CS 234: Assignment #3
Winter 2019

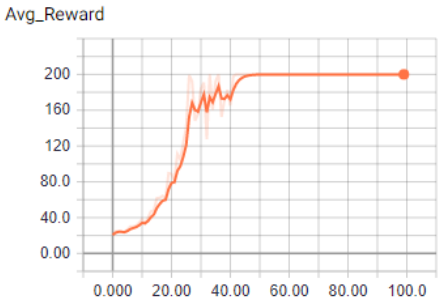
Li Quan Khoo (SCPD)

1 Policy Gradient Methods (50 pts + 15 pts writeup)

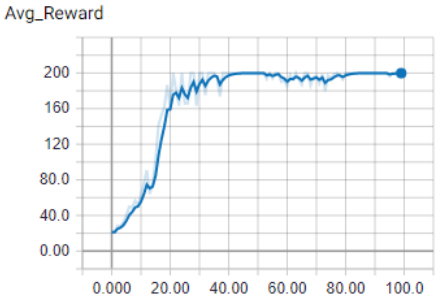
- (code)

1.1 Writeup Questions (15 pts)

(a) (4 pts) (CartPole-v0)



(a) CartPole-v0 with baseline. Average reward over 200 runs.



(b) CartPole-v0 without baseline. Average reward over 200 runs.

Where

$$r(\tau) = \sum_t r(s_t, a_t), \quad and \quad J(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)}[r(\tau)],$$

We have derived the Monte-Carlo estimate of the gradient of $J(\theta)$ for policy gradient to be:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta log \pi_\theta(\tau)[r(\tau) - b]$$

Where b is some constant scalar bias of the reward function. Mathematically, the baseline is just another bias, and we already know that this gradient $\nabla_\theta J(\theta)$ is indifferent to the bias *in expectation*. However, the actual size and value of the bias still influences convergence (consider a bias close to +/- infinity) due to how the probability mass of the policy is shifted when we perform the gradient update, as well as the variance of the quantity $r(\tau) - b$. (Berkeley CS294, lecture 4)

Here, the baseline we are taking is the state value function $V(s_t)$, which represents the average of returns of all actions from that state. Mathematically, $V(s_t)$ already summarizes all future returns, and the advantage $A(s_t, a_t) = r(s_t, a_t) - V(s_t)$ becomes independent of returns of all future time steps *in the current trajectory*. Mathematically, this lowers the variance of our estimated action values. Semantically, we are shifting the probability mass of the policy to make actions which are *better than average* (our chosen baseline) to be more likely.

Empirically, we can see from the plots above that using the baseline gives "better" convergence. Our observation here is that the plateau is more stable.

(b) (4 pts) (InvertedPendulum-v1) Test your implementation on the InvertedPendulum-v1 environment by running

```
python pg.py --env_name pendulum --baseline
```

With the given configuration file `config.py`, the average reward should reach 1000 within 100 iterations. *NOTE: Again, we only require that you reach 1000 sometime during training.*

Include the tensorboard plot for the average reward in your writeup.

Now, test your implementation on the InvertedPendulum-v1 environment without baseline by running

```
python pg.py --env_name pendulum --no-baseline
```

Include the tensorboard plot for the average reward. Do you notice any difference? Explain.

(c) (7 pts) (HalfCheetah-v1) Test your implementation on the HalfCheetah-v1 environment with $\gamma = 0.9$ by running

```
python pg.py --env_name cheetah --baseline
```

With the given configuration file `config.py`, the average reward should reach 200 within 100 iterations. *NOTE: There is some variance in training. You can run multiple times and report the best results or average. We have provided our results (average reward) averaged over 6 different random seed in figure 2* Include the tensorboard plot for the average reward in your writeup.

Now, test your implementation on the HalfCheetah-v1 environment without baseline by running

```
python pg.py --env_name cheetah --no-baseline
```

Include the tensorboard plot for the average reward. Do you notice any difference? Explain.

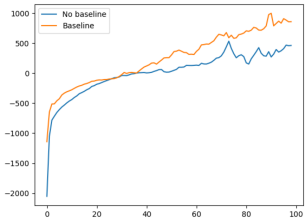


Figure 2: Half Cheetah, averaged over 6 runs

2 Best Arm Identification in Multiarmed Bandit (35pts)

In this problem we focus on the Bandit setting with rewards bounded in $[0, 1]$. A Bandit problem instance is defined as an MDP with just one state and action set $\mathcal{A}$. Since there is only one state, a "policy" consists of the choice of a single action: there are exactly $A = |\mathcal{A}|$ different deterministic policies. Your goal is to design a simple algorithm to identify a near-optimal arm with high probability.
Imagine we have n samples of a random variable x, $\{x_1, \ldots, x_n\}$. We recall Hoeffding's inequality below, where $\overline{x}$ is the expected value of a random variable x, $\widehat{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the sample mean (under the assumption that the random variables are in the interval [0,1]), n is the number of samples and $\delta > 0$ is a scalar:

$$\Pr\left(|\widehat{x} - \overline{x}| > \sqrt{\frac{\log(2/\delta)}{2n}}\right) < \delta.$$

Assuming that the rewards are bounded in $[0, 1]$, we propose this simple strategy: allocate an identical number of samples $n_1 = n_2 = ... = n_A = n_{des}$ to every action, compute the average reward (empirical payout) of each arm $\widehat{r}_{a_1}, \ldots, \widehat{r}_{a_A}$ and return the action with the highest empirical payout $\arg\max_a \widehat{r}_a$. The purpose of this exercise is to study the number of samples required to output an arm that is at least ϵ-optimal with high probability. Intuitively, as n_{des} increases the empirical payout $\widehat{r}_a$ converges to its expected value $\overline{r}_a$ for every action a, and so choosing the arm with the highest empirical payout $\widehat{r}_a$ corresponds to approximately choosing the arm with the highest expected payout $\overline{r}_a$.

(a) (15 pts) We start by defining a *good event*. Under this *good event*, the empirical payout of each arm is not too far from its expected value. Starting from Hoeffding inequality with n_{des} samples allocated to every action show that:

$$\Pr\left(\exists a \in \mathcal{A} \quad s.t. \quad |\widehat{r}_a - \overline{r}_a| > \sqrt{\frac{\log(2/\delta)}{2n_{des}}}\right) < A\delta.$$

In other words, the *bad event* is that at least one arm has an empirical mean that differs significantly from its expected value and this has probability at most $A\delta$.

(b) (20 pts) After pulling each arm (action) n_{des} times our algorithm returns the arm with the highest empirical payout:

$$a^{\dagger} = argmax_a \widehat{r}_a$$

Notice that $a^{\dagger}$ is a random variable. Define $a^{\star}$ as the optimal arm (that yields the highest average reward $a^{\star} = argmax_a \overline{r}_a$). Suppose that we want our algorithm to return at least an ϵ optimal arm with probability $1 - \delta'$, as follows:

$$\Pr\left(\overline{r}_{a^{\dagger}} \geq \overline{r}_{a^{\star}} - \epsilon\right) \geq 1 - \delta'.$$

How many samples are needed to ensure this? Express your result as a function of the number of actions A, the required precision ϵ and the failure probability δ'.