CS 277 (W22): Control and Reinforcement Learning
Assignment 5

Due date: Tuesday, March 15, 2022 (Pacific Time)

Roy Fox
https://royf.org/crs/W22/CS277

Instructions: In theory questions, a formal proof is not needed (unless specified otherwise). instead, briefly explain informally the reasoning behind your answers.

In practice questions, include a printout of your code as a page in your PDF, and a screenshot of TensorBoard learning curves (`episode_reward_mean`, unless specified otherwise) as another page.

Part 1 Actor–Critic PG and Bounded RL (40 points)

Recall that Actor–Critic Policy-Gradient algorithms represent an actor π_θ and a critic V_ϕ. Many such algorithms use a temporal-difference loss to update the critic and a policy-gradient loss to update the actor. The simplest such algorithm we saw gathers on-policy experience (s, a, r, s'), and then uses the target $y_{\bar{\phi}}(r, s') = r + \gamma V_{\bar{\phi}}(s')$ (with $V_{\bar{\phi}}$ a target network) to compute the critic loss $\mathcal{L}_\phi = \frac{1}{2}(y_{\bar{\phi}}(r, s') - V_\phi(s))^2$. It also uses the critic's advantage estimate $A = y_\phi(r, s') - V_\phi(s)$ (usually with $V_\phi(s')$ from the non-target critic network) to compute the actor loss $\mathcal{L}_\theta = A \log \pi_\theta(a|s)$. The algorithm then descends the total loss $\mathcal{L}_{\text{AC}} = \mathcal{L}_\phi + \eta\mathcal{L}_\theta$, with η a coefficient relating the two losses.

Question 1 (5 points) Explain why it makes sense for y_ϕ to serve as an estimator for $Q(s, a)$ (in the actor loss) while its target-network counterpart $y_{\bar{\phi}}$ serves as an estimator for $V(s)$ (in the critic loss). What is the relevant difference between how y_ϕ and $y_{\bar{\phi}}$ are used?

Now recall that, in the Bounded RL framework, the optimal policy is

$$\pi(a|s) = \frac{\pi_0(a|s) \exp \beta Q(s, a)}{Z(s)}, \tag{1}$$

with the normalizer ("partition function") $Z(s) = \mathbb{E}_{(a|s)\sim\pi_0}[\exp \beta Q(s, a)]$. If this policy is plugged into the bounded Bellman recursion, we get

$$V(s) = \frac{1}{\beta} \log Z(s) = \frac{1}{\beta} \log \mathbb{E}_{(a|s)\sim\pi_0}[\exp \beta Q(s, a)]. \tag{2}$$

Rearranging (1) and (2), we get

$$Q(s, a) = V(s) + \frac{1}{\beta} \log \frac{\pi(a|s)}{\pi_0(a|s)}. \tag{3}$$

Question 2 (10 points) Consider implementing the SQL algorithm with a value network $Q_{\theta,\phi} : S \to \mathbb{R}^A$ with the structure in (3). Namely, the network has two heads, $\pi_\theta : S \to \Delta(A)$ and $V_\phi : S \to \mathbb{R}$, which are combined as in (3) to compute $Q_{\theta,\phi}$. There is also a target network that keeps a delayed copy $V_{\bar{\phi}}$ of V_ϕ. What is the gradient of the SQL loss $\mathcal{L}_{\text{SQL}} = \frac{1}{2}(y_{\bar{\phi}}(r, s') - Q_{\theta,\phi}(s, a))^2$ with respect to θ? with respect to ϕ?

Question 3 (10 points) Write an expression for a pseudo-reward[1] $\tilde{r}$, such that the AC critic loss $\mathcal{L}_\phi$ with $y_{\bar{\phi}}(\tilde{r}, s')$ is the same as the SQL loss $\mathcal{L}_{\text{SQL}}$ with $y_{\bar{\phi}}(r, s')$.

Question 4 (10 points) Show that the gradient $\nabla_\theta \mathcal{L}_\theta$ of the AC actor loss $\mathcal{L}_\theta$, with $y_\phi(\tilde{r}, s')$ is the same as the gradient $\nabla_\theta \mathcal{L}_{\text{SQL}}$ of the SQL loss with $y_{\bar{\phi}}(r, s')$, except that the latter is using the target network.

Question 5 (5 points) Under the equivalence in the previous two questions, what is the equivalent of β in this version of AC PG?

Part 2 Option–Critic (60 points)

Question 1 (10 points) Download the following implementation of the Option–Critic algorithm: https://github.com/alversafa/option-critic-arch. Read `option_critic.ipynb`, and make the following changes:

1. In parts 3 and 4, add color bars (see `matplotlib.pyplot.colorbar`) to the heat maps.
2. In part 4, plot the following three histograms:
   (a) For each option h (on the x-axis), the number of times option h was called in an episode.
   (b) For each option h (on the x-axis), the average number of actions option h took each time it was called before it terminated.
   (c) For each option h (on the x-axis), the total number of actions it took in an episode (summed over all times it was called).

   In each of these histograms, plot the average and SEM error bars over 10 episodes.

Run the code, and attach the resulting plots.

Question 2 (5 points) Does the agent seem to be high-fitting (i.e. a single option solves much of the entire task)? Does it seem to be low-fitting (i.e. options terminate very quickly, such that the meta-policy solves much of the entire task)? Explain which results make you think so and why.

[1] $\tilde{r}$ is called a pseudo-reward because it's not a fixed function of s and a, but may change during the run of the algorithm.

Question 3 (10 points) One way to reduce high-fitting is to make the options simpler. In the next question, you'll implement options that try to move towards a single position $\mu_h = [x_h, y_h]$ in 2D space. Specifically, the action policy for option h, parametrized by μ_h, is:

$$\pi_{\mu_h}(a|s) \propto \exp(-d(s', \mu_h)), \tag{4}$$

where s' is the state that would have follow s when action a is taken if there were no walls, and $d(s_1, s_2) = \frac{1}{2}\|s_1 - s_2\|_2^2$. Recall that the option policy gradient in the Option–Critic algorithm is $\nabla_{\mu_h} \mathcal{L}_h(s, a) = -Q_h(s, a) \nabla_{\mu_h} \log \pi_{\mu_h}(a|s)$.

Write an expression for the loss gradient when the policy is given by (4).

Question 4 (25 points) In this question, you'll implement the option class in (4). Read the implementation of the current option policy class `utils.SoftmaxPolicy`. It parametrizes the policy with parameters $\theta_{s,a}$ such that the softmax policy is

$$\pi_\theta(a|s) \propto \exp \tau^{-1} \theta_{s,a},$$

where τ is a temperature hyperparameter. The class originally has the following methods:

- The method `Q_U` just returns the parameters, and is poorly named so don't get confused — it's not returning Q values at all.
- The method `pmf` takes the parameters and applies softmax to get $\pi_\theta(a|s)$ for all actions a in a given state s. Computing softmax can be numerically unstable if parameters become very large or very small, so notice how this function uses `logsumexp` to compute this in a numerically stable way.
- The method `sample` then samples an action for a given state.
- The method `update` takes an option policy gradient step over the parameters. Its argument `Q_h` is the same as what we called Q_h in lecture 16, and is provided to this method by the critic. Note that, in the original parameterization, $\mathcal{L}_\theta(s, a)$ for a given state s and action a depends only on $\theta_{s,a}$ for that action and $\theta_{s,\tilde{a}}$ for other actions. The gradient therefore only touches those parameters, and for them:

  $$\nabla_{\theta_{s,\tilde{a}}} \mathcal{L}_\theta(s, a) = Q_h(s, a) \nabla_{\theta_{s,\tilde{a}}} \log \sum_{\bar{a}} \exp \tau^{-1} \theta_{s,\bar{a}} = \tau^{-1} \pi(\tilde{a}|s) Q_h(s, a),$$

  and similarly

  $$\nabla_{\theta_{s,a}} \mathcal{L}_\theta(s, a) = -\tau^{-1}(1 - \pi(a|s)) Q_h(s, a).$$

  Note how the current code implements this update.

Based on this, implement the new option class, with the new parametrization μ_h. The code for the Option–Critic algorithm will only use methods `sample` and `update` of your class, but you can have any other methods that you find helpful. Some things to note:

- You can initialize the option policy parameters μ_h however you want.
- The `state` argument is an integer. To get the (x, y) position in the grid world, you can use `env.tocell` (see here: https://github.com/alversafa/option-critic-arch/blob/master/fourrooms.py#L42). It may help to pass the `env` to the policy object constructor.
- The `action` argument is also an integer. To get the direction in the grid world, you can use `env.directions`.
- The state that follows `env.tocell(state)` when taking action `env.directions(action)` is their sum (if it's not a wall, but for the purpose of the policy ignore walls).
- Remember to *descend*, rather than ascend, on the loss.

Replace the `option_policies` with your implementation.

Question 5 (10 points) Run your code. Compare the results with different numbers of options. Compare the results with the original code.