Introduction to Artificial Intelligence Homework 4

Cheng-Liang Chi Yi-Wen Liu

Due: May 13, 2025

Contents

1 Introduction

Reinforcement learning is learning what to do - how to map situations to actions - so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.

There are four main components in reinforcement learning:

- Policy: A policy is a mapping from perceived states of the environment to actions to be taken when in those states.

- Reward signal: A reward signal defines the goal in a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number called the reward.
- Value function: The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.
- Model: A model predicts what the environment will do next.

In this homework, we consider the simplest form of reinforcement learning: that in which the state and action space are small enough to be stored as arrays.

And the problem we choose to play with is another simplest special case of reinforcement learning: the multi-armed bandit problem.

2 Multi-armed Bandit Problem

k-armed bandit problem is a simple reinforcement learning problem that models the trade-off between exploration and exploitation. The problem is defined as follows:

- There are k arms, each with an unknown reward distribution.
- At each time step, the agent selects an arm to pull.
- The agent receives a reward from the arm it pulls.
- The goal is to maximize the total reward over a fixed number of time steps.

2.1 Part 1: Game Environment Implementation

For the first part of the homework, you need to implement the game environment for the k-armed bandit problem. The game environment should have the following properties:

- The reward distribution for each arm should be a Gaussian distribution with variance 1 and a randomly generated mean.
- The mean should be generated from a standard normal distribution.
- The number of arms k should be a parameter of the game environment.
- Record the action history and reward history in the game environment.
- The game environment should have the methods `reset()`, `step(action)`, and `export_history()`, which are described below.

The game environment should have the following methods:

- `reset()`: Reset the game environment and the history
- `step(action)`: Take an action and return the reward.
- `export_history()`: Export the action history and reward history.

You are also required to implement it in Python. The example behavior of the game environment is shown in the following code snippet:

```python
from BanditEnv import BanditEnv
import random

env = BanditEnv(10)
env.reset()

actions = []
rewards = []
for i in range(1000):
    action = random.randint(0, 9)
    reward = env.step(action)
    actions.append(action)
    rewards.append(reward)

action_history, reward_history = env.export_history()

for i in range(1000):
    assert actions[i] == action_history[i]
    assert rewards[i] == reward_history[i]
```

The `BanditEnv` class should be implemented in a separate file named `BanditEnv.py`. Your implementation should be able to pass the test code above.

2.2 Part 2: Agent Implementation

For this part, you will need to implement an action-value method with ϵ-greedy exploration.

It is straightforward that choosing the action with the highest expected reward is the best policy. However, in practice, we will not know the true mean reward for each action. So we need to estimate the expected reward of each action based on the rewards we have observed so far.

Please implement a sample-average method to estimate the expected reward of each action. The agent should have the following methods:

- `select_action(self)`: Choose an action based on the estimated expected reward for each action.
- `update_q(self, action, reward)`: Update the estimated expected reward of the chosen action.
- `reset(self)`: Reset the agent.

You are also required to implement it in Python. The example behavior of the game environment is shown in the following code snippet:

```python
from Agent import Agent

k = 10
epsilon = 0.1

```

```python
agent = Agent(k, epsilon)
action = agent.select_action()
reward = 0
agent.update_q(action, reward)
```

The `Agent` class should be implemented in a separate file named `Agent.py`.

2.3 Part 3: Experiment

In this part of the assignment, you will do some experiments using the action-value method with ϵ-greedy exploration that you implemented in the previous part.

You need to do the following experiments 2,000 times independently:

1. Construct a 10-armed bandit environment from Part 1.
2. Construct an agent with ϵ-greedy exploration from Part 2.
3. Run the agent in the environment for 1,000 steps.

Compare a greedy method ($\epsilon = 0$) with two ϵ-greedy methods ($\epsilon = 0.1$ and $\epsilon = 0.01$), and plot two curves:

1. The average reward of the agent over time.
2. The percentage of optimal action selection over time.

The average reward of the agent over time is defined as the average of the rewards the agent receives at each time step, and the percentage of optimal action selection over time is defined as the percentage of the time the agent selects the optimal action at each time step.

3 Non-stationary Environment

In real world, the environment is often non-stationary, meaning that the true mean reward of each action will change over time. Thus, we introduce a non-stationary environment to the multi-armed bandit problem. In this environment, the true mean reward of each action will change every steps.

3.1 Part 4: Environment Implementation

For this part, you will need to modify the environment you implemented in Part 1 to include a non-stationary reward distribution.

Please follow the requirements below:

- Modify the `BanditEnv` class to include a non-stationary reward distribution.
- The true mean reward of each action will change every steps.
- The true mean reward of each action will take independent random walks by adding a normally distributed increment with mean 0 and standard deviation 0.01 to all the true mean rewards.

Your environment should be able to run the following code without any errors:

```python
from Agent import Agent
from BanditEnv import BanditEnv

k = 10
epsilon = 0.1

# Original stationary interface should work as well
# env = BanditEnv(k)

env = BanditEnv(k, stationary=False)
agent = Agent(k, epsilon)
```

3.2 Part 5: Experiment

In this part of the assignment, you will conduct experiments to evaluate the performance of different learning strategies in a non-stationary environment.

Unlike the stationary environment Part 3, where the true mean reward of each action remains constant, in this part, the true mean reward of each action will undergo an independent random walk.

You need to perform the following experiments 2,000 times independently:

1. Construct a 10-armed bandit environment with a non-stationary reward distribution from Part 4.
2. Use the agent from Part 2.
3. Run the agent in the environment for 10,000 steps.

Prepare two plots to visualize the average reward and the average optimal action selection rate over 2,000 independent runs. The plots' requirements are the same as in Part 3.

3.3 Part 6: Constant Step-Size Update Implementation

In this part, you will modify the agent from Part 2 to use a constant step-size update method instead of the sample-average method.

Modify your `Agent` class from Part 2 to:

- Modify the `Agent` class to support a constant step-size update method.
- The constant step-size update method should be implemented in the `update_q` method.
- The `constant` parameter will be given as a parameter to the constructor of the `Agent` class.
- If the step-size parameter neither given nor set to `None`, the agent should fall back to the sample-average method.

The `update_q` method should be modified to use the following formula:

$$Q(a) \leftarrow Q(a) + \alpha \cdot (R - Q(a)) \tag{1}$$

where $Q(a)$ is the estimated expected reward of action a, α is the step-size parameter, and R is the reward received from action a. The `select_action` method should remain unchanged.

Your agent should be able to run the following code without any errors:

```python
from Agent import Agent

k = 10
epsilon = 0.1

# Original stationary interface should work as well
# agent = Agent(k, epsilon)

# Construct with None should behave the same as the original
# agent = Agent(k, epsilon, None)

agent = Agent(k, epsilon, 0.1)
action = agent.select_action()
reward = 0
agent.update_q(action, reward)
```

3.4 Part 7: Experiment

In this part, you will do some experiments using the constant step-size update method in a non-stationary environment.

Run the following experiments 2,000 times independently:

1. Construct a 10-armed bandit environment with a non-stationary reward distribution from Part 4.
2. Use the agent from Part 6.
3. Run the agent in the environment for 10,000 steps.

Prepare two plots to visualize the average reward and the average optimal action selection rate over 2,000 independent runs. The plots' requirements are the same as in Part 3.

4 Report

You should write you report in and include the following sections:

- Implementation Details

  Explain how you implemented the agent. Your implementation correctness will be counted in this part. (7 points)

  Explain how you implemented the environment. Your implementation correctness will be counted in this part. (8 points)

- Experiment Results

  Put the plots from Part 3 and analyze the results. (15 points)

  Put the plots from Part 5 and analyze the results. (15 points)

  Put the plots from Part 7 and analyze the results. (15 points)

- (Optional) Discussion (Bonus up to 5 points)

  Discuss anything you like to share with us.

5 Submission

5.1 Submission Structure

You should submit a zip file named `{student_id}.zip` containing the following directories and files:

- `report.pdf`: This file should contain your report, which includes the implementation details, experiment results, and discussion. See Section 4 for more details.
- `Agent.py`: This file should contain the implementation of your agent.
- `BanditEnv.py`: This file should contain the implementation of your bandit environment.
- `main.py`: This file should contain the main function that runs your agent and environment. This is the only file that will plot the figures. Please do not plot your figures in `Agent.py` or `BanditEnv.py`.

The report should be in PDF format and named `report.pdf`. The submission structure is as follows:

```
{student_id}.zip
├── report.pdf
├── Agent.py
├── BanditEnv.py
└── main.py
```

5.2 Execution Environment

The only external library you need to use is `matplotlib` for plotting. For more information, visit the official Matplotlib documentation.

We will run your code on a Linux machine with Python 3.13.3 and matplotlib 3.10.1 installed. For our convenience, we recommend you to use the same version of Python and matplotlib. But since this assignment is not too complicated, you can use any version of Python and matplotlib that you prefer.

You can use pyenv or micromamba to manage your Python environment. Look for the installation instructions on their official websites.

For your reference, here is the command to create a virtual environment with Python 3.13.3 and matplotlib 3.10.1 using `pyenv`:

```
pyenv install 3.13.3
pyenv virtualenv 3.13.3 ai-hw4
pyenv activate ai-hw4
pip install matplotlib==3.10.1
```

5.3 Submission Deadline

The deadline for this assignment is **May 13, 2025** at 17:00 (GMT+8). Please make sure to submit your assignment to the E3 platform before the deadline.

5.4 Late Submission Policy

The late submission policy is as follows:

- 0-24 hours late: 10% penalty
- 24-48 hours late: 20% penalty
- 48-72 hours late: 30% penalty
- More than 72 hours late: No credit (You can still get credit for the quiz if you come on time)

6 Quiz

We will hold an quiz that will count as 40% of your final score. The further details of the quiz will be announced later.

7 Q/A

If you have any questions about the assignment, please follow the instructions here.

8 Scoring Policy

You can get up to (65%) points for this assignment. And you can get up to (40%) points for the quiz.

Do not palgiarize the code from any other sources. The code you submit should be your own work. If you are caught plagiarizing (including from any LLM services), you will get 0 points for this assignment. Please cleary reference any external sources you use in your report.

Any matters not explicitly mentioned in this document will be decided by the instructor and the teaching assistants. If you have any questions about the assignment, please follow the instructions here and ask questions before the deadline.

References

[1] A. Agarwal, N. Jiang, S. M. Kakade, and W. Sun, "Reinforcement learning: Theory and algorithms," in 2021. [Online]. Available: `https://rltheorybook.github.io/`.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. The MIT Press, 2020.

[3] S. Levine. "Deep reinforcement learning course, fall 2023," UC Berkeley. (2023), [Online]. Available: `https://youtube.com/playlist?list=PL_iWQOsE6TfVYGEGiAOMaOzzv41Jfm_Ps&feature=shared`.