A Simple Reinforcement Learning System Development based on Q-learning

Guangfei Zhu
University of Wisconsin-Madison

Motivation

To help autonomous agents, such as robots, fulfill a wide range of user-specified tasks, it is necessary to train the agents to be capable of learning broadly applicable and general-purpose skill repertoires. However, most of the current reinforcement learning frameworks involving policies learning are designed for specifically individual goals such as pushing [1] and navigation [2], which, in general, are defined by hand-specified reward functions. Some existing reward functions are expressed as the pixel-wise Euclidean distance which is not effective for visual tasks due to the fact that distances between images does not necessarily mean distances between states [3, 4]. Compared with autonomous agents, humans are more proficient at setting environment-conditioned goals for themselves even from early infancy. Based on this, the work published in 2018 [5] is a step toward the challenging goal of general-purpose tasks implementation, and a similar but simplified visual reinforcement learning system is developed in current project.

Research Question

The work in [5] described that the robot was trained to understand and analyze the real-time scenarios, and to push the objects to the corresponding right locations for all scenarios. To achieve this objective, two necessary steps need to be implemented: creating enough real-time scenarios and find a suitable algorithm to train the robot. However, the robot currently is not available for this project. Thus, to mimic the robot, a virtual agent was created based on the Python code [6] to learn the manually generated real-time scenario through Q-leaning [7] algorithm, one famous algorithm in the field of reinforcement learning [8]. In this way, the visual reinforcement learning system was established. One of the reasons why reinforcement learning is preferred is that people want the robots or some agents to find the optimal path to reach the goal from some random initial states in order to reduce the cost.

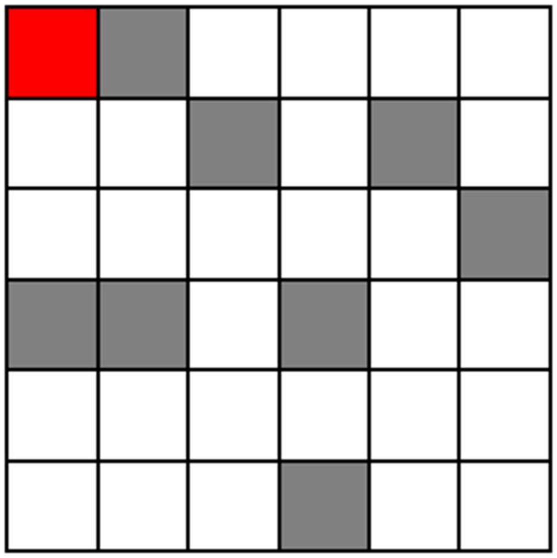
Basically, the paper is laid out as following sections. First, the training data is created. Then, the Q-learning algorithm and its coupling with the epsilon-greedy method are introduced. Followed by this is the results analysis and finally is the future work.

Related Work

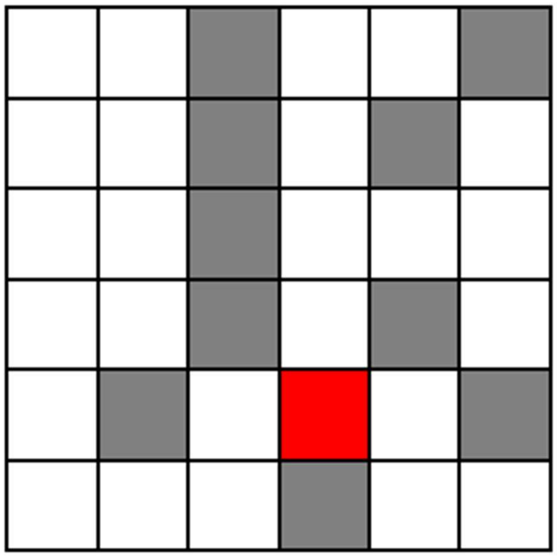
Q-learning developed by Watkins [7] is to apply an action-value function, Q, to directly approximate $\boldsymbol{q}_*$, the optimal action-value function, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs [8]. For Q-learning, one could choose the n-step bootstrapping to approximate $\boldsymbol{q}_*$. For instance, the regular Q-learning is similar to a TD (0). Although Q-learning is a very powerful algorithm, its main weakness is lack of generality. If one views Q-learning as updating numbers in a two-dimensional array (Action Space * State Space), it, in fact, resembles dynamic programming. This indicates that for states that the Q-learning agent has not seen before, it has no clue which action to take. In other words, Q-learning agent does not have the ability to estimate value for unseen states. To deal with this problem, DQN (Deep Q Network) [9] gets rid of the two-dimensional array by introducing Neural Network which is capable of solving more complex tasks, such as the pendulum problem [10]. In current project, the traditional Q-learning is applied for the agent development.

Training Data Generation

It is important to generate the training data for the agent to learn. As a good start, it is better to try some simplified situations and then extend to more complex circumstances. To generate the training data, the VAEs (variational auto-encoders) [11] was employed in ref. [5]. In this project, we focus on 6 by 6 grid cases which means 36 grids/pixels in total. Three colors were used, white, grey, and red, to represent the available path, barrier, and goal, respectively. The virtual agent generated in the Python code is supposed to walk through the white grid (available path) instead of barrier. *Figure 1* shows two examples of the situations for the agent to learn, and 32 different cases were manually generated in this project. The reason why data is manually generated instead of through VAEs because such kind of 36 pixels would lead to very non-accurate random 6 by 6 grid generations based on VAEs. Based on some researches, usually training data with size 28 by28 are good to train the VAEs and then generate some random test data by inputting some noise data to the decoder of VAEs [12].



(a)



(b)

Figure 1. Two manually generated cases for the virtual agent to learn (white grid means available path, grey grid means barriers, and red grid means the goal)

For each grid, four actions are allowed: UP, RIGHT, DOWN, and LEFT. The agent should try to avoid the barrier since it would get a bad reward if it did. The wall/bound should also be avoided. Furthermore, a universal transition matrix P is established for all these 32 cases. The P is a $N_S \times N_A$ matrix with N_S being the total number of states (36 for the 6 by 6 grid in *Figure 1*) and N_A being the total number of available actions for each state (4 in current project).

Reinforcement Learning Algorithm

The algorithm that learn the random case among the 32 manually generated is the Q-learning which is an off-policy algorithm since it tries to find the current best action to implement instead of abiding the previous policy. The formula of Q-learning is given below:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

where Q is the action-value function, S_t, A_t represent the state and action at time t, α is a parameter, R_{t+1} is the reward, and γ is the discount rate. One thing needs to pay attention to is the term $\max_a Q(S_{t+1}, a)$ which means to take the action that maximize the action value at the next time state S_{t+1}. In addition, the sum of R_{t+1} and $\gamma \max_a Q(S_{t+1}, a)$ approximate the true return of Monte Carlo method [8].

As an auxiliary of Q-learning, the epsilon-greedy method was selected based on a given Q-function and epsilon. Assume there are N_A actions for each state. Then the probability of implementing the best action among N_A actions is given as

$$1 - \varepsilon + \frac{\varepsilon}{N_A}$$

For the rest non-best-action $N - 1$, the corresponding probability to implement is

$$\frac{\varepsilon}{N_A}$$

The reason why to choose epsilon-greedy method is due to that this policy could help get out of local optima. Besides, the performance also depends on the epsilon value that one chooses. If a non-suitable value of epsilon was chosen, it could also leads to local optima and could not help the Q-learning find the goal which is the red grid in *Figure 1*. We will show those phenomena and explain the reason in the results section.

Parameters Setting

The parameters used in Q-learning and epsilon-greedy are given in *Table 1*.

Table 1 Parameter setting in Q-learning and epsilon-greedy method

Parameters	α	γ	ε	Episodes
Values	0.2	0.95	0.6	2000

In *Table 1*, the term "Episodes" means the number of iterations for Q-learning. The important parameter ε is set to 0.6 since this value perform the best compared with other values. Moreover, the number of available actions N_A for each state is 4. Thus, the probability to implement the best action is:

$$1-\varepsilon+\frac{\varepsilon}{N_A}=1-0.6+\frac{0.6}{4}=0.55$$

It can be seen that this probability is not as high as 0.8 or above. The reason for that is due to the barrier setting among 32 cases. As a result, the probability to implement the non-best action is:

$$\frac{\varepsilon}{N_A}=\frac{0.6}{4}=0.15$$

Results

To increase the entertainment of the project, the author created some human-computer interaction interface.

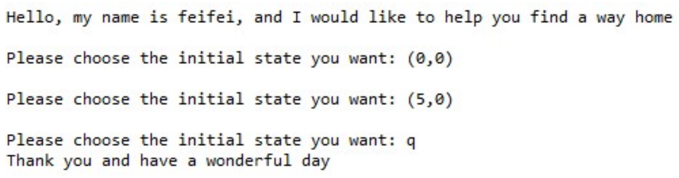
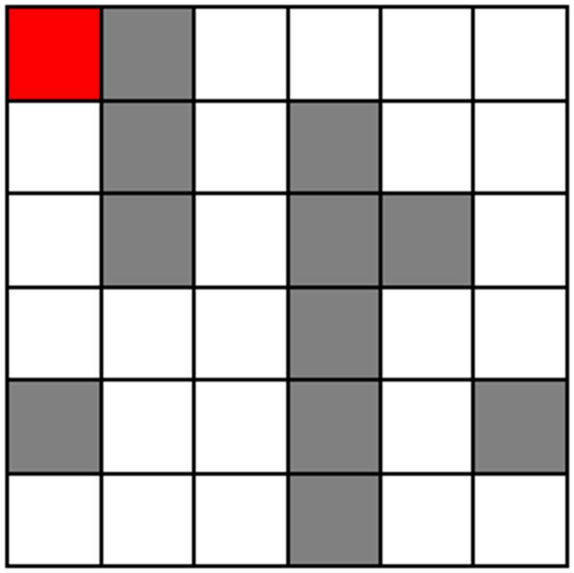
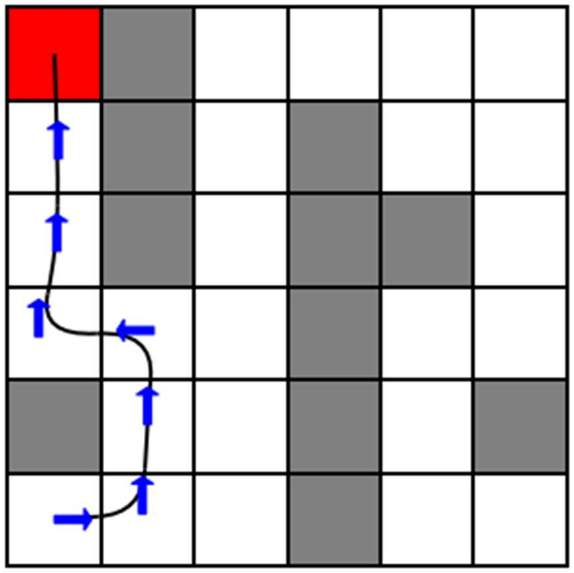

```
Hello, my name is feifei, and I would like to help you find a way home

Please choose the initial state you want: (0,0)

Please choose the initial state you want: (5,0)

Please choose the initial state you want: q
Thank you and have a wonderful day
```

Figure 2 Human-computer interaction interface

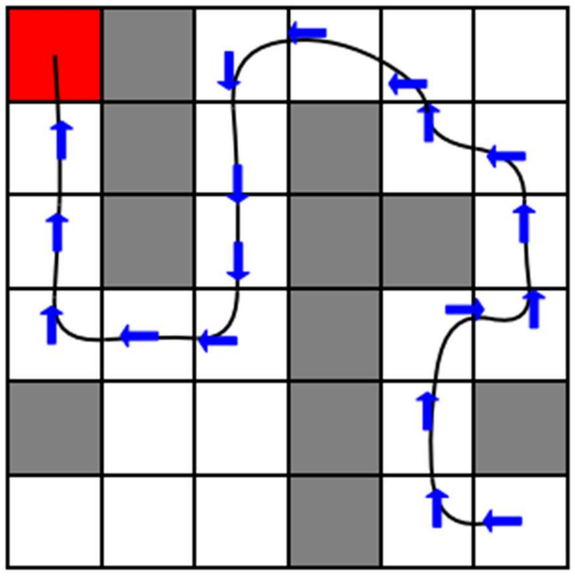
Figure 2 shows an example of human-computer interaction interface. The agent randomly chooses a case among the 32 manually generated cases and learn the optimal policy based on the given situations. After that, the agent asks the person to input a state as a starting state and try to find an optimal path to reach the goal. The state (0, 0) corresponds to the bottom left grid and the state (5, 5) corresponds to the top right grid in *Figure 1*. In *Figure 2*, the person inputs two states (0, 0) and (5, 0). For each input, the agent would display the optimal path to the goal. For instance, as shown in *Figure 3*.



(a)



(b)

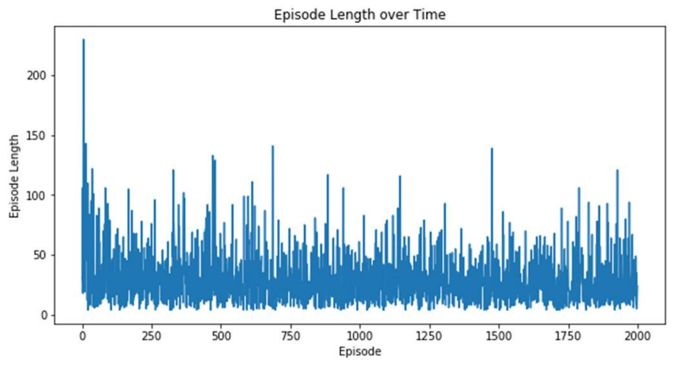


(c)

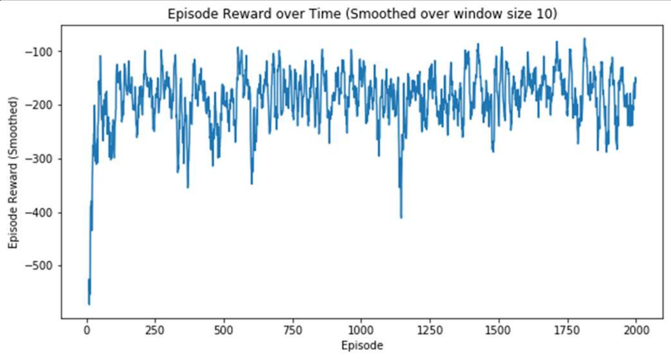
Figure 3. A random case with corresponding optimal path based on human input states ((a). a random chosen case; (b). the optimal path for the input state (0, 0); (c). the optimal path for the input state (5, 0))

In *Figure 3*, *Figure 3 (a)* is a random chosen case. When an initial state (0, 0) is input by the person, the agent displays the optimal path immediately (*Figure 3 (b)*). Then another initial state (5, 0) is input and the agent shows the optimal path again (*Figure 3 (c)*).

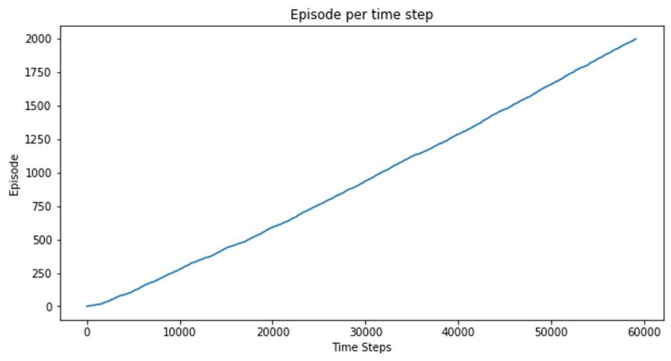
Figure 4 shows some related results with episode. *Figure 4 (a)* is the relationship between episode length and episode with episode length defined as the number of steps needed for each episode. It can be seen that 2000 episode is enough since the episode length keeps relatively stable.



(a)



(b)

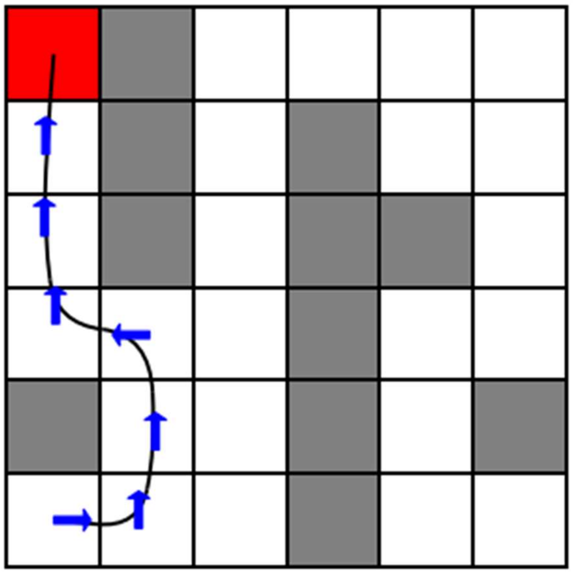


(c)

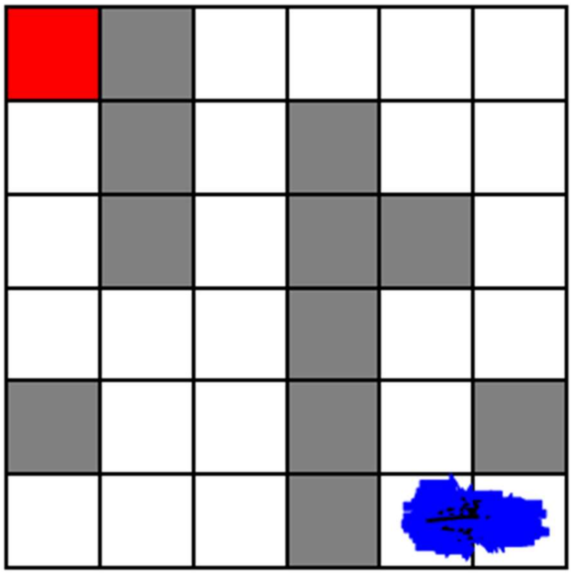
Figure 4. The episode results ((a). the episode length/time steps need to be converged for each episode; (b). episode rewards for each episode; (c). relationship between episode and total time steps)

Figure 4 (b) is the reward as a function of episode. It displays that when the number of episode comes to 2000, the reward fluctuation is relatively small indicates a good stop sign. *Figure 4 (c)* is the total episode number as a function of time steps. It is obvious that only at the very beginning (time steps equals 0) the line is a little bit flat indicates relatively more time steps needed to be converged at the beginning. This is consistent with the reward plot (*Figure 4 (b)*) where low episode rewards are achieved at the first several episode.

It is interesting to investigate the effect of epsilon value in the epsilon-greedy method on policy finding. The value of ε is set 0.1 under which the probability to choose the best action is $1-\varepsilon+\frac{\varepsilon}{N_A}=1-0.1+\frac{0.1}{4}=0.925$.As a result, the probability to implement the non-best action is $\frac{\varepsilon}{N_A}=\frac{0.1}{4}=0.025$. This means almost always to take the best action.



(a)



(b)

Figure 5. Optimal path by setting $\varepsilon = 0.1$ ((a). the optimal path for the input state (0, 0); (b). the optimal path for the input state (5, 0))

Figure 5 shows the results for the case shown in *Figure 3*. It shows that the results for state (0, 0) is not affected by the ε setting while state (5, 0) get trapped.

Summary and Future Work

The Q-learning algorithm is implemented in current project to develop a smart agent to learn real-time scenarios. To help the real-time learning, 32 cases are manually generated and is randomly chosen for the learning. The episode length and rewards are shown as a function of episode number to indicate a criteria for Q-learning. The effect of epsilon value is investigated and a suitable epsilon value needs to be chosen to get good results.

Two more things needs to be done in the future: first, extend the 6 by 6 grid to more complex geometries so that VAEs could be applied; second, two or more goals are expected to set to help the agent to find the best path to the best local goals.

References

1. Finn, C., & Levine, S. (2017, May). Deep visual foresight for planning robot motion. In Robotics and Automation (ICRA), 2017 IEEE International Conference on (pp. 2786-2793). IEEE.
2. Pathak, D., Mahmoudieh, P., Luo, G., Agrawal, P., Chen, D., Shentu, Y., & Darrell, T. (2018, April). Zero-shot visual imitation. In International Conference on Learning Representations.
3. Ponomarenko, N., Jin, L., Ieremeiev, O., Lukin, V., Egiazarian, K., Astola, J., ... & Kuo, C. C. J. (2015). Image database TID2013: Peculiarities, results and perspectives. Signal Processing: Image Communication, 30, 57-77.
4. Zhang, R., Isola, P., Efros, A. A., Shechtman, E., & Wang, O. (2018). The unreasonable effectiveness of deep features as a perceptual metric. arXiv preprint.
5. Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., & Levine, S. (2018). Visual reinforcement learning with imagined goals. In Advances in Neural Information Processing Systems (pp. 9208-9219).
6. Zelle, J. M. (2004). Python programming: an introduction to computer science. Franklin, Beedle & Associates, Inc..
7. Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.
8. Sutton, R. S., & Barto, A. G. (1998). Introduction to reinforcement learning (Vol. 135). Cambridge: MIT press.
9. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.
10. Wen, Z., & Van Roy, B. (2017). Efficient reinforcement learning in deterministic systems with value function generalization. Mathematics of Operations Research, 42(3), 762-782.
11. Doersch, C. (2016). Tutorial on variational autoencoders. arXiv preprint arXiv:1606.05908.
12. Retrieved from the github website: https://github.com/lipiji/variational-autoencoder-theano.