

113-1 (Fall 2024) Semester

Reinforcement Learning

Assignment #1

TA: Chan-Hung Yu (尤展鴻)

Department of Electrical
Engineering
National Taiwan University

Outline

- Tasks
 - Iterative policy evaluation
 - Policy iteration
 - Value iteration
 - Async dynamic programming
- Environment
- Code structure
- Report
- Grading
- Submission
- Policy
- Contact

Tasks

Task 1 - Iterative Policy Evaluation

- Problem
 - Evaluate a given non-deterministic policy (probability distribution)
- Solution
 - Iterative application of Bellman expectation backup

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π , the policy to be evaluated

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$ Synchronous update

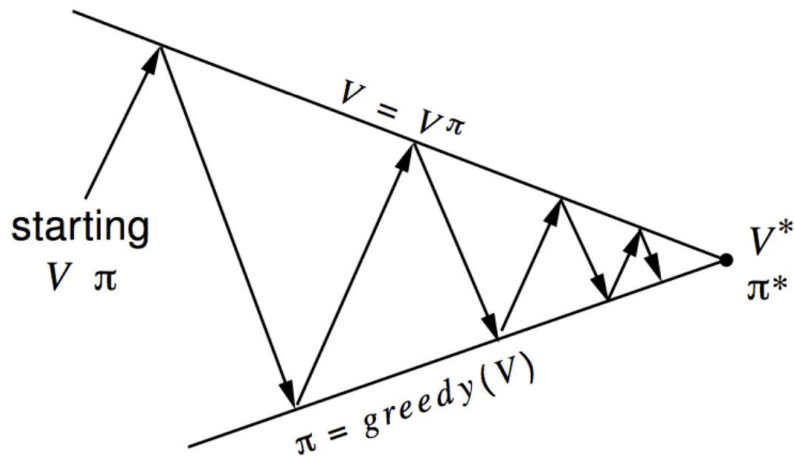
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

[\[source\]](#)

Task 2 - Policy Iteration

- Problem
 - Find the optimal deterministic policy
- Solution
 - Policy evaluation: iterative policy evaluation
 - Policy improvement: greedy policy improvement
 - Eventually converges to optimal policy



[source]

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
 $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$
2. Policy Evaluation
Loop:
 $\Delta \leftarrow 0$
Loop for each $s \in \mathcal{S}$:
 $v \leftarrow V(s)$
 $V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s)) [r + \gamma V(s')]$ Synchronous update
 $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)
3. Policy Improvement
 $\text{policy-stable} \leftarrow \text{true}$
For each $s \in \mathcal{S}$:
 $\text{old-action} \leftarrow \pi(s)$
 $\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$
If $\text{old-action} \neq \pi(s)$, then $\text{policy-stable} \leftarrow \text{false}$
If policy-stable , then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Task 3 - Value Iteration

- Problem
 - Find the optimal deterministic policy
- Solution
 - Iterative application of Bellman optimality backup

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

```
|  $\Delta \leftarrow 0$   
| Loop for each  $s \in \mathcal{S}$ :  
|    $v \leftarrow V(s)$   
|    $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$  Synchronous update  
|    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$   
until  $\Delta < \theta$ 
```

Output a deterministic policy, $\pi \approx \pi_*$, such that

$$\pi(s) = \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

[\[source\]](#)

Task 4 - Async Dynamic Programming

- Problem
 - Find the optimal deterministic policy with better efficiency
 - Less environment interaction
- Solutions
 - In-place dynamic programming
 - Prioritized sweeping
 - Real-time dynamic programming

Environment

Grid World

State space

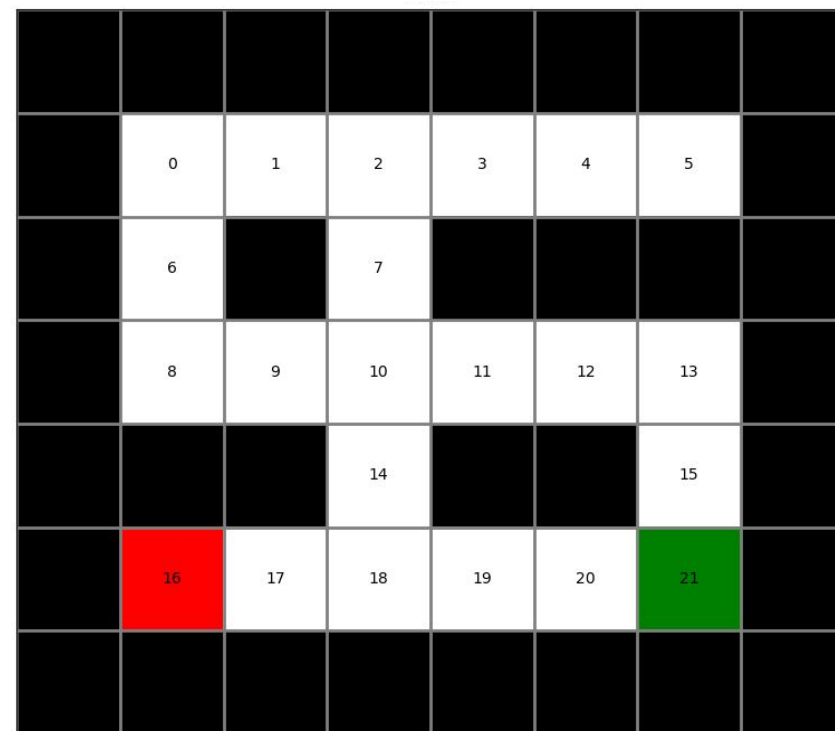
- Nonterminal states: Empty, Wall
- Terminal states: Goal, Trap
- 0-indexed

Action space

- Up, down, left, right
- Hitting the wall will remain at the same state

Reward

- Step reward given at every transition
- Goal reward given after reaching goal state
- Trap reward given after reaching trap state



Done Flag

- Separator for episodes
- Return true from step when doing any action at terminal states
- Need to modify the Bellman equation

$$v_{\pi}(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a) (r + \gamma v_{\pi}(s'))$$

$$v_{\pi}(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a) (r + \gamma v_{\pi}(s') (1 - Done))$$

Code Structure

requirement.txt

- [Conda](#)

```
conda create -n rl_assignment_1 python=3.11
conda activate rl_assignment_1
pip install -r requirement.txt
```

- venv

```
python -m venv venv
source venv/bin/activate
pip install -r requirement.txt
```



DP_solver.py

class **DynamicProgramming**

- Parent class for DP algorithms
- TODO: get_q_value()

class **IterativePolicyEvaluation**

- TODO: get_state_value(), evaluate(), run()

class **PolicyIteration**

- TODO: get_state_value(), policy_evaluation(), policy_improvement(), run()

class **ValueIteration**

- TODO: get_state_value(), policy_evaluation(), policy_improvement(), run()

class **AsyncDynamicProgramming**

- TODO: run()

Feel free to add any function if needed

You must have run() for us to grade your code.

gridworld.py

- Methods:
 - `get_action_space()`: Get the dimension of action space
 - `get_state_space()`: Get the dimension of the state space
 - `step()`: Interact with the environment
 - `reset()`: Reset the environment
 - `visualize()`: Draw the maze with policy and values
 - `run_policy()`: Run the policy from given start state. Output the state history
- Step count:
 - Increment every time when `step()` method is called
 - Private member. Use `get_step_count()` to access.
- Step reward may not be constant at every state transition at private test cases
 - **Must use `step()` to get the reward function**
 - **Don't try to modify or override any private member (double underscore prefix)**
 - **You cannot call `reset()` by yourself. We may rename the function when grading.**

test cases

main.py

- run_policy_evaluation()
- run_policy_iteration()
- run_value_iteration()
- run_async_dynamic_programming()

These methods will call your written functions in DP_solver.py.

The output will be an image and the step count for the algorithm.

sample_solutions\

- iterative_policy_evaluation.png
- policy_iteration.png
- value_iteration.png
- step_counts.txt

The optimal policy might not unique.

Policy in sample solutions are obtained in the traverse order of action index.

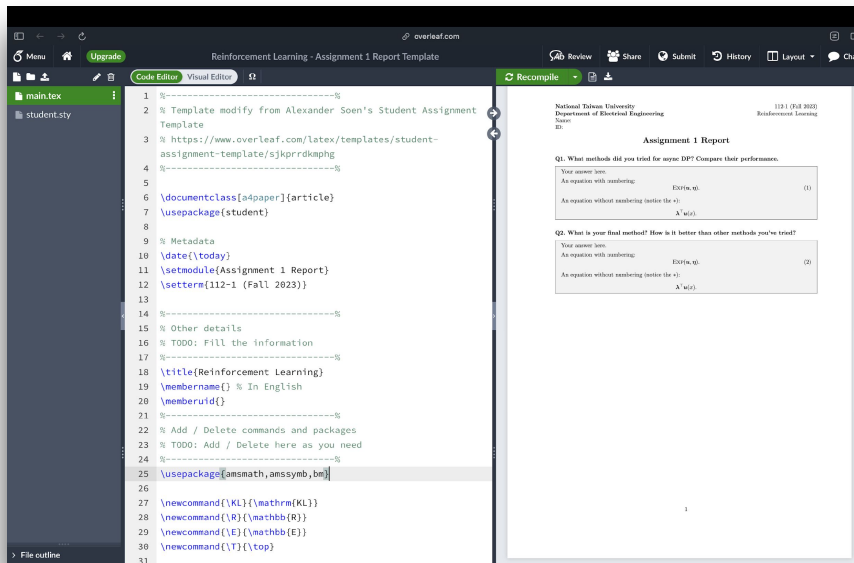
Report

Report

- Q1. What methods have you tried for async DP? Compare their performance. (6%)
 - 2% per method tried with reasonable result and comparison
- Q2. What is your final method? How is it better than other methods you've tried? (4%)
 - 2% for reasonable explanation
 - 2% for novel method (Out of the three methods mentioned in class)
- LaTeX PDF format. Handwriting is forbidden.
 - [Overleaf template](#)
 - Write clear and concise in few sentences

Overleaf

- Online LaTeX editor
- LaTeX
 - Good for math equations, tables and indexes
 - Widely used in paper writing and math writing
- Traditional Chinese will cause some compile problems
- [Official guide](#)



Grading

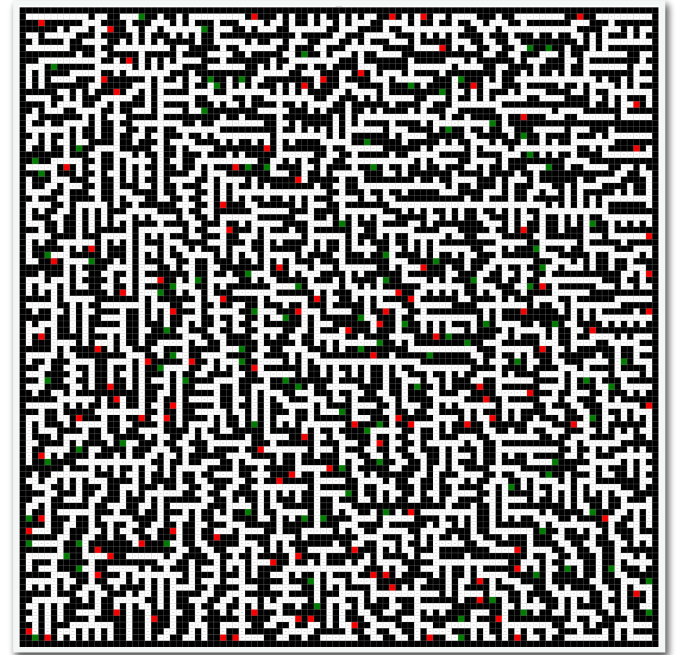
Grading

- Iterative policy evaluation (25%)
 - Test cases (5% x 5 cases)
- Policy iteration (20%)
 - Test cases (4% x 5 cases)
- Value iteration (25%)
 - Test cases (5% x 5 cases)
- Async dynamic programming (20%)
 - Better than your sync DP (5% x 2 cases) (Both policy iteration and value iteration)
 - Report (10%)

The total score is 90%.

Criteria

- Test cases:
 - Call run() and check the final output
 - Task 1: Check the values after evaluation
 - Task 2, 3, 4: Only check if the output policy is optimal
 - Run time limit **3 minute** for each case to avoid infinite loops
 - Up to 1500 states in private test cases
 - Only task 4 considers step count



Submission

Submission

- Submit on NTU COOL with following **zip** file structure
 - Get rid of pycache, DS_Store, __MACOSX, etc.
 - Student ID with lower case
 - **10%** deduction for wrong format

 b09901171.zip



 **b09901171**
├──  DP_solver.py
└──  report.pdf

- **Deadline: 2024/09/26 Thu 09:30am**
- **No late submission is allowed**

Policy

Policy

Package

- You can use any Python standard library (e.g., heap, queue...)
- System level packages are prohibited (e.g., sys, os, multiprocessing, subprocess...) for security concern

Collaboration

- Discussions are encouraged
- Write your own codes

Plagiarism & cheating

- All assignment submissions will be subject to duplication checking (e.g., MOSS)
- Cheater will receive an **F** grade for this course

Grade appeal

- Assignment grades are considered finalized two weeks after release

Contact

Questions?

- General questions
 - Use channel **#assignment 1** in slack as first option
 - Reply in thread to avoid spamming other people
- Personal questions
 - DM me on Slack: **TA 尤展鴻 Chan-Hung Yu**

