Homework 4 : Goal Conditioned and Hierarchical RL

Introduction

This assignment will cover Goal Conditioned Reinforcement Learning (GCRL). You will learn how to train a goal conditioned policy and how to use a pretrained goal conditioned policy to train a hierarchical RL agent.

Part 1: GCRL

We will be building on the code that we have implemented in the first two assignments. We will use the policy gradient algorithm provided (pg). This algorithm will be used to train our goal-conditioned policies.

In order to implement the goal conditionned wrapper, you will be writing new code in the following files:

- `hw4/infrastructure/gclr_wrapper.py`

For this assignment a new environment will be used based on an extension to the ant environment.

Implementation

The first phase of the assignment is to implement a working GCRL method where the policy is also conditioned on a goal $\pi(a|s, g, \theta)$. For this assignment, an *environment wrapper* is used to allow us to modify another RL environment to create a goal conditioned version of the task. Look for the `# TODO` markers in the files listed above for detailed implementation instructions.

When we use goal-conditioned RL we need to modify the observation from the original environment and append the goal. If s was the original state and g is the generated goal your *environment wrapper* should construct a new state $s_{mod} \leftarrow [s, g]$ which results in the policy $\pi(a|s_{mod}, \theta)$

Goal Distributions For this part of the assignment and the first question, generate goals from a uniform distribution with bounds ant = [-4,20], reacher [-0.3, 0.3].

```
def GoalConditionedEnv(object):

  def __init__()
    ## You need to update the size of the self.observation_space to include
    the goal

  def reset():
```

```
    # Add code to generate a goal from a distribution

  def step():
    ## Add code to compute a new goal-conditioned reward

  def createState():
    ## Add the goal to the state
```

Once you implement the wrapper, answering some of the questions may require changing hyperparameters, neural network architectures, and the game, which should be done by changing the command line arguments passed to `run_hw4_gcrl.py`.

To determine if your implementation of GCRL is correct, you should run it with the default hyperparameters on the https://www.gymlibrary.dev/environments/mujoco/reacher/ game for 1 million steps using the command below. Our reference solution gets an average reward of -0.3 in this timeframe. If it takes much longer than that, there may be a bug in your implementation.

Next, you should run your code on the modified Ant environment to evaluate how well your solution works. After *1 million steps* your agent should achieve average reward close to -5.

Evaluation

Once you have a working implementation of GCLR, you should prepare a report. The report should consist of one figure for each question below. You should turn in the report as one PDF and a zip file with your code. If your code requires special instructions or dependencies to run, please include these in a file called `README` inside the zip file. Also provide the log file of your run on Gradescope named as `reacher_1.csv` and `ant_1.csv`.

Question 1: basic GCRL performance. [3 pts] Include a learning curve plot showing the performance of your implementation on `reacher` and *Ant*. The x-axis should correspond to the number of time steps (consider using scientific notation) and the y-axis should show the average per-epoch reward as well as the best mean reward so far. *You should also add logging for the distance between the state and the goal (this is the intrinsic reward you have implemented) and the success.* Make sure they are logged to the `data` folder. Include a plot of these metrics using Tensorboard, wandb.com, etc as in previous assignments or using excel by using the csv file. Be sure to label the y-axis since we need to verify that your implementation achieves similar reward as ours. You should not need to modify the default hyperparameters in order to obtain good performance, but if you modify any of the parameters, list them in the caption of the figure. The final results should use the following experiment name:

```
python ift6131/run_hw4_gcrl.py env_name=reacher exp_name=q1_reacher
python ift6131/run_hw4_gcrl.py env_name=antmaze exp_name=q1_ant
```

Question 2: GCRL with normal distribution. [2 pts] Replace the uniform distribution used for sampling goals with a normal distribution. Compare the **goal distance** performance between using the uniform distribution and the normal distribution. Since there is considerable variance between runs, you must run at least three random seeds for both distributions. You may use `reacher-v1` and `ant-v1` for this question. The final results should use the following experiment names:

```
python run_hw4_gcrl.py env_name=reacher exp_name=q1_reacher_normal goal_dist=
    normal
python run_hw4_gcrl.py env_name=antmaze exp_name=q1_ant_normal goal_dist=
    normal
```

Question 3: Relative Goals [2 pts] So far, we have specified goals in a *global state space*. Using a global state space makes it more complex for the agent to understand how to reach the goal. To make training easier, we can generate and provide the goal in an agent-relative coordinate frame $g_{mod} \leftarrow g - s$. Use this new *relative* goal location g_{mod} as the goal you add to the state $s_mod \leftarrow [s, g_{mod}]$ In addition change, the distribution the goals are generated from to $g \leftarrow N(agent\_position, 3)$ Train the policies again using this representation and plot the distance to the **generated goals**.

```
python run_hw4_gcrl.py env_name=reacher exp_name=q3_reacher_normal_relative
    goal_dist=normal goal_frequency=10 goal_rep=relative
python run_hw4_gcrl.py env_name=antmaze exp_name=q3_ant_normal_relative
    goal_dist=normal goal_frequency=10 goal_rep=relative
```

Submit the run logs for all the experiments above. In your report, make a single graph that averages the performance across three runs for both Uniform and Normal goal distributions. See `scripts/read_results.py` for an example of how to read the evaluation returns from Tensorboard logs.

Question 4: Changing goals during the episode. [3pts] Often, we want a GCRL agent that can learn to achieve many goals in one episode. For this, we will create another wrapper that uses a controlled goal update frequency. We will fill in the missing parts of the *class GoalConditionedWrapperV2()*.

```
def GoalConditionedWrapperV2(object):

  def __init__()
    ## You need to update the size of the self.observation_space to include
    the goal

  def sampleGoal():
    ## This will sample a goal from the distribution

  def reset():
    # Use sampleGoal to get a new goal
```

```
def step():

  ## Now we need to use the goal for k steps and after these k steps sample
   a new goal
  ## Add code to compute a new goal-conditioned reward

def createState():
  ## Add the goal to the state
```

Note: You can subclass the code from Q1 and Q2. This may save you some time as you can reuse some methods.

```
python run_hw4_gcrl.py env_name=reacher exp_name=q3_reacher_normal goal_dist=
    normal goal_frequency=10
python run_hw4_gcrl.py env_name=antmaze exp_name=q3_ant_normal goal_dist=
    normal goal_frequency=10
```

You should try different values $[5, 10]$ for goal_frequency to see how it affects performance.

Saving your Policy For the next part of the assignment you will need to load the policies you have trained. Save and load the policy

Part 2: Hierarchial RL

Implement a Hierarchical policy to plan longer paths across the environment.

In order to implement Hierarchical Reinforcement Learning (HRL), you will be writing new code in the following files:

- `hw4/infrastructure/hrl_wrapper.py`

The higher level HRL policy $\pi(g|s, \theta^{hi})$ is going to use the policy you trained in Q3 as the lower level $\pi(a|s, g, \theta^{lo})$. To accomplish this we will create another environment wrapper that loads the lower level policy and uses it plus the suggested goals from the high level to act in the environment.

Algorithm 1 HRL algorithm

1: init θ^{hi} a random networks, load θ^{low} and $\mathcal{D} \leftarrow \{\}$
2: **for** $t \in 0, \ldots, T$ **do**
3: $g \leftarrow \pi(\cdot|\mathbf{s}_t, \theta^{hi})$ {Take k steps in the environment}
4: **for** $i \in 0, \ldots, k$ **do**
5: compute $\mathbf{a}_t \leftarrow \pi(\cdot|\mathbf{s}_t, g, \theta^{low})$ and receive $\{\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}'_t\}$
6: **end for**
7: Put $\{\mathbf{s}_t, g, r_{t+k}, \mathbf{s}_{t+k}\}$, add to $\mathcal{D}$
8: Update RL algorithm parameters θ^{hi}
9: **end for**

Question 5: Experiments (HRL) [3pts] For this question, the goal is to implement HRL and tune a few of the hyperparameters to improve the performance. Try different *goal_frequency=[5,10]* for the environment. **You should use the lower-level policies you trained for Q3 for each of these frequencies.**

```
python run_hw4.py  exp_name=q4_hrl_gf<b> rl_alg=pg
env_name=antmaze

python run_hw4.py  exp_name=q4_hrl_gf<b> rl_alg=pg
env_name=antmaze

python run_hw4.py  exp_name=q4_hrl_gf<b> rl_alg=pg
env_name=antmaze
```

Submit the learning graphs from these experiments along with the write-up for the assignment. You should plot the environment reward.

1 Bonus [3pts]

You can get extra points if you also implement Hindsight experience replay.

2 Bug Bonus [3pts]

If you find some bugs you can receive bonus marks on the assignment. Describe what bugs you found and how they were fixed.

Submission

We ask you to submit the following content on the course GradeScope :

Submitting the PDF.

Your report should be a PDF document containing the plots and responses indicated in the questions above.

Submitting log files on the autograder.

Make sure to submit all the log files that are requested by GradeScope AutoGrader, you can find them in your log directory `/data/exp_name/` by default.

Submitting code, experiments runs, and videos.

In order to turn in your code and experiment logs, create a folder that contains the following:

- A folder named `data` with all the experiment runs from this assignment. **Do not change the names originally assigned to the folders, as specified by `exp_name` in the instructions.**
- The `roble` folder with all the `.py` files, with the same names and directory structure as the original homework repository (not include the `outputs/` folder). Additionally, include the commands (with clear hyperparameters) and the config file `conf/config_hw4.yaml` file that we need in order to run the code and produce the numbers that are in your figures/tables (e.g. run "python run_hw4.py –ep_len 200") in the form of a README file. Finally, your plotting script should also be submitted, which should be a python script (or jupyter notebook) such that running it can generate all plots from your pdf. This plotting script should extract its values directly from the experiments in your `outputs/` folder and should not have hardcoded reward values.
- You must also provide a video of your final policy for each question above. To enable video logging, set both flags video log_freq to be greater than 0 and render to be true in `conf/config_hw4.yaml` **before** running your experiments. Videos could be fin as `.mp4` files in the folder: `outputs/.../data/exp_name/videos/`. **Note :** For this homework, the atari envs should be in the folder `gym` and the other in the folder `video`.

As an example, the unzipped version of your submission should result in the following file structure. **Make sure to include the prefix `q1_`,`q2_`,`q3_`,`q4_`, and `q5_`.**

```
submit.zip
├── hw1
├── hw2
├── hw3
├── hw4
│   ├── run_logs
│   │   ├── q1
│   │   │   ├── q1_cheetah_n500_arch1x32
│   │   │   │   ├── events.out.tfevents.1567529456.e3a096ac8ff4
│   │   │   │   └── eval_step_x.mp4
│   │   │   └── q1_cheetah_n5_arch2x250
│   │   │       ├── events.out.tfevents.1567529456.e3a096ac8ff4
│   │   │       └── eval_step_x.mp4
│   │   ├── q2
│   │   │   └── ...
│   │   └── ...
│   ├── roble
│   │   ├── agents
│   │   │   ├── dqn_agent.py
│   │   │   └── ...
│   │   ├── policies
│   │   │   └── ...
│   │   └── ...
│   ├── conf
│   │   └── config_hw4.yaml
│   ├── README.md
│   └── ...
├── Dockerfile
├── diff.txt
└── ...
```

You also need to include a diff of your code compared to the starter homework code. You can use the command

```
git diff 8ea2347b8c6d8f2545e06cef2835ebfd67fdd608 >> diff.txt
```

1. If you are a Mac user, **do not use the default "Compress" option to create the zip**. It creates artifacts that the autograder does not like. You may use `zip -vr submit.zip submit -x "*.DS_Store"` from your terminal.
2. Turn in your assignment on Gradescope. Upload the zip file with your code and log files to **HW4 Code**, and upload the PDF of your report to **HW4**.