CS 234: Assignment #3

Due date: May 17, 2024 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 university Honor Code and that of the Computer Science department, 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.

An introduction to reinforcement learning from human preferences

The goal of this assignment is to give you some hands-on experience with reinforcement learning from human preferences (RLHF). You will implement, compare, and contrast several different approaches for a robotics task in MuJoCo.

1 Reward engineering (6 pts writeup + 7 pts coding)

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.

1.1 Written Questions (6 pts)

(a) (2 pts) (**written**) 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) (2 pts) (**written**) 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. (**Optional:** Do you agree with this reward function? Would you change it? How?)

(c) (2 pts) (**written**) 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 (7 pts)

(d) (2 pts) (**coding**) 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.

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

(e) (2 pts) (**written**) 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 SEEDS
```

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?

(f) (2 pts) (**written**) Pick one of the trained policies and render a video of an evaluation rollout.

```
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?

(g) (1 pts) (**written**) Render another video for another policy. How do the two rollouts compare? Do you prefer one over the other?

2 Learning from preferences (5 pts writeup + 25 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 [2]. A reward function $r : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ defines a preference relation $\succ$ if for all trajectories $\sigma^i = (o^i_t, a^i_t)_{t=0,\dots,T}$ we have that

$$\left(\left(o^1_0, a^1_0\right), \dots, \left(o^1_T, a^1_T\right)\right) \succ \left(\left(o^2_0, a^2_0\right), \dots, \left(o^2_T, a^2_T\right)\right)$$

whenever

$$r\left(o^1_0, a^1_0\right) + \dots + r\left(o^1_T, a^1_T\right) > r\left(o^2_0, a^2_0\right) + \dots + r\left(o^2_T, a^2_T\right).$$

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

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

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 $\mu(1)$ and $\mu(2)$:

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

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.

[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

2.1 Written questions (5 pts)

(a) (5 pt) Let $\hat{r}(o,\ a) = \phi_w(o,\ a)$. Calculate $\nabla_w \operatorname{loss}(\hat{r}(o,\ a))$.

2.2 Coding questions (25 pts)

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.

(b) (5 pt) Load one of the samples from the preference dataset we provide you, and render a video of the two trajectories using the following command

```
python render.py --dataset [PATH TO PREFERENCE DATASET FILE] --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 25 timesteps, which means that the resulting videos will have 0.2 seconds. Take note of which sequence was labeled as preferred (this information will appear in the name of the generated videos, but 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 label (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)?

(c) (3 pt) Repeat the previous question for 4 more samples, keeping track of whether you personally agree with the dataset preference label. Use this to estimate how much you agree with whoever ranked the trajectories. How much did you get? Based on this agreement estimate, would you trust a reward function learned on this data?

(d) (8 pt) Implement the functions in the `RewardModel` class (`run_rlhf.py`), which is responsible for learning a reward function from preference data.

(e) (5 pt) 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.

```
python plot.py --rlhf-directory results_rlhf --output \
    results_rlhf/hopper_rlhf.png
```

Do the two correlate?

(f) (1 pt) Is the learned reward function identifiable?

(g) (3 pt) Pick one of the policies and render a video of the agent behavior at the end of training.

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

How does it compare to the behavior of the agent generated by policies trained from scratch? How does it compare to the demonstrations you've seen from the dataset?

3 Direct preference optimization (25 pts coding)

In the previous question we saw how we could train a model based on preference data. However, suppose you are given a pre-trained model and the corresponding preference data. Ideally, you would like to optimize the model directly on the preference data, instead of having to learn a reward function, and then run PPO on it. That is the idea behind direct preference optimization (DPO) [3]. The algorithm proposed in the original paper allows us to skip the reward learning and reinforcement learning steps, and optimize the model directly from preference data by optimizing 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 π_{ref} is the policy from which we sampled y_w and y_l given x, π_θ is the policy we are optimizing, and σ is the sigmoid function.
We will use DPO in this question as well. To provide some context, let us 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 are going to adapt this into a more robust scheme by training our policy to predict a sequence of actions for the next T time steps (where T is the length of the trajectories in our preference dataset), but only take the first action in the plan generated by the policy. In this way, we re-plan our actions at every time step, ensuring the ability to respond to disturbances.

3.1 Coding questions (25 pts)

(a) (9 pts) 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) (3 pts) 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) (5 pts) Implement the `update` method of the `DPO` class. This should minimize the DPO loss described above.

[2]To understand why we are considering sequences of actions rather than a single action for the next time, recall that 25 actions corresponded to 0.2 seconds of video. If you found it difficult to rank a sequence of 25 actions based on such a short video, imagine ranking the effect of a single action!

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

(d) (5 pts) Run DPO for 3 different random seeds (you may want to tweak the number of DPO steps to get better results), and plot the evolution of returns over time.

```
python plot.py --dpo-directory results_dpo --output \
    results_dpo/hopper_dpo.png
```

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) (3 pts) Render a video of an episode generated by the pre-trained policy, and a video of an episode generated by the policy tuned by DPO.

```
python render.py --dpo --checkpoint [PATH TO MODEL CHECKPOINT]
```

How do they compare?

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

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 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 $X_1, \dots, X_n$ are i.i.d. random variables satisfying $0 \le X_i \le 1$ 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) (10 pts) 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) (15 pts) 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) (0 pts) (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 Challenges of RLHF (5 pts)

RLHF and DPO leverage humans providing pairwise comparisons.

(a) (2 pts) Who are human raters likely to be that are employed to provide pairwise preferences for generic LLM RLHF/DPO (in terms of socioeconomic profiles)? What ethical issues may this have, and what impact may this have on the resulting trained models?

(b) (1 pt) Would RLHF be a good way to train a system to make better medical diagnoses? Why or why not?

(c) (2 pts) There are a number of limitations of RLHF. Please describe one here (different than parts a and b). You may find it interesting to read "Open Problems and Fundamental Limitations of Reinforcement Learning from Human Feedback" and you are welcome to describe challenges listed there, or make up your own.