CS 4/5789 - Programming Assignment 4

April 28, 2025

This assignment is due on **Tue May 6, 2025** at 11:59pm.

Section 0: Requirements

This assignment uses Python 3.11 and requires PyTorch, NumPy, Gymnasium, Matplotlib, and YAML. Install the exact versions with:

```
pip install -r requirements.txt
```

Do not add any additional imports; all core functionality must be implemented from scratch. Please make sure that you are using the same python version. You can create a conda environment if that helps. **You may need to install `swig` first before running the regular install!**

Section 1: Direct Preference Optimization (DPO)

In this section we describe our implementation of Direct Preference Optimization (DPO) for the Swimmer-v5 environment. As mentioned in lecture, DPO is a method for off policy RL for optimizing rewards based off the Bradley Terry model. However, it does not always have to be the case. Indeed, for this assignment, we will sample two trajectories and consider them to be the pairwise comparison. We can then sample a label (preference ordering) for the two trajectories based on the BT model and their true reward (r^*). After we collect a dataset, we can use this dataset to train a new policy that optimizes the offline preference data using the standard DPO framework.

1.1 Background and Setting

We will use the OpenAI Gymnasium Swimmer-v5 simulator, in which the agent controls a 2-link swimmer with continuous action space $\mathcal{A} = [-1, 1]^2$. At each timestep the policy outputs a torque vector applied to the joints. The goal is to maximize forward velocity over an episode.

1.2 Assignment Overview

There are three main parts for this assignment: (1) You are to collect a dataset of trajectories from a provided pretrained policy and label them using the method outlined. Then (2) you are to implement DPO using the skeleton code provided and train this model to superior performance, just with your offline data! (3) You are going to repeat this loop, and see how the performance of the model changes as you query online data.

1.3 Gaussian Policy

We implement a Gaussian policy `continuous_policy` in `utils.py`. The network architecture is:

- Two fully-connected hidden layers of 64 units each with ReLU activations
- A linear output layer producing the mean $\mu(x) \in \mathbb{R}^2$
- A learned log-standard-deviation vector $\log \sigma \in \mathbb{R}^2$

We provide the `forward` function. However, you need to implement the following in `model.py`:

- `_gaussian_logp`: computes the log-probability of actions under $\mathcal{N}(\mu, \sigma^2 I)$
- `sample_action`: samples an action $a \sim \pi_\theta(\cdot|x)$ and optionally returns its probability
- `compute_log_likelihood`: computes the log-probability of actions under the policy and states: $\log \pi_\theta(a|s)$

1.4 Batch Trajectory Collection

In order to collect the offline data, you'll need to start with a pretrained policy. We have provided a decent pretrained policy that you can use to collect your offline data. As mentioned above, the steps for this are:

1. Run the current policy for `batch_size` episodes
2. Record the trajectories and their rewards
3. Pair the trajectories according (in any way you want)
4. Label the trajectories according to the BT model
5. Save the dataset to `pair_data.pt` using `torch.save` (**The format for saving the data must be parsed by the DPO dataloader provided since this is what will be used in our tests**)

Using the model provided, you should have a average reward of around 180 (or within 15 reward). How you implement this is up to you, but we suggest you look into vectorized environments:

```
env_fns = [lambda: gym.make("Swimmer-v5") for _ in range(num_envs)]
env = SyncVectorEnv(env_fns)
```

and the provided code as a starter but you don't have to use it.

1.5 Direct Preference Optimization

Now that we have a dataset of trajectories, we want to use DPO to optimize this policy. We provide a skeleton for this in `dpo.py`. The main todo is implement `train`. For now, ignore the iterative DPO flag. We recommend that you use the square version of the loss:

$$\arg\min_{\pi_\theta \in \Pi} \frac{1}{B} \sum_{(y_w, y_l) \in B} \left(\beta \left(\log(\frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)}) - \log(\frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)} \right) - z \right)^2$$

Although you are welcome to use the logistic version.

You should be able to reach an average of 210 reward. **Save this checkpoint as `dpo.pt` and submit this file.**

1.6 Iterative DPO

Now, take into account the iterative DPO flag to perform iterative DPO. For iterative DPO, you need to collect data with the current policy (hopefully using your data collection code from above) and then use this data to update the policy. You should do this only a few times.

This should be able to achieve a reward of > 270. **Save this checkpoint as `dpo_iterative.pt` and submit this file.**

1.7 Finishing Up

Now that you have experimented and implemented the above, please answer the following question:

1. If we were to use iterative DPO forever, what would happen (other than the reward hitting the max possible value)? ie. What could go wrong and cause early convergence?

Section 2: Submission

Now, submit your code and answers in the following format

Also, note that there will be a 20 minute timeout.

```
answers.pdf
batch_collection.py
utils.py
dpo.py
model.py
dpo_iterative.pt
dpo.pt
hparam.yaml
```