AAI4160 Homework 3: Q-Learning

Juii, Kim
2020118019

1 Introduction

The goal of this assignment is to help you understand **Q-learning methods**, including Deep Q-Network (DQN) and Double DQN.

The first part of this assignment includes quizzes about DQN. Then, in the second part of the assignment, you will implement a working version of Q-learning and evaluate Q-learning for playing Atari games. Our code will work with both state-based environments, where the input is a low-dimensional list of numbers (like Cartpole), but we will also support learning directly from pixels (like MsPacman)!

This assignment will be faster to run on a GPU, though it is still possible to complete on a CPU as well. Therefore, we recommend using VESSL AI or Colab if you do not have a GPU available to you.

2 Deep Q-Network Quiz (Total: 2 points)

Let's review what we have learned in the lecture about Deep Q-Network (DQN). Answer the following True/False questions:

I. Q-Learning cannot leverage off-policy samples, resulting in poor sample efficiency.

- ☐ True
- ■ False

II. Without an actor, evaluating Q-values for all possible actions is infeasible with continuous action space.

- ■ True
- ☐ False

III. One of the main challenges in DQN is the moving target, which happens when the agent estimates Q-values and target value using the same neural network. To avoid this, we can use the fixed target network within an inner loop.

- ■ True
- ☐ False

IV. We often use epsilon scheduling to encourage more exploration over time.

- ☐ True
- ■ False

3 Code Structure Overview

The training begins with the script `run_hw3.py`. This script contains the function `run_training_loop`, where the training, evaluation, and logging happens.

You will implement a DQN agent, `DQNAgent`, in `aai4160/agents/dqn_agent.py` and a DQN training loop in `aai4160/scripts/run_hw3.py`. In addition, you should start by reading the following files thoroughly:

- `aai4160/env_configs/dqn_basic_config.py`: builds networks and generates configuration for the basic DQN problems (`CartPole-v1`, `LunarLander-v2`).
- `aai4160/env_configs/dqn_atari_config.py`: builds networks and generates configuration for the Atari DQN problems (`Breakout`).
- `aai4160/infrastructure/replay_buffer.py`: implementation of replay buffer. To efficiently store and sample observations with frame-stack in DQN, we will use `MemoryEfficientReplayBuffer`. Try to understand what each method does (particularly `insert`, which is called after a frame, and `on_reset`, which inserts the first observation from a trajectory) and how it differs from the regular replay buffer.
- `aai4160/infrastructure/atari_wrappers.py`: contains some wrappers specific to the Atari environments. These wrappers can be key to getting challenging Atari environments to work!

Note: We will be using `gymnasium`, which is the latest and stable fork of `OpenAI gym`. Compared to `gym` which we used for Homework 1 and 2, there are two key changes in gymnasium, namely in return items in `env.step` function and `env.reset` function. For further references, please see gymnasium guide. Install `gymnasium` using the following command or simply do `pip install -r requirements.txt`.

```
pip install "gymnasium[classic_control,box2d,atari,accept-rom-license]"
```

3.1 Important Implementation Tricks

The starter code include a few implementation tricks to stabilize training. You do not need to do anything to enable these, but you should look at the implementations and think about why they work.

- **Exploration scheduling for ϵ-greedy actor.** This starts ϵ at a high value, close to random sampling, and decays it to a small value during training (`exploration_schedule` in `aai4160/scripts/run_hw3.py`).
- **Learning rate scheduling.** Decay the learning rate from a high initial value to a lower value at the end of training (`DQNAgent.lr_scheduler`).
- **Gradient clipping.** If the gradient norm is larger than a threshold, scale the gradients down so that the norm is equal to the threshold (`DQNAgent.update_critic`).
- **Atari wrappers.** (in `aai4160/infrastructure/atari_wrappers.py`)
  - **Grayscale.** Convert RGB images $(84 \times 84 \times 3)$ to grayscale images (84×84).
  - **Frame-skip.** Keep the same constant action for 4 steps and ignore intermediate inputs.
  - **Frame-stack.** Stack the last 4 grayscale frames to use as the input $(84 \times 84 \times 4)$.

4 Deep Q-Learning (3 points)

4.1 Implementation

Implement the basic DQN algorithm. You will implement an update for the Q-network and a target network, as well as functions for ϵ-greedy sampling and collecting trajectories:

- Implement a DQN critic update in `update_critic` function by filling in the unimplemented sections (marked with TODO(student)) in `aai4160/agents/dqn_agent.py`.

- Implement `update` of `aai4160/agents/dqn_agent.py`, which calls `update_critic` and updates the target critic, if necessary.

- Implement ϵ-greedy sampling in `get_action` function in `aai4160/agents/dqn_agent.py`.

- Implement the TODOs in `aai4160/scripts/run_hw3.py`.

- Implement the TODOs in `sample_trajectory` function of `aai4160/infrastructure/utils.py`.

  Hint: A trajectory can end in two ways: the actual end of the trajectory (`terminated=True`, usually triggered by catastrophic failure, like crashing), or *truncation* (`truncated=True`), where the trajectory doesn't actually end but we stop simulation for some reason (e.g. exceeding the maximum episode length).

  In `gymnasium`, there are two corresponding boolean values among the return items of `env.step`. Here, `terminated` flag represents the actual end of the trajectory, whereas `truncated` flag represents the truncation of trajectory due to other reasons, including reaching the maximum episode length. In this latter case, you should still reset the environment, but the `done` flag for TD-updates (stored in the replay buffer) should be `False`.

4.2 Experiment

DQN-CartPole. Test your DQN implementation on `CartPole-v1` with `experiments/dqn/cartpole.yaml`. It should reach reward of nearly 500 around 120K steps.

```
python aai4160/scripts/run_hw3.py -cfg experiments/dqn/cartpole.yaml --seed 1
```

- Plot the learning curve with environment steps on the x-axis and eval return (`eval_return`) on the y-axis. You can use `aai4160/scripts/parse_tensorboard.py` as in Homework 1 and 2.

  Answer:

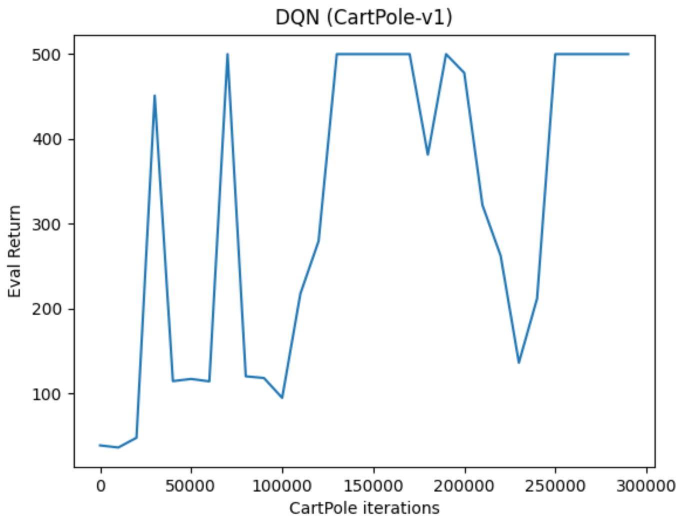

Figure 1: Learning curve of DQN on CartPole-v1

The figure above shows the learning curve of DQN on the CartPole-v1 environment. It achieved reward close to 500 by approximately 120K steps.

- Run DQN with three different seeds on `LunarLander-v2` (No need to provide plots here, since we will use these results and compare in the next section about double-q learning):

```
python aai4160/scripts/run_hw3.py -cfg experiments/dqn/lunarlander.yaml --seed 1
python aai4160/scripts/run_hw3.py -cfg experiments/dqn/lunarlander.yaml --seed 2
python aai4160/scripts/run_hw3.py -cfg experiments/dqn/lunarlander.yaml --seed 3
```

Your code may not reach high return (200) on Lunar Lander yet; this is okay! Your returns may go up for a while and then collapse in some or all of the seeds.

- Run DQN on `CartPole-v1`, but run it with the `experiments/dqn/cartpole_lr_5e-2.yaml`. By using this configuration file, you will change the `learning rate` to 0.05; notice that the default learning rate is 0.001. What happens to (a) the predicted Q-values, (b) the critic error, and (c) the eval returns? Please provide both plots to compare them. Can you relate this to any topics from class or the analysis section of this homework? Provide your reasoning/explanation.

Answer:

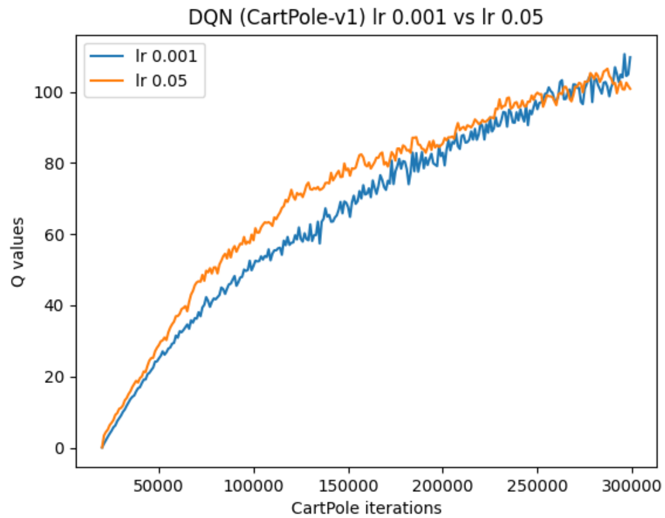

Figure 2: predicted Q-values of DQN lr 1e-3 vs 5e-2

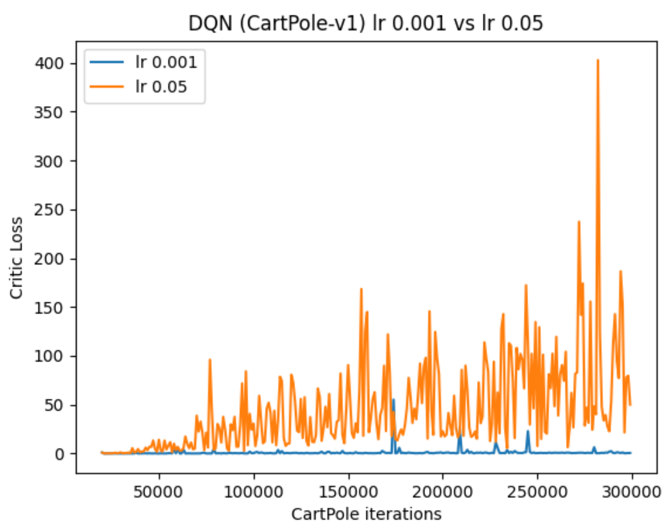

Figure 3: Critic loss of DQN lr 1e-3 vs 5e-2

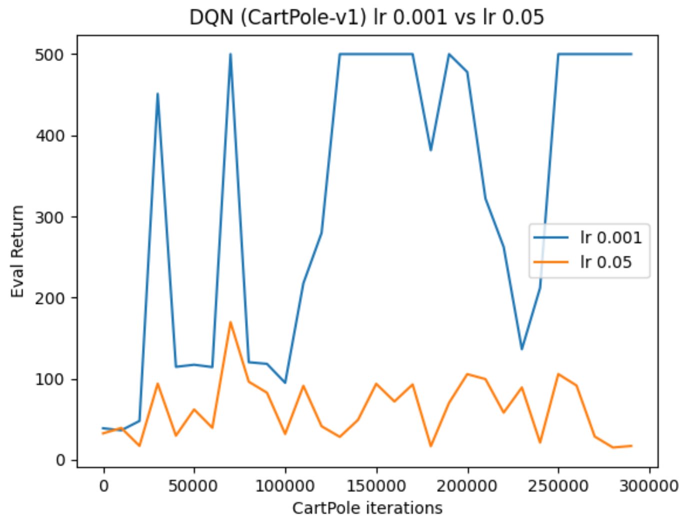

Figure 4: Eval returns of DQN lr 1e-3 vs 5e-2

Through the three figures above, comparing the predicted Q-values between a learning rate of 1e-3 and 5e-2, it is evident that the learning rate of 1e-3 results in more stable learning. Additionally, while a learning rate of 5e-2 leads to significant divergence in the critic loss, 1e-3 maintains stability. Moreover, the evaluation returns show that a learning rate of 1e-3 yields significantly higher returns, indicating that excessively high learning rates hinder stable convergence.

5 Double Q-Learning (4 points)

5.1 Implementation

Let's try to stabilize learning. The double-Q trick avoids overestimation bias in the critic update by using two different networks to *select* the next action a' and to *estimate* its value:

$$a' = \arg\max_{a'} Q_\phi(s', a')$$

$$Q_{\text{target}} = r + \gamma(1 - d_t)Q_{\phi'}(s', a').$$

In our case, we'll keep using the target network $Q_{\phi'}$ to estimate the action's value, but we'll select the action using Q_ϕ (the online Q-network).

Implement this functionality in `aai4160/agents/dqn_agent.py`.

5.2 Experiments

Double DQN-LunarLander and MsPacman.

- Run Double DQN with three seeds for the lunar lander problem:

```
python aai4160/scripts/run_hw3.py -cfg experiments/dqn/lunarlander_doubleq.yaml --
    seed 1
```

```
python aai4160/scripts/run_hw3.py -cfg experiments/dqn/lunarlander_doubleq.yaml --
    seed 2
python aai4160/scripts/run_hw3.py -cfg experiments/dqn/lunarlander_doubleq.yaml --
    seed 3
```

You should expect a return of **200** by the end of training, and it should be fairly stable compared to your policy gradient methods from HW2. (***Disclaimer**: for some seeds, it might not reach the return of **200**. That is fine as far as you observe it can outperform the vanilla DQN on average*).

Plot returns from these three seeds of Double DQN in red, and the "vanilla" DQN results in blue, on the same set of axes. Compare DQN and Double DQN, and describe in your own words what might cause this difference.

Answer:

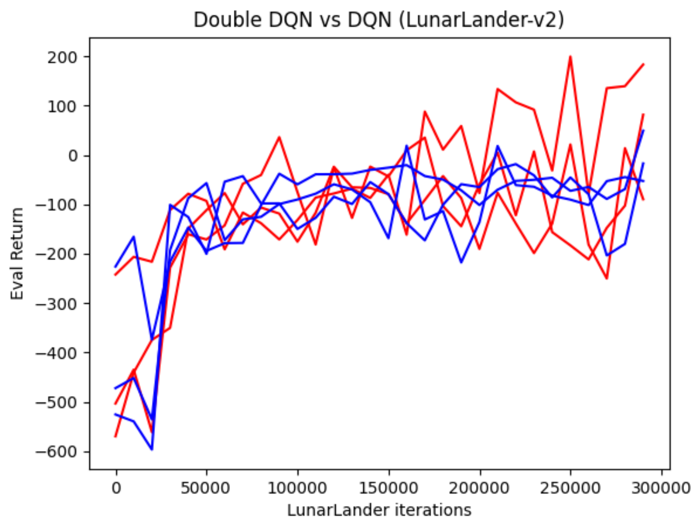

Figure 5: Double DQN vs DQN

Comparing DQN and Double DQN through the figure above, Double DQN seems to achieve higher returns across the different seeds. It's because Double DQN uses both target and online networks (whereas DQN uses only online network) to decompose the max operation in the target into action selection and action evaluation, thereby reducing overestimation in "vanila" DQN.

- Run your Double DQN implementation on the `MsPacman-v0` problem. Our default configuration will use double Q-learning by default. You are welcome to tune hyperparameters to get it to work better, but the default parameters should work (so if they don't, you likely have a bug in your implementation). Your implementation should receive a score of around **1500** by the end of training (1 million steps. **This problem will take about 3 hours with a GPU, or 6 hours without, so start early!**

```
python cs285/scripts/run_hw3.py -cfg experiments/dqn/mspacman.yaml
```

Plot the average training return (`train_return`) and eval return (`eval_return`).

Answer:

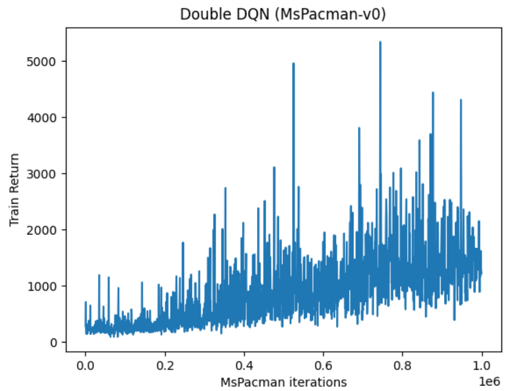

Figure 6: Train/Eval returns of DDQN

6 Experimenting with Hyperparameters (2 point)

Let's analyze the sensitivity of Q-learning to hyperparameters on the `CartPole-v1` environment. Choose one hyperparameter of your choice and run at least three other settings of this hyperparameter, in addition to the default value, and plot eval returns with all four values on the same graph. Explain why you chose this hyperparameter in the caption. Create four config files in `experiments/dqn/hyperparameters` (refer to `aai4160/env_configs/basic_dqn_config.py` to see which hyperparameters you are able to change). You can use any of the base YAML files as a reference.

Hyperparameter options could include:

- Learning rate
- Network architecture
- Exploration schedule (or, if you'd like, you can implement an alternative to ϵ-greedy)

Answer:

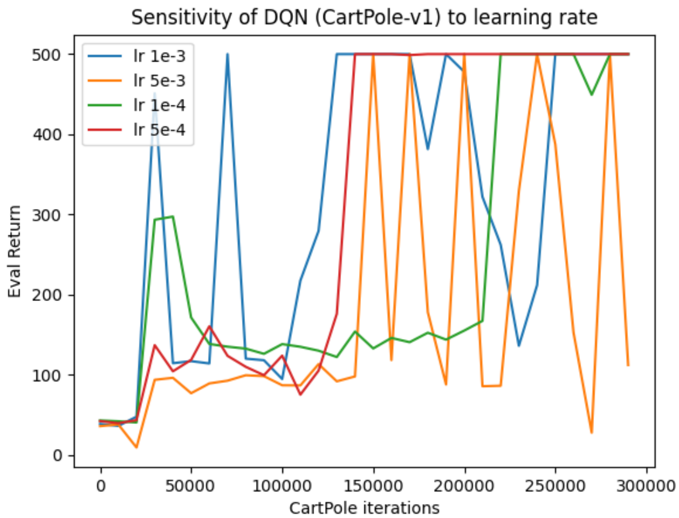

Figure 7: A Comparison of learnig rates for DQN

For the experimentation with hyperparameters, I chose to vary the learning rate. The learning rate plays a crucial role in determining the rate at which the Q-values are updated during training. I experimented with four different learning rates: 1e-3, 5e-3, 1e-4, and 5e-4. Among these, the learning rate of 5e-4 yielded the highest eval return. This result suggests that a moderate learning rate can lead to better performance in the CartPole-v1 environment compared to higher or lower learning rates.

7 Discussion

Please provide us a rough estimate, in hours, for each problem, how much time you spent. This will help us calibrate the difficulty for future homework.

- **Quizzes**: 0.5hours
- **Deep Q-Learning**: 10hours
- **Double Q-Learning**: 5hours
- **Experimenting with Hyperparameters**: 3hours

Feel free to share your feedback here, if any: **We would really appreciate your feedback to improve the reinforcement learning class.**

8 Submission

Please submit the code, tensorboard logs, and the **"report"** in a single zip file, `hw3_[YourStudentID].zip`. Do not include videos as the file size should be less than 30MB. The structure of the submission file should be:

```
hw3_[YourStudentID].zip
├─ hw3_[YourStudentID].pdf
├─ data
│  ├─ hw3_dqn_...
│  │  └─ events.out.tfevents....
│  └─ ...
├─ aai4160
│  └─ ...codes
└─ ...
```