CS 277 (W24): Control and Reinforcement Learning
Exercise 2

Due date: Monday, February 5, 2024 (Pacific Time)

Roy Fox
https://royf.org/crs/CS277/W24

In the following questions, a formal proof is not needed (unless specified otherwise). Instead, briefly explain informally the reasoning behind your answers.

Part 1 Relation between BC and PG (15 points)

Question 1.1 (5 points) Suppose that we want to imitate an unknown expert π^*, but we only have access to a dataset $\mathcal{D}$ of demonstrations provided by a <u>known</u> non-expert policy π_0. Suppose that the exploration policy π_0 supports the actions of π^*; i.e., $\pi_0(a|s) > 0$ for any (s, a) with $\pi^*(a|s) > 0$. Suppose that a teacher provides importance weights $\rho(\xi) = \frac{p_{\pi^*}(\xi)}{p_{\pi_0}(\xi)}$ for each $\xi \in \mathcal{D}$.
We would like to use Behavior Cloning (BC) with a Negative Log-Likelihood (NLL) loss to train a policy $\pi_\theta(a|s)$ on data $\xi \sim \mathcal{D}$ to imitate the expert π^*. What is the loss $\mathcal{L}_\theta^{\text{BC}}(\xi)$ on which we should descend?

Question 1.2 (5 points) Compare the answer to the previous question to the REINFORCE algorithm. How are these algorithms similar? How are they different?

Question 1.3 (5 points) Suppose that, instead of the importance weights $\rho(\xi)$, the teacher labels each trajectory ξ with its return $R(\xi)$. Let $J_\theta = \mathbb{E}_{\xi \sim p_\theta}[R(\xi)]$ be the RL objective for policy π_θ. Write a loss $\mathcal{L}_\theta^{\text{PG}}(\xi)$ whose gradient with respect to θ, on data $\xi \sim \mathcal{D}$, is an unbiased estimate of $\nabla_\theta J_\theta$. Note that only π_θ, π_0, ξ, and $R(\xi)$ are available.

Part 2 Advantage estimators (35 points)

You are playing an infinite sequence of Rock–Paper–Scissors rounds. After each round, you get a reward of 1, 0, or -1, respectively if you win, tie, or lose. Your opponent is a simple bot: it always tries to beat your previous action (e.g. to play Paper if you previously played Rock) with probability 40%, but selects the other two actions with probability 30% each; initially, it plays as if you've just played Paper, i.e. its most likely action is Scissors. Knowing this, you play optimally: Rock, then Scissors, then Paper, and repeat.

Question 2.1 (5 points) What are the states of this system?

Question 2.2 (5 points) For each state s and action a, what is the distribution of the reward r when taking action a in state s? What is its expectation and variance?

Question 2.3 **(5 points)** With discount factor $\gamma = 0.95$, what is the value $V^*(s)$ of the optimal policy in each state s?

Question 2.4 **(5 points)** What is the advantage $A^*(s, a) = Q^*(s, a) - V^*(s)$ of the optimal policy in each state s and action a?

Question 2.5 **(5 points)** Suppose that you estimate (perhaps incorrectly) that your expected future discounted return is $V(s) = 1$ for each state s (there is no variance in this estimate). You use that estimate in a Monte-Carlo advantage estimator

$$A^{MC}(s_0, a_0) = \sum_{t=0}^{\infty} \gamma^t r_t - V(s_0),$$

with on-policy experience $\xi = s_0, a_0, r_0, s_1, a_1, r_1, s_2, \ldots$. Recall that the bias is defined as the difference between the expected estimate and the true advantage. What are the bias and variance of this estimator?

Question 2.6 **(5 points)** Consider the n-step advantage estimator

$$A^n(s_0, a_0) = \sum_{t=0}^{n-1} \gamma^t r_t + \gamma^n V(s_n) - V(s_0),$$

with $n \geq 1$, and again with on-policy experience. What are the bias and variance of this estimator, as a function of n?

Question 2.7 **(5 points)** Consider the GAE(λ) advantage estimator

$$A^\lambda(s_0, a_0) = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} A^n(s_0, a_0) = \sum_{t=0}^{\infty} (\lambda\gamma)^t A^1(s_t, a_t),$$

with $\lambda \in [0, 1]$ and on-policy experience. What are the bias and variance of this estimator, as a function of λ?

Part 3 Model-Free Reinforcement Learning algorithms (50 points)

Question 3.1 **Policy Gradient (10 points)** Download the code at https://royf.org/crs/CS277/W24/CS277E2.zip.
In the function `update` in `reinforce.py`, write PyTorch code that computes the Policy Gradient loss. Hint: arithmetic operators work for PyTorch tensors, and PyTorch has build-in functions for NumPy-like operators, e.g. mean.
In the function `compute_returns_V1` in `reinforce.py`, write NumPy code that computes the return of the entire trajectory. The return will be the sum of rewards along the trajectory. You can discount the sum however you'd like, or not at all. Note that we need `returns` to be a 1-D NumPy

array of the same size as the rewards, i.e. the length of the trajectory. Create an array with the same return repeated in each element.

Train the REINFORCE agent with:

```
python run.py --env CartPole-v1\
              --training-steps 100000\
              --version 1
```

This will create a directory containing the agent's checkpoints. Then, evaluate the agent with:

```
python run.py --eval\
              --env CartPole-v1\
              --checkpoint <checkpoint_directory>
```

Append a printout of your code as a page in your PDF.

Question 3.2   Policy Gradient with Future Return (10 points)   We can reduce the variance of the gradient estimator by not taking into account past rewards. Complete `compute_returns_V2` in `reinforce.py` to sum (with or without discounting, but be consistent with what you did before) only future rewards in each step. Hint: the function `numpy.cumsum` can come in handy, but be careful how you use it.

Train the agent again, but with the new returns:

```
python run.py --env CartPole-v1\
              --training-steps 100000\
              --version 2
```

Evaluate the agent and append a printout of your code as a page in your PDF.

Question 3.3   Policy Gradient with Normalized Future Return (10 points)   We can stabilize training by normalizing the returns in each episode. Complete `compute_returns_V3` in `reinforce.py` to normalize the returns obtained in the previous question. Then train the agent again and evaluate as before.

```
python run.py --env CartPole-v1\
              --training-steps 100000\
              --version 3
```

Append a printout of your code as a page in your PDF.

Question 3.4   DQN (5 points)   Run `dqn.py` using the following command and view the results. You might first need to `pip install swig gymnasium[box2d] stable_baselines3`.

```
python dqn.py --env LunarLander-v2\
              --training-steps 1000000
```

Evaluate the agent with:

```
python dqn.py --env LunarLander-v2\
              --eval
```

If you are running on your local machine, you can pass `--human` to visually render the rollouts to see how the trained agent performs in the simulator. You can also download the trained agent checkpoint and evaluate locally.

`replay_data.dones` (plural of 'done') is a vector of booleans that, for each time step, indicates whether the next state (reached at the end of the step) terminated the episode.

Explain the role of `dones` in line 39.

Question 3.5   Double DQN (10 points)   As we will see in a later class, the Q-learning target tends to overestimate the Q value. Several methods have been proposed to mitigate this, including Double Q-learning and it's function-approximation counterpart, Double DQN.

Complete the function `train` such that, if `self.double_dqn` is `True`, the loss will be the Double DQN loss:

$$\mathcal{L}_\theta(s, a, r, s') = (r + \gamma Q_{\bar{\theta}}(s', \underset{a'}{\mathrm{argmax}}\, Q_\theta(s', a')) - Q_\theta(s, a))^2,$$

where $\bar{\theta}$ are the parameters of the target network.

Train the agent with:

```
python dqn.py --env LunarLander-v2\
              --training-steps 1000000\
              --double-dqn
```

and evaluate as in the previous question.

Append a printout of your code as a page in your PDF.

Question 3.6   Visualize results (5 points)   We can use TensorBoard to visualize the training results of `stable_baselines3`.

Run a TensorBoard web server:

```
tensorboard --logdir <the result parent logdir from the DQN results>
```

Take note of the URL in which TensorBoard is now serving (likely http://localhost:6006/). Open a browser at that URL. Take some time to make yourself familiar with the TensorBoard interface.

You should be able to see all the stable_baseline3 runs on the bottom left, with a color legend. If you happened to execute more runs than the ones detailed in previous questions, uncheck all the other runs.

Find the plot tagged `rollout/ep_rew_mean`. You can find it manually, or use the "Filter tags" box at the top. Enlarge the plot using the left of 3 buttons at the bottom.

On the left you'll find some useful options. Uncheck "Ignore outliers in chart scaling" and note the effect on the plot. Check "Show data download links", download the plot, and append it to your PDF.

Question 3.7 Extra fun For extra fun, repeat the above experiments with other (discrete action space) environments from OpenAI Gym (https://gym.openai.com/envs), such as `Acrobot-v1`, `LunarLander-v2`, `Pong-v4`, and `Breakout-v4`. There's no extra credit, because getting good results will likely take more `--steps` and time than I should ask you to run.