CS 295: Optimal Control and Reinforcement Learning

Winter 2020

Assignment 2

due Tuesday, February 4 2020, 11pm

Part I

1. 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 ($b_i = 0$).

   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 for an instance x and its true label y is $-\log p_\theta(y|x)$. What is the imitation learning loss for dataset $\mathcal{D}$?

   Now suppose we had a teacher label each trajectory with its return R_i. What is the policy gradient loss for dataset $\mathcal{D}$?

   What intuition can you draw from this?

2. In model-based Value Iteration we had a recursion for the state value function $V(s)$

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

   We could also optimize a parametric model, in the hope of approximating the state value function

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

   This method is called *Fitted Value Iteration.*

   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 $Q(s, a)$.

   Why can't we represent only V_θ in a model-free algorithm? (Hint: think about what the algorithm needs to output when it's done.)

Part II

In this part, you will install a Deep RL framework, RLlib (`https://ray.readthedocs.io/en/latest/rllib.html`), and use it to implement a couple of simple Deep RL algorithm, execute them, and evaluate the results. The OS we use in this assignment is Linux / MacOS. If you're using Windows or another OS, please use a local or remote virtual machine.

1 Install dependencies

Make sure you have a recent version of Python installed, such as the latest Python 3.7; but **not Python 3.8**, as RLlib doesn't seem to support it at this time. It is recommended that you create a new `conda` environment for this assignment, instructions here: `https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html`.

RLlib can use any Deep Learning library. Most of the algorithms already implemented in RLlib support TensorFlow (TF), and some support PyTorch. In this assignment we will use TF for CPU. It is also possible to install TF for GPU, and this would allow much faster execution, but is not needed in this assignment.

Install TF:

```
pip install tensorflow
```

2 Install RLlib

RLlib is implemented on top of the Ray distributed execution library.

Install RLlib:

```
pip install ray[rllib]
```

3 Policy Gradient

Download the code at `https://royf.org/crs/W20/CS295/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. `https://www.tensorflow.org/api_docs/python/tf/math/reduce_sum#for_example`.)

In the function `calculate_returns`, write NumPy code that computes the return of `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 (see the previous function for how to extract rewards). 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.

4 Policy Gradient with Future Return

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` 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`.

5 DQN

Create a new trainer (using `build_trainer` as before) that uses the `SimpleQPolicy` (`https://github.com/ray-project/ray/blob/master/rllib/agents/dqn/simple_q_policy.py`).

Run it.

6 Double DQN

Now copy `SimpleQPolicy` to a new file, and fix the loss to 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. Note that the current code actually uses, instead of the square loss, the more commonly used Huber loss

$$\ell(e) = \begin{cases} \frac{1}{2}e^2 & |e| \leqslant 1 \\ |e| - \frac{1}{2} & |e| \geqslant 1. \end{cases}$$

You can either keep the Huber loss or switch to the square loss.

Run your code.

7 Visualize results

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 in your submission.

8 Analyze results

In your submission, include short answers to the following questions:

1. Which algorithms seem to perform better than others according to the plot you got? Try to explain why this is so, at least partially.

2. You may notice some jitter in the curves, which is likely the result of the stochasticity of the environment and of the algorithms. How could your analysis be more robust to these fluctuations and better compare the average performance of the algorithms?