Assignment 4: Markov Decision Processes

CS7641 Machine Learning, 2019 Fall

Hui Xia (hxia40)
903459648
Georgia Institute of Technology

1. Introduction

In many scenarios, the machine learning agent is facing the problem that can be simulated as a Markov Decision Process (MDP). MDP provides a method to model a control process in a discrete manner. In the MDP model, a process will be separated in a limited set of states (s), and an agent could perform a limited set of actions (a). In an MDP model, only the combination of current state and action will affect the result of decision, while the history on how the model end up in such state does not matter. This feature simplified process of decision-making and become well-used in solving problems in various areas, such as game-theory.

Reinforcement learning target to solved MDP problems by estimate the action of agent in a given state, in order to maximize the overall reward. In some (usually simpler) MDP case scenarios (e.g. the game Chess), it is possible to find out all possible state and action combinations, the correlated reward (r) and next state (s') by performing an action in a given state, and the transitional probability (P), if the decision process is stochastic. Such problems could be solved using Value Iteration (VI) and Policy Iteration (PI). In some other case scenarios, the MDP is just too complex to find and calculate on all possible state/action combinations (such as the game Weiqi) [1]. Or, the MDP process itself is not necessarily complex, but the total number of state, transition probability, and/or the reward are unknown. To solve such problem, Q-learning could be used to estimate the optimal policy.

In this report, I will solve two MDP problems: stock-market trading and FrozenLake, using three reinforcement learning methods: VI, PI, and Q-learning. To demonstrate on the strength/weakness of each reinforcement learning methods, MDPS of various size (i.e. number of states) will be investigated.

2. Experiment design

2.1. MDP problems

In this work, two MDP problems are analyzed. The first MDP is derived from OpenAI Gym - anytrading [2]. I have modified the source code, enabling it to generate the transitional probability function (P, s', r, done) for VI and PI. Also, by default, anytrading use the total profit to evaluate a given trading policy, which assume that before the whole trading process, the agent started with a fixed amount of money. This evaluation method is, in my opinion, problematic, as this method allows earlier decisions in the MDP process affect the reward of the latter decisions. For example, if an agent lost 99% of its money in one trading day, then even it will double its money five days in a row, it still underperforms. However, such agent actually has a great performance, if proper risk-management is used (e.g. the agent is given 100 dollars every day). Thus, instead of profit, I use accumulated reward from each tick, which assumes that before each trading tick (in our scenario, each trading day), the agent will start trading with same amount of money (no matter how much the agent has won/lost in its trading history). This assumption simplifies the evaluation of the trading policy, as the MDP decision history does not affect the future trading rewards.

The second MDP is adopted from OpenAI Gym – FrozenLake [3]. This MDP assume that an agent will need to walk through a frozen lake, which is a grid world, to recover an item. Some tiles of the FrozenLake are considered as ice holes. If the agent steps into an ice hole, the agent receives a reward of zero and the game ends. If the agent successfully retrieves the item, it receives a reward of one and the game ends. The map could be stochastic (i.e. 'slippery'), or not. I have tested various versions of the maps, including the default '8x8' map, which is slippery, and has 20% of the tiles as ice holes. I have also investigated a larger, non-slippery map to compare different reinforcement learning algorithms.

2.2. Experiment methods

2.2.1. Value Iteration and Policy Iteration

For an MDP problem, if the transition probability and reward is given, VI and PI are usually implemented to solve the MDP problem. In both of these algorithms, Bellman equation is used to update the Q-value of each state using the known probability and reward:

$$Q((s,a,p,r) \mid \alpha,\gamma) = (1-\alpha) * Q(s,a) + \alpha * (r(s,a) + \gamma * p(s)) \quad (1)$$

Where s is state, a is action, p is probability, r is immediate reward, α is learning rate, γ is reward discount factor, Q(s,a,p,r) represents the value of under certain state and action, when the probability of p happens and the immediate reward r is given. VI solves the MDP by first assuming an arbitrary value score (usually all zeros) to each state, then update the values of each state simultaneously using the Bellman equation. This process iterates until the values from all states converge. Using the optimized values, the optimized policy will then be determined.

Policy iteration solves the MDP by first assuming an arbitrary policy for each state, then calculate the value score with the given policy. Then, the algorithm updates the policy for each state based on the value score. This process iterates until the policy of all states no longer changes. The VI and PI codes have referred from Wei Feng's work [4].

2.2.2. Q-learner

For an MDP problem, if the transition probability and reward is unknown, Q-Learning, which updates states/ transition probability/reward ion a dynamic manner, could be used. Similar to VI and PI, another version of the Bellman equation, which does not rely on the transition probability, is used to update the Q-value of each state:

$$Q((s,a) \mid \alpha,\gamma) = (1-\alpha) * Q(s,a) + \alpha\,(r + \gamma \max_{a'}[Q(s',a')]) \quad (2)$$

In (2), r denotes the discovered reward, instead of the known reward function in (1). The Q-learning have referred from Morvan's work [5].

3. Results and Discussion

3.1. Stock trading

In this part, I have used VI, PI, and Q-learner to decide under a given state, which is the price action of last ten trading days, what kind of trading decision (short or long) should be made for a stock asset. The total number of states is 50 (trading days). Obviously, for any given stock asset, each state (again, the price action of last ten trading days) will be unique. Thus, I expect all three algorithms will perform quite well, given the deterministic nature of this MDP. In my opinion, calling this MDP deterministic is an understatement. Usually, if in an MDP, each action corresponds to multiple different reward and/or next state, we will call this MDP stochastic. On the other hand, if in an MDP, if each action corresponds to only one kind of next state, we will call it deterministic. However, in our case scenario here, both actions correspond to exact same next state (at least different result in different reward. Alas, otherwise it really will be an 'deterministic' MDP world). Although this deterministic seems over-simplified, I would consider it interesting, as analyzing how all three reinforcement algorithms performs could make good comparison among them.

Based on the performance of last 10 trading days, I have performed VI, PI, and Q-learning analysis on $SPY from Jan 1998 to March 1998, among sixty trading days on a

daily basis. As shown in **Table 1** and **Figure 1**, all three algorithms agree on what specific action to take on a given day. Intuitively, the optimized policy for this MDP is, a trading agent should long a stock, if the agent knows stock goes up next day, and short the stock if it to go down the next day. Thus, it is not surprising that with known reward function, VI and PI will find the optimized policy. As the agent's action does not affect 'next state', after two iteration, the Q-learner can also find the optimized policy. The total reward for all three algorithms agrees at 36.46, which means the trading agent obtained 36.46% profit if it starts with same amount of funding at the beginning of each trading day.

Table 1. On the stock trading MDP, converged policy from VI and PI, and converged value table from VI. Action of 0 and 1 indicates longing and shorting, respectively.

Day	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
PI policy	0	0	0	0	0	0	0	0	0	0	1	1
VI policy	0	0	0	0	0	0	0	0	0	0	1	1
VI value	68.0	68.0	68.0	68.0	68.0	68.0	68.0	68.0	68.0	68.0	68.0	67.3
Day	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
PI policy	0	0	1	1	0	0	0	1	1	1	0	0
VI policy	0	0	1	1	0	0	0	1	1	1	0	0
VI value	66.0	65.2	64.5	62.4	59.9	58.9	58.2	57.1	56.5	55.9	54.6	53.4
Day	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**
PI policy	0	0	1	0	1	1	0	1	0	1	0	0
VI policy	0	0	1	0	1	1	0	1	0	1	0	0
VI value	52.1	50.8	50.6	48.8	46.6	44.5	41.0	39.1	39.0	35.6	28.8	25.5
Day	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
PI policy	0	0	0	1	1	1	0	0	0	0	1	0
VI policy	0	0	0	1	1	1	0	0	0	0	1	0
VI value	22.8	22.6	20.6	20.2	20.1	19.8	18.8	18.5	17.3	16.8	16.6	13.8
Day	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**
PI policy	1	1	0	0	0	1	0	0	0	0	1	0
VI policy	1	1	0	0	0	1	0	0	0	0	1	0
VI value	13.1	12.8	10.5	6.6	5.7	5.4	5.0	3.5	3.4	1.7	0.8	0.0

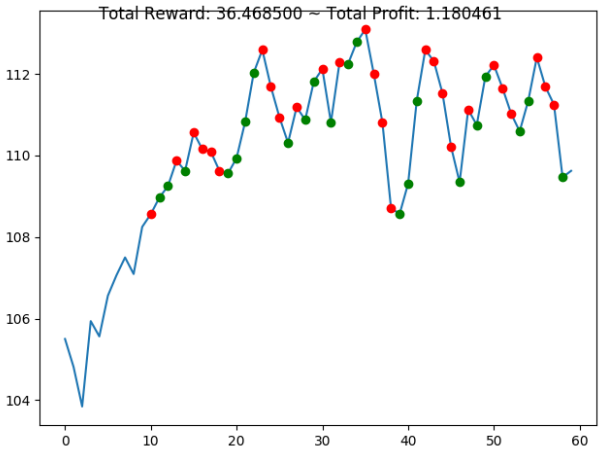

Figure 1. Decided actions from Q-learner

3.2. FrozenLake

All three algorithms are used to investigate the FrozenLake MDP from OpenAI, using default '8x8' map. Compared with the stock trading problem described in **Section 3.1,** While their number of states are comparable (60 for the former and 64 for the latter) this MDP is harder, as it described a stochastic process: as the ice surface is slippery, any action the agent make only have a 33% chance of success, and have a 66% chance to make other actions. Thus, even that know the map, neither VI and PI could perform perfectly: using the optimized policies shown in **Table 2**, VI and PI could only reach an average reward of 0.877 and 0.875, respectively. The VI and PI found convergence after running for 1027 iterations and 7 iterations respectively, and consumed in a total 0f 1.81 and 7.022 sec, respectively. As shown in **Figure 2**, in VI, as the number of iteration increases, the agent smoothly found smaller difference between all variations. While the maximum possible achievable reward is 1. Surprisingly, although neither VI nor PI reached the highest reward score, similar to what we see from **Table 1**, the policy derived from both VI and PI still agree with each other.

Table 2. On the FrozenLake MDP, converged policy from VI and PI, and converged value table from VI. Action of 0 and 1 indicates longing and shorting, respectively.

state	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
PI policy	1	2	2	2	2	2	2	2	3	3	3	3	3	3	3	2
VI policy	1	2	2	2	2	2	2	2	3	3	3	3	3	3	3	2
VI value	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
state	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
PI policy	0	0	0	0	2	3	3	2	0	0	0	1	0	0	2	2
VI policy	0	0	0	0	2	3	3	2	0	0	0	1	0	0	2	2
VI value	1.0	1.0	0.9	0.0	0.9	0.9	1.0	1.0	1.0	0.9	0.8	0.5	0.6	0.0	0.9	1.0
state	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
PI policy	0	3	0	0	2	1	3	2	0	0	0	1	3	0	0	2
VI policy	0	3	0	0	2	1	3	2	0	0	0	1	3	0	0	2
VI value	1.0	0.8	0.5	0.0	0.5	0.6	0.9	1.0	1.0	0.0	0.0	0.2	0.4	0.4	0.0	1.0
state	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
PI policy	0	0	1	0	0	0	0	2	0	1	0	0	1	2	1	0
VI policy	0	0	1	0	0	0	0	2	0	1	0	0	1	2	1	0
VI value	1.0	0.0	0.2	0.1	0.0	0.3	0.0	1.0	1.0	0.7	0.5	0.0	0.3	0.6	0.8	0.0

From an human observer's point of view, if the map (or the probability/reward function) is known, however, it is possible to derive a perfectly doable policy to keep the reward to be not lower than 1: the agent simply just need to make sure to always walk away from any ice-hole, and move toward the target. To realize such algorithm using computational agent, one may need to build a more detailed model of the FrozenLake MDP, highlighting to not drop into an ice-hole.

However, for a model-free algorithm like Q-learning, things get much harder. As shown in **Figure 3**, compared with VI and PI, which achieved 0.87-0.88 of average reward score in seconds, the best reward a Q-learner could get struggles below 0.6, when alpha equal to 0.01. As there is only one reward on the whole FrozenLake map, changing γ did not affect the algorithm performance. On the other hand, neither very high, nor very low α (**Figure 3**) and ε (data not shown, but I found similar trend in experiments) can grant us good performance. I reason this is because that at the first iterations of the experiments, the algorithm will need large value of α and ε to perform fast exploration of the map. After certain amount of iterations, the algorithm will then need a smaller α and ε to slowly tune and exploit

the available information. Thus, smartly setting α and ε as a decreasing function to the increasing iteration number could result a Q(s,a) that is similar to what VI/Pi have. However, finding such function could be computational-expensive and is beyond the scope of this report.

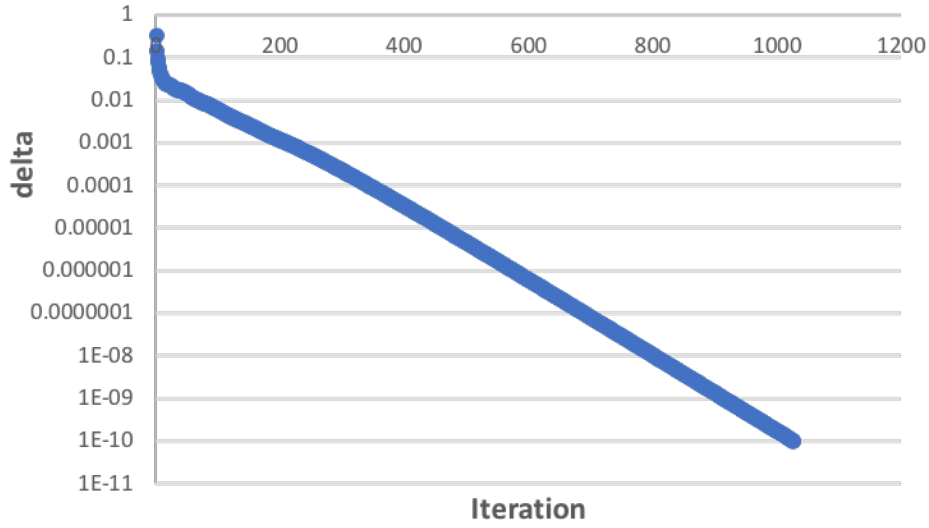

Figure 2. On the FrozenLake MDP 8x8 slippery, convergence of VI on different iterations.

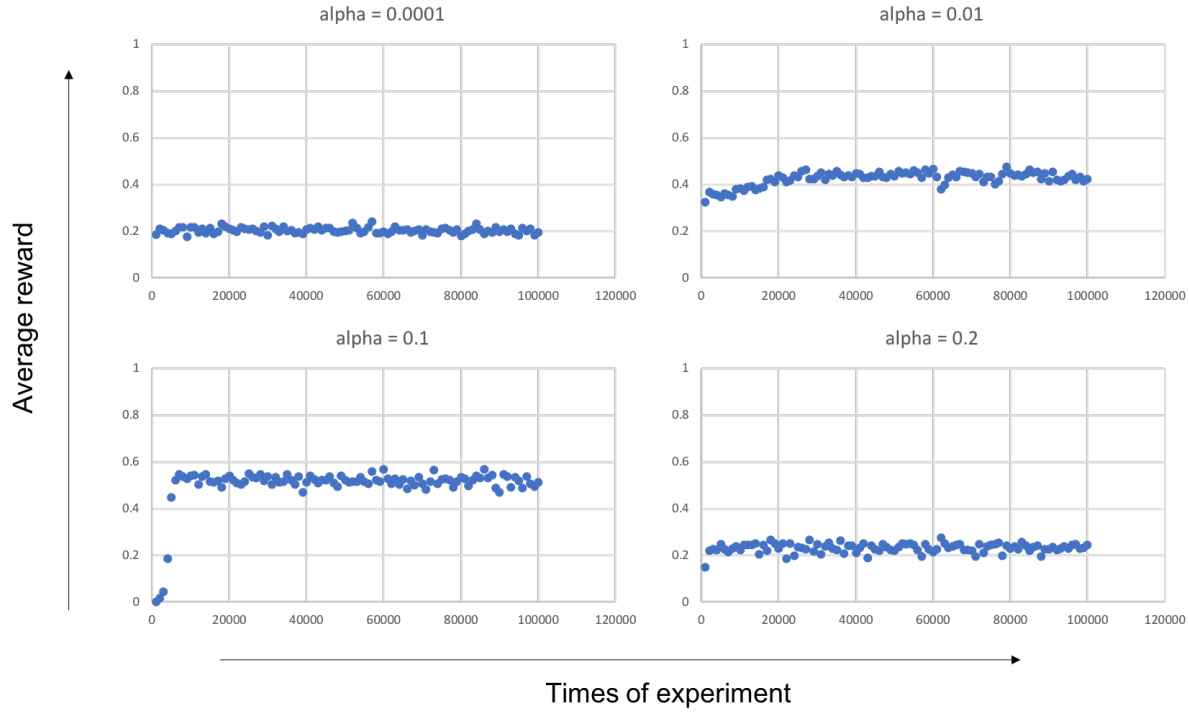

Figure 3. On FrozenLake 8x8 slippery MDP, average reward of Q-learner on different iterations. Alpha is the learning rate.

From **Figures 2** and **3** we can see that Q-learning has reached its limitation in solving complex (in terms of number of states) MDPs. To investigate on how the size of the MDP will affect the performance of Q-learning, I created a series of simplified FrozenLake maps of various size, from 2x2 to 32x32 (i.e. 1024 states). These maps are simplified by being 'non-slippery', and by using fewer ice-holes. Percentagewise speaking, this series of maps will only have 1% of their tiles as ice-holes (compared with 20% in the default '8x8' map used above). This simplification will enable us to create larger maps, while still being

able to obtain meaningful Q-learner performance. **Figure 4** demonstrate on how size of the simplified map affects how quickly and effective the Q-learner could find the solution. When the map is of 24x24 tiles and smaller, the Q-learner could always find the (at least nearly) 'perfect' solution, which will ensure an average reward of 0.99 or higher. However, when using a larger map, the performance of the Q-learner goes lower – it takes the Q-learner longer to reach its plateau of performance, and such performance is not even perfect.

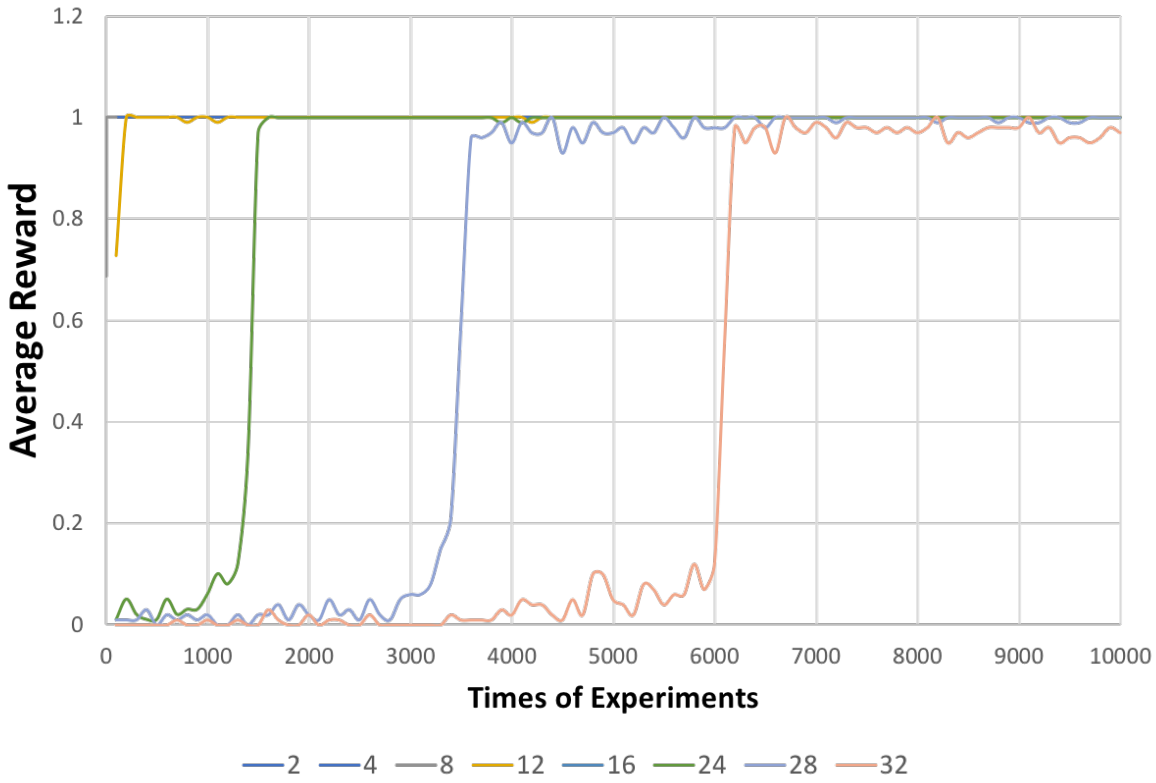

Figure 4. On FrozenLake 32x32 non-slippery MDP, average reward of Q-learner on different iterations. Alpha = 0.01.

On the other hand, similar to what we have seen for the 8x8 slippery FrozenLake, within 2 seconds and 10 seconds, and within 63 and 5 iterations, VI and PI finished analyzing the 32x32 non-slippery FrozenLake map, respectively. As the 32x32 map is non-slippery (i.e. deterministic), both VI and PI for the estimated reward of 1.0. As expected, Both VI and PI are faster and performs better compared with the Q-learner.

4. Conclusion

In this report, I have compared three different reinforcement algorithms over two MDP problems. Based on the above-listed observation, here, I would like to make an analogy: the difference between Q-learner and VI or PI is like the difference between simulated annealing (SA) and genetic algorithm (GA). On both side of the analogy, the former uses only one agent to explore the world of states, and the latter uses many. In case of VI or PI, there will be s agents that perform the exploration simultaneously. If the environment is given, it is institutive that VI/PI or GA will be advantageous. However, when the world is unknown, we are forced to rely on Q-learner. Unfortunately, Q-learner only has one agent. How to find the algorithm, to encourage this agent could firstly explore the world as thoroughly as possible, then encourage this agent to exploit its knowledge to the world as effectively as possible would improve the Q-learner.

References

[1]. Go (game). Retrieved from https://en.wikipedia.org/wiki/Go_(game)
[2]. gym-anytrading. (2019) Retrieved from https://github.com/AminHP/gym-anytrading
[3]. FrozenLake-v0 (2019). Retrieved from https://gym.openai.com/envs/FrozenLake-v0/
[4]. 强化学习基础篇: 价值迭代 (Value Iteration) (2019). Retrieved from https://zhuanlan.zhihu.com/p/33229439
[5]. Q-learning 算法更新(2019). Retrieved from https://morvanzhou.github.io/tutorials/machine-learning/reinforcement-learning/2-2-tabular-q1/