CS 277 (W26): Control and Reinforcement Learning
Exercise 5

Due date: Tuesday, March 17, 2026 (Pacific Time)

Roy Fox
https://royf.org/crs/CS277/W26

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)

Consider the following Actor–Critic Policy Gradient algorithm. It represents an actor π_θ and a critic V_ϕ, as well as a slowly-updating target network $V_{\bar{\phi}}$. Then with on-policy experience (s, a, r, s') it uses a target $y_{\bar{\phi}}(r, s') = r + \gamma V_{\bar{\phi}}(s')$ to compute the Temporal-Difference (TD) critic loss

$$\mathcal{L}_\phi = \tfrac{1}{2}(y_{\bar{\phi}}(r, s') - V_\phi(s))^2. \tag{1}$$

It also uses the critic's advantage estimate

$$A = y_{\bar{\phi}}(r, s') - V_\phi(s) \tag{2}$$

to compute the Policy-Gradient actor loss

$$\mathcal{L}_\theta = -A \log \pi_\theta(a|s). \tag{3}$$

The algorithm then descends the total loss

$$\mathcal{L}_{\text{AC}} = \mathcal{L}_\phi + \eta \mathcal{L}_\theta, \tag{4}$$

with η a coefficient relating the two losses.

Question 1.1 (5 points) When using (2) for (3), it would be more standard to use the non-target network ϕ for both terms. Explain why using ϕ rather than $\bar{\phi}$ would be a reasonable choice in (3), but less so in (1).

Question 1.2 (5 points) Point out another difference (in addition to the one in the previous question) between the above algorithm and the Actor–Critic PG algorithm (V version) that we saw in class. Point out one difference between the above algorithm and the Advantage Actor–Critic (A2C) algorithm that we saw in class.

Question 1.3 (5 points) The SQL algorithm represents an action-value function $Q_\beta(s,a)$ that induces the soft-greedy policy

$$\pi_\beta(a|s) = \frac{\pi_0(a|s)\exp \beta Q_\beta(s,a)}{\exp \beta V_\beta(s)}, \tag{5}$$

with the log-normalizer

$$V_\beta(s) = \tfrac{1}{\beta}\log \mathbb{E}_{(a|s)\sim\pi_0}[\exp \beta Q_\beta(s,a)]. \tag{6}$$

Rearranging (5), we get

$$Q_\beta(s,a) = V_\beta(s) + \tfrac{1}{\beta}\log\frac{\pi_\beta(a|s)}{\pi_0(a|s)}. \tag{7}$$

Consider implementing SQL with a value network $Q_{\theta,\phi} : S \to \mathbb{R}^A$ that has the structure in (7). Namely, the network has two heads, $\pi_\theta : S \to \Delta(A)$ and $V_\phi : S \to \mathbb{R}$, which are combined as in (7), with $\pi_\beta = \pi_\theta$ and π_0 a fixed prior, to compute $Q_{\theta,\phi}$. There is also a target network that keeps a slowly-updating copy $V_{\bar{\phi}}$ of V_ϕ, and the TD loss is now

$$\mathcal{L}_{\text{SQL}} = \tfrac{1}{2}(y_{\bar{\phi}}(r,s') - Q_{\theta,\phi}(s,a))^2, \tag{8}$$

with $y_{\bar{\phi}}(r,s') = r + \gamma V_{\bar{\phi}}(s')$.
What is the gradient of $\mathcal{L}_{\text{SQL}}$ with respect to θ? with respect to ϕ?

Question 1.4 (10 points) Consider (a) how the SQL loss (8) depends on the V-network $V_\phi(s)$ in that algorithm. Now consider (b) how the AC critic loss (1) depends on the V-network $V_\phi(s)$ in that algorithm, but with a different reward: instead of r, we'll use a pseudo-reward $\tilde{r}(s,a)$. Write an expression for $\tilde{r}$ such that (a) and (b) show the same dependence on their respective V-networks. Hint: $\tilde{r}$ is called a pseudo-reward because it's not a fixed function of s and a, but may additionally depend on quantities that change during the run of the algorithm.

Question 1.5 (10 points) Consider the θ gradient $\nabla_\theta \mathcal{L}_{\text{SQL}}$ of the SQL loss (8). Now consider the gradient $\nabla_\theta \mathcal{L}_\theta$ of the AC actor loss (3), but with the pseudo-reward $\tilde{r}$ from the previous question. Show that the expressions for these gradients have the same dependence, up to a constant, on their respective policy networks π_θ.

Question 1.6 (5 points) Under the equivalence in the previous two questions, what quantity in the above AC PG algorithm plays a role equivalent to that of β in SQL?

Part 2 Option–Critic (60 points)

Question 2.1 (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. Update `utils.py`: replace `from scipy.misc import logsumexp` with the following for compatibility:

```
try:
    from scipy.special import logsumexp
except ImportError:
    from scipy.misc import logsumexp
```

2. In parts 3 and 4, add color bars (see `matplotlib.pyplot.colorbar`) to the heat maps.

3. 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 2.2 (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.

Question 2.3 (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:

$$\pi_{\mu_h}(a|s) \propto \exp(-d(s', \mu_h)), \tag{9}$$

where s' is the state that <u>would have</u> followed s when action a is taken <u>if there were no walls</u>, 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 (9).

Question 2.4 (25 points) In this question, you'll implement the option class in (9). 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

$$\pi_\theta(a|s) \propto \exp \tau^{-1}\theta_{s,a}, \tag{10}$$

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_U` is the same as what we called Q_h, and is provided to this method by the critic. Note that, in the original parametrization, $\mathcal{L}_\theta(s,a)$ 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:

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

  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 (9). The code for the Option–Critic algorithm will only use the methods `sample` and `update` of your class, but you can use or add 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`). Taking the `env` object as argument when constructing the policy object may therefore help.
- 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 2.5 (10 points) Run your code. Compare the results with different numbers of options. Compare the results with the original code.
Due to the instability of the Option–Critic algorithm, you can repeat each experiment for up to 10 independent runs and report the best run. In this case, report how many runs you used and include a representative of unsuccessful runs.