Homework 1: Policy Gradients

CMU 10-703: Deep Reinforcement Learning (Fall 2025)

OUT: Wednesday September 3rd, 2025
DUE: Monday September 12th, 2025 by 11:59pm EST

Instructions: START HERE

Note: This homework assignment requires a significant implementation effort. Please plan your time accordingly. General tips and suggestions are included in the 'Guidelines on Implementation' section at the end of this handout.

- **Collaboration policy:** You may work in groups of up to three people for this assignment. It is also OK to get clarification (but not solutions) from books or online resources after you have thought about the problems on your own. You are expected to comply with the University Policy on Academic Integrity and Plagiarism[1].
- **Late Submission Policy:** You are allowed a total of 10 grace days for your homeworks. However, no more than 2 grace days may be applied to a single assignment. Any assignment submitted after 2 days will not receive any credit. Grace days do not need to be requested or mentioned in emails; we will automatically apply them to students who submit late. We will not give any further extensions so make sure you only use them when you are absolutely sure you need them. See the Assignments and Grading Policy here for more information about grace days and late submissions: https://cmudeeprl.github.io/703website_f25/
- **Submitting your work:**
  - **Gradescope:** Please write your answers and copy your plots into the provided LaTeX template, and upload a PDF to the GradeScope assignment titled "Homework 1." Additionally, zip all the code folders into a directory titled `<andrew_id>.zip` and upload it the GradeScope assignment titled "Homework 1: Code." Each team should only upload one copy of each part. Regrade requests can be made within one week of the assignment being graded.
  - **Autolab:** Autolab is not used for this assignment.

This is a challenging assignment. **Please start early!**

[1] https://www.cmu.edu/policies/

Introduction

The goal of this homework is to give you experience implementing and analysing Deep Reinforcement Learning algorithms. We'll start with one of the oldest Reinforcement Learning algorithms, REINFORCE, then build our way up to N-step Advantage Actor-Critic (A2C). Although these algorithms were proposed many years ago, they serve as the foundation of many current state of the art approaches.

Problem 0: Collaborators

Please list your name and Andrew ID, as well as those of your collaborators.

Problem 1: Policy Gradient Algorithms (48 pts)

Problem 1.1: REINFORCE (10 pts)

We begin this homework by implementing episodic REINFORCE [4], a policy-gradient RL algorithm. Use the conda environment defined in `code/environment.yaml` for compatibility. Please write your code in `code/pg/pg.py`; the template code should give you an idea on how you can structure your code for this problem and next two problems.

In [3], the authors present a version of REINFORCE that performs a single gradient update to the policy for every timestep of a collected episode. Here, we ask you to consider a slightly different version that instead performs a single policy gradient update for every *episode* of collected data, treating the entire episode as a minibatch of experience when determining the policy gradient. By performing a single gradient update per episode, we can obtain policy gradient estimates with (slightly) less variance, and we reduce the risk of significantly changing the policy if we encounter a particularly long episode of experience, which generally tends to result in learning instabilities. We also ask you to use the Adam optimiser instead of performing vanilla gradient ascent. If you're not familiar with Adam, you can think of it as a fancy version of gradient descent that introduces momentum and an adaptive learning rate for each dimension. Pseudo-code for the required version of REINFORCE is provided in Algorithm 1. Recommended hyperparameter values are included in the code template. The network settings and hyperparameters for the policy are provided to you in `code/pg/pg.py`.

Evaluate your implementation of Algorithm 1 on the `CartPole-v1` environment by running 5 IID trials[2] with $E = 3,500$. During each trial, freeze the current policy every 100 training episodes and run 20 independent test episodes. Record the mean **un**discounted return obtained over these 20 test episodes for each trial and store them in a matrix $D \in \mathbb{R}^{\text{number of trials} \times \text{number of frozen policies per trial}} = \mathbb{R}^{5 \times (3,500/100)} = \mathbb{R}^{5 \times 35}$. Note that each entry

[2]We ask you to run multiple trials as DRL algorithms tend to exhibit a significant deal of random variation both across implementations and across random seeds for a given implementation. This is well documented in the literature [1] and we encourage you all to reflect on the various sources of randomness that make DRL algorithms hard to compare.

Algorithm 1 REINFORCE

procedure REINFORCE
    Start with policy network π_θ
    repeat for E **training episodes:**
        Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ *following* $\pi_\theta(\cdot)$
        for $t = 0, 1, \ldots, T-1$:
            $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} R_k$
        $L(\theta) = -\frac{1}{T} \sum_{t=0}^{T-1} G_t \ln \pi_\theta(A_t|S_t)$
        Update π_θ *using* `Adam` $(\nabla_\theta L(\theta))$
end procedure

in D is an average of 20 values, and any particular column of D gives us a "snapshot" of your implementation at some point in time during training on `CartPole-v1`.

In one figure, plot the **mean of the mean undiscounted return** on the y-axis against **number of training episodes** on the x-axis. In other words, plot the average of the entries in columns of D on the y-axis. On the same figure, also plot a shaded region showing the maximum and minimum mean undiscounted return against number of training episodes, where the max and min are performed across the trials (i.e. show the the max and min of each column in D). The plotting format is given to you in `code/pg/pg.py`.

As a brief aside, in RL, the term "expected return" refers to $\mathbb{E}_{\tau \sim \pi_\theta} \{\sum_{t=0}^{\infty} \gamma^t R_t\}$, where the expectation is with respect to the randomess in the MDP under a particular policy (the initial state, any randomness in the state transitions, any randomness in the action selection, and any randomness in the reward generation). When we ask you to plot the mean of the mean cumulative undiscounted reward, the first mean refers to the randomness in the DRL algorithm (e.g. random weight initialisation), and the second mean refers to the randomness in the underlying MDP itself. When we look at a particular column of D, the variability amongst these 5 values is attributable to the DRL algorithm itself, not the underlying MDP. For the purposes of comparing DRL algorithms, this variability is of critical importance.

Runtime Estimation: To help you plan your time, note that our implementation of Algorithm 1 takes 8 minutes to complete a single trial with $E = 3,500$ on a laptop.

Problem 1.2: REINFORCE with Baseline (10 pts)

An important early development to REINFORCE was the introduction of a baseline. In this part of the homework, we ask you to make a small change to your implementation of REINFORCE to examine the impact of this on the same `CartPole-v1` test environment.

Again, we ask you to consider a slightly modified version of the algorithm presented in Sutton & Barto's text. The pseducode is provided in Algorithm 2. For the baseline, we recommend making changes to the output layer of the network provided in `code/pg/pg.py`. Generate the same plot as for REINFORCE (i.e. run 5 IID trials and plot the mean, max and min of the mean undiscounted returns across trials). Take $E = 3,500$.

Runtime Estimation: Note that our implementation of Algorithm 2 takes 10 minutes to complete a single trial with $E = 3,500$ running on a laptop.

Algorithm 2 REINFORCE with Baseline

procedure REINFORCE WITH BASELINE
    Start with policy network π_θ *and baseline network* b_ω
    repeat for E **training episodes:**
        Generate an episode $S_0, A_0, R_0, \ldots, S_{T-1}, A_{T-1}, R_{T-1}$ *following* $\pi_\theta(\cdot)$
        for $t = 0, 1, \ldots, T-1$:
            $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} R_k$
        $L(\theta) = -\frac{1}{T} \sum_{t=0}^{T-1} (G_t - b_\omega(S_t)) \ln \pi_\theta(A_t|S_t)$
        $L(\omega) = \frac{1}{T} \sum_{t=0}^{T-1} (G_t - b_\omega(S_t))^2$
        Update π_θ *using* `Adam` $(\nabla_\theta L(\theta))$
        Update b_ω *using* `Adam` $(\nabla_\omega L(\omega))$
end procedure

Problem 1.3: N-step Advantage Actor Critic (20 pts)

Another important REINFORCE development was in choosing the baseline to be an approximate state value function, and to use this baseline to bootstrap estimates of the return. We call such a baseline a *critic*, and it carves out a family of algorithms depending on the degree of bootstrapping performed. See chapter 13 of the Sutton & Barto text for more details on this. In this part of the homework, we ask you to implement this algorithm for a variety of different bootstrapping strategies. In particular, we ask you to implement the version of N-step A2C given in Algorithm 3.

Once again, please use the provided network architectures and hyperparameters.

Please provide 3 separate figures similar to the previous problems for $N = 1, 10, 100$. For each value of N please run 5 IID trials as in the previous parts of the homework problem. Take $E = 3,500$ for each value of N.

Runtime Estimation: Note that our implementation of Algorithm 3 takes 11 minutes to complete a single trial with $E = 3,500, N = 100$ running on a 2020 MacBook Pro.

Algorithm 3 N-step Advantage Actor-Critic

procedure N-STEP ADVANTAGE ACTOR-CRITIC
  Start with actor network π_θ *and critic network* V_ω
  repeat:
    Generate an episode $S_0, A_0, R_0, \ldots, S_{T-1}, A_{T-1}, R_{T-1}$ *following* $\pi_\theta(\cdot)$
    for $t = 0, 1, \ldots, T-1$:
      $$V_{\text{end}} = \begin{cases} V_\omega(S_{t+N}) & \text{if } t+N < T \\ 0 & \text{otherwise} \end{cases}$$
      $$G_t = \left(\sum_{k=t}^{\min(t+N-1,T-1)} \gamma^{k-t} R_k\right) + \gamma^N V_{\text{end}}$$
    $L(\theta) = -\frac{1}{T}\sum_{t=0}^{T-1}(G_t - V_\omega(S_t))\ln\pi_\theta(A_t|S_t)$
    $L(\omega) = \frac{1}{T}\sum_{t=0}^{T-1}(G_t - V_\omega(S_t))^2$
    Update π_θ *using* `Adam` $(\nabla_\theta L(\theta))$
    Update V_ω *using* `Adam` $(\nabla_\omega L(\omega))$
end procedure

1.4 N-step A2C & REINFORCE with Baseline (4 pts)

How does N-step A2C relate to REINFORCE and REINFORCE with Baseline? (i.e. under what conditions do these algorithms become equivalent)

1.5 REINFORCE with & without Baseline (4 pts)

Does adding a baseline improve the performance? Please briefly explain why you think this happens.

Problem 2: Question Answering (12 pts)

1. SARSA and Q-learning converge to the same Q function but with different exploration strategies. Answer true/false and justify in 1-3 sentences.
2. Q-learning does not need exploration during training as it selects actions using the greedy policy over the computed Q function. Answer true/false and justify in 1-3 sentences.
3. In a finite MDP, consider we are given the value functions v_{π^*} that correspond to an optimal policy π^* and v_π that correspond to a policy π. The value $v_{\pi^*}(s)$ can be lower than $v_\pi(s)$ for some state s. Answer true/false and justify in 1-3 sentences.
4. Actor-critic methods cannot be used to optimize policies over discrete actions. Answer true/false and justify in 1-3 sentences.
5. Actor-critic methods use epsilon-greedy to explore. Answer true/false and justify in 1-3 sentences.
6. In a finite MDP, suppose we know the true Q function, q_π, for a specific policy π. Now suppose we find a state s for which $q_\pi(s, a_1) > q_\pi(s, a_2)$, but π at state s selects a_2 with probability 1.

   - Switching to a_1 for state s will provide us with a policy better than π.
   - Switching to a_1 for state s will increase the value for s but may decrease it for other states.
   - Switching to a_1 for state s will give us an optimal policy for this MDP.

   Select all that apply and justify.

Feedback

Feedback: You can help the course staff improve the course by providing feedback. What was the most confusing part of this homework, and what would have made it less confusing?

Time Spent: How many hours did you spend working on this assignment? Your answer will not affect your grade.

Guidelines on Implementation

This homework requires a significant implementation effort. It is hard to read through the papers once and know immediately what you will need to implement. We suggest you to think about the different components (e.g., network definition, network updater, policy runner, ...) that you will need to implement for each of the different methods that we ask you about, and then read through the papers having these components in mind. By this we mean that you should try to divide and implement small components with well-defined functionalities rather than try to implement everything at once. Much of the code and experimental setup is shared between the different methods so identifying well-defined reusable components will save you trouble.

A couple of points which may make life easier:

- Using a debugger like `pudb` or built in debuggers in IDEs are **extremely** useful as we start to consider more sophisticated implementations.
- Consider dumping your data (the matrix D) after every trial instead of generating the required plots in the same script.

Some hyperparameter and implementation tips and tricks:

- For efficiency, you should try to vectorize your code as much as possible and use **as few loops as you can** in your code. For example, in lines 5 and 6 of Algorithm 1 (REINFORCE) you should not use two nested loops. How can you formulate a single loop to calculate the cumulative discounted rewards? Hint: Think backwards!
- We recommend using a discount factor of $\gamma = 0.99$.

References

[1] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. How many random seeds? statistical power analysis in deep reinforcement learning experiments. *arXiv preprint arXiv:1806.08295*, 2018.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[3] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[4] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256, 1992.