Homework 1: Policy Gradient

1 Introduction

The goals of this document is two-fold. First, we will present a short review on *Policy Gradient (PG)* algorithm and some of it's variations. Then, you will be asked to conduct a series of experiments that showcase your understanding about the concepts. You will start by implementing the vanilla-PG (VPG) and then training a simple agent with it on an Atari game environment to verify it's soundness. Then, you will implement a few other improvements over VPG and compare the performance gain across these versions.

2 Review

2.1 Policy Gradient

Recall that in VPG, we are presented with the challenge of maximizing a goal function $J(\theta)$, that represents the cumulative-reward or return $R(\tau)$ from a sampled trajectory τ with the policy π_θ.

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\big[R(\tau)\big] \tag{1}$$

Here, θ is the policy parameters, and in our case it is the weights of our neural network. The return R can be represented as $R(\tau) = \sum_{t=0}^{T} r_t$, where r_t is the reward at time t and T is the final or max time-step of τ.

Additionally, a trajectory τ is represented as a sequence of state-action tuples and the probability of τ can be expressed as a joint-probability over these transitions. If the initial state is s_0, then we can write the probability of τ as

$$P(\tau) = P(s_0)\prod_{t=1}^{T}\Big[\pi_\theta(a_{t-1}|s_{t-1})P(s_t|s_{t-1},a_{t-1})\Big] \tag{2}$$

To find the maximum value, we will apply the derivative operator on the both side of eq 2 and can simplify the expression as follows

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\big[R(\tau)\big] \\
&= \nabla_\theta \int \big[P(\tau)R(\tau)\big] \\
&= \int \Big[\big[\nabla_\theta P(\tau)\big]R(\tau)\Big] \\
&= \int \Big[P(\tau)\big[\nabla_\theta \log P(\tau)\big]R(\tau)\Big] \\
&= \int \left[P(\tau)\left(\nabla_\theta \log P(s_0) + \nabla_\theta \sum_{t=1}^{T}\log \pi_\theta(a_{t-1}|s_{t-1}) + \nabla_\theta \sum_{t=1}^{T}\log P(s_t|s_{t-1},a_{t-1})\right)R(\tau)\right] \\
&= \int \left[P(\tau)\left(0 + \sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t) + 0\right)R(\tau)\right] \\
&= \int \left[P(\tau)\left(\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t)\right)R(\tau)\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t)\right)R(\tau)\right]
\end{aligned} \tag{3}$$

This expectation value can be approximated by sampling a batch of $N = 1, 2, ..., k$ trajectories from the environment and then taking the mean. Hence, we will have,

$$\begin{aligned}\nabla_\theta J(\theta) \approx& \frac{1}{N}\sum_k \left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^k|s_t^k)\right) R^k(\tau)\right] \\ \approx& \frac{1}{N}\sum_k \left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^k|s_t^k)\right)\left(\sum_{t=0}^{T} r_t^k\right)\right]\end{aligned} \tag{4}$$

Effectively, eq 4 is telling us to take the sum of derivatives of all the action-log-probability from τ_k. Then, multiply this sum with our return $R^k(\tau)$. We repeat these two steps k times, and take their mean to approximate the expected gradient over our objective function $J(\theta)$. And finally we update our network weight as

$$\hat{\theta} \leftarrow \theta + \nabla_\theta J(\theta) \tag{5}$$

Note that, in standard ML libraries like PyTorch or TensorFlow, we minimize the goal or objective or cost, rather than maximizing it. Hence, for an implementation that use those libraries, eq 5 will take the following form

$$\nabla_\theta J(\theta) \approx -\frac{1}{N}\sum_k \left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^k|s_t^k)\right)\left(\sum_{t=0}^{T} r_t^k\right)\right] \tag{6}$$

2.2 Reducing variance of VPG

As you remember from the class lectures that the VPG algorithm suffers heavily from high-variance in the estimation of gradient. From eq 6, we can see that there are two components that contribute to the gradient estimation. The action-log-probability is directly dependent upon the policy that we are trying to optimize. However, the reward signal is coming from the environment and in it's raw form can have a very distribution with very high-variance. The are many methods that have been proposed over the years that aim to reduce this variance.

2.2.1 Reward to Go

Under this method, we exploit the fact that an action can only affect rewards that come from that point onwards, i.e. the causality. Using this we can define the reward-to-go for action a_t as the sum of rewards from time t to the end of trajectory. And we can modify the eq 6 as

$$\nabla_\theta J(\theta) \approx -\frac{1}{N}\sum_k \left[\sum_{t=0}^{T-1} \left(\nabla_\theta \log \pi_\theta(a_t^k|s_t^k) \sum_{t'=t}^{T} r_{t'}^k\right)\right] \tag{7}$$

2.2.2 Reward Discounting

We can introduce a discounting factor γ that will put more weights on near-future rewards than far-future rewards. This helps sum-of-reward to converge better to a less-variance distribution. Also intuitively, we care more about rewards that we are about to get, compared to some reward that we will get in distant future. To include the discount-factor on top of the reward-to-go, we update eq 7 as

$$\nabla_\theta J(\theta) \approx -\frac{1}{N}\sum_k \left[\sum_{t=0}^{T-1} \left(\nabla_\theta \log \pi_\theta(a_t^k|s_t^k) \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}^k\right)\right] \tag{8}$$

3 Starter Code

This section talks about the provided starter code (available as zip file). It includes partial implementation of this assignment. There are two versions. One that you can run in your local machine from the CLI. Alternatively, you can also find a Colab notebook that you can run on Google Colab. The colab version takes the simulation parameters as a dictionary.

You should find out the tags marked as ***TODO*** and complete them accordingly. In most cases, some hints are provided for you.

Here is a list of files that you can find within the starter code folder

- main_hw1.py

  This is the main file that you will use to run all the simulations/experiments from section 4. Running this script should generate a .pkl file as the log of the experiment. It will also create a video containing a single episode with the policy you have just trained!

Table 1: List of arguments and their default values

argument	default value
env_name/e	CartPole-v1
rng_seed/rng	6369
reward_to_go/rtg	False
reward_discount/rd	False
n_rollout/nr	10
n_trajectory_per_rollout/ntr	2
hidden_dim/hdim	64
lr/lr	3e-3
exp_name/xn	my_exp

- main_generate_plt.py

  This is the main file that you can use to generate visualizations for section 4. You can use other graphing methods, but you must adhere to this visualization guideline.

  After you have multiple runs (applicable for section 4.2 bonus) of one PG version, you can merge multiple .pkl files into one pandas dataframe to create a mean and standard-deviation plot. An example is given here.

- learning_algorithms.py

  This file contains most of the codes behind the PG implementation. The TODO's here are

  - Calculate avg reward for this rollout
  - Compute loss function
  - Define the policy net
  - Forward pass of policy net
  - Get action from the policy
  - Step environment

- utils.py

  - Compute rtg_reward (as a list) from raw_reward
  - Compute discounted_rtg_reward (as a list) from raw_reward

- requirements.txt

  A text file that has a list of required packages to run this assignment.

- hw1.ipynb

  A notebook, which you can upload into your Google drive to run the experiments there. This can save you from creating an environment locally. :)
  Don't forget to upload your scripts with TODO's as well. One possible solution is discussed here.

4 Tasks for You

4.1 Part 1 - Completion of TODOs

Complete the TODOs in the started code.

4.2 Part 2 - Experiment I (CartPole)

In this section, you will run couple of experiments with your completed implementation and the environment CartPole-v1. The goal of this experiment is to train the agent on CartPole environment using the given hyper-parameters and answer couple of questions based on the observed results.

Trials

- **T0**: python main_hw1.py -e CartPole-v1 -nr 100 -ntr 10 -hdim 64 -lr 3e-3 -xn CartPole_v1_t0
- **T1**: python main_hw1.py -e CartPole-v1 -rtg -nr 100 -ntr 10 -hdim 64 -lr 3e-3 -xn CartPole_v1_t1
- **T2**: python main_hw1.py -e CartPole-v1 -rd -nr 100 -ntr 10 -hdim 64 -lr 3e-3 -xn CartPole_v1_t2

Deliverables for report

1. Take snapshots from your code that show your implementation of eq 6, 7, 8 and include in the report.
2. Create a graph that compares the learning curve from the four trials above. Label the curves as t0, t1, t2.
3. Answer the following questions
   (a) Which version of PG is best performing (among return-based, reward-to-go-based and discounted-reward-based)?
   (b) **BONUS:** If you run the same trials multiple times (with different random seeds) and then plot the learning curves together, it should give a visual estimation of average-trajectory rewards variance (higher spread means higher variance and vice-versa). Can you generate such plots (one for each version) and comment on the resulting variance (which one has higher or lower variance)?

What to expect

- The average-trajectory-reward for the best configuration (among the three listed above) should converge around **480**. Expected results from the Colab version was tested without any hardware acceleration.

4.3 Part 3 - Experiment II (LunarLander)

In this section, you will run couple of experiments with your completed implementation and the environment LunarLander-v2. The goal of this experiment is to train the agent on LunarLander environment using the given hyper-parameters and answer couple of questions based on the observed results.

Trials

- **T0**: python main_hw1.py -e LunarLander-v2 -rd -nr 125 -ntr 5 -hdim 128 -lr 3e-3 -xn LunarLand_v2_t0
- **T1**: python main_hw1.py -e LunarLander-v2 -rd -nr 125 -ntr 20 -hdim 128 -lr 3e-3 -xn LunarLand_v2_t1
- **T3**: python main_hw1.py -e LunarLander-v2 -rd -nr 125 -ntr 60 -hdim 128 -lr 3e-3 -xn LunarLand_v2_t2

Deliverables for report

1. Create a graph that compares the learning curve from the three trials above. Label the curves as t0, t1, t2.
2. Answer the following questions
   (a) How does the number-of-trajectory-per-rollout effect learning performance? How can you justify this relationship?

What to expect

- The average-trajectory-reward for the best configuration (among the three listed above) should converge around **200**. Expected results from the Colab version was tested without any hardware acceleration.

5 Submission Guideline

Please read the following very carefully. Unable to meet any of the requirements will cause you to lose points-

1. You are **highly encouraged** to create a GitHub repo and perform frequent commit. Attach the repo link within your report. This will serve as a history of your activity for this assignment.
2. Ideally you should not require to create any other additional functions or classes except the ones that are already provided. You are only expected to fill-in the TODOs.
3. Submit as a zipped file named as cse6369_a1__StudentID__StudentName.zip. A student named "John Doe" with ID 0123456789 should have a zipped file name cse6369_a1__0123456789__john_doe.zip.

   Make sure this zip file contains the following folder/files.

   - report.pdf
   - main_hw1.py
   - main_generate_plot.py
   - learning_algorithms.py
   - utils.py
   - requirements.txt
   - (optional) Two .mp4 files from best configuration of each environment.
4. All the submitted code and the report should be your own work. If you have used any reference, make sure to use the source as a reference. If any form of plagiarism is found, you won't receive any credit for this assignment and it will reflect negatively on your final grade as well.
5. Your implementations must be written in **Python 3.8** or later versions.
6. You are not allowed to import any additional libraries or packages except the ones that are already imported. Please consult with the TA if you think you need any other libraries apart from those.
7. Make sure to fill in student name and ID in the Colab file.

8. Do not submit figures as stand-alone files, please embed them into your report.
9. Make sure to include your name and student ID in the report.