Assignment 4c: Reinforcement Learning

Goal: Study Reinforcement Learning algorithms and compare them.

Submission: The assignment consists of three parts: a proof, implementation and an analysis. You are supposed to present results for all parts in the following manner:

1. Upload your code.
2. Prepare a report with the proof and an analysis of results obtained at home.

The code and the report must be uploaded due to the deadline to Canvas.

UPLOAD A **SINGLE FILE** (a zip file) containing your code and the report. Name your file as follows: [vunetid]_[assignment number].

Introduction

Part 1: Proofs

First, we will prove that REINFORCE with correct credit assignment is an unbiased estimate of the policy gradient. To help you get started, we divide the proof into three parts to guide you in the correct direction.

In this exercise, we will prove that

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T-1} \gamma^t G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

That is, the REINFORCE estimate is an unbiased estimate of the policy gradient. See the slides for notation details and please keep to it in your answers. (Don't worry about the colors though :))

Question 1: Show that

$$\nabla_\theta J(\theta) = \sum_{t'=0}^{T-1} \nabla_\theta \mathbb{E}_{p(\tau|\theta)}[\gamma^{t'} r_{t'+1}]$$

Question 2: Show that

$$\nabla_\theta \mathbb{E}_{p(\tau|\theta)}[\gamma^{t'} r_{t'+1}] = \mathbb{E}_{p(\tau|\theta)}[\gamma^{t'} r_{t'+1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)]$$

Question 3: Prove that (these two are not equal: The sum of the first is to T-1, the second is to t'-1!)

$$\mathbb{E}_{p(\tau|\theta)}[\gamma^{t'} r_{t'+1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)]$$

$$= \mathbb{E}_{p(\tau|\theta)}[\gamma^{t'} r_{t'+1} \sum_{t=0}^{t'-1} \nabla_\theta \log \pi_\theta(a_t|s_t)]$$

Hint: Write out the expectation over trajectories as a sum over states, actions and rewards.

Question 4: Prove that

$$\sum_{t'=0}^{T-1} \gamma^{t'} r_{t'+1} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

$$= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'+1}$$

Hint: A proof by induction will work here. If you don't know how to do a proof by induction, then a convincing argument why it should hold is also okay!

Question 5: Using the previous four results, prove that

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T-1} \gamma^t G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

i.e., REINFORCE is a Monte Carlo estimate of the policy gradient.

Part 2: implementation

Setting up Python environment

- Ensure PyTorch is installed
- Install openAI gym: `pip install gym`
- More information on installing Gym with special environments is at https://github.com/openai/gym#installation

Choose a gym environment

A list of environments can be found at https://github.com/openai/gym/blob/master/docs/environments.md . Try to start with simple environments, like those in classic control and algorithmic. If those are easy to solve with your algorithms, you can move on to more complex environments.
Note that to get good results on atari, for example, a huge amount of training iterations might be required. We don't expect you to do this! This assignment is supposed to get you to understand RL algorithms. If running times are prohibitively large, move back to a simpler environment.

Question 6: Implement simple REINFORCE and REINFORCE with lookahead

Implement both the simple REINFORCE algorithm (lecture 9, slide 22) and the 'normal' REINFORCE algorithm (lecture 9, slide 32). Slide 33 contains PyTorchy pseudo code to get you started.
Also consider problems like batching: Do you first collect multiple trajectories, possibly in parallel, and batch them together?

Question 7: Implement a more advanced RL algorithm

Implement a more advanced RL algorithm. While you are free to choose this, good options are Deep Q-Learning (DQN) and Advantage Actor-Critic (A2C) that are introduced in lecture 11.
REMARKS:

- Do not worry if you do not achieve strong scores. RL is notoriously hard to get working, and high scores are not an important part during scoring your report!
- Please focus on reasonable architectures that you can learn on your machine. No need to use gigantic neural networks.
- Implement only so many hacks as you have time for! Of course, the more stuff you implement ánd experiment with the better, but most important is a good report of your experiments. A simple implementation of an actor-critic or DQN algorithm is sufficient for a good grade.
- Try to Google around for tips on good implementations of these algorithms: They usually require some stupid hacks to get working, but there are good blog-posts and lectures that can help you out.

Actor-critic

For actor-critic, some tricks are very useful to implement.

- Share parameters between actor and critic
- Use parallel workers (I'd recommend doing synchronized A2C)

Q-learning

For DQN, I'd recommend watching Sergey Levine's lectures to get a good implementation, such as this lecture with practical tips. A stable DQN implementation could use (in order of importance):

- Experience replay

- Target networks (this video also contains pseudocode for the classical Google DQN algorithm)
- Double Q-learning
- Multi-step returns

Part 3: Experiments

Performance:
Run the algorithms you implemented on the chosen environment(s). Think of one or two measures of performance to compare the algorithms on. Examples are: Speed of getting to max performance, max performance, total online reward, best average reward after X episodes…

Because of the high amount of stochasticity involved in RL, make sure to evaluate on multiple runs.

Question 8: *Analysis*
Compare the results of the different algorithms. Try to think about some research question that you want to answer, together with a hypothesis of the answer. Some ideas: Why do you think some algorithms work better than others? Is this different from your expectations? Does this depend on the environment?

Available implementations

- PyTorch offers a tutorial on DQN https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html
- Stable baselines offers the most thorough implementations of RL algorithms, including DQN and A2C. https://stable-baselines.readthedocs.io/en/master/index.html Obviously, we don't expect your implementation to be on this level!
- https://openai.com/blog/openai-baselines-dqn/

Grading (10pt)

- Question 1: 0.5pt
- Question 2: 0.5pt
- Question 3: 0.5pt
- Question 4: 0.5pt
- Question 5: 0.5pt
- Question 6: 2.5pt

- Question 7: 2.5pt
- Question 8: 2.5pt