Exercise Set 2: Policy Gradient & Actor Critic Algorithm

Alexandre Carlhammar & Theo Le Fur

March 23, 2024

1 Introduction

The goal of this assignment is to experiment with policy gradient and its variants, including variance reduction tricks such as implementing reward-to-go and neural network baselines.

2 Policy Gradient

Recall that the reinforcement learning objective is to learn a parameter θ^* that maximizes the objective function:

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[r(\tau)\right] \tag{1}$$

where each rollout τ is of length T, as follows:

$$\pi_\theta(\tau) = p(s_0, a_0, \ldots, s_{T-1}, a_{T-1}) = p(s_0)\prod_{t=1}^{T-1} p(s_t|s_{t-1}, a_{t-1})\pi_\theta(a_t|s_t) \tag{2}$$

and the reward for the trajectory τ is given by:

$$r(\tau) = r(s_0, a_0, \ldots, s_{T-1}, a_{T-1}) = \sum_{t=0}^{T-1} r(s_t, a_t). \tag{3}$$

The policy gradient approach is to directly take the gradient of this objective:

$$\nabla_\theta J(\theta) = \nabla_\theta \int \pi_\theta(\tau) r(\tau)\, d\tau \tag{4}$$

$$= \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) r(\tau)\, d\tau \tag{5}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\nabla_\theta \log \pi_\theta(\tau) r(\tau)\right] \tag{6}$$

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

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau_i) r(\tau_i) \tag{7}$$

$$= \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_{it}|s_{it})\right)\left(\sum_{t=0}^{T-1} r(s_{it}, a_{it})\right). \tag{8}$$

Here we see that the policy π_θ is a probability distribution over the action space, conditioned on the state. In the agent-environment loop, the agent samples an action a_t from $\pi_\theta(\cdot|s_t)$ and the environment responds with a reward $r(s_t, a_t)$.

3 Variance Reduction

3.1 Reward-to-go

One way to reduce the variance of the policy gradient is to exploit causality: the notion that the policy 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."

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_{it}|s_{it})\right)\left(\sum_{t'=t}^{T-1} r(s_{it'}, a_{it'})\right). \tag{9}$$

3.2 Discounting

Multiplying a discount factor γ to the rewards can be interpreted as encouraging the agent to focus more on the rewards that are closer in time, and less on the rewards that are further in the future. This can also be thought of as a means for reducing variance (because there is more variance possible when considering futures that are further into the future). We saw in lecture that the discount factor can be incorporated in two ways, as shown below.

The first way applies the discount on the rewards from full trajectory:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_{it}|s_{it})\right)\left(\sum_{t'=0}^{T-1} \gamma^{t'} r(s_{it'}, a_{it'})\right). \tag{10}$$

And the second way applies the discount on the "reward-to-go:"

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_{it}|s_{it})\right)\left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r(s_{it'}, a_{it'})\right). \tag{11}$$

3.3 Baseline

Another variance reduction method is to subtract a baseline (that is a constant with respect to τ) from the sum of rewards:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[r(\tau) - b\right]. \tag{12}$$

This leaves the policy gradient unbiased because

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[b] = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\nabla_\theta \log \pi_\theta(\tau)\cdot b\right] = 0. \tag{13}$$

In this assignment, we will implement a value function V_ϕ^π which acts as a state-dependent baseline. This value function will be trained to approximate the sum of future rewards starting from a particular state:

$$V_\phi^\pi(s_t) \approx \sum_{t'=t}^{T-1} \mathbb{E}_{\pi_\theta}\left[r(s_{t'}, a_{t'})|s_t\right], \tag{14}$$

so the approximate policy gradient now looks like this:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_{it}|s_{it})\right)\left(\sum_{t'=t}^{T-1}\gamma^{t'-t} r(s_{it'}, a_{it'}) - V_\phi^\pi(s_{it})\right). \tag{15}$$

4 Policy Gradient

4.1 Vanilla Policy Gradient

Within the `pg_agent.py` file, implement the vanilla version of the Q value function in the function `_calculate_q_vals` by also defining the `_discounted_return` function:

$$r(\tau_i) = \sum_{t'=0}^{T-1}\gamma^{t'} r(s_{it'}, a_{it'}), \tag{16}$$

4.2 Reward-To-Go Policy Gradient

Within the `pg_agent.py` file, implement the Reward-To-Go version of the Q value function in the function `_calculate_q_vals` by also defining the `_discounted_reward_to_go` function:

$$r(\tau_i) = \sum_{t'=t}^{T-1}\gamma^{t'-t} r(s_{it'}, a_{it'}), \tag{17}$$

4.3 Estimate Advantage Method

Within the `pg_agent.py` file, implement the `_estimate_advantage` method of the PGAgent in the case `self.critic = None`.

4.4 Update Method

Within the `pg_agent.py` file, implement the `update` method of the PGAgent (Step 1 to 3, not `self.critic != None`).

4.5 Base MLP policy

Within the `policies.py` file, implement the `get_action` and `forward` methods for the `MLPPolicy` class.

4.6 MLP Policy PG

Within the `policies.py` file, implement the `update` method for the `MLP Policy PG` class.

4.7 Trajectory sampling

Within the `utils.py` file, implement the `sample_trajectory` function.

4.8 Training Loop

Within the `run_hw2.py` file, complete the training loop.

4.9 Experiments

Run multiple experiments with the PG algorithm on the discrete CartPole-v0 environment, using the following commands:

```
# Basic Experiment
python run_hw2.py --env_name CartPole-v0 \
-n 100 -b 1000 --exp_name cartpole

# Using Reward-to-Go
python run_hw2.py --env_name CartPole-v0 \
-n 100 -b 1000 -rtg --exp_name cartpole_rtg

# Normalizing Advantages
python run_hw2.py --env_name CartPole-v0 \
-n 100 -b 1000 -na --exp_name cartpole_na

# Reward-to-Go with Normalized Advantages
python run_hw2.py --env_name CartPole-v0 \
-n 100 -b 1000 -rtg -na --exp_name cartpole_rtg_na

# Repeated with Increased Batch Size (4000)
python run_hw2.py --env_name CartPole-v0 \
-n 100 -b 4000 --exp_name cartpole_lb
---
python run_hw2.py --env_name CartPole-v0 \
-n 100 -b 4000 -rtg --exp_name cartpole_lb_rtg
---
python run_hw2.py --env_name CartPole-v0 \
-n 100 -b 4000 -na --exp_name cartpole_lb_na
---
python run_hw2.py --env_name CartPole-v0 \
-n 100 -b 4000 -rtg -na --exp_name cartpole_lb_rtg_na
```

What's happening here:

- **-n**: Number of iterations.
- **-b**: Batch size (number of state-action pairs sampled per iteration).

- **-rtg**: If present, sets `reward_to_go=True`. Otherwise, `reward_to_go=False`.
- **-na**: If present, sets `normalize_advantages=True`, normalizing the advantages within a batch.
- **–exp_name**: Specifies the name for the experiment, used in data logging.

Various other command line arguments will allow you to set batch size, learning rate, network architecture, and more.

What to Expect:

- The best configuration of CartPole in both the large and small batch cases should converge to a maximum score of 200.

5 Policy Gradient with a Neural Network as a baseline

5.1 Critic Neural Net

Within the `critics.py` file, implement the `forward` and `update` functions.

5.2 Estimate Advantage Method

Within the `pg_agent.py` file, implement the `_estimate_advantage` method of the PGAgent in the case `self.critic != None`.

5.3 Experiments

Next, you will use your baselined policy gradient implementation to learn a controller for HalfCheetah-v4.

Run the following commands:

```
# No baseline
python cs285/scripts/run_hw2.py --env_name HalfCheetah-v4 \
-n 100 -b 5000 -rtg --discount 0.95 -lr 0.01 \
--exp_name cheetah

# Baseline
python cs285/scripts/run_hw2.py --env_name HalfCheetah-v4 \
-n 100 -b 5000 -rtg --discount 0.95 -lr 0.01 \
--use_baseline -blr 0.01 -bgs 5 --exp_name cheetah_baseline
```

6 Note on Launching TensorBoard

To visualize the results of your experiments with TensorBoard, use the following command in your terminal. Ensure you replace `<path_to_your_logs>` with the actual directory path where your TensorBoard event files are stored. This directory is the one specified during the training process for saving logs.

```
tensorboard --logdir=<path_to_your_logs>
```

After running the command, TensorBoard will start and provide a URL, typically `http://localhost:6006`, which you can open in a web browser to view your experiment metrics.

If you want to visualize also pictures and 3D renders, you can add the following flags to your run command.

```
-- video_log_freq 10
```