Deep Reinforcement Learning

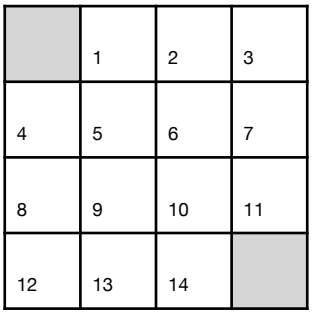
Homework #1

Due: 2024/3/26 (Tue.) 23:59

【UPDATE】 typo issue, step function for testing file, folder architecture, and check file.

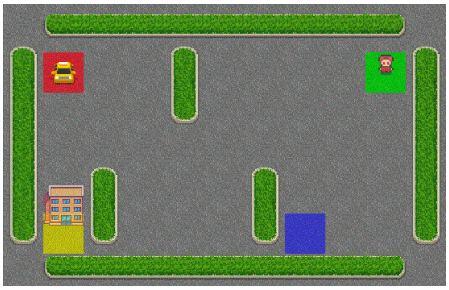
Problem Description

1. Random Policy Evaluation ("**<Student_ID>_hw1_1.py**")
   - Consider the following environment:



   - Detail of the environment:
     - Normal states: {1 - 14}.
     - Terminal states: grey states.
     - Action space: {*right*, *left*, *top*, *bottom*}.
     - Rewards: -1 for all transitions.
     - If the agent hits the wall (e.g., move left at state 8), it stays at the same state in the next timestep.
     - Random Policy: $\pi(a \mid s) = 0.25, \forall a, s$.
   - Calculate the state value (v_π) of the environment:
     - Output the learned (converged) v_π for state 1-14 (order matters). The output should be rounded to the second decimal place.
     - Calculate the correct results of v_π with $\gamma = 0.9$, and store the result in "**<Student_ID>_hw1_1_data_gamma_0.9**".
     - Calculate the correct results of v_π with $\gamma = 0.1$, and store the result in "**<Student_ID>_hw1_1_data_gamma_0.1**".
     - Explain your code and results. How does the iteration stop? How does the discount factor γ affect the results?
     - Sample Output: (numbers separated by spaces, no trailing characters) See "**hw1-1_sample_output**".

2. Compare Q-Learning with SARSA
   - Apply SARSA and Q-Learning algorithms in OpenAI gym environment "**Taxi-v3**".



   - The two algorithms should learn different policies and have different learning curves in this environment.
   - Please name your files as "**<Student_ID>_hw1_2_taxi_sarsa.py**" and "**<Student_ID>_hw1_2_taxi_qlearning.py**".
   - Please save the Q table for SARSA and Q-Learning with following names: "**<Student_ID>_hw1_2_taxi_sarsa.pth**" and "**<Student_ID>_hw1_2_taxi_qlearning.pth**".
   - We load the trained model using following code: "**torch.load(path)**". We will run your saved table with following code:

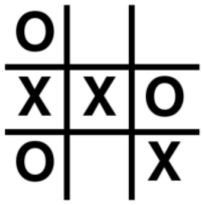
```
agent = torch.load(path)
done = False
state, _ = env.reset()

while not terminated:
    action = torch.argmax(agent[state]).item()
    next_state, reward, done, _, _ = env.step(action)
```

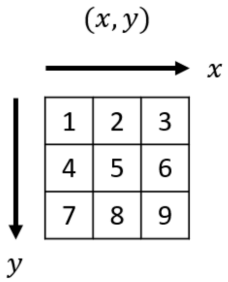
   - Explain the behaviors of these algorithms.
   - We decide grade according to following rules:
     - Testing on same environment 10 times and record the episodic cumulated reward.
     - Get full score if the average cumulated rewards exceeds 0.
     - Get zero score if the average cumulated rewards doesn't exceed 0.

3. Tic-tac-toe (3x3) ("**<Student_ID>_hw1_3_<train|test>.py**")

- Learn a policy for playing Tic-tac-toe:



- State space: 3x3 integer array, e.g., the array [[−1,0,0], [1,1,−1], [−1,0,1]] corresponds to:
    1. 1 represents 'X'
    2. -1 represents 'O'
    3. 0 represents empty space
- Action space: integer coordinates (x, y). The coordinate represents the position for your move.
- Goal: win the game.
- Use Q-learning, SARSA or other tabular method (without function approximation).
- You may store your learned results in an external file "**./hw1_3_data**" (readonly, max 10 MB), and access it with your program (during testing).
- Explain your training procedure and how you implemented the environment.
- The input integers are structured with the order shown in right:
  (The cell number corresponds to the i^{th} input, and the index of coordinate starts from 0.)



- In your test file, please make a class named as "Agent" with functions "load_policy" and "choose_action". We will use the following program to load your policy and play against the baselines:

```
from <Student_ID>_hw1_3_test import Agent
your_agent = Agent()
Agent.load_policy() # load policy from your own folder
action = Agent.choose_action(state: np.array) # generate action
```

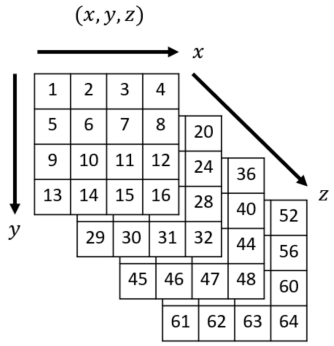
4. 3D Tic-tac-toe (4x4x4) ("**<Student_ID>_hw1_4_<train|test>.py**")

- State space: 4x4x4 integer array.

  1 represents 'X'

  1. -1 represents 'O'
  2. 0 represents empty space

- Action space: integer coordinates (x, y, z)
- Goal: win the game.
- Implement your policy with **Monte Carlo Tree Search (MCTS)**
- Other improvements are allowed. The detailed rules are specified in the next section.
- The input integers are structured with the follow order:



(The cell number corresponds to the i^{th} input)

a) You may store your learned results in an external file "**./hw1_4_data**" (readonly, max 50 MB), and access it with your program (during testing).

b) Explain your training / testing procedure and how you implemented the environment.

- In your test file, please make a class named as "Agent" with functions "load_policy" and "choose_action". We will use the following program to load your policy and play against the baselines:

```
from <Student_ID>_hw1_4_test import Agent
your_agent = Agent()
Agent.load_policy() # load policy from your own folder
action = Agent.choose_action(state: np.array) # generate action
```

Detailed Rules for 3D Tic-tac-toe

1. If your program outputs invalid moves, you lose and the game ends immediately.
2. Time limit for each move is 5 seconds, and the memory limit is 4 GB. (Note that the 5 second duration may vary depending on different processors.)
3. You are allowed to access an external file (readonly, max 50 MB) for storing your learned policy. You can read the file at the following path: "**./hw_1_4_data**".
4. You can keep overriding your move during the time limit. Only the last successful output move is selected.
5. You are allowed to use the following Python packages for 2D Tic-tac-toe:
   (a) Numpy, Scipy, gym, pandas, Tensorflow, PyTorch (non-GPU accelerated)\
   (b) You are allowed to use Python default installed packages. (e.g., sys, time, pickle, random, etc.)
   (c) If you need to use other packages, state your reasons and post on **eeclass**.
6. You are not allowed to use the following:
   (a) Other External API/libraries (e.g. System calls)
   (b) Inline assembly (e.g. asm)
   (c) Vectorization instructions (e.g. SSE, AVX)
   (d) Multi-threading (e.g. multiprocessing package)
   (e) GPU acceleration (e.g. CUDA code)
   (f) Networking (e.g. Sockets, MPI)

Check File

We provide a check file "**checker.py**" which you can use it to check whether your saved model (problem 2) and test files (problem 3, 4) are available to be accessed by our evaluation file. If all check points are passed, you can see the output "**<Student_ID> problem {#}: Pass**", otherwise, it will show "**<Student_ID> problem {#}: Failed**".

Program Submission

1. For each problem, please use Python to implement with a single source file.
2. Your files must be named as:
   (a) "**<Student_ID>_hw1_1.py**"
   (b) "**<Student_ID>_hw1_1_data_gamma_0.9**"
   (c) "**<Student_ID>_hw1_1_data_gamma_0.1**"
   (d) "**<Student_ID>_hw1_2_taxi_sarsa.py**"
   (e) "**<Student_ID>_hw1_2_taxi_qlearning.py**"
   (f) "**<Student_ID>_hw1_2_taxi_sarsa.pth**"
   (g) "**<Student_ID>_hw1_2_taxi_qlearning.pth**"
   (h) "**<Student_ID>_hw1_3_train.py**"
   (i) "**<Student_ID>_hw1_3_test.py**"
   (j) "**<Student_ID>_hw1_3_data**"
   (k) "**<Student_ID>_hw1_4_train.py**"
   (l) "**<Student_ID>_hw1_4_test.py**"
   (m) "**<Student_ID>_hw1_4_data**"
   (n) "**<Student_ID>_hw1_report.pdf**"
   (o) Please make sure that all characters of the filename are in lower case.
   (p) Please compress all files as "**<Student_ID>.zip**"
   Architecture of the folder after unzip shown as follow:

```
<Student_ID>
    |——— <Student_ID>_hw1_1.py
    |——— <Student_ID>_hw1_1_data_gamma_0.9
    |———  …..
    |——— <Student_ID>_hw1_report.pdf
```

3. Your program will be ran in a GNU/Linux environment with Python 3.8.
4. 0 points will be given to Plagiarism. NEVER SHOW YOUR CODE to others and you must write your code by yourself. If the codes are similar to other people and you can't explain your code properly, you will be identified as plagiarism.
5. 0 points will be given if you violate the rules above.
6. If you use modularized / OOP code and want to use multiple files to keep your code structured, please email us beforehand.

Report

1. Elaborate on how you design your tic-tac-toe. The report is graded directly. So, make sure you have included enough details and figures to help TAs grade your report.
2. TAs will not refer to your code when grading your report, so make sure you have taken a screenshot of the important code snippets.
3. The report filename must be "**<Student_ID>_hw1_report.pdf**" and please make sure that all characters of the filename are in lower case.

Grading Policy

1. The project accounts for 15 points (tentative) of your total grade.
2. You must submit both your source code and report. Remember the submission rules mentioned above, or you will be punished on your grade. Late submission rules are specified in the Lecture 1 Slides.
3. Compress all your files directly (do not compress the folder containing your files) and upload to **this Google Form** before the deadline. (Only 1 compressed file!!!)
4. The TA code will not be released. Your code will be tested against them after the submission deadline.

- **Random Policy Evaluation**
  - Converged Value Function (10%)
  - Report (Discussion) (5%)
- **Compare Q-Learning with SARSA**
  - Compare with baseline (20%)
  - Report (Discussion) (5%)
- **Tic-tac-toe**
  - Never Lose the Random Agent (5%)
  - Never Lose the TA Agent (15%)
  - Report (Discussion) (5%)
- **3D Tic-tac-toe**
  - Compete with Random Agent (10%)
  - Compete with Your Classmates (20%)
  - Report (Discussion) (5%)