Reinforcement Learning for Snake

Adam Reevesman
Evan Liu
June 28, 2019

Introduction

The computer game *Snake* originated from the 1976 arcade game *Blockade*. Player maneuvers a snake with arrow keys to find an apple inside of a grid, and the snake grows in length when it reaches the apple. The walls of the grid and the body of the snake are obstacles. If the snake runs into the wall or its own body, the game is over. In this project, we attempt to use a reinforcement learning algorithm to train a computer to play snake in the most optimal way possible.

Project Goals

Simply put, the goal of this project is to teach a computer how to play snake using techniques from reinforcement learning. A reinforcement learning framework assumes that there is an *agent* (the snake) that acts in an *environment* (the grid). The agent can be in a set of *states* (different positions in the grid) and collect *rewards* as it moves. We started by using Q-learning. Q-learning is a technique for estimating the value associated with being in one state and moving to another. The value takes into account both the immediate reward from moving to the next state and the estimated future reward from states that might be reached later.
It is a powerful approach, but can be limited by memory requirements when there are many states to model.

Setup

A naive approach would be to create a different state for all possible snake and apple positions. However, this would require far too much memory so instead we consider just the locations of the snake's head and tail, relative to the apple. This reduced the computational cost of solving the problem.

The snake collects rewards each time it makes a move. If the snake loses the game by hitting either the wall or itself, large negative rewards were given to discourage this behavior. If the move resulted in the snake eating the apple, a large positive reward was given to persuade the snake eat more. In order to encourage the snake to take the quickest path to the apple, small negative rewards were given for just traveling in the grid.

In our implementation of deep Q-learning, we wanted to simplify the state space so that it would help with our memory issues and improve the learning process. Instead of using relative positions between a few pieces of the snake and the apple, we instead checked just six binary conditions. We checked whether or not the snake loses the game if it takes one step forward, to the left, and/or to the right. In addition, we checked whether or not the apple somewhere in front, to the left, and/or to the right of the snake. This cut our state space from O(grid_width^4) to 2^6, independent of the size of the grid.

We then modeled the Q-values of each state-action pair using a neural network with one hidden layer where that takes six inputs and produces three outputs. Experience replay was used to estimate the target Q-values for the model. After each move, observations of the resulting old state, action, new state, and reward were saved to a memory that would be randomly sampled from as inputs to the model.

Note: During experimentation phases, *prioritized experience replay* was used so that the model would be trained on data that it did not predict well. The sample of training observations was weighted so that those with larger prediction error were sampled at a higher rate. This appeared to work well before simplifying the state space, but was unnecessary and performed worse once the new states were defined.

Results

When using Q-learning with complicated states, the agent did not perform well. After about 100 moves, it learned how to survive for about 10 steps. This is not much better than random and the agent did not improve much with more training. It also had trouble learning to eat the apple.

In comparison, our deep Q-learning algorithm learns an optimal path to the apple at a quicker pace. After about 50 deaths, the agent regularly began collecting apples. After around 300 deaths, it regularly collected more than five apples. Over the course of future iterations the agent received high scores of up to 25 apples. It also made its way to them quickly.

We went back and applied Q-learning with the simplified state space as well. We found that agent initially learned more quickly than with deep Q-learning but it only achieved high scores of around 15 apples.

Conclusion

Setting up and defining the environment, states, and actions were the most challenging parts of this project, yet simplifying the state space went a long way compared to any model change or hyperparameter tuning. Therefore, future work would focus on improving the current state space. For example, the agent tends to lose when it gets so large that it is trapped by body parts on either side. A few features that ask if the majority of the body parts are in front, to the left, or to the right of the head would help address this issue in a similar way to how adding the safety features helped the snake quickly learn not to run into the wall or itself. As with any machine learning project, correctly defining the problem is the most important step to solving it.