ECEN 689: Reinforcement Learning

HW 4: Policy Gradient

In this HW, we would be coding a class of algorithms that is a to-go approach for solving reinforcement learning (RL) tasks: Policy Gradient Algorithms. The learning outcomes of this assignment are:

1. Introduction to Policy Gradient variants.
2. Introduction to RL coding pipeline.
3. Introduction to Deep RL framework PyTorch.

1 Review

Let $V_{\pi_\theta} = V_{\pi_\theta,\mu_0} = \mathbb{E}_{x\sim\mu_0(\cdot)}\left[V_\pi(x)\right]$, where $V_\pi(x) = \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t R\left(x_t, a_t\right) \mid x_0 = x\right]$. We have looked at the following variants of basic policy gradient algorithms in the class:

$$\nabla V_{\pi_\theta} = \mathbb{E}_{\tau\sim P_{\text{traj},\theta}(\cdot)}\left[R(\tau)\sum_{t=0}^{\infty}\nabla\log\pi_\theta\left(x_t, a_t\right)\right], \tag{1}$$

$$\nabla V_{\pi_\theta} = \mathbb{E}_{\tau\sim P_{\text{traj},}\ (\tau)}\left[\sum_{t=0}^{\infty}\gamma^t G_t \nabla\log\pi_\theta\left(x_t, a_t\right)\right], \tag{2}$$

$$\nabla V_{\pi_\theta} = \mathbb{E}_{\tau\sim P_{\text{traj},}\ (\tau)}\left[\sum_{t=0}^{\infty}\gamma^t A_{\pi_\theta}(x_t, a_t)\nabla\log\pi_\theta(x_t, a_t)\right]. \tag{3}$$

Here, $\tau = \left(x_0, a_0, x_1, a_1, \ldots, x_T, a_T\right), a_t \sim \pi_\theta\left(x_t, \cdot\right), x_{t+1} \sim P\left(\cdot \mid x_t, a_t\right), x_0 \sim \mu_0(\cdot)$ represents one trajectory sampled from the the distribution $P_{\text{traj},\theta}(\tau) = \mu_0\left(x_0\right)\prod_{t\geq 0}\pi_\theta\left(x_t, a_t\right)P\left(x_{t+1} \mid x_t, a_t\right)$. The cumulative discounted reward of trajectory τ is given by $R(\tau) = \sum_{t=0}^{\infty}\gamma^t R\left(x_t, a_t\right)$, and the reward to go, is given by $G(t) = \sum_{t'=t}^{\infty}\gamma^{t'-t}R\left(x_{t'}, a_{t'}\right)$. In Eqn. 1 we scale the gradient of the log of the policy using the cumulative reward accumulated in that trajectory. This method results in high variance in the policy gradient update. In Eqn. 2 we exploit causality (rewards in the past do not affect the policy) to reduce variance. The variance in the policy gradient can be further reduced by subtracting a baseline **(generally the value function)** as in Eqn. 3.

Each variant above is an instance of the same algorithm: Policy Gradient Algorithm. The difference stems from which quantity the expectation is taken over and how the quantities within the expectation are estimated. Since the equations above are expectations, they can not be used directly. A sampling-based estimator of Eqn. 1 that can be coded up is:

$$\nabla V_{\pi_\theta} \approx \frac{1}{N}\sum_{i=1}^{N}\left(R(\tau_i)\left(\sum_{t=0}^{T}\nabla_\theta\log\pi_\theta(s_t^i, a_t^i)\right)\right). \tag{4}$$

The equation above is an empirical mean of scaled gradient terms. Similar sampling-based estimators can be written for all the other variants.

2 Environment

For this assignment, we will be studying Policy Gradients using the following two environments:

1. Point-v0: This is a simple custom 2D 'reaching' environment designed to help us understand the basics of policy gradient algorithms. At the beginning of each episode, a point agent is thrown at a random location on a 2D square (The state space is the 2D grid of points $\{x, y : -1 \leq x \leq 1, -1 \leq y \leq 1\}$). The goal of the agent is to move to the origin asap. The action space is 2D continuous vector, (dx, dy), that encodes the agent movement along any co-ordinate direction. A multivariate Gaussian distribution with identity co-variance matrix is used as the action distribution. The mean of multivariate Gaussian distribution is given by $\mu = \theta^\top \tilde{s}$, where $\tilde{s}$ is the vector obtained by appending 1 to the state s (1 is appended to capture bias term often used in machine learning implementations). This implies that PDF of the policy is given by $\frac{1}{2\pi} e^{\frac{-1}{2}(a-\mu)^\top I (a-\mu)}$. Note that the policy is linear in randomly initialized parameters ($\theta \sim \mathcal{N}(0, 0.01, \text{size} = (|\mathcal{A}|, |\mathcal{S}| + 1))$). For further details on the environment, look into the file `point_env.py`. A sample rendering of the environment is provided in `assets/Point-v0_rendering`.

2. CartPole-v0: This environment from OpenAI Gym consists of a pole that is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of $+1$ or -1 to the cart. The pole starts upright, and the goal is to prevent it from falling over. A reward of $+1$ is provided for every timestep that the pole remains upright. CartPole-v0 defines *solving* as getting an average reward of 195.0 over 100 consecutive episodes.

3 Questions

3.1 Point-v0

1. Recall the PDF of π_θ given in the environment description. Analytically compute a closed form expression for the gradient of the policy for a given action and state, *i.e.*, gradient of the term $\nabla_\theta \log \pi_\theta(a|s)$. Include the expression in your submission report.

2. Implement policy gradient variants specified in Eqn. 1 and Eqn. 2 for Point-v0. For this part you would have to update the following files:

   (a) `main_loop.update_params` method in `main.py`.

   (b) `estimate_net_grad` method `solutions/point_mass_solutions.py`

   Please feel free to modify the method arguments as needed. We have only provided a basic outline. We recommend a learning rate of ~ 0.1 for this environment. Plot the training curves for each of these variants (discounted sum of rewards per iteration of the training procedure). Summarize your observations. How fast does the algorithm learn? What average return do we see after 100 iterations of the algorithm? **Include a rendering of the final policy.**

3. **Bonus**: Do you think including a baseline would improve the performance of these algorithms. Try implementing a simple time dependent baseline as follows: Simply set the baseline to the average of all the returns ($\text{R}(\tau)$) among the trajectories collected from the previous iteration. If there are no trajectories long enough at a certain time step, we can set the baseline to 0.

3.2 CartPole-v0

1. Implement the REINFORCE Algorithm using the policy gradient described in Equation 1 on the CartPole-v0 environment. Plot the un-discounted cumulative reward per episode. (Note: We do not expect you to solve CartPole-v0 using this approach)

2. Implement the REINFORCE Algorithm using the policy gradient described in Equation 2 on the CartPole-v0 environment. Plot the un-discounted cumulative reward per episode. (Note: We expect you to solve CartPole-v0 using this approach)

3. Implement the REINFORCE Algorithm using the policy gradient described in Equation 3 on the CartPole-v0 environment. Plot the un-discounted cumulative reward per episode. (Note: We expect you to solve CartPole-v0 using this approach)

4. Summarize your observations on the 3 approaches to implement the REINFORCE algorithm.

4 Coding Pipeline

Summary: Sample trajectories → Estimate returns/advantages → train/update parameters. The basic code flow for this assignment (and reinforcement learning algorithms in general) is included in Fig. 1. We define an agent class that includes capability to interact with the environment and generate batch data (Fig 2) of the form $\{(s_t^i, a_t^i, r_t^i, s\prime_t^i, \text{mask})_t^i\}_{t=0,i=1}^{t=T,i=N}$ (This step is covered by numbers (1) and (2) in Fig. 1). The last term 'mask' is a binary variable that captures end of the episode: it is 0 only when the episode ends and always taking a value 1 otherwise. That is, the agent samples N trajectories of T tines steps each at each iteration of the algorithm.

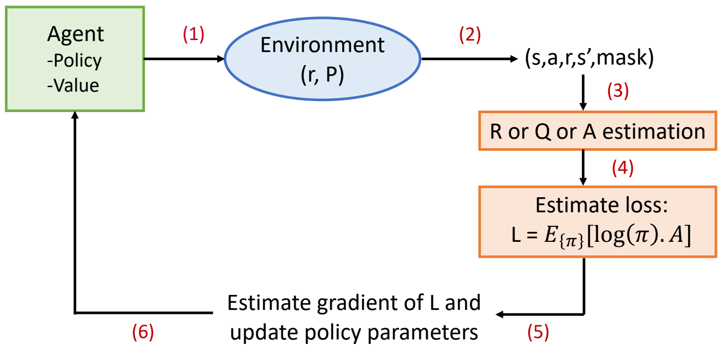

Figure 1: Coding pipeline for policy gradient algorithms

We include a single parameter for the batch size called `min_batch_size` in `main.py`. The variable `batch` defined `main.py` takes the following form:

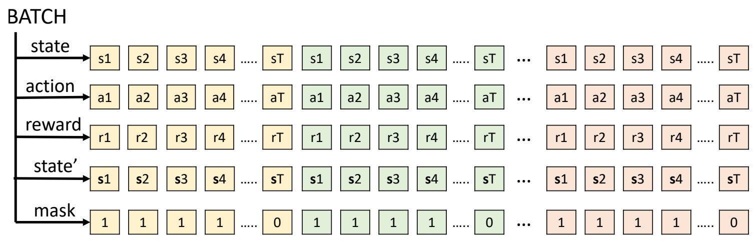
BATCH
state
action
reward
state'
mask
s1 s2 s3 s4 ..... sT s1 s2 s3 s4 ..... sT ... s1 s2 s3 s4 ..... sT
a1 a2 a3 a4 ..... aT a1 a2 a3 a4 ..... aT ... a1 a2 a3 a4 ..... aT
r1 r2 r3 r4 ..... rT r1 r2 r3 r4 ..... rT ... r1 r2 r3 r4 ..... rT
s1 s2 s3 s4 ..... sT s1 s2 s3 s4 ..... sT ... s1 s2 s3 s4 ..... sT
1 1 1 1 ..... 0 1 1 1 1 ..... 0 ... 1 1 1 1 0

Figure 2: Batch

In each iteration upon collection of the `batch` data, the function `update_params` is called to:

1. estimate the returns/advantages (step (3) in Fig. 1),

2. compute policy gradient (steps (4) and (5) in Fig. 1), and
3. finally to update policy parameters (step (6) in Fig. 1).

5 General Instructions

The basic code is implemented in Python 3.9 and PyTorch 1.8.1 and uses OpenAI-Gym. It is advisable to set up a virtual environment (like an anaconda environment) for solving this assignment.

Please feel free to modify the method arguments as needed. We have only provided a basic outline.

While we highly recommend you to use the existing code skeleton for this homework, you may implement all the questions in your own methods as well.

IMPORTANT: Remember that while solving coding assignments Stack-Overflow is your best friend. Please search on Stack-Overflow when you hit roadblocks.

6 Submission Details

1. Submit a zip file consisting of completed code and a report that lists your finding and answers to all the questions above. **Do not add your code in the report**.
2. Put the following files in a folder named **LastName_FirstName**
   - All the codes
   - A PDF report of your findings.
3. Zip the **folder** in 2 and submit on Canvas.