Introduction to Machine Learning
Deep Q-Learning for Playing Snake

Brian Chesko

Advisors Glenn Dawson, Muhammad Umer, and Robi Polikar

Signal Processing & Pattern Recognition Laboratory, Department of Electrical & Computer Engineering
Rowan University, Glassboro, NJ 08028

Signal Processing and Pattern Recognition Laboratory

Background

Reinforcement learning (RL) is a growing subfield of machine learning which deals with how a software agent should behave in order to maximize a reward over the course of execution. Much research has been dedicated to the use of RL methods towards optimal game-playing,.

- DeepMind's AlphaStar defeated professional player Grzegorz "MaNa" Komincz 5-0 in real-time strategy game StarCraft II (2019).

https://www.engadget.com/2019/07/11/starcraft-ii-players-can-play-against-alphastar-ai/

Snake is a simpler game with simpler rules:

- A snake is placed on a board, typically in the center
- The object is to acquire a piece of food which is placed randomly on the board
- User can turn the snake in the four directions
- If the snake hits a wall or itself, the game ends

Because each food eaten increases the length and randomizes the position of the next food, Snake is a suitably difficult target for a reinforcement learning algorithm.

Deep Q-Learning Overview

Q-Learning is a very common RL algorithm in which the optimal behavior is learned over time through exploration and the updating of a function named the Q-function

- Q(s, a) describes the expected "quality" of a particular action '*a*' the agent can take at a given state '*s*' of the environment
- When the state/action space grows too quickly to explore fully with the simpler Q-table algorithm, this function can be approximated with a neural network

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

Q-value equation the Q-network attempts to approximate

- γ defines the proportion of future rewards counted towards value of current state
- r(s, a) is the reward obtained by taking action *a* in state *s*
- A common augmentation of this is a secondary network to stabilize the error during training, referred to as Double Deep Q-Learning

Constraints

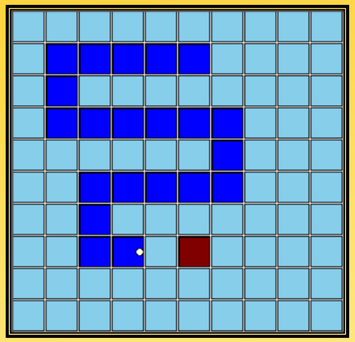
- Training was done on a single laptop machine with an NVIDIA GeForce GTX 1060 with 6 GB memory.
- Neural networks need the entire observation every time (no missing data)

Implementation

- Game implemented in Python
  - Model-View-Controller (MVC) design pattern
  - Utilizes NumPy for encoding board and direction
  - 8x8 board used for training and testing

NumPy

- RL environment implemented using OpenAI Gym Python package
  - Provides a common interface for RL agents to use
- Keras-RL Python package used for RL agent, training and testing
  - Interfaces with Keras, a wrapper for deep learning functionality
  - Utilizes Tensorflow as backend



Screenshot from 10x10 game board. The front of the snake is represented with a white dot, and the food with a red square.

- Action selection done using epsilon-greedy policy
  - Epsilon 'eps' is a hyperparameter
  - With probability eps, the next action is chosenly randomly from action space
  - Otherwise action with highest q-value chosen
  - Eps linearly reduced from 1.0 to 0.35 over 1 million actions
  - During testing, eps is a constant 0.05

Observations

A single observation of the state is a 3-dimensional array *obs*:

- $obs_{x,y,0}$ is the type of object at board cell (x, y)
  - -1 represents a food piece
  - 0 represents an empty square
  - 1 represents a snake segment
- $obs_{x,y,1}$ is the index of the snake segment at (x, y)
  - 0 represents no snake segment at this location
  - > 0 represents a snake segment. A segment distance *i* away from the head of the snake is given the value (1 - (*i* ÷ total length of snake))
    - Value 1 is always the head of the snake

Actions

- Agent can input LEFT, RIGHT, UP, or DOWN from any state
- Encoded in the Q-network as one-hot encoded outputs in a layer with 4 nodes

Reward Structure

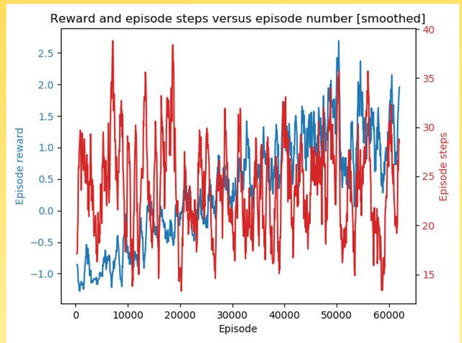
- -1 on death
- -0.01 for every move, regardless of outcome
- +1 for encountering food

Discount factor γ was set to 0.9 for the purposes of training.

- 0.9 emphasizes long-term gains instead of solely focusing on the reward of the immediate action
  - Less "near-sighted"

Results

- After 1.5 million actions, the agent understood the objective of the game
- Approximately 65,000 games played, each played to 100 moves or death, whichever came first
- Agent tended to obtain between 5 and 10 food before becoming trapped or otherwise dying
  - Most common cause of death was trapping itself in a corner
  - With no way to escape it immediately died



Plot showing average episode reward and length across entire training
Note: 1 episode = 1 game

Impact and Considerations

Because RL algorithms learn much like humans by receiving positive and negative feedback on choices they make, they are key candidates for replacing human involvement in some areas. Supplementing other technologies, they can greatly change society:

Public Health/Welfare Impacts

- RL agents trained on simulated and real-time system information may contribute to self-driving vehicle accuracy and safety, reducing loss of life
- ML is used in medical fields to analyze health records and predict correct treatments

Pictured: A Waymo self-driving minivan, https://www.therobotreport.com/self-driving-cars-disengagements/

Economic Impacts

- They have been used for predicting stock movements and improving trade execution time

Global Impacts

- RL agents have the potential to be used during future wars to reduce loss of human life for the wielders, although this may also trivialize warfare.

Standards

- Software adheres to PEP8 coding standard
- OpenAI Gym environment specifications