CS 277: Control and Reinforcement Learning (Winter 2021)
Assignment 2

Due date: Friday, January 29, 2021 (Pacific Time)

Roy Fox
https://royf.org/crs/W21/CS277/

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 (25 points)

Consider a dataset $\mathcal{D}$ of trajectories, some of which are expert teacher demonstrations and some random policy rollouts. Suppose each trajectory ξ_i is flagged with a bit b_i indicating whether it was generated by the teacher ($b_i = 1$) or the random policy ($b_i = 0$).

1. First, we'd like to learn a policy by imitating the expert with a cross-entropy loss. Recall that, in supervised learning, the cross-entropy loss (also called negative log-likelihood) for an instance x and its true label y is $\mathcal{L}_\theta(x, y) = -\log p_\theta(y|x)$. What is the Behavior Cloning loss for dataset $\mathcal{D}$? (10 points)
2. Now suppose we had a teacher label each trajectory ξ_i with its return $R_i = R(\xi_i)$. What is the Policy Gradient loss (as in the REINFORCE algorithm) for dataset $\mathcal{D}$? (10 points)
3. What intuition can you draw from this? (5 points)

Part 2 Model-Free Value Representation (25 points)

In model-based Value Iteration we have a recursion for the optimal state–value function $V(s)$:

$$V(s) \leftarrow \max_a \mathbb{E}[r + \gamma V(s')|s, a].$$

In Fitted Value Iteration, we optimize a parametric model $V_\theta(s)$, in the hope of approximating the optimal state–value function. We minimize the square error in the above recursion, with a stabilized target network $V_{\bar\theta}$ updated slowly from V_θ:

$$\min_\theta (\max_a \mathbb{E}[r + \gamma V_{\bar\theta}(s')|s, a] - V_\theta(s))^2.$$

Once we wanted to be model-free, i.e. not assume knowledge of the dynamics $p(s'|s, a)$ (nor perhaps the reward $r(s, a)$), we switched to the state–action value function $Q(s, a)$, instead of $V(s)$.

Why is it impossible to have a model-free reinforcement learning algorithm that represents only $V(s)$, and nothing else, as its solution? (Hint: is such a solution useful?)

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

1 Policy Gradient (5 points)

Download the code at https://royf.org/crs/W21/CS277/A2/pg.py.

In the function `policy_gradient_loss`, write TensorFlow code that computes the Policy Gradient loss. (Hint: arithmetic operators work for TF tensors, and TF has build-in functions for NumPy-like operators, e.g. reduce_sum.)

In the function `calculate_returns`, write NumPy code that computes the return of steps in `sample_batch`. `sample_batch` is guaranteed to represent part of a single trajectory, and in this assignment we'll assume it's 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 `returns` is expected to be a 1-D NumPy array of the same size as the rewards, i.e. the length of the trajectory. Repeat the same total episode return for the entire array.

Run `pg.py`. Take note of the "Result logdir" that RLlib prints after each evaluation. When running the algorithm, you can specify a different logdir, or just use the default like we did here.

Behold your creation:

```
rllib rollout <logdir>/<checkpoint_num>/<checkpoint-num> --run PG --env CartPole-v1
  --steps 2000
```

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

2 Policy Gradient with Future Return (15 points)

We can reduce the variance of the gradient estimator by not taking into account past rewards. Copy `pg.py` as `pg2.py`, and change `calculate_returns` to sum (with or without discounting, but be consistent with what you did before) only future rewards in each step. (Hint: the functions `numpy.cumsum` and `ray.rllib.evaluation.postprocessing.discount_cumsum` can come in handy, but be careful how you use them.)

Tip: Don't forget to change the `name` of the `PG_Trainer`.

Run `pg2.py` and view the results.

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

3 DQN (10 points)

Download the code at `https://royf.org/crs/W21/CS277/A2/dqn.py`. Run it and view the results.

```
rllib rollout <DQN logdir>/<checkpoint_num>/<checkpoint-num> --run DQN --env CartPole-v1
   --config '{"dueling": false}' --steps 2000
```

`dones` 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 43.

4 Double DQN (15 points)

Fix `dqn.py` such that, if `policy.config["double_q"]` is `True`, the loss will be the Double DQN loss:

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

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

Change `policy.config["double_q"]` to `True`, run your code, and view the results.

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

5 Visualize results (5 points)

TF comes with a utility for visualizing training results, called TensorBoard.

Run a TensorBoard web server:

```
tensorboard --logdir <the result logdir from the previous sections>
```

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 RLlib runs on the bottom left, with a color legend. If you happened to execute more runs than the four detailed above, uncheck all the other runs.

Find the plot tagged "tune/episode_reward_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.

Unfortunately, there's currently no good way to save the plot as an image, so just take a screenshot, and include it as a page in your PDF.

6 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 can require.