Reinforcement Learning Report

Introduction

This project focuses on writing an AI which is capable of learning how to play the game "Snake". This game involves controlling an agent composed out of several blocks to eat food within a closed boundary, with the aim being to eat as many blocks as possible without dying. The following rules apply within the game:

- If the snake hits the edge of the boundary the game is over
- If the snake hits itself the game is over
- Each time the snake eats a piece of food a block is added to its length

To complete this task reinforcement learning will be employed and, in particular, Q-learning is used.

Outline of Workflow

The general workflow of this task is now detailed. There are two main parts to this:

- Game, implemented using Pygame. Takes in an action from the game and returns the reward, whether the game is over and the current score
- Model, implemented using PyTorch. A deep neural net which takes in the current state and returns an action

The training has the following form:

1. State = get_state(Game)
2. Action = get_action(state)
   → Model.predict()
3. Reward, game_over, score = Game.play_step(action)
4. New_state = get_state(Game)
5. Remember previous states and new states
6. Model.train()

Further Detail of Methodology

Some other parameters in the game must also be defined before implementing this game.
The rewards given for different outcomes at each time step:

- +10 if the snake eats a piece of food
- -10 if the snake bumps into a boundary or itself
- 0 otherwise

How we define an action in the game. Actions will be defined as three dimensional vectors. It is expected that there would be four actions for four directions. However, if the snake takes the action of the opposite direction, it will immediately bump into itself and die so this action is not considered. Therefore, actions are:

- [1,0,0] = straight (maintain current direction)
- [0,1,0] = right
- [0,0,1] = left (both left and right are in relation to current direction)

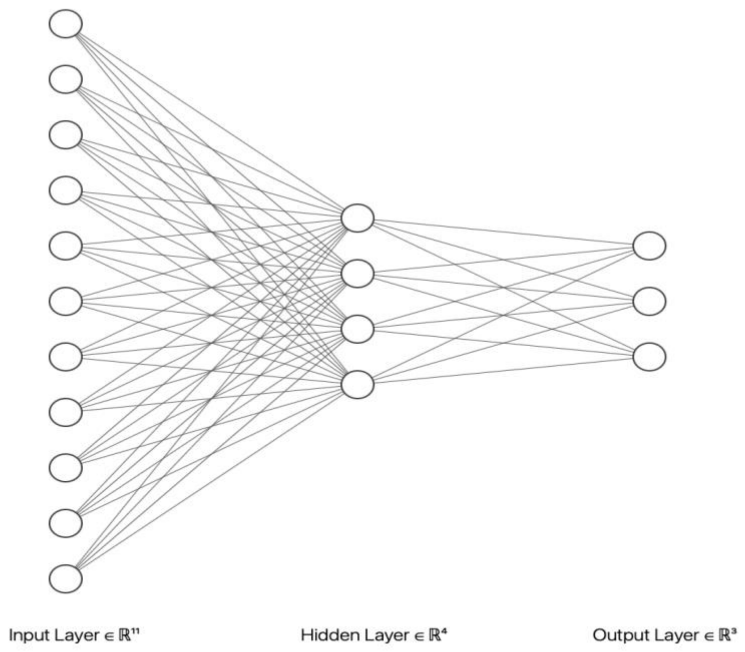
The "state" is the current information about different aspects of the environment and will be composed of eleven values, all of which are boolean:

- Danger straight
- Danger right
- Danger left
- Direction left
- Direction right

- Direction up
- Direction down
- Food left
- Food right
- Food up
- Food down

Network Architecture

As for the neural network, a relatively simple architecture will be used to make decisions. This has eleven input values for the states, one hidden layer and an output layer with three nodes. A graph of this is given below. Note that a hidden layer size of four is used in this graph but this does not represent the final size used in experiments.



The max value of the three outputs is then chosen as the action, i.e., $[5.5, 1.2, 3.4] \rightarrow [1, 0, 0]$.

Q-Learning Description

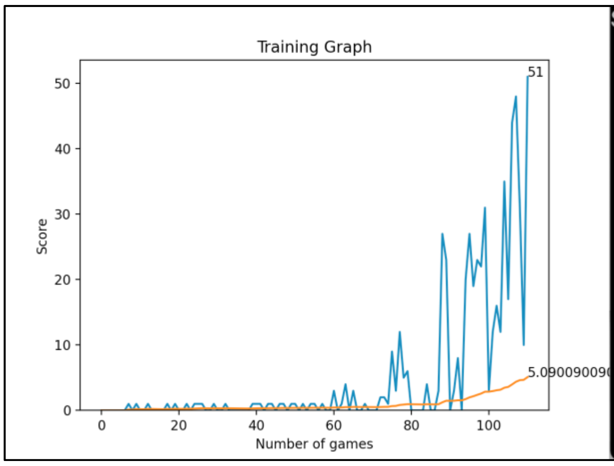
Q-learning is then implemented using the Bellman equation which is detailed in the equation below.

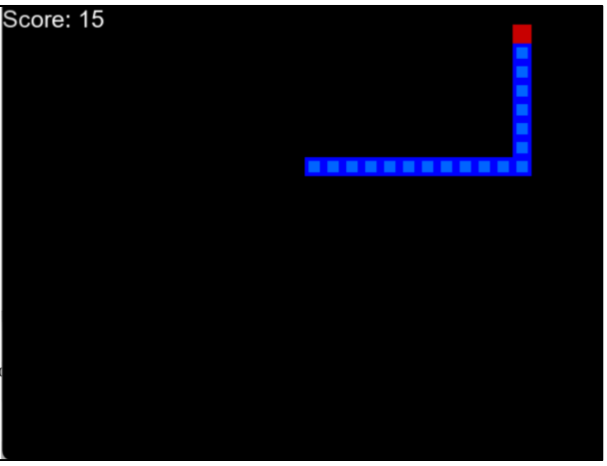
$$\boxed{\text{New } Q(s,a)} = \boxed{Q(s,a)} + \boxed{\alpha}\,[\boxed{R(s,a)} + \boxed{\gamma}\,\boxed{\max Q'(s',a')} - \boxed{Q(s,a)}]$$

- 🟥 New Q Value for that state and the action
- ⬛ Learning Rate
- 🟫 Reward for taking that action at that state
- 🟪 Current Q Values
- 🟩 Maximum expected future reward given the new state (s') and all possible actions at that new state.
- 🟧 Discount Rate

Result of Implemented Code

The code written for this task displays the snake game as the algorithm learns the optimal behaviour and plots a graph showing the score after each game, along with the mean score over all iterations, against number of games.





As shown by the above graph on the left, the algorithm gradually learns how to avoid dying from hitting the boundaries or itself and learns to find the food (red square). As the number of games increases, the mean score and record score increases. Some of the parameter values were altered to find an optimal combination of parameter values. The parameters explored were: hidden layer size (n), learning rate (α) and discount factor (γ). After some experimentation it was found that a good combination of parameter values was $n = 512, \alpha = 0.001, \gamma = 0.95$. There were some circumstances where the algorithm spent time stuck in a loop which made training time longer, but this was a rare occurrence and never resulted in an infinite loop.