CS 330 Autumn 2021/2022 Homework 3

Goal Conditioned Reinforcement Learning and Hindsight Experience Replay

Due Wednesday October 27th, 11:59 PM PT

SUNet ID:
Name:
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Overview

In this assignment we will be looking at goal-conditioned reinforcement learning and hindsight experience replay (HER). In particular, you will:

1. Adapt an existing model (a Deep Q-Network) to be goal-conditioned
2. Run goal-conditioned DQN on two environments
3. Implement Hindsight Experience Replay (HER) [1,2] on top of a goal-conditioned DQN
4. Compare the performance with and without HER

We have provided you with starter code, which can be downloaded from the course website. In this assignment, we will be running goal-conditioned Q-learning on two problems: (a) a toy problem where we flip bits in a bit vector to match it to the current goal vector and (b) move the end effector of a simulated robotic arm to the desired goal position.

Submission: To submit your homework, submit one PDF report to Gradescope containing written answers and Tensorboard graphs (screenshots are acceptable) to the questions below, as well as `trainer.py` and `run_episode.py` in a single zip file. The PDF should also include your name and any students you talked to or collaborated with. **Any written responses or plots to the questions below must appear in your PDF submission.**

Setup: This assignment requires the use of physics simulator MuJoCo, and installing the related python bindings. While we have provided instructions to setup MuJoCo and install the corresponding python bindings `mujoco-py` in `README.md`, the setup can be dependent on the machine and it may require additional steps to be setup successfully.

We recommend completing the setup as soon as possible. In particular, we recommend setting up the virtualenv and installing the requirements, and then running `python test_installation.py`. The script should print `Dependencies successfully installed!` if the setup was successful. In case the local setup is unsuccessful after following the troubleshooting guide, we recommend completing the assignment by creating a virtual machine on Azure. Please follow the detailed instructions in `README.md`.

Note for students using a Mac with M1 chip: You will need to use Azure to complete this assignment (unless you can arrange for a different machine). MuJoCo does not support the M1 processor yet.

Code Overview: The code consists of several files to enable Q-learning. You are expected to write the code in the following files: `run_episode.py, trainer.py`. A brief description for the code base is provided here:

- `trainer.py`: The main training loop `train`. Alternates between collecting transitions using Q-value networks and training the networks on collected transitions.
- `run_episode.py`: Collect an episode using the current Q-value network, and return the transitions, collected reward and whether the agent was successful during the episode.
- `replay_buffer.py`: Buffer for storing the transitions collected in the environment.
- `q_network.py`: Create a feedforward neural network with one hidden layer for Q-network.
- `bit_flip_env.py`: The source code for the bit flipping environment, setup to follow the gym API with an `__init__, reset` and `step` functions.
- `sawyer_action_discretize.py`: Wraps the `SawyerReachXYEnv` from `multiworld` environment and converts the continuous action space into a discrete action space with a simplified 2D observation space.
- `main.py`: Main files to configure the experiment.

A detailed description for every function can be found in the comments. You are not expected to change any code except for sections marked with `TODO`. Next, we provide description of the environments we will be looking at and then we look at the exact expectations for this assignment.

Environments

You will be running on two environments:

Environment 1: Bit Flipping Environment

In the bit-flipping environment, the state is a binary vector with length `n`. The goal is to reach a known goal vector, which is also a binary vector with length `n`. At each step, we can flip a single value in the vector (changing a 0 to 1 or a 1 to 0). This environment can very easily be solved without reinforcement learning, but we will use a DQN to understand how adding HER can improve performance.

The bit flipping environment is an example of an environment with sparse rewards. At each step, we receive a reward of -1 when the goal and state vector do not match and a reward of 0 when they do. With a larger vector size, we receive fewer non-negative rewards.

Environment 2: 2D Sawyer Arm

The Sawyer Arm is a multi-jointed robotic arm for grasping and reaching (https://robots.ieee.org/robots/sawyer/). The arm operates in a 2D space, and the goal is to move the robot to a set of coordinates. The sawyer reach is an example of a dense reward environment, where the reward is given by negative Euclidean distance between the robot arm and the goal state.

Problem 1: Implementing Goal-conditioned RL

We have a goal-conditioned DQN. The Q-function takes in the concatenated state and goal as input. You can think of the goal-conditioned implementation as an extended Markov decision process (MDP), where your state space contains both the original state and the goal. We will use this goal-conditioned Q-network to collect episodic data in `run_episode.py`.

For this part, complete `run_episode.py` and run the following command:

```
python3 main.py --env bit_flip --num_bits 6 --num_epochs 250 --her_type no_hindsight
```

The evaluation metrics should be available in tensorboard events logged in `logs/` by default. **Verify** the `eval_metrics`, that is `total_reward` should be above -40.0 and `success_rate` should be 1.0. This plot illustrates the performance without using HER. You do **not** need to include this in the homework.

Implementation notes:

- For simplicity, we will only consider the *greedy* action with respect to Q-network. Pass this action to `env.step`.
- The `env.step` function returns `next_state, reward, done, info`, where `info` is a dictionary containing the a boolean under the key `'successful_this_state'`, indi-

cating whether the state was successful or not. Use this value to update `succeeded`, such that `run_episode` returns `True` if the *policy was successful at any step of the episode*. To understand more about `env.step`, read the documentation for the function in `bit_flip_env.py`.

- Ensure that floating point numpy arrays use `np.float32`. You may need to recast some of the arrays to ensure that.

Problem 2: Adding HER to Bit Flipping

With HER, the model is trained on the actual (state, goal, reward) tuples along with (state, goal, reward) tuples where the goal has been relabeled. The goals are relabeled to be what state was *actually reached* and the rewards correspondingly relabeled. In other words, we pretend that the state we reached was always our goal. HER gives us more examples of actions that lead to positive rewards. The reward function for relabeled goals is the same as the environment reward function; for the bit flipping environment, the reward is -1 if the state and goal vector do not match and 0 if they do match.

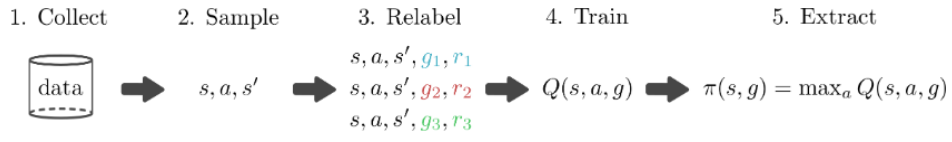

Figure 1: Illustration of HER. The goals and rewards are relabeled so that we have more positive rewards to learn from. In practice, these steps are performed iteratively, returning to step 1 after step 5.

There are three different variations of HER: `final`, `random`, and `future`. In each variation, the goal is relabeled differently:

- `final`: The final state of the episode is used as the goal
- `random`: A random state in the episode is used as the goal
- `future`: A random future state of the episode is used as the goal

Implementation Notes:

- Always use `copy()` when assigning an existing numpy array to a new variable. NumPy does not create a new copy by default.

- When choosing a new goal for relabelling, choose the state corresponding to `next_state` from the experience.

More details of these three implementations are in the comments of `trainer.py`. Implement the three variations of HER in the function `update_replay_buffer` in `trainer.py`. You **do not** need to submit a plot for this.

Problem 3: Analyzing HER for Bit Flipping Environment

Once you have completed the previous problems, we can analyze the role of HER in goal-conditioned RL. We analyze the performance of HER as the size of the bit vector is increased from 6 to 25. For each of the parts (a) to (d), submit a tensorboard screenshot showing the `eval_metrics` for different runs on the same plot (check the correct event files). A total of 4 screenshots should be submitted for this section.

a) Run the following commands:

```
python3 main.py --env=bit_flip --num_bits=6 --num_epochs=250 --her_type no_hindsight
python3 main.py --env=bit_flip --num_bits=6 --num_epochs=250 --her_type final
```

b) Run the following commands:

```
python3 main.py --env=bit_flip --num_bits=15 --num_epochs=500 --her_type no_hindsight
python3 main.py --env=bit_flip --num_bits=15 --num_epochs=500 --her_type final
```

c) Run the following commands:

```
python3 main.py --env=bit_flip --num_bits=25 --num_epochs=1000 --her_type no_hindsight
python3 main.py --env=bit_flip --num_bits=25 --num_epochs=1000 --her_type final
```

d) Finally, we will compare the three versions of HER, with the baseline of not using HER:

```
python3 main.py --env=bit_flip --num_bits=15 --num_epochs=500 --her_type no_hindsight
python3 main.py --env=bit_flip --num_bits=15 --num_epochs=500 --her_type final
python3 main.py --env=bit_flip --num_bits=15 --num_epochs=500 --her_type random
python3 main.py --env=bit_flip --num_bits=15 --num_epochs=500 --her_type future
```

Since two of the commands (HER final and HER none) are identical to part (b), you do **not** need to run them again.

e) For the above, explain your findings and why you expect the methods to perform in the observed manner for varying numbers of bits and varying relabeling strategies. Keep your write-up concise and summarize your observations across all parts.

Problem 4: Analyzing HER for Sawyer Reach

If implemented correctly, HER should work for the second environment, `Sawyer Reach`.

Compare the performance of the Sawyer arm with and without HER. Run the following commands:

```
python3 main.py --env=sawyer_reach --num_epochs=1000 --her_type no_hindsight
python3 main.py --env=sawyer_reach --num_epochs=1000 --her_type final
```

For this part,

a) Submit the tensorboard screenshot comparing the `eval_metrics` in your report.

b) Discuss your findings: Compare the role of HER in Bit Flipping Environment and Sawyer Reach. Comment on the differences between the contribution of HER, if any.

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, Wojciech Zaremba. Hindsight Experience Replay. Neural Information Processing Systems (NeurIPS), 2017. https://arxiv.org/abs/1707.01495

[2] Leslie Kaelbling. Learning to Achieve Goals. International Joint Conferences on Artificial Intelligence (IJCAI), 1993. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.3077