However, their values are calculated using the initial value of θ. Therefore, you will need to use the `Tensor.detach()` function to detach the value of $p_{\theta_t}(x_{t,n}, a)$ from the gradient calculation.

(b) (15%) Find the value of M and N that lead to the best performance. Show the values of M and N you have tried and their corresponding average reward.

(c) (15%) Implement the simplest baseline where $b_t(x) = b$ and $c_t(x, a) = 0$ in Eq. (8). Adjust the value of b and plot the performance (x-axis: time t, y-axis: the learner's average reward in $[1, t]$).

(d) [optional] (15%) Implement any more adaptive baseline methods. For example, have another network learning the value of context $b_t(x)$ or the value of context-action pair $c_t(x, a)$ on the fly. Compare the performance with the non-adaptive ones. Describe how you implement those baselines.

(e) [optional] (15%) Find another (preferably larger-scale) dataset and repeat (b)-(d).

(f) (10%) Elaborate on any interesting observations or tricks you have found in the experiments.

4 Survey

(a) (5%) How much time did you use to complete each of the problems in this homework? Do you have any suggestion for the course? (e.g., the pace of the lecture, the length of the homework)