First International Conference on Computer Engineering
International Journal of Scientific Research in Science and Technology
Print ISSN: 2395-6011 | Online ISSN: 2395-602X (www.ijsrst.com)
Volume 5 Issue 8, November-December-2020

Training an AI agent to play a Snake Game via Deep Reinforcement Learning

Reetej Chindarkar[1], Kartik Kaushik[1], Rutuja Vetal[1], Ronak Thusoo[1], Prof. Pallavi Shimpi[2]
[1]Department of Computer Engineering, Dr. D.Y.Patil School of Engineering, Lohegaon,Maharashtra India
[2]Assistant Professor, Department of Computer Engineering, Dr. D.Y.Patil School of Engineering, Lohegaon, Maharashtra India

ABSTRACT

Deep Reinforcement Learning (DRL) has become a normally adopted methodology to alter the agents to be told complex management policies in varied video games, after Deep-Mind used this technique to play Atari games. In this paper, we will develop a Deep Reinforcement Learning Model along with Deep Q-Learning Algorithm that will enable our autonomous agent to play the classical snake game. Specifically, we will employ a Deep Neural Network (DNN) trained with a variant of Q-Learning. No rules about the game are mentioned, and initially the agent is provided with no information on what it needs to do. The goal for the system is to figure out the rules and elaborate a method to maximize the score or reward.

Keywords: Deep reinforcement learning, Snake Game, Autonomous agent, Deep Learning, Experience replay

I. INTRODUCTION

One of the most well-known models of Reinforcement Learning used for playing games is called TD-gammon [1] which was developed decades ago. It was used to play the Backgammon game and it surpassed human-level performance. However, this technique shows very little generalization to different games and failed to attract wide attention. As a recent breakthrough in deep learning, DeepMind creatively combined deep learning with reinforcement learning and came up with the distinguished deep Q-learning network (DQN) model [2]. DQN outperforms all the previous approaches on six games and surpasses human-level performance on 3 games. This breakthrough lit up researchers' passion and lots of similar researches (e.g., [3, 4]) presently emerged.

However, DQN might not be straightforwardly applied to any or all scenarios as a result of its naïve reward mechanism solely produces thin and delayed rewards which will cause ineffective learning of correct policies [5]. In most reinforcement learning problems, to decrease the correlation of sampled experiences when training the network, a technique named Experience Replay is usually adopted [6]. However, this method samples previous experiences haphazardly while not considering their quality. To solve this downside, an improved approach was proposed by Schaul et al. [7] namely "Prioritized Experience Replay".

In Reinforcement Learning, we have two components: The Environment and The Agent. Every time the agent performs an action, the environment provides a reward to the agent, which might be positive or negative that depends on how smart the action was from that of the specified state. The goal of the agent is to find what actions will maximize the reward, according to the possible state. States are the observations that the agent receives at every iteration from the environment. A state may be its position, its speed, or whatever array of variables describes the environment. The Reinforcement Learning notation that is used for the decision-making method that the agent adopts is termed as policy. On a theoretical level, a policy could be a mapping from the state space (the space of all the attainable observations that the agent will receive) into the action space (the space of all the actions the agent will take, say UP, DOWN, LEFT and RIGHT). The optimal agent is in a position to generalize over the entire state space to consistently predict the best possible action, even for those situations that the agent has never seen before.

II. GAME ENVIRONMENT AND TECHNICAL FOUNDATION

This section describes the proposed Snake Game Environment and the technical foundation required of a typical Deep Reinforcement Learning Model.

A. *Game Environment*

Snake Game is a classical digitized game, throughout which the player will control the snake to maximize the score by eating the apples that are spawned at random places. Only one apple appears within the game screen at any time. Moreover, because the snake can grow one grid long by eating an apple, avoiding collision is vital to its survival.

In this work, we will implement the Snake Game in Python as the testbed of autonomous agents. The Snake and the apple will be randomly deployed when the game starts. The game score will be initialized at 0 and will increase by 1 as the snake reaches a target. Moreover, after the collision of the snake the game will end and the game score will be reset to 0 at the start of the new game. After the previously given apple is eaten a new apple respawns and the target of the snake changes during the game when it reaches a previously determined target.

Therefore, having the ability to localize new targets in an adaptive manner is crucial for the agent playing the Snake Game. The number of control signals within the Snake Game are four, i.e. UP, DOWN, LEFT and RIGHT. At each step in time, the snake will move one grid forward on its course direction, unless the control signal it receives is in an orthogonal direction.

B. *Technical Foundation*

Deep Q-Network (DQN) was firstly presented by Mnih et al. [2] which was used to play Atari 2600 video games using the Arcade Learning Environment (ALE) [8]. DQN demonstrates its ability to successfully learn complicated control policies directly from raw pixel inputs. DQN is a convolutional neural network (CNN) trained by a variation of the classical Q-learning algorithm [9]. DQN algorithm advances traditional Reinforcement Learning techniques because it utilizes CNN to estimate the Q-function that provides a mechanism to approximate Q-values of feasible actions directly from the most recently observed states (pixels). To train the neural network and to keep the iterative evaluations stable, DQN uses mini-batches of experience. Each experience is manifested as a four-tuple (s; a; r; s'), where s denotes the state of the observed environment, a denotes the action that agent performs in state s. After the agent executes action (a) in state (s), it receives the reward r from the environment and goes into the next state (s'). Along game play, agent stores the experience in memory for future sampling and training of CNN, which is known as experience replay [6].

Additionally, DQN uses former network parameters to check the Q-values of the next state, which provides a stable training target for CNN [10]. To understand how the agent makes decisions, it’s necessary to understand what a Q-Table is. A Q-table is a matrix that correlates the state of the agent with each and every potential action that the agent will take-on. The values in the table are the action’s chances of success, based on the rewards it got throughout the training. The values within the Q-Table represent the expected reward of taking action a from a state s. This table is the policy of the agent that we mentioned before: it determines what actions ought to be taken from every state so as to maximize the expected reward. The problem with this can be that the policy is a table hence it can only handle a finite state space. That is to say, we cannot have an infinitely large table with infinite states. This can become a problem for those situations where we are expecting the number of possible states to be very large. Deep Q-Learning increases the capability of Q-Learning, since the policy isn't a table but a Deep Neural Network. The Q-values are updated in accordance with the Bellman equation [11].

$$\underbrace{NewQ(s,a)}_{\text{New Q-Value}} = \underbrace{Q(s,a)}_{\text{Current Q-Value}} + \underset{\text{Learning rate}}{\alpha}[\underbrace{R(s,a)}_{\text{Reward}} + \underset{\text{Discount rate}}{\gamma} \underbrace{max\, Q'(s',a')}_{\text{Maximum predicted reward, given new state and all possible actions}} - Q(s,$$

Fig. 1. Bellman equation

1) *State:* A state can be the representation of a situation in which the agent will find itself. The state will also represent the input the Neural network will take.
   In our case, the state can be an array which will contain 11 Boolean variables, which will take into account:
   - ❖ If the snake is in danger from its immediate proximity i.e. Left, Right and Straight.
   - ❖ If the snake moves in the direction Up, Down, Left or Right.
   - ❖ If the food is Left, Right, Above or Below.

2) *Loss:* The Deep neural network optimizes the output (action) to a specific input (state) trying to maximize the expected reward. The Loss function gives the value that expresses how accurate the prediction is compared to the truth. The job of a neural network is to reduce the loss and to abridge the difference between the real target and the predicted one.

3) *Reward:* The AI tries to maximize the expected reward in any given circumstance. A positive reward is only given to the agent once it eats the food target (+10). If the snake hits a wall or hits itself, negative reward (-10) is given. In addition, there can be a positive reward for each and every step that the snake takes without dying. In that case, the agent might just decide to run in a loop, since it would get positive rewards for each step that it takes. Reinforcement Learning agents sometimes present us with flaws in our strategy that we did not anticipate, as such outsmarting us in that way.

General Algorithm:

- ❖ The Q-value is randomly initialized at the beginning of the game.
- ❖ The system gets the present state ‘s’ (the observation).
- ❖ Based on the state 's', it executes an action, randomly or based on its neural network. During the primary phase of the training, the system usually adopts random actions to maximize exploration. Henceforth, the system depends more and more on its neural network.
- ❖ When the AI chooses and performs an action, the environment gives it a reward. After that the
- ❖ agent reaches a new state s' and it updates its Q-value as per the Bellman equation. Also, for each and every move that the agent makes, it stores the original state (s), the action (a), the state reached after performing that action (s'), the reward (r) obtained and whether the game is terminated or not. This data is then used a sample to train the neural network. This process is known as Replay Memory.
- ❖ These last two operations can be recursive in nature until an explicit condition is met (example: the game ends).

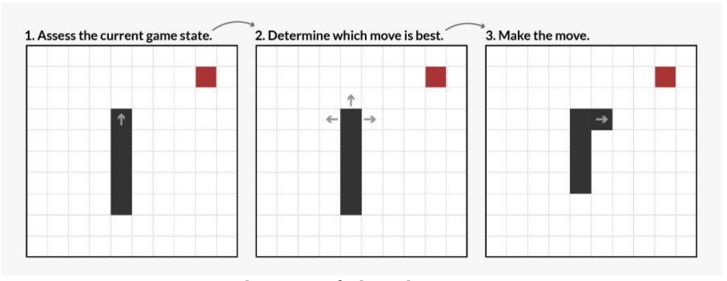
1. Assess the current game state.
2. Determine which move is best.
3. Make the move.

Fig. 2. Sample Image of Algorithm

III. LITERATURE SURVEY

Refer Table I for the Literature Survey

IV. CONCLUSION

In this paper, we have discussed how Reinforcement Learning can be used to train an agent that will learn how to play the classical snake game while being trained using DQN (Deep Q-Network). Specifically, we will propose a rigorously designed reward mechanism to resolve the sparse and delayed reward issue. Also, we will employ a training gap strategy to eliminate improper training experiences and implement a dual experience replay method to further improve the training efficacy. Using the State-Action-Reward-State-Action (SARSA) algorithm the proficiency of the game will increase due to the quick task performance. It will provide relatively better results as when compared to the existing techniques.

TABLE I

Sr. No.	Paper Name	Author	Methods Proposed	Advantages	Limitations
1.	Autonomous Agents in Snake Game via Deep Reinforcement Learning	Zhepei Wei, Di Wang, Ming Zhang, Ah-Hwee Tan, Chunyan Miao, You Zhou.	Deep Q-learning	1. Using Deep Q-learning, issue of reward mechanism to solve the sparse and delayed reward issue, employ the training gap strategy to exclude improper training experiences, and implement a dual experience replay method to further improve the training efficacy has solved.	1. The first several games, both the game scores and the survival time are expected to be low because the agent only chooses random actions.
2.	Exploration of Reinforcement Learning to Play Snake Game	Ali Jaber Almalki, Pawel Wocjan	Deep Q-Learning, State-Action-Reward-State-Action (SARSA) Algorithm and Reinforcement Learning.	1. Deep Q-Learning helps to increase the efficiency of the Q-Learning and provide a steady flow to control the snake's position in the game.	1. The performance of the SARSA algorithm depends upon the user's instruction and state of the changing in the action occurred accordingly.
3.	Exploration of Reinforcement Learning to Snake	Bowei Ma, Meng Tang, Jun Zhang	Q-Learning, State-Action-Reward-State-Action (SARSA) Algorithm and Reinforcement Learning.	1. SARSA is the on-policy algorithm that helps to make decisions effectively. 2. SARSA algorithm supports agent to effectively interact with the environment.	1. Even with a decreasing exploration probability, the performance of Q-learning algorithm is not very stable. 2. Other approximation of the state space could be explored for better performance. 3. To further improve the learning rate of the Snake agent, Expected SARSA could be used.
4.	Solving the Classic Snake Game Using AI	Shubham Sharma, Saurabh Mishra, Nachiket Deodhar, Akshay Katageri, Parth Sagar	Breadth-first search, Almighty move.	1. AI Bot is trained to achieve the maximum score possible in the minimum number of steps. 2. This can also be used in other games of bigger	1. Almighty move is not used from the first iteration as the number of steps required increases to a large extent, thus increasing the time complexity. 2. In BFS, the

				size, which are the part of "Electronic Sport" to train the players.	limitation is that, it guarantees the snake till length 4, because for length greater than 4 the snake can bite itself.
5.	Automated Snake Game Solvers via AI Search Algorithms	Shu Kong, Joan Aguilar Mayans	A*Search, A* Searching with Forward, Random move.	1.The informed search algorithms can also show a reasonable reliability and the highest efficiency at the beginning of the run this property disappear at the end game. 2. In contrary almighty move is a slow algorithm at the beginning but has guaranteed a max score at end. 3. Combination of the different algorithm can achieve perfect reliability.	1. A* search algorithm is dependent on the cost of the path to reach the current fruit from the starting, and the heuristic distance from the head of the snake to the next fruit. 2. It only checks the path till the fruit is reached, with the knowledge of the previous path cost. 3. Random Move can easily reach a dead end since it blindly moves forward.
6.	Snake Played by a Deep Reinforcement Learning Agent	K. Hornik, M. Stinchcombe, H. White. Mnih et al.	Deep Reinforcement Learning, Reinforcement Learning	1. By using pixels and Convolutional Neural Networks in the state space it is possible for the agent to 'see' the whole game, instead of just nearby obstacles. 2. It can learn to recognize the places it should go to avoid enclosing and get the maximum score.	1. The agent learns to avoid obstacles directly surrounding the snake's head, but it can't see the whole game. So, the agent will enclose itself and die, especially when the snake is longer.
7.	An adaptive Strategy via Reinforcement Learning for Prisoner's Dilemma Game	Lei Xue, Changyin Sun, Donald Wunsch, Yingjiang Zhou, Fang Yu.	Reinforcement Learning, Temporal difference learning, Complex Learning.	1. The agent with adaptive strategy can make decisions under a consideration of the long-term reward. 2. The adaptive agents were able to cooperate with their opponents without losing competitiveness.	1. It is very difficult for the agents to achieve mutual cooperation by the characteristic of the scale-free network. 2. The condition of the hubs is significant for the scale free network.
8.	Deep Reinforcement	Ruben Rodriguez	DQN, Prioritized Duelling DQN and	1. It serves two advantages: it demonstrates the	1. DQNs and A2C perform badly on games with a binary

	Learning for General Video Game AI	Torrado, Julian Togelius, Jialin Liu, Diego Perez-Liebana	Advance Actor-Critic (A2C)	strengths and weaknesses of the current generation of reinforcement learning algorithms. 2. It allows results achieved on GVGAI to be compared to other existing environments.	score (win or lose, no intermediate rewards). 2. High dependency of the initial conditions which suggests that running multiple times is necessary for accurately benchmarking DQN algorithms.

V. REFERENCES

[1]. G. Tesauro, “Temporal difference learning and TDgammon,” Communications of the ACM, vol. 38, no. 3, pp. 58–68, 1995.

[2]. 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.

[3]. E. A. O. Diallo, A. Sugiyama, and T. Sugawara, “Learning to coordinate with deep reinforcement learning in doubles pong game,” in Proceedings of IEEE International Conference on Machine Learning and Applications (ICMLA), 2017, pp. 14–19.

[4]. S. Yoon and K. J. Kim, “Deep Q networks for visual fighting game AI,” in Proceedings of IEEE Conference on Computational Intelligence and Games (CIG), 2017, pp. 306–308.

[5]. M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel,and W. Zaremba, “Hindsight experience replay,” in Proceedings of Annual Conference on Neural Information Processing Systems, 2017, pp. 5055–5065.

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

[7]. T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” Computing Research Repository, vol. abs/1511.05952, 2015.

[8]. M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” Journal of Artificial Intelligence Research, vol. 47, pp. 253–279, 2013.

[9]. P. D. Christopher J. C. H. Watkins, “Q-learning,” Machine Learning, vol. 8, no. 3-4, pp. 279–292, 1992.

[10]. M. Roderick, J. MacGlashan, and S. Tellex, “Implementing the deep q-network,” ArXiv e-prints, 2017.

[11]. Ali Jaber Almalki, Pawel Wocjan, “Exploration of Reinforcement Learning to Play Snake Game” in Proceedings of International Conference on Computational Science and Computational Intelligence (CSCI), 2019, pp. 377-381.