Homework #3

CSC2547 – Introduction to Reinforcement Learning
(Spring 2021)

- **Deadline:** Monday April 19, 2021 at **16:59**.
- **Submission:** You need to submit two files through MarkUs. One is a PDF file including all your answers and plots. The other is a source file that reproduces your answers. You can produce the file however you like (e.g. LaTeX, Microsoft Word, etc) as long as it is readable. Points will be deducted if we have a hard time reading your solutions or understanding the structure of your code. If the code does not run, you may lose most/all of your points for that question.
- **Late Submission:** 10% of the marks will be deducted for each day late, up to a maximum of 3 days. After that, no submissions will be accepted.
- **Collaboration:** Homework assignments must be done individually, and you cannot collaborate with others.

Gradient-based policy search

The goal of this question is to get familiar with the policy gradient methods. In this question you will experiment with two policy gradient methods, REINFORCE [Williams, 1992] and Advantage Actor Critic (A2C) [Mnih et al., 2016]. You try them on two types of environments: one with discrete action space and another with continuous action space.

The framework for the these methods is setup in `main.py` and everything that you need to implement are in files `network.py`, `BaseAgent.py` `REINFORCE_agent.py`, and `A2C_agent.py`. Each file has detailed instructions for each implementation task, but an overview of the key steps in the algorithm is provided here.

REINFORCE

Recall that the reinforcement learning objective is to find a policy $\pi : \mathcal{X} \times \mathcal{A} \to [0, 1]$, that maximizes the expected return $J(\pi) = \mathbb{E}_{\tau \sim \pi_{\theta(\tau)}}[r(\tau)]$ where each rollout τ is of length T with probability $\pi_\theta(\tau)$[1]

$$\pi_\theta(\tau) = p(x_0, a_0, ..., x_{T-1}, a_{T-1}) = p(x_0)\pi_\theta(a_0|x_0) \prod_{t=1}^{T-1} p(x_t|x_{t-1}, a_{t-1})\pi_\theta(a_t|x_t)$$

and the return

$$r(\tau) = r(x_0, a_0, ..., x_{T-1}, a_{T-1}) = \sum_{t=0}^{T-1} \gamma^t r(x_t, a_t).$$

One way to address this problem is to directly optimize the expected return $J(\pi_\theta)$ by performing stochastic gradient ascent on the parameters θ of a family of policies, π_θ. The policy gradient approach is to directly take the gradient of this objective:

$$\nabla_\theta J(\theta) = \nabla_\theta \int \pi_\theta(\tau) r(\tau) d\tau \tag{1}$$

$$= \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(\tau) r(\tau)\right]. \tag{2}$$

[1] The notation in the homework is slightly different from the lectures. For example, we often used G as return. This difference gives you the opportunity to learn about other commonly used notations.

In practice, the expectation over trajectories τ can be approximated from a batch of N sampled trajectories:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau_i) r(\tau_i) \tag{3}$$

$$= \frac{1}{N}\sum_{i=1}^{N} \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{i,t}|x_{i,t})\right)\left(\sum_{t=0}^{T-1} \gamma^t r(x_{i,t}, a_{i,t})\right). \tag{4}$$

Notice that in this approximation, we multiply the episode's return to the gradient of $\log \pi_\theta$. One way to reduce the variance of this approximation is to notice that the policy at time t cannot affect rewards in the past. This yields the following modified objective, where the sum of rewards here does not include the rewards achieved prior to the time step at which the policy is being queried. This sum of rewards is a sample estimate of the Q function, and is referred to as the reward-to-go.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{i,t}|x_{i,t}) \left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r(x_{i,t'}, a_{i,t'})\right). \tag{5}$$

As you might have noticed, REINFORCE relies on an estimated return by Monte-Carlo methods using episode samples to update the policy. The policy gradient theorem [Sutton et al., 1999] generalizes this result giving the policy gradient of the form:

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_{x\in\mathcal{X}} d^{\pi_\theta}(x) \sum_{a\in\mathcal{A}} \nabla_\theta \pi_\theta(a|x) Q^{\pi_\theta}(x,a) \\ &= \mathbb{E}_{x\sim d^{\pi_\theta}, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|x) Q^{\pi_\theta}(x,a)\right],\end{aligned} \tag{6}$$

where $d^{\pi_\theta}(x)$ is a discounted weighting of states encountered starting at $x_0 \sim p(x_0)$ and then following policy π: $d^{\pi_\theta}(x) = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}\{x_t = x | x_0, \pi_\theta\}$.

Advantage Actor Critic

Actor-Critic methods use bootstapping to find a better estimate of the action-value function and use that estimated action-value function $Q_w^{\pi_\theta}$ to find the gradient of expected return in the policy gradient theorem:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a|x) Q_w^{\pi_\theta}(x,a)\right]. \tag{7}$$

The policy gradient theorem can be generalized to include a comparison of the action-value to an arbitrary baseline $b(x)$:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a|x)(Q_w^{\pi_\theta}(x,a) - b(x)\right]. \tag{8}$$

An intuitive choice for baseline is the state-value function V^{π_θ}. This new critic is called Advantage function and is defined as: $A^{\pi_\theta}(x,a) = Q^{\pi_\theta}(x,a) - V^{\pi_\theta}(x)$. When using Advantage function as the critic in the Actor-Critic method it is called Advantage Actor Critic (A2C). Algorithm 1 from Sutton and Barto [2018] shows the complete pseudocode for A2C algorithm when using one step bootstraping for learning state-value function.

Algorithm 1: One-step Actor-Critic for estimating $\pi_\theta \approx \pi^*$

Input: a differentiable policy parameterization $\pi(a|x,\theta)$
Input: a differentiable state-value function parameterization $\hat{v}(x,w)$
Algorithm parameter: step size $\alpha^\theta > 0$, $\alpha^w > 0$
Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$
for *each episode* **do**
    Initialize X (first state of episode)
    $I \leftarrow 1$
    for *each time step* t **do**
        $A \sim \pi(\cdot|X,\theta)$
        Take action A, observe X', R
        $\delta \leftarrow R + \gamma\hat{v}(X',w) - \hat{v}(X,w)$ (if X' is terminal, then $\hat{v}(X',w) \doteq 0$)
        $w \leftarrow w + \alpha^w \delta \nabla_w \hat{v}(X,w)$
        $\theta \leftarrow \theta + \alpha^\theta I \delta \nabla_\theta \log \pi(A|X,\theta)$
        $I \leftarrow \gamma I$
        $X \leftarrow X'$
    end
end

1 Writeup Questions (40 pts)

(a) **[10pts]** To compute the REINFORCE estimator, you will need to calculate the values $(G_t)_{t=1}^T$, where

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}.$$

Computing all these values naively takes $O(T^2)$ time. Describe how to compute them in $O(T)$ time.

(b) **[15pts]** Assuming that our gradient estimate will be:

$$\widehat{\nabla_\theta J(\theta)} = (Q^{\pi_\theta}(x, a) - b(x)) \nabla_\theta \log \pi_\theta(a|x) \tag{9}$$

where $x \sim d^{\pi_\theta}(x)$ and $a \sim \pi_\theta(\cdot|x)$.

1. **[5pts]** Prove this estimate is unbiased estimate of the true gradient.
2. **[10pts]** Find $b(x)$ that leads to the minimum variance estimate of the true gradient.

(c) **[5pts]** Prove that $\delta_t(x_t, a_t) = r_t + \gamma \hat{V}(x_{t+1}) - \hat{V}(x_t)$ is an unbiased estimate of $A^\pi(x_t, a_t)$ when $\hat{V} = V^\pi$

(d) **[10pts]** Why do we have $I \leftarrow \gamma I$ step in the Algorithm 1? Most policy gradient methods drop the discount factor (as well as the methods we implement here). What happens if we remove it?

2 Coding Questions (60 pts)

The functions that you need to implement in `network.py`, `REINFORCE_agent.py`, `A2C_agent.py` and `BaseAgent.py` are enumerated here. Detailed instructions for each function can be found in the comments in each of these files.

(a) [**30pts**] We will implement a simple version of REINFORCE using equation 5 to approximate the gradient of policy for the both case of continuous and discrete action spaces.

1. [**20pts**] Implement the following functions in the provided code:
   - In `component/network.py`: `GaussianPolicyNet.forward()`, `CategoricalPolicyNet.forward()`
   - In `agent/REINFORCE_agent.py`: `step()`
   - In `agent/BaseAgent.py`: `eval_episode()`
2. [**10pts**] Report training return, policy loss and the final policy's evaluation return on `CartPole-v0` and `Pendulum-v0` (you can set the environment in `main.py`). The hyperparameters given in `main.py` are not tuned to be optimal and you may experiment with these to get better performance. Test your code with multiple random seeds to make sure the performance is consistent (you can set the random seed using `set_seed` in `main.py`).

(b) [**30pts**] Algorithm 1 shows the Pseudocode code for one-step Advantage Actor-Critic. In this question we will implement the extension of this algorithm when using **n-step** bootstraping.

1. [**20pts**] Implement the following functions in the provided code:
   - In `component/network.py`: `GaussianActorCriticNet.forward()`, `CategoricalActorCriticNet.forward()`
   - In `agent/A2C_agent.py`: `step()`
2. [**10pts**] Report training return, policy loss, value loss and the final policy's evaluation return on `CartPole-v0` and `Pendulum-v0`. The hyperparameters given in `main.py` are not tuned to be optimal and you may experiment with these to get better performance. Test your code with multiple random seeds to make sure the performance is consistent across them. Do you notice a difference between REINFORCE and A2C?

References

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org, 2016. 2

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. 4

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 1057–1063. The MIT Press, 1999. 3

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256, 1992. doi: 10.1007/BF00992696. 2