GCB6206 Homework 3: Q-Learning

[Your name]
[Your student ID]

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 BankHeist)!

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. Section 4.2 takes about 15 minutes with a GPU and 30 minutes without it, while Section 5.2 requires approximately 12 hours with a GPU and 24 hours without it in total.

2 Deep Q-Network Quiz

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 `gcb6206/agents/dqn_agent.py` and a DQN training loop in `gcb6206/scripts/run_hw3.py`. In addition, you should start by reading the following files thoroughly:

- `gcb6206/env_configs/dqn_basic_config.py`: builds networks and generates configuration for the basic DQN problems (`CartPole-v1`, `LunarLander-v2`).
- `gcb6206/env_configs/dqn_atari_config.py`: builds networks and generates configuration for the Atari DQN problems (`Breakout`, `BankHeist-v5`).
- `gcb6206/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.
- `gcb6206/infrastructure/atari_wrappers.py`: contains some wrappers specific to the Atari environments. These wrappers can be key to getting challenging Atari environments to work!

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 `gcb6206/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 `gcb6206/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

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 `gcb6206/agents/dqn_agent.py`.

- Implement `update` of `gcb6206/agents/dqn_agent.py`, which calls `update_critic` and updates the target critic, if necessary.
- Implement ϵ-greedy sampling in `get_action` function in `gcb6206/agents/dqn_agent.py`.
- Implement the TODOs in `gcb6206/scripts/run_hw3.py`.
- Implement the TODOs in `sample_trajectory` function of `gcb6206/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 does not 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 300K steps (around 15 minutes).

```
python gcb6206/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 `gcb6206/scripts/parse_tensorboard.py` as in Homework 1 and 2.

  Answer:

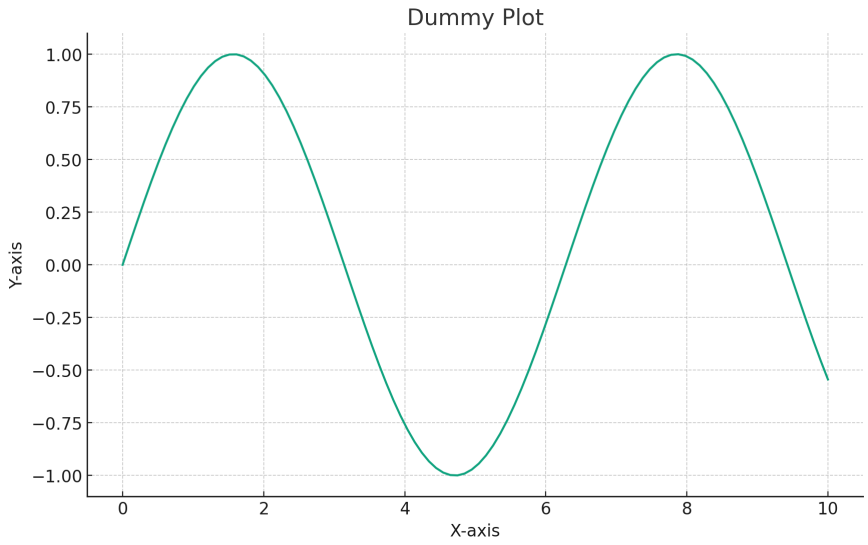
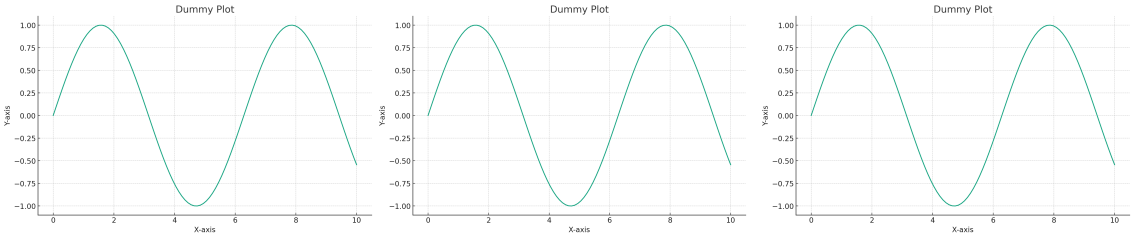


Figure 1: Replace this with your plot, and add the caption.

Add your explanation.

- Run DQN on `CartPole-v1`, but create `experiments/dqn/cartpole_lr_5e-2.yaml` by modifying the config file `experiments/dqn/cartpole.yaml` (change the `learning rate` to 0.05, where the default learning rate is 0.001), and run with this config file. What happens to (a) the predicted Q-values, (b) the critic error, and (c) the eval returns? Please provide three plots to compare the results of two different learning rates. Can you relate this to any topics from class? Provide your reasoning/explanation.

  Answer:



(a) caption (b) caption (c) caption

Figure 2: Replace this with your plot, and add the caption.

Add your explanation.

5 Double Q-Learning

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:

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

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

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 `gcb6206/agents/dqn_agent.py`.

5.2 Experiments

For this problem, each experiment will take about 2 hours with a GPU (12 hours in total), or 4 hours without (24 hours in total), so start early!

- Run DQN with three different seeds on `BankHeist-v5`:

```
python gcb6206/scripts/run_hw3.py -cfg experiments/dqn/bankheist.yaml --seed 1
python gcb6206/scripts/run_hw3.py -cfg experiments/dqn/bankheist.yaml --seed 2
python gcb6206/scripts/run_hw3.py -cfg experiments/dqn/bankheist.yaml --seed 3
```

  Your returns should improve until around 150.

- Run Double DQN with three seeds on `BankHeist-v5`:

```
python gcb6206/scripts/run_hw3.py -cfg experiments/dqn/bankheist_ddqn.yaml \
  --seed 1
python gcb6206/scripts/run_hw3.py -cfg experiments/dqn/bankheist_ddqn.yaml \
  --seed 2
python gcb6206/scripts/run_hw3.py -cfg experiments/dqn/bankheist_ddqn.yaml \
  --seed 3
```

You should expect a return of 300 by the end of training. (Disclaimer: for some seeds, it might not reach the return of 300. That is fine as far as you observe it can outperform the vanilla DQN on average).

Plot the 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:

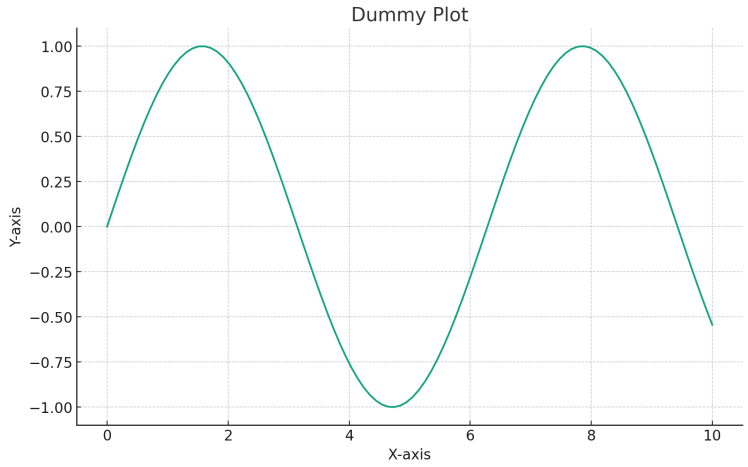


Figure 3: Replace this with your plot, and add the caption.

Add your explanation.

6 Experimenting with Hyperparameters

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 `gcb6206/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:

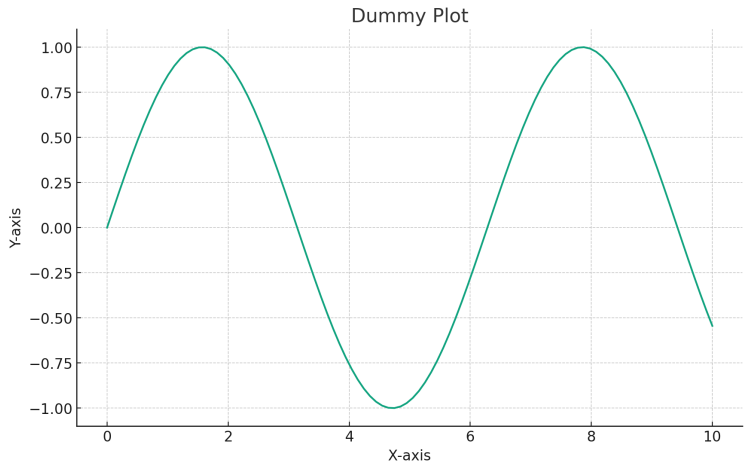


Figure 4: Replace this with your plot, and add the caption.

Add your explanation.

7 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 50MB. The structure of the submission file should be:

```
hw3_[YourStudentID].zip
├─ hw3_[YourStudentID].pdf
├─ gcb6206/
│  └─ ...codes
├─ data/
│  ├─ hw3_dqn_...
│  │  └─ events.out.tfevents....
│  └─ ...
└─ ...
```