Reinforcement Learning using Direct Utility Estimation
Problem (21.6)
Team (1)

Mohamed Shawky Zaky AbdelAal Sabae
Section:2,BN:15

Remonda Talaat Eskarous
Section:1,BN:19

Mohamed Ramzy Helmy
Section:2,BN:13

Mohamed Ahmed Mohamed Ahmed
Section:2,BN:10

***Abstract*— Direct Utility Estimation (DUE) is a simple method for passive reinforcement learning. It's simply an estimation of utility function using random trials and moving average at each state. In this work, we discuss our implementation of DUE using both tabular representation and function approximation. We compare between both based on performance and execution time using grid world problem.**

I. INTRODUCTION

Direct Utility Estimation **(DUE)** is a simple method of estimating utility function in *passive reinforcement learning* by keeping moving average of the utility at each state. There are many representations of a utility function, which include *tabular representation* and *function approximation*. In this work, we use three setups of the grid world problem to compare between both representations according to *direct utility estimation*, as well as checking the performance of *direct utility estimation* on this problem.

II. IMPLEMENTATION DETAILS

A. Grid Worlds

Three grid world configurations are used:

- $4X3$ gird world described in chapter 21
- $10X10$ grid world with reward of $+1$ at $(10, 10)$
- $10X10$ grid world with reward of $+1$ at $(5, 5)$

Keep in mind that any **non-terminal** state has a reward of -0.04. The implementation of the three grid worlds can be found in `grid.py` file.

B. Single Random Trial

Performing a single random trial is considered a **common** function between both *tabular representation* and *function approximation*. It is performed as follows:

- First, a random starting state *(cell)* is chosen to provide more coverage to all states.
- Then, the agent loops until it reaches a terminal states. At each step:
  - The agent samples an action according to the given policy **(user input)**, with probability of 80% of choosing the policy action and 20% of choosing left or right of the policy action.
  - Then, the agent applies the sampled action and add the next state rewards to all the visited states, so far.
  - If the agent visits the same state twice, it's considered a different entry.
- Finally, the agent returns the set of visited states and their corresponding utilities.

The implementation of a single random trial can be found in `utils.py` file and the policies input can be changed in the text files inside `policies` folder.

C. DUE with tabular representation

The direct utility estimation using tabular representation goes as follows:

- First, a grid is initialized to store the estimated utilities **(with Nones)**.
- The agent loops over large number of trials $(10,000)$. At each step:
  - It performs a single random trial **(described above)**.
  - It summarizes the trial results, by **collecting** the repeated states and **average out** their estimated utilities.
  - It updates estimated utilities table by taking the moving average **(mean)** of the current utility and the estimated utility in the current trial for each cell.

The implementation of DUE with tabular representation can be found in `due_tabular.py` file.

D. DUE with function approximation

The used utility function contains 3 parameters and if formulated as: $U_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y$. The direct utility estimation using function approximation goes as follows:

- The agent initializes the parameters uniformly from 0 to 1
- The agent loops over large number of trials $(10,000)$. At each step:
  - It performs a single random trial **(described above)**.

  - It uses **Widrow-Hoff update rule** to update the parameters with the trial results *(a default learning rate is set to* 0.001*)*.

The implementation of DUE with function approximation can be found in `due_fa.py` file.

III. CODE USAGE

- First, update the policies in text files inside `policies` folder. Each text file is named by the order of its corresponding grid world.
- Second, run the code using:
  `python run.py <agent_number> <grid_number> <path/to/policy/text/file>`
  Specifying the agent number *(1 for tabular representation / 2 for function approximation)*, the grid number and the path to the policy text file.
- For example:
  `python run.py 1 1 policies/1.txt` runs DUE with tabular representation on the $4X3$ grid world.

IV. EXPERIMENTAL RESULTS

A. Performance and Correctness

The policies used for these experiments is a **near-optimal** policies *(input from user and can be found in policies folder)*. The actions sampling mechanism is described above. Also, note that we consider the **top-left** corner as the *origin* of the world.

1) 4X3 Grid World:

- For the $4X3$ grid world, the tabular representation DUE yielded the following estimated utilities:

0.839	0.920	0.960	1.000
0.815	None	0.916	-1.000
0.772	0.708	0.790	0.665

- However, the function approximation DUE yielded the following estimated utilities (with parameters: $\theta_0 = 0.923/\theta_1 = 0.0153/\theta_2 = -0.166$):

0.924	0.939	0.954	0.970
0.757	None	0.787	0.803
0.590	0.605	0.620	0.636

- We can see that the tabular representation performs **better than** the function approximation on that grid configuration, as it can accurately predict the utilities of the terminal states and predict reasonable utilities for all grid cells. This is mainly because the parameters update suffers from some *inconsistency*, due to the negative terminal, which leads to negative utilities. This problem can be **mitigated** by **increasing the number of parameters (features)** in the function approximator to adapt with both positive and negative utilities.

2) 10X10 Grid World with Reward at (10, 10):

- For the $10X10$ grid world with reward $+1$ at $(10, 10)$, the tabular representation DUE yielded the following estimated utilities:

0.534	0.564	0.622	0.678	0.722	0.774	0.827	0.880	0.925	1.000
0.492	0.504	0.585	0.621	0.670	0.742	0.771	0.835	0.880	0.959
0.444	0.374	0.407	0.645	0.694	0.732	0.746	0.787	0.850	0.904
0.308	0.309	0.402	0.480	0.532	0.647	0.730	0.779	0.822	0.852
0.264	0.377	0.414	0.472	0.532	0.634	0.666	0.728	0.781	0.802
0.282	0.215	0.365	0.433	0.523	0.540	0.629	0.670	0.665	0.746
0.194	0.326	0.332	0.441	0.482	0.505	0.566	0.605	0.645	0.705
0.141	0.238	0.268	0.305	0.361	0.412	0.491	0.525	0.578	0.650
0.100	0.146	0.131	0.252	0.353	0.387	0.415	0.454	0.551	0.560
0.114	0.191	0.135	0.118	0.174	0.283	0.397	0.398	0.440	0.484

- However, the function approximation DUE yielded the following estimated utilities (with parameters: $\theta_0 = 0.526/\theta_1 = 0.055/\theta_2 = -0.055$):

0.527	0.582	0.637	0.693	0.748	0.803	0.858	0.914	0.969	1.024
0.472	0.527	0.582	0.637	0.693	0.748	0.803	0.858	0.914	0.969
0.416	0.472	0.527	0.582	0.637	0.693	0.748	0.803	0.858	0.914
0.361	0.416	0.472	0.527	0.582	0.637	0.692	0.748	0.803	0.858
0.306	0.361	0.416	0.471	0.527	0.582	0.637	0.692	0.748	0.803
0.250	0.306	0.361	0.416	0.471	0.527	0.582	0.637	0.692	0.748
0.195	0.250	0.306	0.361	0.416	0.471	0.527	0.582	0.637	0.692
0.140	0.195	0.250	0.306	0.361	0.416	0.471	0.526	0.582	0.637
0.084	0.140	0.195	0.250	0.305	0.361	0.416	0.471	0.526	0.582
0.029	0.084	0.140	0.195	0.250	0.305	0.361	0.416	0.471	0.526

- We can see that both performs well on this particular grid configuration, as both can predict the terminal state right and, also, give very reasonable assumptions about the utilities of other states *(cells)*.

3) 10X10 Grid World with Reward at (5, 5):

- For the $10X10$ grid world with reward $+1$ at $(5, 5)$, the tabular representation DUE yielded the following estimated utilities:

0.589	0.607	0.694	0.719	0.787	0.763	0.728	0.651	0.453	0.468
0.683	0.599	0.743	0.772	0.837	0.761	0.669	0.726	0.573	0.490
0.664	0.678	0.742	0.849	0.874	0.879	0.797	0.732	0.561	0.543
0.706	0.747	0.839	0.870	0.915	0.920	0.832	0.794	0.627	0.657
0.751	0.795	0.836	0.919	1.000	0.957	0.888	0.787	0.703	0.654
0.707	0.813	0.827	0.914	0.910	0.775	0.832	0.759	0.685	0.572
0.719	0.765	0.823	0.835	0.915	0.880	0.515	0.632	0.701	0.518
0.642	0.709	0.742	0.785	0.858	0.839	0.703	0.591	0.698	0.658
0.611	0.670	0.687	0.673	0.795	0.778	0.694	0.537	0.601	0.599
0.509	0.609	0.605	0.576	0.749	0.734	0.591	0.553	0.438	0.502

- However, the function approximation DUE yielded the following estimated utilities (with parameters: $\theta_0 = 0.934/\theta_1 = -0.0047/\theta_2 = -0.0125$):

0.935	0.930	0.925	0.921	0.916	0.911	0.906	0.902	0.897	0.892
0.922	0.917	0.913	0.908	0.903	0.899	0.894	0.889	0.884	0.880
0.910	0.905	0.900	0.896	0.891	0.886	0.881	0.877	0.872	0.867
0.897	0.892	0.888	0.883	0.878	0.874	0.869	0.864	0.859	0.855
0.885	0.880	0.875	0.870	0.866	0.861	0.856	0.852	0.847	0.842
0.872	0.867	0.863	0.858	0.853	0.848	0.844	0.839	0.834	0.830
0.860	0.855	0.850	0.845	0.841	0.836	0.831	0.827	0.822	0.817
0.847	0.842	0.838	0.833	0.828	0.823	0.819	0.814	0.809	0.805
0.835	0.830	0.825	0.820	0.816	0.811	0.806	0.801	0.797	0.792
0.822	0.817	0.813	0.808	0.803	0.798	0.794	0.789	0.784	0.779

- Here, we can see that the function approximation fails to estimate any reasonable utilities. At the same time the tabular representation can obtain pretty reasonable utility estimates. This is mainly because the true utility is no more linear *(like the previous configurations)*, however it's more like a pyramid. This makes it very difficult for a function approximator with very few parameters (3) to estimate reasonable utilities. Also, all of the included parameters is *linear*, which cannot solve the pyramid hierarchy given by this configuration.
- A reasonable solution for that is to add **non-linearity** to the function approximator. One possible suggestion *(by the textbook)* is to use the **Euclidean distance** between current state and the goal, as a third term in the equation. This will improve the utility estimates, however we didn't implement it due to *time constraints*.

B. Execution Time

Configuration	Tabular	Func. Approx.
4X3	0.173	0.172
10X10 (10,10)	0.669	0.847
10X10 (5, 5)	0.376	0.445

TABLE I: Execution time of both methods in different grid configurations (measured in seconds)

It's clear that, **at small grid sizes**, both methods takes almost the same time. However, as the grid size **increases**, the function approximation becomes **slower than** the tabular representation. This is mainly because the number of *parameter updates* increases, which is more computationally expensive than just keeping the moving average of the observed utilities.

V. CONCLUSION

To sum up, *Direct Utility Estimation* is a very simple method to estimate state utilities. It can be implemented using either *function approximation* or *tabular representation*. Both methods perform well with linear utilities, however the function approximation method can suffer from some inconsistencies, due to *negative terminal states*. However, for **non-linear** utilities, a function approximator with only few linear parameters can miserably fail. This can be mitigated by adding more *non-linear* terms to the equation. On the other hand, function approximation is a good method for large state spaces given enough features. Also, a well-fitted function approximator can generalize very well to unseen states, unlike the tabular representation method that only works well for observed states.

REFERENCES

[1] Russell, Stuart J. (Stuart Jonathan). Artificial Intelligence : a Modern Approach. Upper Saddle River, N.J. :Prentice Hall, 2010.