Exploration-Exploitation Strategies in Deep Q-Networks Applied to Route-Finding Problems

1st Pengyuan Wei

School of International Education. Beijing University of chemical technology, Beijing, China

a2017100054@mail.buct.edu.cn, 1527755386@qq.com

Abstract—Reinforcement learning is a class of algorithm that allows computers to learn how to accumulate rewards effectively in the environment and ultimately get excellent results. Among them, the exploration-exploitation tradeoff is a very important concept, since a good strategy can improve learning speed and final total reward. In this work, we applied DQN algorithm with different exploration-exploitation strategies to solve traditional route-finding problems. The experimental results show that the epsilon greedy strategy with a parabolic drop in epsilon value over reward improvement is the best, while it is not satisfactory after incorporating the softmax function. We hypothesized that the simplicity of the maze we use in this work in which the agent attempts to find the shortest path leads to the inadequacy of applying softmax to further encourage exploration. Future work thus involves experimenting with mazes at different scales and complexities and observing which exploration-exploitation strategies work best in each condition.

1. INTRODUCTION

Deep Reinforcement Learning is a new class of algorithm that combines deep learning and reinforcement learning to achieve end-to-end learning from perception to action. More similar to how human learns by trial and error, deep reinforce learning is a framework where an intelligent system makes a sequence of decisions, which acts upon the environment, and perceives the consequence of each decision [1,2]. Perceived information at each step would influence how the intelligent system adjusts its knowledge, embodied by a deep neural network, and makes its next decision. There is no hand-crafted engineering work between perception and action.

Deep reinforcement learning has the potential to enable robots to truly and autonomously learn new skills [3-5]. Although the idea of combining deep learning and reinforcement learning was proposed decades ago, DeepMind's Playing Atari with Deep Reinforcement Learning published at NIPS 2013 is the real beginning, where the name Deep Reinforcement Learning was proposed for the first time, and a DQN (Deep Q-Network) algorithm was proposed to achieve the results of playing Atari games from pure image input through learning [1].

Based on the above research, we want to use different deep reinforcement learning approaches to let the computer solve the maze problem, that is, find the shortest path of the maze. We applied the epsilon-greedy algorithm, changed the size and descending rule of the epsilon value to find the most effective epsilon changing rule [6-8]. Further, we incorporated softmax with DQN to study the feasibility of Deep Q- learning with Softmax (DQwS). We hypothesized that using softmax would increase the relative speed of the learning process.
Through experiments, we found that when the initial epsilon value is 0.5 and the parabola is reduced, the system’s learning efficiency is the optimal. However, the effect of its combination with softmax is not ideal. The application of softmax in DQN is worth further exploration.

2. BACKGROUND

2.1 Q-learning

The Q-learning is a kind of algorithm for reinforcement learning [9]. Reinforcement Learning is a type of Machine Learning, also a branch of Artificial Intelligence. RL uses the Markov Decision Process to automatically determine the ideal behavior within a specific context, in order to maximize its performance. MDP is the most common method that is used in reinforcement learning, which includes five elements: an agent, states, actions, rewards and policies. The simple reward feedback is required for the agent to learn its behavior, which is known as the reinforcement signal. In Q-learning, “Q” represents the “quality” of the policy. It has the Q-table that has all possible combinations of action and state and their corresponding reward values. These rewards are called Q-values. In the processes of learning, the agent uses a function called Q-function to get the action based on a determined state and to find the maximum discounted reward in the future. The Q-function can gives agent the best action when the agent is situated in a certain state, and gives the maximum possible value at the end of the training. When training begins, an initialized Q-function is given and the agent gets an initial state. What agent does is using the function to find the best action that will lead to the best end result under the known “Q-function”. The main idea in the Q-learning is using the Bellman equation to iteratively approximate the Q-function [9]. The Bellman equation is the maximum future reward for a state and an action and is equal to the immediate reward plus maximum future reward for the next state. To use the game to illustrate the work flow of Q-learning: the agent uses Q-function to play the video game repeatedly to approximately learn all the reward values in Q-table, and constantly update the Q-function. After that, the agent will know the optimal policy to get the maximum profit at the end. However, there is a shortcoming in the Q-learning. For many problems in the reality, their Q-tables have too many entries to learn, sometimes more than the number of atoms in our universe. Agents in those situations are unable to reach and learn all entries in the Q-table.

2.2 Deep Q-learning

Deep Q-learning addresses this problem in effectively [1,2]. Since many states in Q-table will never occur, we can safely use a sparse table containing only occurred states to represent Q-table. Even so, the most of states are very rarely visited and it will spend a very long time to be discovered by training. Ideally, people would also like to have a good guess for Q-values for states people have never seen before. Since the shortcomings of Q-learning, people tend to use the Deep Q Network, also called deep Q-learning, to solve the problem. The DQN is the combination of deep learning and reinforcement learning. Deep learning is also a type of Machine Learning, which in this context leads to replacing Q-tables with neural networks. The deep neural network works just like the Q table in that it takes agent’s state as input and returns the Q values associated with all possible actions the agent can take. What makes deep neural network superior is that it is known to be powerful function approximator [10-12], and in this case it could approximate the impractically large Q table pretty well with a compact neural network model when using good learning techniques such as experience replay and target network separation. Deep Q-learning can be applied in many fields. Beyond playing video games, the fastest path in logistics warehouse for transport robot can also be determined by this algorithm [13,14]. Prior works have also explored using deep reinforcement learning along with an advanced contextual text generation model to produce highly readable summaries of long texts [15].

3. EXPLORATION-EXPLOITATION STRATEGY

In reinforcement learning, there have two very important concepts: the exploration and the exploitation. Exploration and exploitation are a pair of contradictory elements. Exploration refers to the selection of
actions that have not been performed before to explore more possibilities. Exploitation refers to the selection of actions that have been performed to refine the models of known actions and to leverage existing knowledge. However, the problem is, finding best long-term strategy may involve short-term sacrifices, and making the best overall decisions usually requires the possession of enough information. How to balance between exploration and exploitation is an important issue and challenge in RL. In making this tradeoff, plenty of strategies have been explored in prior works:

- The **greedy algorithm** always exploits and never explores. This can lock the agent into performing sub-optimal action forever.
- The **epsilon-greedy algorithm** continues to explore. With probability $1 - \epsilon$ the agent exploits the known best action, and with probability $\epsilon$ the agent explores, where $\epsilon$ is a value between 0 and 1. This algorithm brings the agent closer to finding optimal action and policy.
- The **optimistic initialization** initializes all Q-values to high value to encourage more thorough, systematic exploration early on.
- The **decaying epsilon-greedy algorithm** has logarithmic asymptotic total regret, but scheduling the epsilon decay requires advance knowledge of rewards.

In this work, we applied the different epsilon and Boltzmann Exploration (softmax) strategies to explore the mechanics of exploration and exploitation.

4. **EXPERIMENT SETUP**

Our goal is to apply various DQN techniques to solve traditional route-finding problems and to study the relationship between different exploration-exploitation strategies and our DQN agent’s learning time. The experiment use two 6×6 mazes. The maze has two kind of cells: free cells and occupied cells. The agent starts from the top left cell, ends at the bottom right cell, and can only move along the free cells. In formal reinforcement learning, the final solution is such that the agent finds the optimal sequence of states, where the cumulative sum of rewards is the largest. In our model, the solution is defined such that the program finds the shortest path from start to end. To incentivize the discovery of the shortest path, the reward policy is designed as follows: Each move will consume 0.04 points. Trying to reach the occupied cell will deduct 0.75 points, trying to get out of the map boundary will deduct 0.8 points, and trying to visit any previously visited cell will deduct 0.25 points. The agent is rewarded with 1 point when it successfully reaches the end point. When the total reward is lower than -3 points, the game is deemed to have failed. Experience replay is used in the learning.

For this work, we used Keras version 4.2.2 on top of Google's TensorFlow version 0.14.1. We ran our experiments with Intel core i5-6300HQ CPU and Nvidia CeForce GTX 960M GPU. The computer uses a 8GB Hynix's random access memory. There are three parameters in our experiments: the epsilon value, the decrease rate of epsilon and the temperature in the softmax function. This is the code to judge the combination of algorithm:

```python
# determine whether to use softmax and get the next action
if np.random.rand() < epsilon:
    # explore
    action = random.choice(valid_actions)
elif softmax == True:
    # exploit with softmax-ed probabilities
    p = SOFTMAX(experience.predict(prev_envstate), temperature)
    action = generate_action(p)
else:
    # exploit with model-predicted probabilities
    action = np.argmax(experience.predict(prev_envstate))
```
For each parameter setting, the agent trains on the same maze for 50 times, each time with its neural network weights randomly initialized. We measured the spent time and value of epoch for each training and charted them for ease of comparison.

5. **Observation Results**

![Histogram of epoch values of epsilon-greedy algorithm.](image1.png)

**Fig. 1.** Histogram of epoch values of epsilon-greedy algorithm.

When the epsilon value is 0.4, the mathematical expectation and variance of the epoch value are both minimum. When the epsilon value goes larger or smaller from 0.4, the variance increases. When the epsilon value is close to 1, the characteristics of the epoch value become worse, far exceeding the situation when the epsilon value approaches 0.

![Histogram of average time per epoch of epsilon-greedy algorithm.](image2.png)

**Fig. 2.** Histogram of average time per epoch of epsilon-greedy algorithm.

The change of epsilon value has little effect on the value of average time per epoch. The average time per epoch value is basically in the interval of 2-6 seconds, and the mathematical expectation is about 3.5 seconds. When the epsilon value is 0.5, the variance of the average time per epoch value is the largest, and the value is highly discrete. When the epsilon value gradually deviates from 0.5, the variance will gradually decrease. However, this change is insignificant. When the epsilon value approaches 0, the characteristics of the average time per epoch value become the worst.

![Histogram of epoch values of linear decrease epsilon-greedy algorithm.](image3.png)

**Fig. 3.** Histogram of epoch values of linear decrease epsilon-greedy algorithm. Value of epsilon indicates the initial epsilon from which the linear decrease begins.
From Fig. 3 we can see that when the initial value of epsilon is between 0.5 and 0.7, the mathematical expectation and variance are not much different. Only when the initial epsilon value is close to 1 does the mathematical expectation and variance of the epoch value increase significantly. However, from figure 3 it is clear that, as the initial value of epsilon increases, the distribution of epoch values gradually moves to the right. When the initial epsilon value is 0.4, the mathematical expectation and variance of the epoch value are the smallest.

![Figure 3. Histogram of average time per epoch of linear decrease epsilon-greedy algorithm. Value of epsilon indicates the initial epsilon from which the linear decrease begins.](image3)

The change of epsilon value has little effect on the value of average time per epoch. The average time per epoch value is basically in the interval of 1-7 seconds, and the mathematical expectation is about 3.5 seconds. When the epsilon value is 0.4, the characteristics of the average time per epoch value become very bad. The variance is the largest, and the value of average time per epoch is highly discrete. When the epsilon value gradually deviates from 0.4, the variance will gradually increase. Combining Fig. 4 and Fig. 3, we can conclude that changing the initial value of epsilon influences the number of epochs the agent takes to finish training, but the length of each epoch in time does not change much.

![Figure 4. Histogram of epoch values of parabolic decrease epsilon-greedy algorithm.](image4)

The situation is similar to the linear decrease, where the change of the initial epsilon value has little effect on the mathematical expectation and variance of the epoch value. When the initial epsilon value is 0.5, the mathematical expectation and variance of the epoch value are the smallest.
Fig. 6. Histogram of average time per epoch of parabolic decrease epsilon-greedy algorithm.

When the epsilon value is 0.8-0.9, the characteristics of the average time per epoch value are the worst. When the initial epsilon value is 0.7, the mathematical expectation and variance of the average time per epoch value are the smallest.

Fig. 7. Histogram of epoch values of parabolic decrease epsilon-greedy algorithm with softmax, initial epsilon equal to 0.5.

As the temperature value increases, both the mathematical expectation and the variance increase. When the temperature value is 1, the characteristics of the data are the best. However, when applying softmax with increasing temperature values, the mathematical expectation and the variance increase significantly.

Fig. 8. Histogram of average time per epoch of parabolic decrease epsilon-greedy algorithm with softmax, initial epsilon equal to 0.5.

The average time per epoch value has no obvious relationship with the value of softmax temperature, but the expectation and variance increase significantly compared to when softmax is not used.
6. DISCUSSION AND CONCLUSION
Experimental data is the same as expected, when epsilon is equal to 0.5, the agent has the shortest learning time and smaller epoch value. When the epsilon value is larger, the agent has a greater probability to do exploration, randomly choose a direction to proceed. This leads to many poorly performing moves, gets bad rewards, and the learning time becomes longer. Especially when the epsilon value is constant, if a lot of exploration is still carried out in the late stage, it will seriously impact the agent’s ability to learn and reinforce good action policy. As learning progresses, the agent obtains better policies through early exploration, so the agent should naturally exploit more in later stages. Moreover, agents with decreasing epsilon over time perform better overall than those having a constant epsilon values.

In machine learning, especially deep learning, softmax is a very common and important function, especially used in multi-class classification tasks. The softmax function allows actions with high rewards to be frequently taken, while actions with smaller rewards have an increased probability of being occasionally taken, thereby encouraging reasonable level of exploration. However, it can be seen from the data charted above that in the context of finding the shortest path in a maze, the application of softmax function is not an effective method. We hypothesize that, because the maze is too small, further encouragement of exploration by applying softmax does not help with speeding up route-finding but instead hurting the agent’s ability to exploit its existing knowledge at every step. How and when to use the softmax function in the maze problem is worth further exploration. For future work we will explore larger and more complex mazes and study the relationship between maze characteristics and best exploration-exploitation strategies.

ACKNOWLEDGMENT
Many thanks to my mentor Thomas Xue for his help and support. He led me to further study, and provided timely help to my research direction and revision of the thesis. Without his help, I wouldn't complete this series of work.

REFERENCES
[1] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013).
[2] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al.: Humanlevel control through deep reinforcement learning. Nature, 518(7540):529–533 (2015).
[3] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft and Peter Corke: Towards Vision-Based Deep Reinforcement Learning for Robotic Motion Control. arXiv:1511.03791 (2015).
[4] Y. Zhu et al.: Target-driven visual navigation in indoor scenes using deep reinforcement learning, 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, pp. 3357-3364 (2017).
[5] Sridhar Mahadevan, Jonathan Connell: Automatic programming of behavior-based robots using reinforcement learning, Artificial Intelligence, Volume 55, Issues 2–3, Pages 311-365, ISSN 0004-3702 (1992).
[6] http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/XX.pdf, last accessed 2020/1/21.
[7] https://www.samyzaf.com/ML/rl/qmaze.html, last accessed 2020/2/3.
[8] https://zhuanlan.zhihu.com/p/21262246, last accessed 2020/1/29.
[9] Christopher JCH Watkins and Peter Dayan. Q-learning. Machine learning, 8(3-4):279–292, 1992.
[10] G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems (1989).
[11] K. Hornik. Approximation capabilities of multilayer feedforward networks. Neural networks (1991).
[12] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. Neural networks (1989).

[13] Wei C., Ni F.: Tabu Temporal Difference Learning for Robot Path Planning in Uncertain Environments, Towards Autonomous Robotic Systems, TAROS 201 8. Lecture Notes in Computer Science, vol 10965. Springer, Cham (2018).

[14] XIA C., EL KAMEL A.: A Reinforcement Learning Method of Obstacle Avoidance for Industrial Mobile Vehicles in Unknown Environments Using Neural Network, Proceedings of the 21st International Conference on Industrial Engineering and Engineering Management (2015).

[15] Hongyu Guo: Generating Text with Deep Reinforcement Learning, arXiv:1510.09202 (2015).