Assignment 5
GridWorld

B4B36ZUI

Abstract—The assignment 5 focuses on Markov decision processes (MDPs) used for solving GridWorld examples and is worth 10 points in total. The GridWorld consists of a rectangular grid of cells, and the goal is to navigate an agent to reach the highest reward using non–deterministic actions. The assignment is to be implemented in Python 3.8 using provided codes. The assignment consists of both the implementation part and the experimental part.

1 Introduction

The goal of this assignment is to find optimal decision policy of an agent maximizing its reward in a GridWorld MDP problem (see slides or Wikipedia for details). A GridWorld consists of a rectangular grid of $m \times n$ cells. The cells are described using a coordinate system with $[0, 0]$ in the top left corner. An agent can make for different actions — it can go one cell north, east, south, and west from the cell it is occupying. The actions are not deterministic and intended action will be executed only with probability p (action_proba) and some other action will be executed with probability $1 - p$. More specifically, only the action that are neighboring the intended cardinal direction might be executed instead and each of them with uniform probability, i.e., if the agent's intended action is *north*, it will be executed with probability p and action *east* or *west* will be executed instead with probability $\frac{1-p}{2}$. Each action has an associated cost c (`action_cost`), which, for the purposes of this assignment, is identical for each action. This cost is applied only when the cell where the action would have ended has no associated reward — if the cell $[i, j]$ has an associated reward $r_{i,j}$, the reward overrides the cost.

In all the predefined worlds, the cells with defined rewards are also set to be terminal states where the agent ends when he reaches them. It is recommended for your experiments to keep the set of reward states the same as the set of the terminal state. A reward (or a cost if it is negative) is obtained if the associated cell is reached by the agent.

2 Algorithms

While the MDPs can be solved by both linear programming and dynamic programming, this task focuses on the latter — namely, only two variants of dynamic programming for solving the MDP are needed: *value iteration* and *policy iteration*. The dynamic programming approach consists of the iterative estimation of the values of individual states $V(s)$ and choosing the best policy $\pi(s)$ for deciding for action a from the set of actions $\mathbb{A}$ for each state 0 s from the set of states $\mathbb{S}$.

First, the action are evaluated using known valuation of states $V_{n-1}(s)$ and the transitional probabilities $P(s'|a, s)$ where s' is the target state, s is the current state, and a is the executed action:

$$Q_n(s, a) = R(s) + \sum_{s' \in \mathbb{S}} P(s'|a, s)\gamma V_{n-1}(s') \tag{1}$$

where $R(s)$ is the reward for going from state s and $\gamma \in [0, 1]$ is the discount factor.

The optimal policy $\pi_n(s)$ is chosen as

$$\pi_n(s) = \arg\max_{a \in \mathbb{A}} Q_n(s, a) \tag{2}$$

and finally, the valuation of states V_n is recomputed:

$$V_n = R(s) + \sum_{s' \in \mathbb{S}} P(s'|\pi_n(s), s)\gamma V_{n-1}(s') \tag{3}$$

2.1 Value iteration

The *value iteration* algorithm skips explicitly computing the optimal policy and consists of a single step repeated until the convergence criterion is met:

$$V_n = \max_{a \in \mathbb{A}} \left(R(s) + \sum_{s' \in \mathbb{S}} P(s'|a, s)\gamma V_{n-1}(s') \right) \tag{4}$$

2.2 Policy iteration

The *policy iteration* consists of two steps — computation the optimal policy $\pi_n(s)$ and evaluation of states $V_n(s)$ given the policy $\pi_n(s)$. The evaluation of states for a given policy might be either computed solving a set of equations or iteratively similarly as in the *value iteration*. The assignment uses the iterative version, which consists of repeated computation of $Q_n(s, a)$ and V_n until the V_n meets the convergence criterion.

3 Implementation

There are several implementation details that are worth. First, while, the functions are defined as multidimensional arrays — $R(s)$ is replaced by an array $\mathbf{R}$ of shape $|\mathbb{S}|$ where $\mathbf{R}[s] = R(s)$, $P(s'|a, s)$ is replaced by $\mathbf{P}$ of shape $|\mathbb{S}| \times |\mathbb{A}| \times |\mathbb{S}|$ where $\mathbf{P}[s, a, s'] = P(s'|a, s)$. Also $V_n(s)$ is saved as an array $\mathbf{V}_n$ of shape $|\mathbb{S}|$ and $Q_n(s, a)$ as $\mathbf{Q}_n$ of shape $|\mathbb{S}| \times |\mathbb{A}|$.

The set of states $\mathbb{S}$ consists of all cells from the grid and a **terminal sink state**, which is an added state that cannot be left (all action with lead back to it with probability 1) and which has no reward for reaching it. All actions from the states **on** the grid that are considered to be terminal leads to the sink state to prevent a repeated application of rewards that are associated with such states.

3.1 Environment

The task is to be implemented and the provided codes are in **python 3.8**. The other necessary packages are *NumPy* Walt et al. 2011, *matplotlib* Hunter 2007, and *seaborn* Waskom et al. 2016.

It is recommended to use Anaconda distribution, which also contains a package manager which allows installing many pre-compiled packages (which is especially beneficial when using Windows as it is often problematic to compile packages there).

4 Task

The task consists of two parts, an implementative one and an experimental one. The goal of the implementative part is to implement missing parts of several functions, while the experimental part consists of several small experiments with the GridWorld. The output of the assignment are the implemented codes, script (or Jupyter notebook) for launching the experiments, and report.

4.1 Implementative part [4p]

Your goal is to implement several helper functions (`Q_from_V`, `Q2V`, `Q2Vbypolicy`, and `Q2policy`), evaluation of the MDP for given policy (`evaluate_policy`), value iteration (`value_iteration`), and policy iteration (`policy_iteration`) in the file `ZUI_MDP.py`. You are also provided a set of test cases (`test_ZUI_MDP.py`) that can be used for testing your implementation using e.g. `unittest`, `nosetests`, or `pytest` (as in previous tasks in the course) modules. It is not necessary to use the tests but it is recommended as the correctness of your implementation will be tested similarly (mostly same tests with just different parameterization).

4.1.1 Unittests

While the unitests are not necessary for fulfilling the assignment, it is recommended to use them to check your implementation.

The basic usage when using nosetests is (run it from the folder containing `test_data`)

```
nosetests test_ZUI_MDP.py
```

and the output should be similar to

```
Ran 63 tests in 0.511s

OK
```

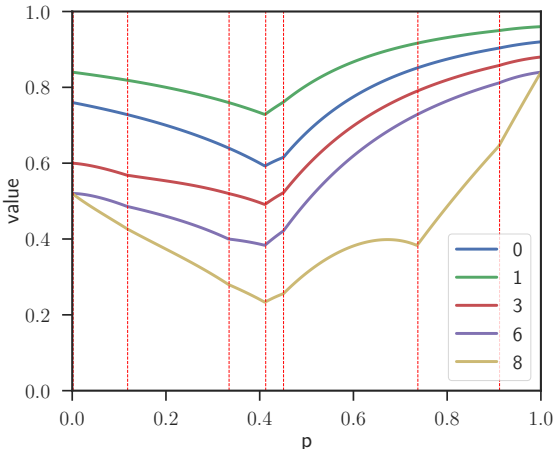

Figure 1. Example of output of Experiment 1 for the GridWorld `3x3` and states 0, 1, 3, 6, and 8

4.2 Experimental part [4.5p]

Once you have correctly implemented all the necessary functions, your goal is to experiment with the GridWorld MDPs. You are required to do at least 3 different experiments from which two are assigned (and are the same for all of you), and you are required to come up with your own ideas for the remaining one. All the experiments have to be thoroughly described in the report — the settings, used GridWorld parametrization, goals of the experiments, results (including visualization), and (if necessary) a conclusion.

4.2.1 Experiment 1: Policy switching based on action probability

Your goal is to analyze how the optimal policy changes with the changes of action probability p for the predefined GridWorld `3x4`. The output is to be a list of values of p for which there occurs a change in the optimal policy and also a plot showing the valuation of states 0, 3, 6, 8, 9, 10, and 11 together with thresholds where the policy change occurs. The plot should be similar to plot fig. 1 which shows the Experiment 1 for the GridWorld `3x3` and states 0, 1, 3, 6, and 8. The probability p should be on the x-axis and the valuation on the y-axis.

4.2.2 Experiment 2: Policy switching based on action cost

Your goal is to analyze how the optimal policy changes with the changes of action cost $c \in [0, \infty]$ for the predefined GridWorld `5x5`. The output is to be a list of values of c for which there occurs a change in the optimal policy. You also should show plots of the several most interesting policies (you can use method `GridWorld.plot`) — the exact policies you show are up to you.

4.2.3 Experiments 3 (or more)

You are required to come up with your own experiment that should be at least as complex as the previous two experiments. The possibilities are almost endless, e.g., measuring the runtime based on the size of the grid, the influence of both action cost c and action probability p on certain states (you would plot a heatmap where on the x axis would be

action probability p, on the y axis would be action cost c and the color would encode the value of the given state), or simple evolutionary algorithms for finding good policies (using function `evaluate_policy` for evaluation the objective or writing a custom evaluation function that solves equations for evaluating the policy instead of iterating). **Your** experiment **should be different** from the experiments described above (i.e., it is not sufficient to just take the experiments described above and just change the GridWorld instance to another predefined GridWorld instance).

You are required to include stand-alone scripts named `<username>_experiment_<n>.py` (e.g. `kuncvlad_experiment_2.py`) that are launchable in the described environment. For experiments running longer than 5 minutes, write the approximate running time in the report. Your experiment can be also in Jupyter notebooks (`.ipynb`).

4.3 Report [1.5p]

You are required to describe all the steps in a short report. The report has no fixed length limit, but it is evaluated on the basis of completeness and correctness of presented information. You can get up to **1.5 points** for the neatness and formal requirements of the report — e.g., your figures should have captions summarizing what is there, you should reference figures from the text (usage of `cref{}` is recommended), your report should have a logical structure, etc. You are required to hand in the report in the PDF format (any PDF format that graders can open is acceptable, but it is guaranteed that format `PDF/A` is fine). You are **not required** to use LaTeX but it is **highly recommended**, furthermore a LaTeX template `ZUI_template.tex` is provided for your use (and it is also **recommended** to use the template).

5 Grading

The whole assignment is worth **10 points in total**. The implementative part is for **4 points in total** — implementation of the helper functions is worth **1 point** and the implementation of `evaluate_policy`, `value_iteration`, and `policy_iteration` is worth **1 point for each function**. The experimental part is worth **4.5 points in total** where each experiment is worth **1.5 point** (Note that the points are transferable between experiments — a great experiment might offset poorer experiment). The final **1.5 points** are for the report itself. Summary of the grading is in table 1.

subtask	points
helper functions	1
`evaluate_policy`	1
`value_iteration`	1
`policy_iteration`	1
Experiment 1	1.5
Experiment 2	1.5
Experiment 3	1.5
report	1.5
total	**10**

Table 1
Summary of points receivable for the assignment.

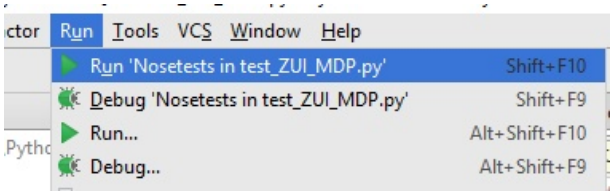

Figure 2. You can run the tests similarly as any other script in PyCharm — just keep active the test file and click on `Run`

6 Deadline

The deadline for submission into the upload system is 12.5.2020 23:59:59 CES.

References

[Hun07] John D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/mcse.2007.55. URL: http://dx.doi.org/10.1109/MCSE.2007.55.

[Was+16] Michael Waskom et al. *seaborn: v0.7.1 (June 2016)*. June 2016. DOI: 10.5281/zenodo.54844. URL: https://doi.org/10.5281/zenodo.54844.

[WCV11] Stefan van der Walt, S. Chris Colbert, and Gaël Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science & Engineering* 13.2 (Mar. 2011), pp. 22–30. DOI: 10.1109/mcse.2011.37. URL: http://dx.doi.org/10.1109/MCSE.2011.37.

Appendix A Notes

Please follow the course webpage and the forum for updates and additional information. In case of any questions, please write to the forum or write an email to `kuncvlad@fel.cvut.cz`.

A.1 LaTeX editor

Unless you have installed LaTeX locally, it is recommended to use Overleaf v2 which provides an online editor and also has many predefined templates and also allows online collaboration (which might be useful for your other projects). The Overleaf v2 is the result of a merge of Overleaf v1 and ShareLatex several years ago.

If you have LaTeX installed **locally**, it is recommended to set the `matplotlib` with LaTeX which allows using LaTeX code inside the figure — e.g. for the legend or axis labels.

A.2 Using Unittests

While the tests can be launched from a terminal, you can launch them also directly from the PyCharm IDE. The usage is very simple and is shown in figs. 2 and 3. Similarly, the tests also can be run in another popular editor Visual Studio Code. Example of the result is shown in fig. 4.

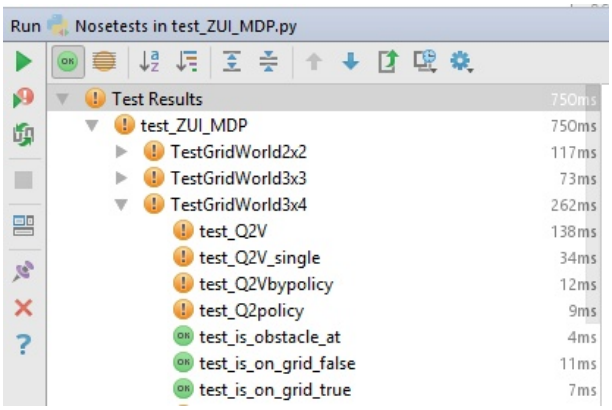

Figure 3. The results of the tests in PyCharm are shown in the lower left corner by default. You can click on any of the tests to see details (and the script output in case you print anything).

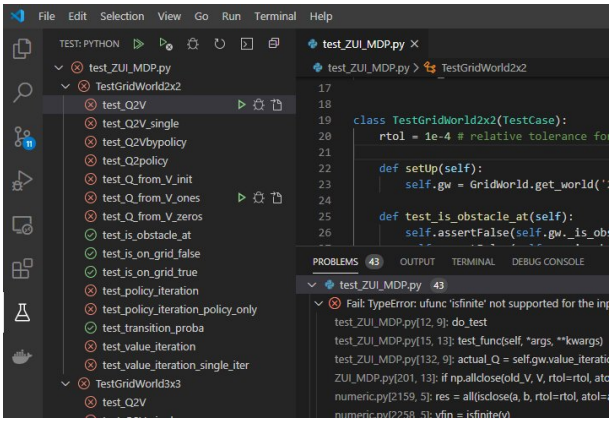

Figure 4. The results of the tests as shown in Visual Studio Code. You can click on any of the tests to rerun it or debug it. You can also go through more detailed outputs using the buttons above the tests.

A.3 Virtual environment

It is recommended to use a separate virtual environment for your assignment. If you are using the Anaconda distribution, you can create a new environment using

```
conda create -n MDP python=3.8 numpy seaborn matplotlib
```

which can be then activated using `conda activate MDP` (or using `activate MDP` on Windows and `source activate MDP` on Linux for older versions of conda). More details are available in the documentation.

If you choose to check your implementation using the provided tests (as recommended), you also have to install the `nosetests` or `pytest` in the activated conda environment:

```
conda install nose
```

or

```
conda install pytest
```

A.4 Docker

There is also a prepared docker image `felzui/zui-mdp` with all necessary packages, however, unless you connect the container to an X server, it is terminal only and thus the visualizations will not work (details of connecting to an X server are out of scope of this short document).

For an interactive session, run the following command:

```
docker run -v /home/user/MDP:/MDP -it felzui/zui-mdp:latest bash
```

where `/home/user/MDP` is your local absolute path to the downloaded assignment. This command runs a docker container and mounts the supplied path to the directory `/MDP`.

A.5 Saving figures

You should save the figures in a vectorized format, which is better for publication. This can be done easily in python using

```
plt.savefig('your_filename.pdf', dpi=500, transparent=True)
```