DS 669
Reinforcement Learning
Programming Assignment #1

In this assignment, you will have the opportunity to get familiar with Gymnasium, a popular Python library for developing and comparing reinforcement learning algorithms. A starter code has been provided to help you set up the environment. You will implement value iteration (VI) and policy iteration (PI) for the Frozen Lake environment and run various experiments to understand the effects of different parameter settings.

Setup: You need Python3 (versions>=3.8) to complete this assignment. You also need to install the libraries required by running:

- **pip install gymnasium==0.29.1**
- **pip install numpy==1.26.3**

Please use **pip3** instead of **pip** to install the libraries if you also have Python 2.x on your computer.

Submission:

- You need to finish the assignment by finishing all parts marked with **Question** in each section and taking screenshots if asked.
- Make sure you number each of your responses so it is easy to match responses with exercises.
- When finished, submit your report in **PDF** form and `code_base.py` file on Canvas. **Please DO NOT** zip all your files together.

I Implementation of basic function (10%)

In this section, you need to read through `code_base.py` and `get_args.py`. Finish your code in `code_base.py` between the pound sign lines denoted with "Your Code" in each function. Test your code on `FrozenLake-v1 8*8` deterministic environment. Suggestions:

- Please fully understand the `FrozenLake` game, analyze the structure of the programs, especially the input and output of each function according to the comments, and read the notes under each "Your Code" carefully before implementation. The tutorial of the `FrozenLake` is in here.
- Please do not change the default settings in `get_args.py` for this section.
- Please use `np.max` and `np.argmax` if you need to return or find the maximum in a numpy array.
- All functions needed for policy iteration and value iteration are defined and technically you do not need to create new ones.

(a) Implement `policy_evaluation` and `policy_improvement`, and then use these two functions to finish `policy_iteration`.
(**Hint 1**: Please be careful about the definition of iteration. The iteration does not include the steps of policy evaluation.)
(**Hint 2**: Please update the value function using the synchronous update method (refer to specific hints in the corresponding code sections).)

(b) Implement `value_iteration`.

II Evaluate the performance (20%)

In this section, you will evaluate the performance of the methods you just finished in Section I. Use parameter: `method` in `get_args.py` to select to run `policy_iteration` or `value_iteration`.
(**Hint**: Iteration refers to the process of repeatedly updating the policy or value function. The steps in policy evaluation or policy improvement are not included.)

(a) Set `seeds` $= 1$ in `get_args.py` and take the screenshots of the all the output results after running `policy_iteration` and `value_iteration`.
(The screenshots should include the method you are using, the policy generated, the total running time and the episode reward.)

(b) Please modify the parameter `seeds` in `get_args` and run each of `policy_iteration` and `value_iteration` **50 times** and take the screenshots of the total running time starting with the line: "`Total running time is`".

(c) **Question:** What is the time complexity of **policy** iteration in one iteration? Please provide your answer in terms of $|S|$ (the length of the state vector) and $|A|$ (the length of the action vector).
(**Hint:** "in one iteration" means we only consider the running time for just one run of policy evaluation and policy improvement function. You may also refer to the annotation in the codebase to calculate the running time.)

(d) **Question:** What is the time complexity of **value** iteration in one iteration? Please provide your answer in terms of $|S|$ (the length of the state vector) and $|A|$ (the length of the action vector). (**Hint:** "in one iteration" means we only consider the running time for just one time's value updating. You may also refer to the annotation in the codebase to calculate the running time.)

(e) **Question:** Based on your answer for part(c) and part(d), theoretically which algorithm is faster in one iteration? Aside from the running time in a single iteration, what other factors could influence the **total** running time?

III Effect of parameter epsilon ε (20%)

In this section, you will use value iteration to further understand the effect of `epsilon` ε. The definition of ε is introduced in the class. The setting of ε is in the $14th$ line of `get_args.py` named as ε, and you can change `default` to the desired value.

(a) Reset the parameter `seeds` as 1 in `get_args.py`. Use the value iteration method by setting the `method` as `value_iteration`. Set value the of ε as 0.5. Rerun the program and take the screenshot of all the output results.
(The screenshots should include the method you are using, the policy generated, the total running time and the episode reward.)

(b) **Question**: How many iterations does the value iteration method need to generate the policy? Is the policy the optimal policy?

(c) **Question:** How many iterations does value iteration method need to generate the policy from part II.(a)? Is it less or more than the iterations you get when setting $\varepsilon = 0.5$?

(d) **Question**: Based on the comparison between part (b) and part (c), what will happen if ε is decreased? Conversely, what will happen if ε is increased? Can you explain the reason?
(**Hint:** Based on the definition from lecture that the condition of convergence is the maximum difference $\|\mathbf{V_{k+1}} - \mathbf{V_k}\|_\infty \leq \varepsilon$, how does ε affect the number of iteration and the optimal policy?)

IV Effect of parameter gamma γ (30%)

In this section, you will still use the value iteration to further understand the effect of `gamma` γ. The definition of γ is introduced in the class. The setting of γ is in the $16th$ line of `get_args.py`, and you can change `default` to modify the value of the parameters. Remember to recover ε to $1e-3$. You can print out the value function of the states to find some clues about the answers.

(a) Set $\gamma = 0$, and take the screenshot of the all the output results from your code.
(The screenshots should include the method you are using, the policy generated, the total running time and the episode reward.)

(b) **Question:** A computer science student Micheal conducted an experiment setting the value of γ to be 0, 1 and 2, and respectively obtained the results shown in the screenshots of Figure 1. Based on his results, can you analysis:
When $\gamma = 0$, does the value iteration generate the optimal policy? If not, can you explain the

```
---- Value Iteration----epsilon0.001----gamma0----init_value0----

There are 2 itereations in value iteration.
policy: [['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'D']
 ['L' 'L' 'L' 'L' 'L' 'L' 'R' 'L']]
Total running time is: 1.395934820175171  average iteration is: 2.0
The agent didn't reach a terminal state in 100 steps.

Process finished with exit code 0
```

(a) Value iteration with $\gamma = 0$

```
---- Value Iteration----epsilon0.001----gamma1----init_value0----

There are 15 itereations in value iteration.
policy: [['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'D' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'D' 'L']
 ['L' 'L' 'L' 'L' 'D' 'L' 'L' 'L']
 ['L' 'L' 'L' 'D' 'L' 'L' 'L' 'D']
 ['L' 'L' 'D' 'L' 'L' 'D' 'L' 'D']
 ['L' 'L' 'L' 'L' 'D' 'L' 'L' 'L']]
Total running time is: 1.344228744506836  average iteration is: 15.0
The agent didn't reach a terminal state in 100 steps.

Process finished with exit code 0
```

(b) Value iteration with $\gamma = 1$

```
---- Value Iteration----epsilon0.001----gamma2----

 ['L' 'L' 'L' 'L' 'D' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'D' 'L']
 ['L' 'L' 'L' 'L' 'D' 'L' 'L' 'L']
 ['L' 'L' 'L' 'D' 'L' 'L' 'L' 'D']
 ['L' 'L' 'D' 'L' 'L' 'D' 'L' 'R']
 ['L' 'L' 'L' 'L' 'D' 'L' 'L' 'L']]
Total running time is: 1.5405659675598145  average iteration is: 1026.0
The agent didn't reach a terminal state in 100 steps.

Process finished with exit code 0
```

(c) Value iteration with $\gamma = 2$

Figure 1: Results of value iteration with different γ values

reason?
When $\gamma = 1$, does the value iteration generate the optimal policy? If not, can you explain the reason?
When $\gamma = 2$, can you explain why we got the value function and the policy as shown?
(**Hint**: when $\gamma = 0, 1, 2$, how does the value function update? You may print out the value function to help you analysis this question.)

(c) Set $\gamma = 0.5$, and take the screenshot of the output.
Question: Does the value iteration generate the optimal policy? How is γ related with ε?

(d) **Question:** If the policy found in the previous question is not optimal, can you help the value iteration find the optimal policy by modifying the value of ε? Please show your modified value of ε and provide a screenshot.

V Effect of initialization setting (20%)

In this section, you will use policy iteration to further understand the effects of initial settings. Please remember to recover $\gamma = 0.9$ and $\varepsilon = 1e - 3$.

For the previous sections, we use random action for each state as the initial policy:

```
init_policy = np.random.randint(0, nA, nS) if init_action == -1
else np.ones(nS, dtype=int) * init_action
```

(a) Please change the initial action to "go left" for all states and rerun the policy iteration. Take the screenshot of all the output results from your code. Please use the `init_action` in `get_args.py` to control the initialization of the policy.
(The screenshots should include the method you are using, the policy generated, the total running time and the episode reward.)

(b) **Question:** Is the policy the optimal policy? How many iterations does the policy iteration need?

(c) Please try "go right", "go up", and "go down" for all states and rerun the policy iteration method. Take a screenshot for each case.

(d) **Question:** Which action(s) initialization can reduce the number of iterations? Can you explain the reason?

(e) In `policy_evaluation` function of the policy iteration method, we initialize the value function with all 0 by: `value_function = np.zeros(nS)`

Question: How many evaluation steps do policy evaluation need to converge? Please print the `evaluation_steps` for each iteration and take a screenshot.

(f) Can you optimize the initialization to reduce the number of evaluation steps? You can add a new parameter to the policy_iteration function and use this parameter to control the initialization. Please print out **evaluation_steps** for each iteration and take the screenshot.

Question: Please describe the main idea of your initialization method.