International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 05 Issue: 06 | June - 2021 ISSN: 2582-3930

Reinforcement Learning in Game Playing

Sahil Khunger[1], Naman Tyagi[2], Vipul Chaudhary[3], Amit Pandey[4]

[1,2,3] Student, [4]Assistant Professor

Department of Information Technology,

[1,2,3,4] Dr. Akhilesh Das Gupta Institute of Technology and Management, New Delhi

***Abstract*- The single-player variant of Snake is a well-known and fashionable computer game that requires a player to navigate a line-based illustration of a snake through a two-dimensional piece of a grid, while avoiding collisions with the walls of the playing area and the body of the snake itself. The game keeps on getting more challenging as the snake navigates its way to the treats. The score is also increased accordingly. We have used Deep Reinforced Learning (DRL) to achieve AI in game playing. Deep Q-Learning is a particular type of Deep Reinforcement Learning. "Q-Function" is learnt by the network, which takes as input the current state of the environment and outputs a vector containing rewards for each possible action. The agent will then decide the action that maximizes the Q function. Based on this action, the environment is updated to another state by the game and a reward is assigned.**

I. INTRODUCTION

Since video games are challenging while easy to formalize, it has been a popular area of artificial intelligence research for a long time. For decades, game developers have attempted to create the agent with simulate intelligence, specifically, to build the AI player who can learn the game based on its gaming experience, rather than merely following one fixed strategy. The dynamic programming could solve the problem with relatively small number of states and simple underlying random structure, but not the complex one. One of the most fascinating technology of Machine Learning is Reinforcement Learning, which uses hit and trial to learn the most optimal action. Therefore, we chose the Snake game to explore the performance of reinforcement learning. The ambition behind this project is to develop the same game and embed a trained AI to maximise the performance.

Q-Learning is a type of a reinforcement learning technique used to train an agent to incorporate an optimal strategy for solving a task. In this an agent tries to learn the optimal strategy from its history of interaction with the surrounding environment, and we call the agent's knowledge base as "Q-Factor". However, it is not feasible to store every Q-factor separately, when the game needs a large number of action state pairs, so we could introduce Neural Network to store Q-factor of each state.

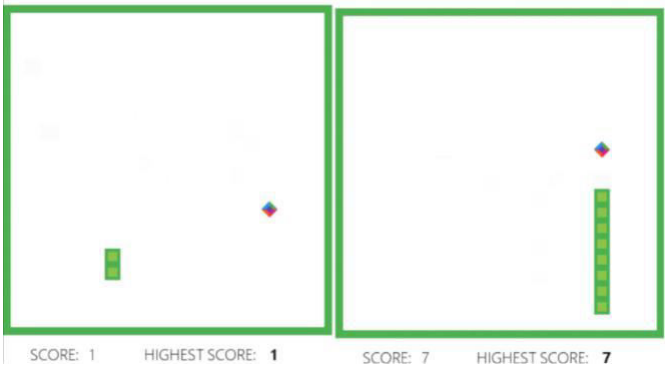

Fig 1. Game Snake

II. WORK REVIEW

There are abundant works about the artificial intelligence research for game agent. In [1] it has been shown by Miikkulainen that technologies such as soft computational intelligence (CI) such as neural networks have proven to be excellent in fields where the standard hard work intensive scripting and authoring models failed. At the same time, recent research shows that trends have moved from games in a compact form which used symbolic representations such as boards and cards game to more complex and immersive video games. The most successful backgammon playing program which was developed in the 90s called TD-gammon is a perfect example of reinforcement learning. In [2], a model-free reinforcement learning algorithm was used by TD-gammon which was similar to Q-learning, whilst using a multi-layer perceptron with one hidden layer gave the approximated value function. It was shown by Tsitsiklis and Van Roy [3] that combining model free reinforcement learning algorithms such as Q-learning with non-linear function approximators, or with off-policy learning could cause the Q-network to diverge. Eventually, linear function approximators became the work of interest as they had better convergence guarantees. With the redeveloping interest in combining deep learning with reinforcement learning, Sallans

and Hinton [4] illustrated that deep neural networks could be used to estimate the environment, while restricted Boltzmann machines could benefit the estimation of the value function.

III. TECHNICAL FOUNDATION

A. Game Environment

Snake Game is a classic 2-Dimension game, during which the player controls the snake to maximise the score by eating apples spawned arbitrarily at places on the board. An apple appears on the game screen at any time. The goal of the snake is to grow in length by eating apple after apple by avoiding collision, which is the key to its survival. In this project, we implemented the Snake Game in Python as the testbed of autonomous agents. We changed the settings of the Snake Game as recommended by [5] in their paper. Specifically, the size of the game map is set to 250×250 pixels and cut into 12×12 large grids. As recommended the starting length of the snake is 3, starting direction is set to the right, and the apples as well as the snake is randomly deployed whenever a game starts. The game score is set to 0 in the start which increases one by one as the snake reaches a target. Whenever, there is a collision of snake with anything it will automatically lead to the end of the game and the game score will be reset to 0 at the start of the new game. Whenever an apple is eaten and a new apple appears on the board, the target of the snake changes during the game trial upon reaching the previously determined target. Therefore, being able to localize new targets in an adaptive manner is crucial to the agents playing the Snake Game. The number of operations or controls available to the Snake in the Game is four, which are UP, DOWN, LEFT and RIGHT. The Game has a very complex and challenging reinforcement learning environment that has often been studied in the past many times. In this paper, we propose a refined Deep Q-learning Neural model with mainly three technical improvements and apply it to enable an artificial agent to play the Snake Game.

B. Technical Foundation

Deep Q-Network (DQN) was first presented by Mnih et al. [6] to play Atari 2600 video games in the Arcade Learning Environment (ALE) [7]. Deep Q-Network shows the ability to successfully learn challenging control policies directly from crude pixel inputs. Actually, Deep Q-Network is a convolutional neural network trained by a variation of the classical Q-learning algorithm [8], using the stochastic gradient descent method to change the weights. Deep Q-Network improves over traditional reinforcement learning methods as it uses Convolutional Neural Networks to approximate the Q-function, which provides a great way to estimate Q-values of possible actions directly from the most recently observed states (pixels).

In Q-learning, an agent tries to learn the best policy by remembering its history of interactions with the environment. By history we mean a sequence of state-action-rewards:

$$< s0, a0, r1, s1, a1 \cdots > \ldots (1)$$

Here *s0* is the state of the observed environment, *a0* is the action that the agent performs in state *s0*. After agent executes an action *a0* in state *s0*, it receives the reward *r0* from the environment and goes into the next state s1 or s′. After every game ends, the agent remembers and stores the experience in memory for subsequent sampling and training of Convolutional Neural Network. This is known as experience replay [9]. In addition, Deep Q-Network uses former network parameters to calculate the Q-values of the next state, which provides a optimal training target for Convolutional Neural Network.[10].

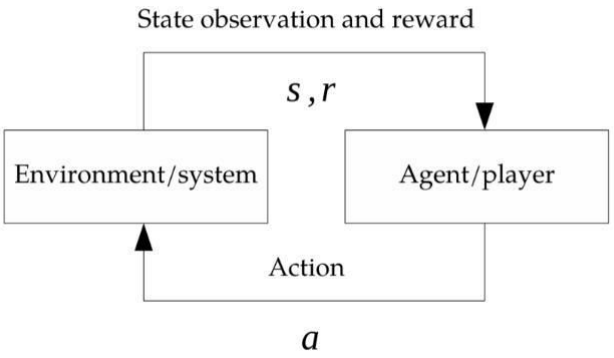

Fig 2. Dynamics of Markov Process with a reward

Actually, the process in the Snake game is indeed a Markov Decision Process, and the agent only needs to remember the last state information in which it was present. So we define the memory or the experience as a tuple of $< s, a, r, s_{next} >$. These memories or the experiences will be the data from which the agent will learn what to do. As the aim for the agent is to maximize the value of the total payoff $Q(s, a)$ in decision-theoretic planning, which in our case is the discounted reward. In Q-learning, which is off-policy, we use the Bellman equation as an alternative update

$$Q_{i+1}(s, a) = E_{s'\sim\epsilon}\{r + \gamma \max Q_i(s', a' \mid s, a)\} \ldots (2)$$

In the above equation, *s, s′* are the current and next state, *r* is the reward, γ is the discount factor and is the environment. And it can be shown that the Bellman equation will converge to the optimal Q-function. Since the distribution and transition probability is unknown to the agent, in our approach we use a neural network to approximate the value of the Q-function. This can be achieved by using the temporal difference formula to update each iteration's as

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \max Q(s', a') - Q(s, a)) \ldots (3)$$

An ϵ greedy approach is used here. The exploration probability is ϵ, it is changed from 0.75 to 0.9 with a constant density 0.03 during iterations. When it reaches 0.03, it remains constant. So it prompts the agent to explore a lot of possibilities in the starting of the game when it doesn't know how to play the game. This leads to a huge number of random actions which enables the agent to narrow down the optimal actions.

To understand however, how the agent takes choices, first it's necessary to understand what a Q-Table is. A Q-table is simply a table that matches the state of the agent with the potential actions that the agent can adopt. The values in the table are the action's probability of success, the highest value action is selected and that action is performed.

State	Right	Left	Up	Down
1	0	0.31	0.12	0.87
2	0.98	-0.12	0.01	0.14
3	1	0.10	0.12	0.31
4	0.19	0.14	0.87	-0.12

Table 1.

As we can see in the example table 1, first we will want to move DOWN and then move RIGHT when we are in State 2, and then we would want to go DOWN if we are in State 3 and so on. So, like this we can move through the environment increasing the reward. This table is the cheat sheet or the map for the agent: it determines what actions should be performed in every state in order to optimize the equation and maximize the expected reward. But there is more to that, the policy is a table, hence it can only handle a limited state space, In other words what would happen if we would have 10000 states, 10000 actions at each state, this number can be even more huge. So, this is only optimal for a smaller number of states and actions.

So, Deep Q-Learning increases the muscle power of Q-Learning, as the policy is not a matrix or a table but a DNN and the Q-values are only updated according to the Bellman equation.

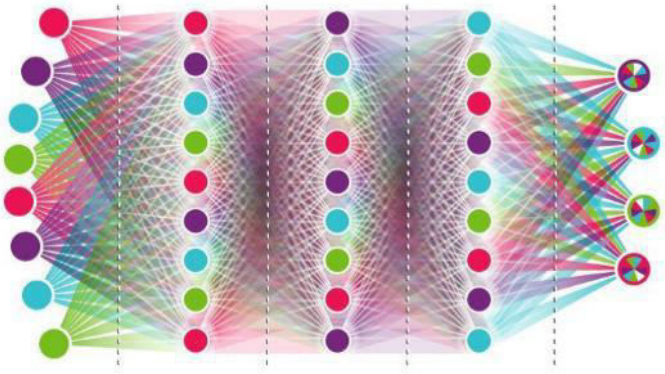

Fig 3. Deep Neural Network

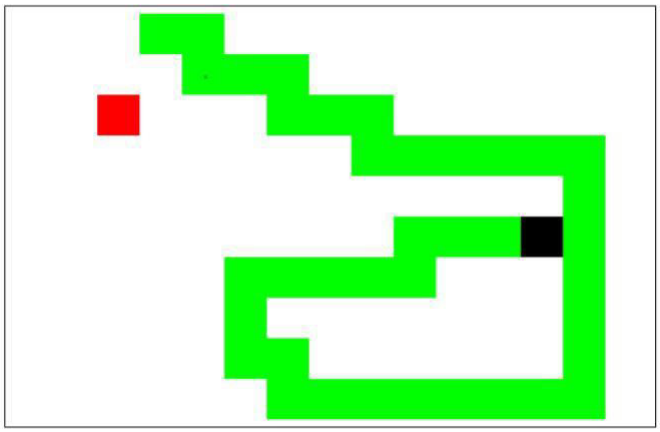

Fig 4. The snake at a score of 36 points. In this situation it is vital for the agent to understand that turning upwards is the move to make—if it turns downwards, it will have no way of surviving to eat another apple.

IV. RESULTS

A. Tuning Parameters

γ, which is the discount factor was set to be 0.95, relatively decreasing learning rate starting from $\alpha = 0.1$ and the rewards were set as shown in Table 2. We want the agent to control the snake to go to the food quickly, and the last column in table 2 is a punishment for taking for one movement, which encourages the agent to traverse shorter walk to the food. This negative reward acts as similar function as the discount factor γ. We performed trial and error on different combinations of reward for different cases to get current combination of reward values as one optimal combination among all the trials.

Case	Eat Food	Hit Wall	Hit Snake	Else
Reward	+500	-100	-100	-10

Table 2.

At the end of the execution, our AI scores 50 points on average in a 25*25 game board (each treat eaten rewards one point). The record is 86 points. To visualize the learning process and how effective the approach of Deep Reinforcement Learning is, a graph has been plotted below with scores vs. number of games played. It is evident from the plot below that during the first 50 games the AI scores imperfectly: less than 11 points on an average. This was anticipated as during this phase, random actions are being undertaken by the agent in order to traverse the board and the different states, actions and rewards are being stores in its memory. It is in the last 50 games that our agent is making educated decisions based on the weights of our neural network.

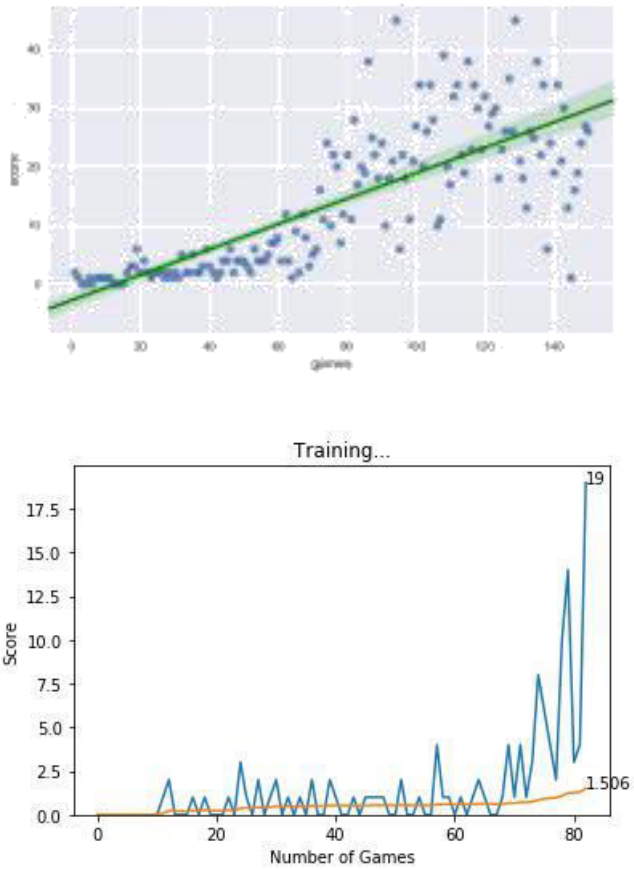

Fig 5: Plots between score and number of games.

V. FUTURE WORK

A. While exploring the Q-learning algorithm and its stability, it encountered that in a few training cases the performance of Q-learning algorithm was not very stable even with a reducing exploration probability. So, exploring other principles and methods will come handy to further improve the stability of the algorithm. A simple approach to solve this problem might be to add various tuning parameters to better the probability of convergence.

B. Study the other state space approximation methods. Other approximation of the state space could be explored for better performance. Currently, we use a quadrant view state mapping technique. By using this method, the agent will tend to hit itself repeatedly. Hence, a more rigorous state mapping technique should be developed. Such reduction mapping must not only approximate the relative position of the head and the food, but also obtain a precise idea of the position of the body. The size of the state space must be kept in mind to avoid increasing it dramatically.

C. SARSA Model

In order to further improve the learning rate of the Snake agent, Expected SARSA could be used to provide a theoretical and empirical analysis of Expected SARSA, and found it to have significant advantages over more commonly used methods like SARSA and Q-learning.

VI. CONCLUSIONS

In this paper, we portray how to refine a widely adapted DQN model and apply it to enable an autonomous agent to learn how to play Snake Game from scratch. Specifically, we propose a carefully designed reward mechanism to solve the sparse and delayed reward issue, employ the training gap strategy to exclude improper training experiences, and a dual experience replay method is executed to further improve the training efficacy. Experimental values show that our refined DQN model outperforms the baseline model. It is more encouraging to find out the performance of our agent surpasses human-level performance.

Going forwards, we shall harvest more computing resources to find out the convergence requirement in this Snake Game and conduct more benchmarking experiments. Additionally, we shall apply our refined DQN model to other continuous reallocated targets (such as the re-spawned treats) along with their applications and gradually increasing restrictions (such as the increasing length of the snake).

REFERENCES

[1]. Risto Miikkulainen, Bobby Bryant, Ryan Cornelius, Igor Karpov, Kenneth Stanley, and Chern Han Yong. *Computational Intelligence in Games.* URL: ftp://ftp.cs.utexas.edu/pub/neuralnets/papers/miikkulainen.wcci06.pdf

[2]. Gerald Tesauro. *Temporal difference learning and td-gammon*. Communications of the ACM, 38(3):5868, 1995.

[3]. John N Tsitsiklis and Benjamin Van Roy. *analysis of temporal difference learning with function approximation.* Automatic Control, IEEE Transactions on, 42(5):674690, 1997.

[4]. Brian Sallans and Geoffrey E. Hinton. *Reinforcement learning with factored states and actions.* Journal of Machine Learning Research, 5:10631088, 2004.

[5]. A. Punyawee, C. Panumate, and H. Iida, "Finding comfortable settings of Snake Game using game refinement measurement," Advances in Computer Science and Ubiquitous Computing, pp. 66–73, 2017.

[6]. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves,I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," ArXiv e-prints,2013.

[7]. M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling,"The arcade learning environment: An evaluation plat-form for general agents," Journal of Artificial IntelligenceResearch, vol. 47, pp. 253–279, 2013

[8]. P. D. Christopher J. C. H. Watkins, "Q-learning," Ma-chine Learning, vol. 8, no. 3-4, pp. 279–292, 1992.

[9]. L.-J. Lin, "Reinforcement learning for robots using neural networks," Ph.D. dissertation, Pittsburgh, PA, USA,1992, UMI Order No. GAX93-22750.

[10]. Van Seijen, H., van Hasselt, H., Whiteson, S. and Wiering, M. (2009). *A theoretical and empirical analysis of Expected Sarsa, 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning pp. 177184.*
URL: http://goo.gl/Oo1lu