XCS234 Assignment 4

Due Sunday, October 12 at 11:59pm PT.

Guidelines

1. If you have a question about this homework, we encourage you to post your question on our Slack channel, at http://xcs234-scpd.slack.com/
2. Familiarize yourself with the collaboration and honor code policy before starting work.
3. For the coding problems, you must use the packages specified in the provided environment description. Since the autograder uses this environment, we will not be able to grade any submissions which import unexpected libraries.

Submission Instructions

Written Submission: Some questions in this assignment require a written response. For these questions, you should submit a PDF with your solutions online in the online student portal. As long as the PDF is legible and organized, the course staff has no preference between a handwritten and a typeset LaTeX submission. If you wish to typeset your submission and are new to LaTeX, you can get started with the following:

- Type responses only in `submission.tex`.
- Submit the compiled PDF, **not** `submission.tex`.
- Use the commented instructions within the `Makefile` and `README.md` to get started.

Coding Submission: Some questions in this assignment require a coding response. For these questions, you should submit only the `src/submission.py` file in the online student portal. For further details, see Writing Code and Running the Autograder below.

Honor code

We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions independently, and without referring to written notes from the joint session. In other words, each student must understand the solution well enough in order to reconstruct it by him/herself. In addition, each student should write on the problem set the set of people with whom s/he collaborated. Further, because we occasionally reuse problem set questions from previous years, we expect students not to copy, refer to, or look at the solutions in preparing their answers. It is an honor code violation to intentionally refer to a previous year's solutions. For SCPD classes, it is also important that students avoid opening pull requests containing their solution code on the shared assignment repositories. More information regarding the Stanford honor code can be found at https://communitystandards.stanford.edu/policies-and-guidance/honor-code.

Writing Code and Running the Autograder

All your code should be entered into `src/submission.py`. When editing these files, please only make changes between the lines containing `### START_CODE_HERE ###` and `### END_CODE_HERE ###`. Do NOT make changes to files other than `src/submission.py`.

The unit tests in `src/grader.py` (the autograder) will be used to verify a correct submission. Run the autograder locally using the following terminal command within the `src/` subdirectory:

```
$ python grader.py
```

There are two types of unit tests used by the autograder:

- `basic`: These tests are provided to make sure that your inputs and outputs are on the right track, and can be run locally.

- `hidden`: These unit tests are NOT visible locally. These hidden tests will be run when you submit your code to the Gradescope autograder via the online student portal, and will provide feedback on how many points you have earned. These tests will evaluate elements of the assignment, and run your code with more complex inputs and corner cases. Just because your code passed the basic local tests does not necessarily mean that they will pass all of the hidden tests.

For debugging purposes, you can run a single unit test locally. For example, you can run the test case `3a-0-basic` using the following terminal command within the `src/` subdirectory:

```
$ python grader.py 3a-0-basic
```

Before beginning this course, please walk through the Anaconda Setup for XCS Courses to familiarize yourself with the coding environment. Use the env defined in `src/environment.yml` to run your code. This is the same environment used by the online autograder.

Test Cases

The autograder is a thin wrapper over the python `unittest` framework. It can be run either locally (on your computer) or remotely (on SCPD servers). The following description demonstrates what test results will look like for both local and remote execution. For the sake of example, we will consider two generic tests: `1a-0-basic` and `1a-1-hidden`.

Local Execution - Basic Tests

When a basic test like `1a-0-basic` passes locally, the autograder will indicate success:

```
----- START 1a-0-basic:  Basic test case.
----- END 1a-0-basic [took 0:00:00.000062 (max allowed 1 seconds), 2/2 points]
```

When a basic test like `1a-0-basic` fails locally, the error is printed to the terminal, along with a stack trace indicating where the error occurred:

```
----- START 1a-0-basic:  Basic test case.
<class 'AssertionError'>
{'a': 2, 'b': 1} != None   <-- This error caused the test to fail.
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/grader.py", line 23, in test_0   <-- In this case, start your debugging in line 23 of grader.py.
    submission.extractWordFeatures("a b a"))
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
----- END 1a-0-basic [took 0:00:00.003809 (max allowed 1 seconds), 0/2 points]
```

Remote Execution

Basic and hidden tests are treated the same by the remote autograder, however the output of hidden tests will only appear once you upload your code to GradeScope. Here are screenshots of failed basic and hidden tests. Notice that the same information (error and stack trace) is provided as the in local autograder, now for both basic and hidden tests.

1a-0-basic) Basic test case. (0.0/2.0)

```
<class 'AssertionError'>:  {'a': 2, 'b': 1} != None
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/autograder/source/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/grader.py", line 23, in test_0
    submission.extractWordFeatures("a b a"))
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
```

Just like in the local autograder, this error caused the test to fail.

Just like in the local autograder, start your debugging in line 23 of grader.py.

1a-1-hidden) Test multiple instances of the same word in a sentence. (0.0/3.0)

```
<class 'AssertionError'>:  {'a': 23, 'ab': 22, 'aa': 24, 'c': 16, 'b': 15} != None
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/autograder/source/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/grader.py", line 31, in test_1
    self.compare_with_solution_or_wait(submission, 'extractWordFeatures', lambda f: f(sentence))
  File "/autograder/source/graderUtil.py", line 183, in compare_with_solution_or_wait
    self.assertEqual(ans1, ans2)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
```

This error caused the test to fail.

Start your debugging in line 31 of grader.py.

Finally, here is what it looks like when basic and hidden tests pass in the remote autograder.

1a-0-basic) Basic test case. (2.0/2.0)

1a-1-hidden) Test multiple instances of the same word in a sentence. (3.0/3.0)

0 Introduction

In this assignment, we will be using the Hopper environment where we will first derive reward functions for specific tasks, moving then to adding human preferences to learn the reward function and then ending the process with an end-to-end approach, where we skip the reward learning and reinforcement learning steps, and optimize the model directly from preference data.

Deliverables:

For this assignment, please submit the following files to gradescope to receive points for coding questions:

- `src/submission/__init__.py`
- `src/submission/rlhf.py`
- `src/submission/dpo.py`

1 Reward engineering

In Assignment 3 you applied Policy Gradient methods to solve an environment with a provided reward function. The process of deriving a reward function for a specific task is called reward engineering.

(a) **[1 point (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) **[1 point (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) **[1 point (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.

(d) **[0 points (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) **[1 point (Written)]**

The plot of the episodic returns along training, with and without early termination is seen below. You can generate the plot by running

```
$ python plot.py --directory results --output ppo_hopper.png --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?

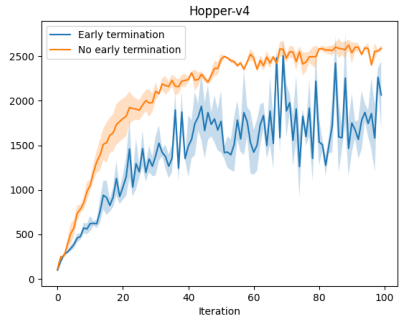

Figure 1: PPO results

(f) **[1 point (Written)]**

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

```
$ python render.py --checkpoint results/[MODEL]/model.zip
```

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 point (Written)]**

Render another video for another policy. How do the two rollouts compare? Do you prefer one over the other?

2 Learning from Preferences

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 $\sigma^i = (o_t^i, a_t^i)_{t=0,\ldots,T}$ 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 [1], we can calculate the probability of one trajectory σ^1 being preferred over σ^2 as follows:

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

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 $\mu(1)$ 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.

(a) **[4 points (Written)]**

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

(b) **[2 points (Written)]**

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.

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 ./data/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 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) **[2 points (Written)]**

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?

[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

(d) **[8 points (Coding)]**

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

Afterwards run the following command to train the reward model using the provided preference data:

```
$ python run_rlhf.py
```

(e) **[3 points (Written)]**

Train a model using PPO and the learned reward function with 3 different random seeds. The expected plots of the average returns for both the original reward function and the learned reward function are given below.

```
$ python plot.py --rlhf-directory results_rlhf --output results_rlhf/hopper_rlhf.png --seeds
    0
```

Do the two correlate?

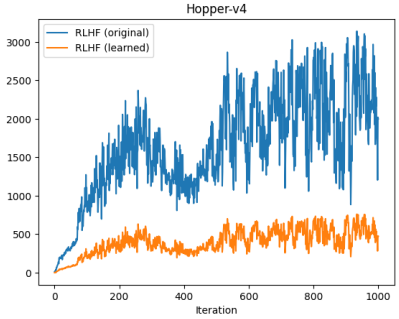

Figure 2: RLHF results

(f) **[2 points (Written)]**

Is the learned reward function identifiable?

(g) **[2 points (Written)]**

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

```
$ python render.py --checkpoint ./results_rlhf/[MODEL]/model.zip
```

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 (DPO)

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.

(a) **[18 points (Coding)]**

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) **[4 points (Coding)]**

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) **[4 points (Coding)]**

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

Afterwards run the following command to train the DPO model using the provided preference data:

```
$ python run_dpo.py
```

[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) **[2 points (Written)]**

Run DPO for 3 different random seeds (you may want to tweak the number of DPO steps to get better results). The plots of the evolution of returns over time are illustrated below.

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

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

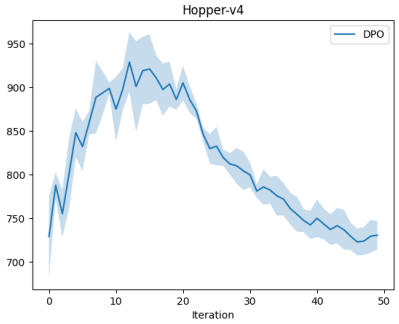

Figure 3: DPO results

(e) **[2 points (Written)]**

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 ./results_dpo/[MODEL]/model.pt
```

How do they compare?

4 Challenges of RLHF

RLHF and DPO leverage humans providing pairwise comparisons.

(a) **[2 points (Written)]**

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) **[2 points (Written)]**

Would RLHF be a good way to train a system to make better medical diagnoses? Why or why not?

(c) **[2 points (Written)]**

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.

This handout includes space for every question that requires a written response. Please feel free to use it to handwrite your solutions (legibly, please). If you choose to typeset your solutions, the —README.md— for this assignment includes instructions to regenerate this handout with your typeset LaTeX solutions.

1.a

1.b

1.c

1.e

1.f

1.g

2.a

2.b

2.c

2.e

2.f

2.g

3.d

3.e

4.a

4.b

4.c