CS 234: Assignment #3

Due date: Feb 20, 2025 at 6:00 PM (18:00) PST

These questions require thought but do not require long answers. Please be as concise as possible.

We encourage students to discuss in groups for assignments. **However, each student must finish the problem set and programming assignment individually, and must turn in her/his assignment.** We ask that you abide by the class Honor Code (see the course website), university Honor Code and the Computer Science department Honor Code, and make sure that all of your submitted work is done by yourself. If you have discussed the problems with others, please include a statement saying who you discussed problems with. Failure to follow these instructions will be reported to the Office of Community Standards. We reserve the right to run a fraud-detection software on your code.

Please review any additional instructions posted on the assignment page at http://web.stanford.edu/class/cs234/assignments.html. When you are ready to submit, please follow the instructions on the course website.

Note: We are now requiring students to typeset their homeworks.

Submission guidelines

You will be submitting the following on Gradescope:

1. PDF of this writeup
2. Latex submission .zip file. Do **not** include an enclosing top-level folder; the ZIP file must unpack directly to:

```
main.tex
img/
    hopper.png
    hopper_rlhf.png
    hopper_dpo.png
```

3. Code submission .zip file. See starter code README for instructions.

An introduction to reinforcement learning from human preferences

Reinforcement learning from human preferences (RLHF) was a key tool used in enabling the impressive performance of ChatGPT. However, the concept of RLHF came earlier, and is best known from a seminal paper "Deep reinforcement learning from human preferences." The goal of this assignment is to give you some hands-on experience with RLHF in the context of a robotics task in MuJoCo. You will also get a chance to explore the performance of Direct Preference Optimization (DPO), an alternative to RLHF that allows one to directly learn from preferences without inferring a reward model. You will implement, compare, and contrast several different approaches, including supervised learning (behavior cloning). These methods are all popular approaches used in large language model training, and many other machine learning tasks.

Environment setup:
Please see the starter code README for setting up the environment for this *entire* assignment!

1 Reward engineering (13 pts writeup)

In Assignment 2 you applied PPO to solve an environment with a provided reward function. The process of deriving a reward function for a specific task is called reward engineering. Each question in this problem shall be answered **concisely** with a few sentences.

1.1 Written Questions

(a) Why is reward engineering usually hard? What are potential risks that come with specifying an incorrect reward function? Provide an example of a problem and a reward function that appears to be adequate but may have unintended consequences.

(b) Read the description of the Hopper environment. Using your own words, describe the goal of the environment, and how each term of the reward functions contributes to encourage the agent to achieve it.

(c) By default, the episode terminates when the agent leaves the set of "healthy" states. What do these "healthy" states mean? Name one advantage and one disadvantage of this early termination.

1.2 Coding Questions

(d) Use the provided starter code to train a policy using PPO to solve the Hopper environment for 3 different seeds. Do this with and without early termination. **Each seed can take up to 90 minutes to run so please start this early!**

```
python ppo_hopper.py [--early-termination] --seed SEED
```

Attach here the plot of the episodic returns along training, with and without early termination. You can generate the plot by running

```
python plot.py --directory results --seeds 1,2,3 --output results/hopper.png
```

where `SEEDS` is a comma-separated list of the seeds you used. Comment on the performance in terms of training epochs and wall time. Is the standard error in the average returns high or low? How could you obtain a better estimate of the average return on Hopper achieved by a policy optimized with PPO?

(e) Pick one of the trained policies and render a video of an evaluation rollout.

```
# on linux
MUJOCO_GL=egl python render.py --checkpoint [PATH TO MODEL CHECKPOINT]

# on mac
python render.py --checkpoint [PATH TO MODEL CHECKPOINT]
```

Does the agent successfully complete the assigned task? Does it complete it in the way you would expect it to, or are you surprised by the agent behavior?

(f) Render another video with another policy. How do the two rollouts compare? Do you prefer one over the other?

2 Learning from preferences (19 pts writeup + 8 pts coding)

In the previous part you trained multiple policies from scratch and compared them at the end of training. In this section, we will see how we can use human preferences on two roll-outs to learn a reward function. We will follow the framework proposed by [1]. A reward function $r : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ defines a preference relation $\succ$ if for all trajectories $\sigma^i = (o_t^i, a_t^i)_{t=0,\ldots,T}$ we have that

$$\left(\left(o_0^1, a_0^1\right), \ldots, \left(o_T^1, a_T^1\right)\right) \succ \left(\left(o_0^2, a_0^2\right), \ldots, \left(o_T^2, a_T^2\right)\right)$$

whenever

$$r\left(o_0^1, a_0^1\right) + \cdots + r\left(o_T^1, a_T^1\right) > r\left(o_0^2, a_0^2\right) + \cdots + r\left(o_T^2, a_T^2\right).$$

Following the Bradley-Terry preference model, we can calculate the probability of one trajectory σ^1 being preferred over σ^2 as follows:

$$\hat{P}\left[\sigma^1 \succ \sigma^2\right] = \frac{\exp \sum \hat{r}\left(o_t^1, a_t^1\right)}{\exp \sum \hat{r}\left(o_t^1, a_t^1\right) + \exp \sum \hat{r}\left(o_t^2, a_t^2\right)},$$

where $\hat{r}$ is an estimate of the reward for a state-action pair. This is similar to a classification problem, and we can fit a function approximator to $\hat{r}$ by minimizing the cross-entropy loss between the values predicted with the above formula and ground truth human preference labels μ.

$$\mathcal{L}(\hat{r}) = -\sum_{(\sigma^1, \sigma^2, \mu) \in \mathcal{D}} \mu \log \hat{P}\left[\sigma^1 \succ \sigma^2\right] + (1-\mu) \log\Big(1 - \hat{P}\left[\sigma^1 \succ \sigma^2\right]\Big).$$

Once we have learned the reward function[1], we can apply any policy optimization algorithm (such as PPO) to maximize the returns of a model under it.

2.1 Written questions

We parameterize $\hat{r}(o, a)$ with a neural network θ, i.e. $\hat{r}_\theta(o, a)$. Now we want to derive $\nabla_\theta \mathcal{L}(\hat{r})$, the gradient of the loss w.r.t. θ.

(a) First write $\nabla_\theta \log \hat{P}[\sigma^1 \succ \sigma^2]$ as a function of $\nabla_\theta \hat{r}_\theta(o_i^j, a_i^j)$ where $i = 0, ..., T$ and $j = 1, 2$. You may find it useful to rewrite $\hat{P}$ as $\sigma(z_\theta)$ where σ is sigmoid and z_θ is some function of θ.

(b) Then write $\nabla_\theta \mathcal{L}(\hat{r})$ as a function of $\nabla_\theta \hat{r}_\theta(o_i^j, a_i^j)$ where $i = 0, ..., T$ and $j = 1, 2$.

2.2 Coding questions

In this problem we are trying to solve the same task as in the previous part, but this time we will learn a reward function from a dataset of preferences rather than manually specifying a reward function.

(c) You can load a sample from the provided **long preferences** dataset and render a video of the two trajectories using the following command

[1] Recent work on RLHF for reinforcement learning suggests that the pairwise feedback provided by humans on partial trajectories may be more consistent with regret, and that the learned reward function may be better viewed as an advantage function. See Knox et al. AAAI 2024 "Learning optimal advantage from preferences and mistaking it for reward." https://openreview.net/forum?id=euZXhbTmQ7

```
# on linux
MUJOCO_GL=egl python render.py --dataset data/long-prefs-hopper.npz --idx IDX

# on mac
python render.py --dataset data/long-prefs-hopper.npz --idx IDX
```

where `IDX` is an index into the preference dataset (if ommitted a sequence will be chosen at random). Bear in mind that each sequence in the dataset has 200 timesteps which results in an 8-second video.

Load 5 different samples from the dataset. For each, take note of which sequence was labeled as preferred (for the coming parts it is helpful to know that 0 means the first sequence was preferred, 1 means the second one, and 0.5 means neither is preferred over the other). Do you agree with the labels (that is, if shown the two trajectories, would you have ranked them the same way they appear in the dataset, knowing that we are trying to solve the Hopper environment)?

From your answers, how often do you estimate you agree with whoever ranked the trajectories? Based on this estimate, would you trust a reward function learned on this data?

(d) Implement the functions in the `RewardModel` class (`run_rlhf.py`), which is responsible for learning a reward function from preference data.

(e) Train a model using PPO and the learned reward function with 3 different random seeds. Plot the average returns for both the original reward function and the learned reward function and include it in your response.

```
# run rlhf
# Note: You may want to look at the original_scores.png
# and learned_scores.png generated in the output folder
python run_rlhf.py --seed SEED

# plot
python plot.py --rlhf-directory results_rlhf \
    --output results_rlhf/hopper_rlhf.png --seeds 1,2,3
```

Do the two correlate?

(f) Given enough preference pairs sampled under the Bradley-Terry model, can we recover the original reward function they were derived from?

(g) Pick one of the policies and render a video of the agent behavior at the end of training.

```
# on linux
MUJOCO_GL=egl python render.py --checkpoint [PATH TO MODEL CHECKPOINT]

# on mac
python render.py --checkpoint [PATH TO MODEL CHECKPOINT]
```

How does it compare to the behavior of the policies trained with the ground truth reward in problem 1? How does it compare to the demonstrations you've seen from the dataset?

3 Direct preference optimization (6 pts writeup + 19 pts coding)

In the previous question we saw how we could train a model based on preference data. In general you may be given access to a pre-trained model and the corresponding preference data. Learning a reward model and then optimizing a policy for that reward model can have some limitations. An alternative is Direct Preference Optimization (DPO): directly optimize a policy using the preference data, without learning a reward model. DPO in its original form is focused on bandit problems, and proposes optimizing the policy using the following loss:

$$\mathcal{L}_{\text{DPO}}\left(\pi_\theta; \pi_{\text{ref}}\right) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta\left(y_w \mid x\right)}{\pi_{\text{ref}}\left(y_w \mid x\right)} - \beta\log\frac{\pi_\theta\left(y_l \mid x\right)}{\pi_{\text{ref}}\left(y_l \mid x\right)}\right)\right],$$

where x is the context/state, and y_w and y_l are two actions (in LLM terms, responses) sampled from the reference policy π_{ref}. y_w is the response/action that was preferred (the "winning" response/action) and y_l is the other ("losing") response. π_θ is the policy to be learned, and σ is the sigmoid function.
In this part of the assignment you will get to use DPO for learning policies for the same MuJoCo task. While DPO is designed for bandit problems, we will use a simple adaptation of it to handle our RL setting.
First, to provide some context, consider the general approach for RLHF for text generation:

1. Train a large language model (LLM) to do next token prediction given a context (the tokens that came previously).
2. Given a fixed context x, generate possible next token sequence predictions y_1 and y_2, and store the triple (x, y_1, y_2).
3. Ask human supervisors to rank y_1 and y_2 given x according to individual preference.
4. Update the LLM to maximize the probability of giving the preferred answers using reinforcement learning.

In a similar way, given an observation x we could have two ranked sequences of actions $a^1_{1:T}$ and $a^2_{1:T}$, train the model to generate the preferred sequence of actions, and then execute them all[2]. If the length of the generated action sequence is equal to the environment time horizon, this is called open-loop control. However, this approach lacks robustness, since the plan of actions will not change in response to disturbances or compounding errors. Instead we will use the common approach of receding horizon control (often also called model predictive control), where a multi-step action plan is computed, the first action in that plan is taken, and then a new action plan is computed. MPC/RHC is well known to improve performances, since it allows the agent to react to disturbances.

Note that there are other algorithms that directly tackle learning from preferences in the multi-step RL setting, such as later work on Contrastive Preference Learning (CPL). Indeed CPL showed that DPO can be viewed as a special case of their setting, for the bandit setting. In this homework we focus on DPO because it is widely used in LLM training, and is a simpler setting which still provides key insights into the difference between RLHF and learning policies directly from preferences.

3.1 Coding questions

In this coding question you will need to modify the `run_dpo.py` file. You do not need to modify any other files.

[2]To understand why we are considering sequences of actions rather than a single action for the next time, recall that 50 actions corresponded to 2 seconds of video. If you found it difficult to rank a sequence of 50 actions based on such a short video, imagine ranking the effect of a single action!

(a) Implement the `ActionSequenceModel` class instance methods. When called, the model should return a probability distribution for the actions over the number of next time steps specified at initialization. Use a multivariate normal distribution for each action, with mean and standard deviation predicted by a neural network (see the starter code for more details).[3]

(b) Implement the `update` method of the `SFT` class. This class will be used to pre-train a policy on the preference data by maximizing the log probabilities of the preferred actions given the observations in the dataset.

(c) Implement the `update` method of the `DPO` class. This should minimize the DPO loss described above.

(d) Run SFT and DPO for 3 different random seeds each, and plot the evolution of returns over time.

```
# run SFT
python run_dpo.py --seed SEED --algo sft

# run DPO
python run_dpo.py --seed SEED --algo dpo

# plot
python plot.py --dpo-directory results_dpo \
    --output results_dpo/hopper_dpo.png --seeds 1,2,3
```

Include that plot in your response. How does it compare to the returns achieved using RLHF? Comment on the pros and cons of each method applied to this specific example.

(e) Take the best DPO training run and render videos of episodes generated by the SFT policy and the DPO policy. The following command will render 10 episodes from SFT DPO side-by-side. The left one is from SFT policy while the right one is from the DPO policy.

```
# Note: --checkpoint argument should be the path to the dpo.pt the script will
# find a sft.pt with the same seed following our default naming convention

# on linux
MUJOCO_GL=egl python render.py --dpo --checkpoint [PATH TO DPO CHECKPOINT]

# on mac
python render.py --dpo --checkpoint [PATH TO DPO CHECKPOINT]
```

How do they compare? Note that both of them may not look great because they are only trained on a small amount of offline data and have not interacted with the environment during training. But you may still observe that one is slightly better than the other in some videos.

4 Best Arm Identification in Multi-armed Bandit (25 pts writeup)

In many experimental settings we are interested in quickly identifying the "best" of a set of potential interventions, such as finding the best of a set of experimental drugs at treating cancer, or the website design that maximizes user subscriptions. Here we may be interested in efficient pure exploration, seeking to quickly identify the best arm for future use.

In this problem, we bound how many samples may be needed to find the best or near-optimal intervention. We frame this as a multi-armed bandit with rewards bounded in $[0, 1]$. Recall a bandit problem can be

[3] We have prepared a notebook to illustrate the behavior of `torch.distributions.Independent`.

considered as a finite-horizon MDP with just one state ($|\mathcal{S}| = 1$) and horizon 1: each episode consists of taking a single action and observing a reward. In the bandit setting – unlike in standard RL – the action (or "arm") taken does not affect the distribution of future states. We assume a simple multi-armed bandit, meaning that $1 < |\mathcal{A}| < \infty$. Since there is only one state, a policy is simply a distribution over actions. There are exactly $|\mathcal{A}|$ different deterministic policies. Your goal is to design a simple algorithm to identify a near-optimal arm with high probability.

We recall Hoeffding's inequality: if $X_1, \ldots, X_n$ are i.i.d. random variables satisfying $0 \leq X_i \leq 1$ with probability 1 for all i, $\overline{X} = \mathbb{E}[X_1] = \cdots = \mathbb{E}[X_n]$ is the expected value of the random variables, and $\widehat{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ is the sample mean, then for any $\delta > 0$ we have

$$\Pr\left(|\widehat{X} - \overline{X}| > \sqrt{\frac{\log(2/\delta)}{2n}}\right) < \delta. \tag{1}$$

Assuming that the rewards are bounded in $[0, 1]$, we propose this simple strategy: pull each arm n_e times, and return the action with the highest average payout $\widehat{r}_a$. The purpose of this exercise is to study the number of samples required to output an arm that is at least ϵ-optimal with high probability. Intuitively, as n_e increases the empirical average of the payout $\widehat{r}_a$ converges to its expected value $\overline{r}_a$ for every action a, and so choosing the arm with the highest empirical payout $\widehat{r}_a$ corresponds to approximately choosing the arm with the highest expected payout $\overline{r}_a$.

(a) We start by bounding the probability of the "bad event" in which the empirical mean of some arm differs significantly from its expected return. Starting from Hoeffding's inequality with n_e samples allocated to every action, show that:

$$\Pr\left(\exists a \in \mathcal{A} \quad \text{s.t.} \quad |\widehat{r}_a - \overline{r}_a| > \sqrt{\frac{\log(2/\delta)}{2n_e}}\right) < |\mathcal{A}|\delta. \tag{2}$$

Note that, depending on your derivation, you may come up with a tighter upper bound than $|\mathcal{A}|\delta$. This is also acceptable (as long as you argue that your bound is tighter), but showing the inequality above is sufficient.

(b) After pulling each arm (action) n_e times our algorithm returns the arm with the highest empirical mean:

$$a^{\dagger} = \arg\max_a \widehat{r}_a \tag{3}$$

Notice that $a^{\dagger}$ is a random variable. Let $a^{\star} = \arg\max_a \overline{r}_a$ be the true optimal arm. Suppose that we want our algorithm to return at least an ϵ-optimal arm with probability at least $1 - \delta'$, as follows:

$$\Pr\left(\overline{r}_{a^{\dagger}} \geq \overline{r}_{a^{\star}} - \epsilon\right) \geq 1 - \delta'. \tag{4}$$

How accurately do we need to estimate each arm in order to pick an arm that is ϵ-optimal? Then derive how many total samples we need total (across all arms) to return an ϵ-optimal arm with prob at least 1- δ' (that satisfies Equation 4). Express your result as a function of the number of actions, the required precision ϵ and the failure probability δ'.

(c) (Optional challenge, will not be graded) The above derivation only assumed the outcomes were bounded between 0 and 1. In practice people often assume outcomes are drawn from a parametric distribution, and under mild assumptions, one can use the central limit theorem to assume the average outcomes for an arm will follow a normal distribution. Repeat the above analysis under this assumption, for a multi-armed bandit with two arms. Is the resulting number of samples significantly smaller under these assumptions? In real settings it is often very expensive to run experiments. Do you think the method and bound derived in (a-b) would be preferable to making a normal assumption and why or why not?

5 Stated vs. Revealed Preferences (4 pts writeup)

Context: You are designing a reinforcement learning algorithm to power a news recommendation app. The app has two types of data about users:

- User Profiles: Users specify what news topics they are interested in reading about, and their preferred format (videos, podcasts, etc).
- Interaction data: what news articles the user lingered on, how long they lingered on it, and if and who they shared articles/podcasts.

Additionally, the app has metadata about users, such as:

- Location, browser type, etc.
- Engagement Metrics: Frequency of app usage, average time spent on the app
- Revenue Potential: Whether they are paying users and their likelihood to subscribe to premium services.

1. Which of the given data about users represent the stated preferences of the user, and which represents the revealed preferences?
2. What reward function might a company pick?
3. Assume the company wants to optimize its news feed to cater to user's preferences. What are the ethical considerations of prioritizing the user's stated preferences vs revealed preferences?
4. Suggest a way to incorporate exploration to test whether a user's preferences (stated or revealed) might evolve over time.

References

[1] Paul F Christiano et al. "Deep Reinforcement Learning from Human Preferences". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd Paper.pdf.