CS 234: Assignment #3

Due date: 2/23 11:00 PM PST

These questions require thought, but do not require long answers. Please be as concise as possible.

We encourage students to discuss in groups for assignments. **However, each student must finish the problem set and programming assignment individually, and must turn in her/his assignment.** We ask that you abide by the university Honor Code and that of the Computer Science department, and make sure that all of your submitted work is done by yourself. If you have discussed the problems with others, please include a statement saying who you discussed problems with. Failure to follow these instructions will be reported to the Office of Community Standards. We reserve the right to run a fraud-detection software on your code.

Please review any additional instructions posted on the assignment page at http://cs234.stanford.edu/assignment3. When you are ready to submit, please follow the instructions on the course website.

1 Policy Gradient Methods (45pts coding + 20pts writeup)

The goal of this problem is to experiment with policy gradient and its variants, including variance reduction methods. Your goals will be to set up policy gradient for both continuous and discrete environments, and implement a neural network baseline for variance reduction. The framework for the vanilla policy gradient algorithm is setup in `pg.py`, and everything that you need to implement is in this file. The file has detailed instructions for each implementation task, but an overview of key steps in the algorithm is provided here.

1.1 REINFORCE

Recall the vanilla policy-gradient theorem,

$$\nabla_\theta V(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)\right]$$

REINFORCE is a monte-carlo policy gradient algorithm, so we will be using the sampled returns G_t as unbiased estimates of $Q^{\pi_\theta}(s,a)$. Then the gradient update can be expressed as maximize the following reward function:

$$V(\theta) = \frac{1}{D}\sum_{\tau \in D}\sum_{t}^{T} \log(\pi_\theta(s_t,a_t))G_t$$

where D is the set of all trajectories collected by policy π_θ, and $\tau = (s_0, a_0, r_0, s_1 ...)$ is a trajectory.

1.2 Baseline

One difficulty of training with the REINFORCE algorithm is that the monte-carlo estimated return G_t can have high variance. To reduce variance, we subtract a baseline $b(s)$ from the estimated returns when computing the policy gradient. A good baseline is the state value function, $b(s) = V^{\pi_\theta}(s)$, which requires a training update to minimize the mean-squared error loss:

$$V(b) = \frac{1}{D}\sum_{\tau \in D}\sum_{t}^{T} (b(s_t) - G_t)^2$$

1.3 Advantage Normalization

A second variance reduction technique is to normalize the computed advantages so that they have mean 0 and standard deviation 1. From a theoretical perspective, we can consider centering the advantages to be simply adjusting the advantages by a constant baseline, which does not change the policy gradient. Likewise, rescaling the advantages effectively changes the learning rate by a factor of $1/\sigma$, where σ is the standard deviation of the empirical advantages.

1.4 Writeup Questions

(a) (8 pts) (CartPole-v0) Test your implementation on the CartPole-v0 environment with the given configuration parameters (in `config.py`) by running

```
python pg.py
```

With the given configuration file, this should achieve a perfect score of 200 within the 100 iterations. *NOTE: training may repeatedly converge to 200 and diverge. Your plot does not have to reach 200 and stay there. We only require that you achieve a perfect score of 200 sometime during training.* Include in your writeup the tensorboard plot for the average reward. Start tensorboard with:

```
tensorboard --logdir=results/CartPole-v0
```

and then navigate to the link it gives you. Click on the "SCALARS" tab to view the average reward graph.

Now, modify the configuration file to try running with larger and smaller batch sizes. What happens as you increase/decrease the batch size, and why? Note that as you perform multiple runs, tensorboard will generate multiple result files, so you should either remove previous run's results, or change the output path for each run in the configuration file. Try running without the baseline, and include the plot. Do you notice any difference? Explain.

(b) (5 pts) (InvertedPendulum-v1) Test your implementation on the InvertedPendulum-v1 environment by modifying `config.py`. To use this environment (and HalfCheetah-v1), you will have to install MuJoCo and mujoco-py. Instructions for installation are in the README.txt file in the starter code. We have ran into errors installing MuJoCo on Windows, so Windows users may instead need to use the FarmShare servers.

Find configuration parameters so that training reaches the maximum episode reward of 1000 within 100 iterations. *Again, training will probably reach 1000, and then repeatedly diverge and converge. We only require that you reach 1000 sometime during trainig.* Include the tensorboard plot in your writeup.

(c) (7 pts) (HalfCheetah-v1) Test your implementation on the much more difficult HalfCheetah-v1 environment, with $\gamma = 0.9$ so that training reaches an average episode reward of more than 200 within 100 iterations. Include the tensorboard plot, and explain your parameter choices you make in the writeup. Hint: a large batch size of 50000 works well for us, and it may help to change the size of the policy network.

2 Expected Regret Bounds (35pts)

Assume a reinforcement learning algorithm A for discounted infinite-horizon MDPs has expected regret

$$\mathbb{E}_* \left[\sum_{t=1}^{T} r_t \right] - \mathbb{E}_A \left[\sum_{t=1}^{T} r_t \right] = f(T)$$

for all $T > 0$, where $\mathbb{E}_*$ is over the probability distribution with respect to the optimal policy π_* and $\mathbb{E}_A$ is the expectation with respect to the algorithm's behavior. We assume that $\gamma \in [0, 1)$ is the discount factor and that rewards are normalized, i.e., $r_t \in [0, 1]$.

(a) (15 pts) Let π be an arbitrary policy or algorithm. Show that for any $\epsilon' > 0$ and $T' \geq \log_{\frac{1}{\gamma}} \frac{H}{\epsilon'}$ where $H = 1/(1-\gamma)$, we have

$$\left| V_\pi(s) - \sum_{t=1}^{T'} \gamma^{t-1} \mathbb{E}_\pi[r_t | s_1 = s] \right| \leq \epsilon' \,, \text{ for all state } s.$$

Note V_π is the value function associated with π and $\mathbb{E}_\pi$ is the expectation with respect to the randomization of π.

(b) (20pts) From the regret guarantee of algorithm A and Part a), it follows that for any $\epsilon' > 0$ and $T' \geq \log_{\frac{1}{\gamma}} \frac{H}{\epsilon'}$, we have

$$\mathbb{E}_*[V_*(s_{T+1})] - \mathbb{E}_A[V_A(s_{T+1})] \leq f(T' + T) - f(T) + 2\epsilon' \,, \text{ for } T > 0,$$

where V_A is the value function of the (possibly nonstationary) policy that algorithm A follows. Assume now $f(T) = \sqrt{T}$. Show that for any $\epsilon > 0$ and $t \geq 1 + \frac{1}{\epsilon^2} \left(\log_{\frac{1}{\gamma}} \frac{4H}{\epsilon} \right)^2$, we have

$$\mathbb{E}_*[V_*(s_t)] - \mathbb{E}_A[V_A(s_t)] \leq \epsilon \,.$$

Hint: It may be helpful to set ϵ' to be some function of ϵ and choose an appropriate value of T'.