CHI LI
HENRY NGUYEN
HUIDONG WANG

FINAL PROJECT WRITE-UP PLAYING SNAKE THROUGH REINFORCEMENT LEARNING

A CLASSIFICATION ANALYSIS OF A DEEP-Q NETWORK - FOR PREDICTING THE NEXT BEST MOVE.

Portland State University
CS445/545-002 Machine Learning
Fall 2021

Contributions

While we all communicated and dabbled over each part of the project, there were areas of the project in which we individually spent most of our time and focus:

- Huidong led the development of the deep reinforcement learning agent.
- Henry led the development of the control method (random inputs at time intervals).
- Chi led the development of the human-playable Snake game.

Introduction

In this project, we applied a deep reinforcement learning algorithm so that an agent learns to play the game "Snake", as best as it can. "Snake" consists of a board where the snake starts as a single block unit and tries to collect pellets that appear somewhere in the board. For each time a pellet is collected by the snake, the length of the snake is increased by appending another block unit to its "tail" end. The game typically plays indefinitely until the snake's "head" collides with the borders of the board or with its own body, concluding the game. The objective of the game is to collect as many pellets as possible in a single life (or to maximize the snake body length). We built the snake game in Python with the turtle library.

Deep-Q Network Implementation

The Deep-Q Network consists of two main components: a deep neural network as well as a Q-Reinforcement Learning algorithm.

The Deep Neural Network component is made up of an input layer, 3 hidden layers, and an output layer. The neural network takes in a set of states as the input before using the hidden layers, each layer consisting of 128 nodes, to calculate an output, which results in a single value as the output which represents the action that the snake should perform. For the process of updating the weights of the neural network to help the neural network make better action selections in the future, the resulting reward from the action performed is used (rewards are further explained below).

The Reinforcement Learning component of the project takes on a Q-Learning technique in which a Q-Table is used to track the value of each state action pair, which is a mapping between a position on the board and the action that the snake agent can perform. The snake agent may choose from one of 4 actions: move upwards, move leftwards, move downwards, and move rightwards. The Q-Learning algorithm performs such that a set number of episodes, or epochs, are performed where one episode lasts from the beginning of a new snake game to until the snake dies or a specified maximum number of moves (we set the max moves to 10,000 moves). For each move in an episode, the agent decides on an action to perform and then carries out the planned action. This action will return a reward value depending on the result of the action. All of the possible rewards resulting from the agent's actions are shown below:

- Snake agent picks up a pellet (+10)
- Snake agent collides with its own body or a wall (-100)
- Snake agent moves closer to a pellet (+1)
- Snake agent moves farther from a pellet (-1)

The agent is looking to maximize the total reward that it can earn, so positive rewards represent "good" actions that the AI will continue to try to repeat while the negative rewards represent "bad" actions that the AI will try to avoid repeating in the future.

When selecting an action, the agent follows an ε-greedy approach. There is a hyperparameter, epsilon (ε), that defines the rate at which the agent will choose to perform a uniform, random action. If the agent chooses not to perform a random action, then the agent will deterministically choose the action for that state that has the highest value, or most likely to result in a positive reward. Epsilon will also decay over the course of time, decreasing the chances of random actions as the agent continues to play the game.

After the agent receives its reward for the turn, the value of the state action pair for that move is calculated. The formula to calculate the value for a state action pair is shown below:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma * max[Q(s'a')] - Q(s,a))$$

In this equation, s = current state, a = current action, ɑ = learning rate, γ = discount factor, r = action reward, s' = next state, a' = next action. The aim is that by continuously updating the values of the Q-Table, the snake agent learns to perform actions that are likely to result in a positive reward more frequently while avoiding the actions that are likely to result in negative rewards.

<u>**World and Experiment**</u>

The Snake game board consists of a 500 pixel x 500 pixel world with the snake's head and each of its body parts taking up 20 pixels, effectively turning the board into a 25 x 25 grid. This grid results in a total of 625 possible spaces that the snake head, body, and pellets can be positioned in.

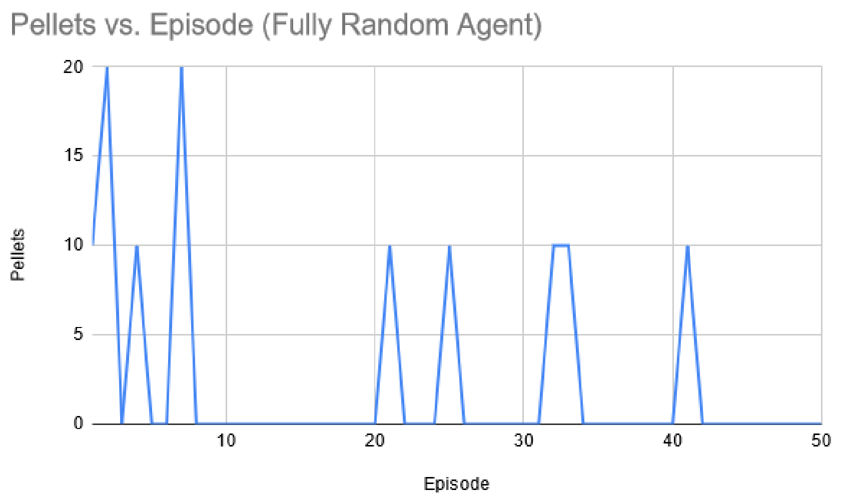
Our main experiment for this project will be to compare our Deep-Q Reinforcement Learning algorithm to a much simpler "test" case where the test case consists of an agent that simply randomly chooses to make a random turn. For the current configuration of this simple agent, the agent has a 50% chance to decide whether it wants to make a turn. If the agent doesn't randomly choose to make a turn, it will simply continue moving in the direction of its current orientation. Otherwise, it will make a turn to the left or right with a 50% chance of turning either direction.

We will run 50 episodes of each agent and then compare the differences in the average number of pellets collected per episode between each agent.

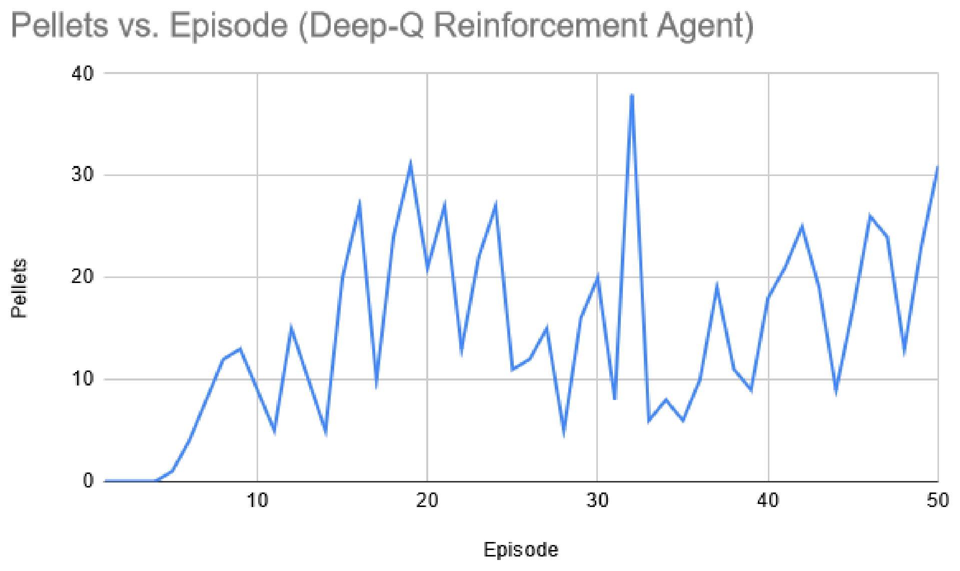
Results

Below are the plots that show the performance of our simple random agent and our Deep Q-Learning agent in terms of the number of pellets collected per episode.

Here is the plot showing the results of our simple agent:



Additionally, here is the plot showing the results of our Q-Learning agent:



As the plots show, the Q-Learning agent overall performed much better than the simple random agent (i.e. it often collected much more pellets per episode than the simple agent did). The average number of pellets that the Q-Learning agent collected in a single episode was about 14.48 pellets per episode while the simple agent collected an average of 0.22 pellets per episode.

There are a couple reasons as to why our Q-Learning agent has a much better performance than the simple random agent. One of the reasons is because the simple agent works purely off of randomness with no other factors determining its next move. Each step of the game has the simple agent simply make a fully random decision to determine if it should take action or not, and if action should be taken, whether it takes a left or right turn is also fully random.

Contrast this simple agent to the Q-Learning agent and we can see that this agent is much more complex. The Q-Learning agent has both a deep neural network and a Q-Table to determine its next move, and both components are frequently updated, after every step in our implementation, in order to help the agent choose the best action for its current state as possible. When the agent receives a positive reward, the weights of the neural network are updated to reflect the positive reward feedback. Similarly, the Q-Table is updated so that the value corresponding to that state-action pair is increased. This effectively allows for the Q-Learning agent, should it reach the same state again, see that the action corresponding to that state has a more positive value, which leads to a higher likelihood of the same action being chosen by the neural network. This works on the opposite end too. Should the Q-Learning agent receive a negative reward, The state-action pair corresponding to the action performed in that state in the Q-Table has its value reduced so that the action is less likely to be chosen again should the snake reach that same state again.

During our testing and data collection, we did notice that our Q-Learning agent isn't exactly perfect. While the agent generally is able to reach the pellet and we can see from the data that it does indeed improve its performance over time, we noticed that the snake often dies and ends the episode by colliding with its own body much more frequently than by colliding with the borders of the game. A likely reason for this behavior is because the snake agent cares about the difference between its head position and the pellet position without accounting for the position of the rest of its body. When the snake's head gets closer to the pellet, the agent is rewarded positively. The snake may keep approaching closer and closer to its own body and be continuously rewarded as long as the head and pellet distance decreases until the agent ultimately receives the negative reward for colliding with its own body. The agent does sometimes avoid its own body by doing maneuvers like traversing long straights on the edges of the board, but oftentimes, the snake agent will try to take the shortest path to the pellet, irregardless of the position of the rest of its body. This could explain why in the data collected above, we can see episodes where the snake may collect dozens of pellets in a single episode only to follow up on the next episode with a much smaller number of pellets collected. This is of course on top of the randomness factor that is in the game itself, such as the random placement of the pellet and the occasional random decision influenced by the epsilon parameter.

Conclusion

Overall, we developed a Deep Q-Learning agent that is able to play the game "Snake" at a pretty proficient level. Using the rewards it receives as well as the neural network and the Q-Table, the snake is able to gain information about the previous states of the game and use that knowledge to predict the best next move. The Q-Learning agent may have some of its own imperfections, but it definitely achieved better results than a truly random agent.

Citations

[1] Build the Famous Snake Game with Python’s Turtle Module -

https://medium.com/edureka/python-turtle-module-361816449390

[2] Practical Reinforcement Learning — 02 Getting started with Q-learning

https://towardsdatascience.com/practical-reinforcement-learning-02-getting-started-with-q-learning-582f63e4acd9

[3] Snake Played by a Deep Reinforcement Learning Agent -

https://towardsdatascience.com/snake-played-by-a-deep-reinforcement-learning-agent-53f2c4331d36