CS 330 Autumn 2019/2020 Homework 3

Goal Conditioned Reinforcement Learning and
Hindsight Experience Replay
Due Wednesday November 6th, 11:59 PM PST

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 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 for each environment
4. Compare the performance with and without HER

Submission: To submit your homework, submit one pdf report and one zip file to Gradescope, where the report will contain answers to the deliverables listed below and the zip file contains your code `bits_main.py` and `sawyer_main.py` with the filled in solutions.

Code Overview: The code consists of four files. You should only make modifications to two of them.

- `BitFlip.py`: Bit flipping environment for problems 1-3. You should not modify this file, though it may be helpful to look at it and understand the environment.
- `buffers.py`: Buffers for storing experiences. You should not modify this file, though it may be useful to look it.
- `bits_main.py`: Main loop and helper functions for solving the bit flipping environment. You will add your solutions to problems 1 and 2 to this file and run it for problem 3.

- `sawyer_main.py`: Main loop and helper functions for solving the Sawyer arm environment. You will add your solutions to problem 4 to this file and run it for problem 5.

Dependencies: We expect code in Python 3.5+ with `Pillow, scipy, numpy, tensorflow, gym, mujoco, multiworld` installed.

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 demonstrate 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. As the size of the vector grows, we receive fewer and fewer non-negative rewards. Adding HER helps us train in an environment with sparse rewards (more details later).

The bit flipping environment is included in the homework zip file as `BitFlip.py` and does not require additional installation.

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.

To run the Sawyer Arm environment, you will have to install several packages. If you have any trouble with installation, **please post on Piazza**; it's likely that other students have run into the same problem.

Installing gym

Gym is a package for comparing reinforcement learning algorithms. It is a widely used library, and contains several simulated tasks for testing algorithms such as teaching robots to walk and balancing a pole on a cart. Instructions for installing gym are here `http://gym.openai.com/docs/`. Run:

```
pip install gym
```

Installing Mujoco

Mujoco is a physics simulation engine that runs with gym. You will be emailed a class license for Mujoco. **DO NOT REDISTRIBUTE THE LICENSE KEY**. Instructions for installing Mujoco can be found here: `https://github.com/openai/mujoco-py`. Run:

```
pip install mujoco-py
```

Installing multiworld

To install multiworld, run

```
git clone https://github.com/vitchyr/multiworld
```

navigate to the `setup.py` and run

```
python setup.py install
```

Problem 1: Implementing Goal-Conditioned RL on Bit Flipping

The file `bits_main.py` contains a DQN that runs in a bit-flipping environment. Currently the DQN is not goal-conditioned and does not have HER implemented. The Q-function takes in only the state as the input, and does not consider the goal. To modify the Q-function to be goal-conditioned, concatenate the observation vector with the goal vector and pass the combined vector to the Q-function. 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.

a) Run `bits_main.py` with the following arguments:

   ```
   python bits_main.py --num_bits=7 --num_epochs=150
   ```

   Save the generated plot and include it in your homework. This plot illustrates the performance without goal conditioning.

b) Modify the DQN so that it is goal-conditioned. The places where you will need to make modifications are marked in `bits_main.py`. You should not make modifications to `buffers.py` or `BitFlip.py`. Hint: the bit flipping environment returns the state and goal vector when `reset()` is called.

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.

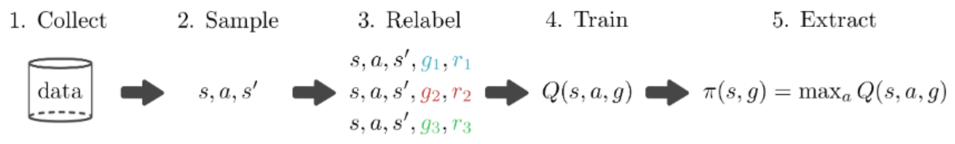

Figure 1: Illustration of HER. The goals and rewards are relabeled so that we have more positive rewards to learn from. In your implementation, it may be easier to relabel the goals and rewards prior to step 1.

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

More details of these three implementations are in the comments of `sawyer_main.py`.

Implement the three variations of HER in the function `update_replay_buffer()`. You may modify other parts of the code, but this is not necessary.

Problem 3: Bit Flipping Analysis

Once you have completed the previous problems, run the bit flipping environment both with and without HER:

a) Run `bits_main.py` with the following arguments:

```
python bits_main.py --num_bits=7 --num_epochs=250 --HER=None
python bits_main.py --num_bits=7 --num_epochs=250 --HER=final
```

b) Run `bits_main.py` with the following arguments:

```
python bits_main.py --num_bits=15 --num_epochs=500 --HER=None
python bits_main.py --num_bits=15 --num_epochs=500 --HER=final
```

c) Run `bits_main.py` with the following arguments:

```
python bits_main.py --num_bits=25 --num_epochs=1000 --HER=None
python bits_main.py --num_bits=25 --num_epochs=1000 --HER=final
```

Each run should generate a separate plot showing the success rate of the DQN. Include these six plots in your writeup.

d) Next you will compare the three versions of HER. Plot the performance of the three versions of HER along with running without HER for the following arguments:

```
--num_bits=15 --num_epochs=500
```

Include these four plots in your homework. Since two of the plots (HER final and HER none) are identical to part b), feel free to reuse your prior plots.

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. Your write-up should include around 2-3 sentences for each part.

Problem 4: Implementing Goal-Conditioning and HER on Sawyer Reach

You will now implement goal-conditioning and HER on a second environment. In the file `sawyer_main.py` implement goal-conditioning and HER for the Sawyer arm environment. Your implementation will be similar to the modifications you made to `bits_main.py`.

Hint: The reward function for the Sawyer environment is the negative Euclidean distance between the goal and state vector.

Problem 5: Sawyer Analysis

Compare the performance of the Sawyer arm with and without HER. Run the environment with the following arguments, and include the plots in your report. If you would like to render the Sawyer Arm and see it move in the environment, add the argument `--render=True`.

```
python sawyer_main.py --num_epochs=150 --steps_per_episode=50 --HER=None
python sawyer_main.py --num_epochs=150 --steps_per_episode=50 --HER=final
```

Finally, discuss your findings.

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