Snake Game

Ayush Tankha , Jatin Singh and Nishant Dave

Ms Data Science and Business Analytics – ESSEC x CentraleSupelec

Abstract. The aim of this project is to develop a self-playing RL agent for the snake game, we will try to observe the learning rate of the agent and its limitations. The agent will be self-learn optimal move strategies for maximizing the score. Our investigation centers around the application of Q-learning algorithm to assess it’s effectiveness in guiding the AI towards mastering game strategies.

Keywords: Snake Game, Reinforcement Learning, Q-Learning

1 Introduction and Motivation

1.1 An introduction to Snake Game

Once a mainstay of early computer gaming, the Snake game has evolved beyond its simple gameplay to become a burgeoning topic of research in reinforcement learning (RL) and artificial intelligence (AI). This game invites players to steer a constantly expanding snake in the direction of food while avoiding collisions with the game's limits and with itself. The game is a great testbed for reinforcement learning algorithms because, despite its apparent simplicity, it encompasses a rich strategic depth.

1.2 Motivation

In this research, we explore how these RL tactics are applied to the game of Snake, aiming to provide insights into how an AI agent might independently design effective strategies. Our study of the agent's learning curve and strategic improvement contributes to the current conversation about AI's capacity to overcome difficult problems and lays the groundwork for its wider use in robotics and autonomous systems. With our research, we hope to highlight the useful applications of reinforcement learning both inside and outside of the gaming industry, showcasing the strategic sophistication and flexibility AI may achieve in constrained but complex environments.

2 Introduction and Motivation

2.1 Environment

In the Snake game, the snake's location, the food's location, and the existence of obstacles (the snake's body and the game's edges) determine the environment. To help the agent learn, we created a reward system that is essential to teaching it what acts are desirable.

2.2 State Space

The state space includes information about the immediate danger for the snake, the current direction of movement, and the relative position of food. For this game, the state space is an array with 11 boolean values representing different conditions the snake can encounter within the game at any given moment. Here's what each of the 11 states: danger straight, danger right, danger left, direction left, direction right,direction up direction down, food left, food right, food up and food down.

2.3 Action Space

The action space is a vector of size 3, each corresponding to a possible decision for the snake. [1,0,0] means to turn left, [0,1,0] means to go straight and [0,0,1] means to turn right with respect to the snakes initial direction.

2.4 Policy Function

The policy in terms of action selection is seen in the *get_action* method of the Agent class, which uses the model's output (the Q-values) to select an action either by exploiting the best-known action (choosing the action with the highest Q-value) or by exploring (choosing a random action).

2.5 Value Function

The value function is represented by the Q-value function that the neural network (*Linear_QNet*) is approximating. The Q-value function is a state-action value function that gives the expected utility of taking a given action in a given state and following the optimal policy thereafter.

The *train_step* method in the *QTrainer* class updates the neural network parameters to minimize the difference between the predicted Q-values and the target Q-values.

3 Agent Description

The agent consists of several components that allow it to perceive its environment (the Snake game), decide on the best course of action, and learn from the outcomes of its actions:

Neural Network (Linear_QNet): The agent has a neural network that serves as a function approximator. This network takes the game's current state as input (an 11-dimensional vector) and outputs a prediction of the Q-values for each possible action. The neural network has one hidden layer, which allows it to learn complex strategies.

Memory (deque): The agent possesses a memory buffer that stores its experiences, which include the current state, the action taken, the reward received, the next state, and whether the game ended. This memory is used to learn from past actions, particularly during training sessions.

Training Modules (QTrainer): The agent has a training module that uses the stored experiences to update the weights of the neural network. This training happens in two forms: short-term (using the most recent experience) and long-term (using a batch of experiences sampled from memory).

4 Agent Functionality

The agent's functionality revolves around the core loop of observing the state, taking an action, receiving a reward, and learning:

State Observation: Using the *get_state* function, the agent observes the current state of the game, which includes information on the immediate danger of collision, the current direction of movement, and the location of the food relative to the snake.

Action Decision: When it is time to decide on an action, the agent employs an ε-greedy policy (embedded within the *get_action* function). This policy either chooses a random action (with probability ε, encouraging exploration) or the best action according to the neural network's predictions (encouraging exploitation).

Learning: After taking an action and observing the outcome, the agent uses the *train_short_memory* function to immediately learn from the experience. Additionally, at the end of each game, the agent uses *train_long_memory* to learn from a wider range of past experiences.

Performance Improvement: The goal of the agent is to maximize the score in the Snake game, which is done by eating food without crashing. As the agent plays more games and learns, it is expected to improve its decision-making, resulting in higher scores.

Model Saving: When the agent achieves a new high score, it saves its neural network's state to a file. This allows the learned behavior to be retained and potentially reused or further improved upon later.

5 Training

Both short-term and long-term memory components are trained for. After every action, the agent's short-term memory is refreshed, enabling instantaneous learning from recent encounters. At the conclusion of every game, long-term memory training takes place, using a batch of experiences to generalise learning across different states and circumstances. With this method, the agent can improve its decision-making and strategic planning over time by gradually fine-tuning its policy. In order to maintain a steady convergence towards optimal strategies, the learning rate (LR = 0.001) and discount rate (gamma = 0.9) parameters were carefully selected to balance the relevance of current benefits vs future rewards.

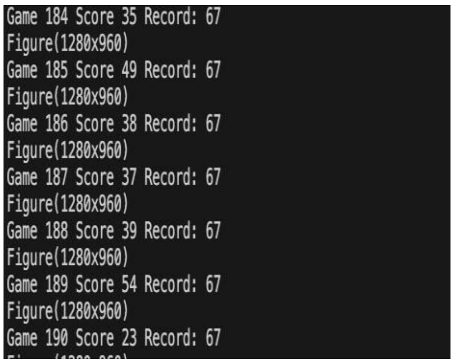
6 Results

We ran 3 test cases to compare our results and learning curve for our Q-Learning agent. For each test we looked across 190 games to not comprisable difference.

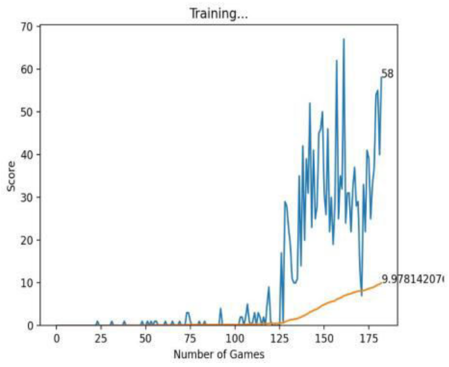
6.1 Single Layer Architecture (Baseline)

For this we used the following architecture –

Input Layer - 11 neurons (state representation)
Hidden Layer - 256 neurons, ReLU activation
Output Layer - 3 neurons (Q-values for each action)



```
Game 184 Score 35 Record: 67
Figure(1280x960)
Game 185 Score 49 Record: 67
Figure(1280x960)
Game 186 Score 38 Record: 67
Figure(1280x960)
Game 187 Score 37 Record: 67
Figure(1280x960)
Game 188 Score 39 Record: 67
Figure(1280x960)
Game 189 Score 54 Record: 67
Figure(1280x960)
Game 190 Score 23 Record: 67
```

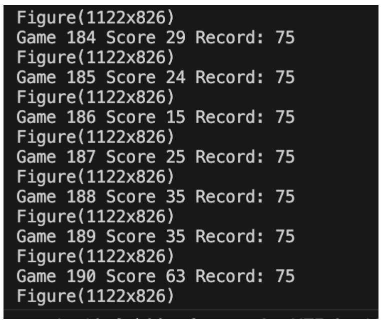


Here we noticed a breakthrough in learning around 130 games.

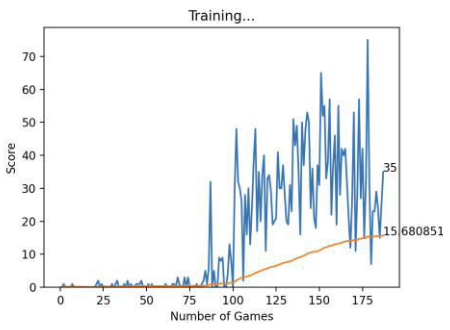
6.2 Multi-Layer Architecture (3 hidden layers)

Input Layer - 11 neurons (state representation)
Hidden Layer - 256 neurons, ReLU activation
Hidden Layer - 256 neurons, ReLU activation

Hidden Layer - 256 neurons, ReLU activation
Output Layer - 3 neurons (Q-values for each action)



```
Figure(1122x826)
Game 184 Score 29 Record: 75
Figure(1122x826)
Game 185 Score 24 Record: 75
Figure(1122x826)
Game 186 Score 15 Record: 75
Figure(1122x826)
Game 187 Score 25 Record: 75
Figure(1122x826)
Game 188 Score 35 Record: 75
Figure(1122x826)
Game 189 Score 35 Record: 75
Figure(1122x826)
Game 190 Score 63 Record: 75
Figure(1122x826)
```



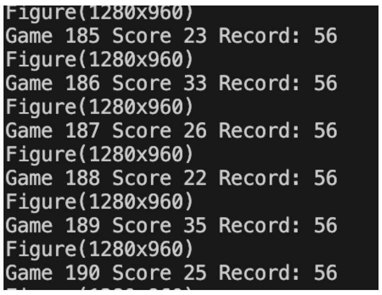
In this case we notice an improvement in highest score as well as average score. The breakthrough is also achieved much earlier (around 100 games)

6.3 Multi-Layer Architecture + Epsilon tuning

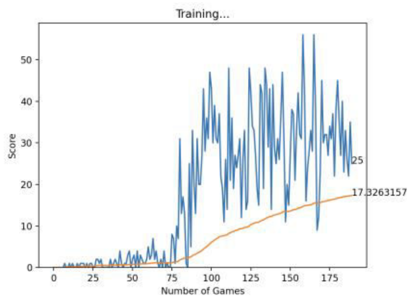
In earlier cases our epsilon didn't have a lower bound and was linearly decreasing with an increase in number of games. (80 – epsilon value).

New update: - *self.epsilon = max(80 - self.n_games, minimum_epsilon_value)*

Here *minimum_epsilon_value* is set at 0.01 so that when it takes a negative value, we still have a lower bound of 0.01 which keeps the probability for the snake to have exploration.



```
Figure(1280x960)
Game 185 Score 23 Record: 56
Figure(1280x960)
Game 186 Score 33 Record: 56
Figure(1280x960)
Game 187 Score 26 Record: 56
Figure(1280x960)
Game 188 Score 22 Record: 56
Figure(1280x960)
Game 189 Score 35 Record: 56
Figure(1280x960)
Game 190 Score 25 Record: 56
```



Even though we don't have a better high score from the previous test case, but we have significantly improved the average score which implies that a room for randomness allows the snake to have an option for exploration even in later training stage. Here we can also notice that the learning breakthrough comes much earlies , that is at 90 games.

7 Conclusion

The test change emphasized the benefits of using Q-learning in the context of the Snake game, which is a dynamic and unpredictable environment. Q-learning's dynamic exploration-exploitation balance and adaptability demonstrated to be very appropriate for these kinds of situations.

However, our research also revealed areas that still need work. The AI occasionally displayed less-than-ideal behavior patterns, including times where the snake would go in circles infinitely, indicating that the learning techniques may be improved. Rather than viewing these instances as negatives, we saw them as chances to expand on our knowledge of the possible uses and powers of reinforcement learning. This realisation has motivated us to keep researching the best ways to adapt RL methods for intricate, dynamic situations.

References

1. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.
2. Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. Nature.
3. Silver, D., et al. (2016). Mastering the game of Go with deep neural networks and tree search. Nature.
4. Z. Wei, D. Wang, M. Zhang, A. -H. Tan, C. Miao and Y. Zhou, "Autonomous Agents in Snake Game via Deep Reinforcement Learning," 2018 IEEE International Conference on Agents (ICA)
5. A. Sebastianelli, M. Tipaldi, S. L. Ullo and L. Glielmo, "A Deep Q-Learning based approach applied to the Snake game," 2021 29th Mediterranean Conference on Control and Automation (MED), PUGLIA, Italy, 2021
6. A. J. Almalki and P. Wocjan, "Exploration of Reinforcement Learning to Play Snake Game," 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2019
7. R. Cai and C. Zhang. Train a snake with reinforcement learning algorithms. 2020