Deep Learning Snake

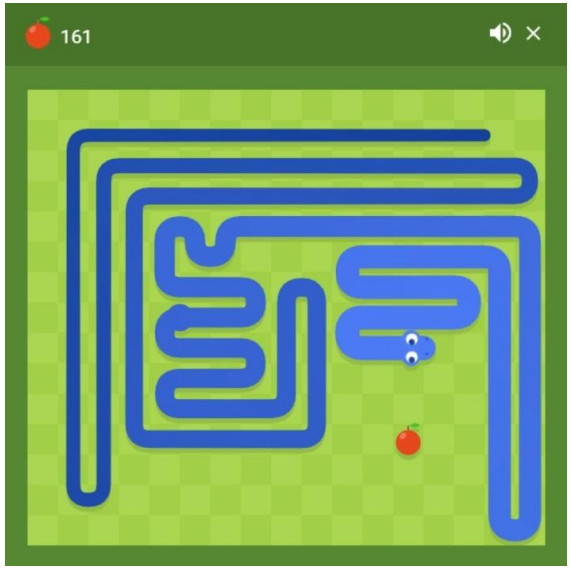
Mehrab Hafiz

Overview

- Majority of games released in the last decade still use rudimentary forms of AI systems.
- As a result developers are forced to generate all possible game states and hard-code agent behavior for each scenario.
- AI can become predictable/repetitive.
- Not always possible to generate every game state.
- A trained Deep Learning agent can work around these problems.

Goal

- Utilize Deep Reinforcement Learning to improve the state of autonomous agents in video games.
- Solving Snake using Deep Learning is a great example.
- As the snake grows in length, the game becomes increasingly difficult.
- Deep Learning agent playing snake must be able to react to the ever changing environment.

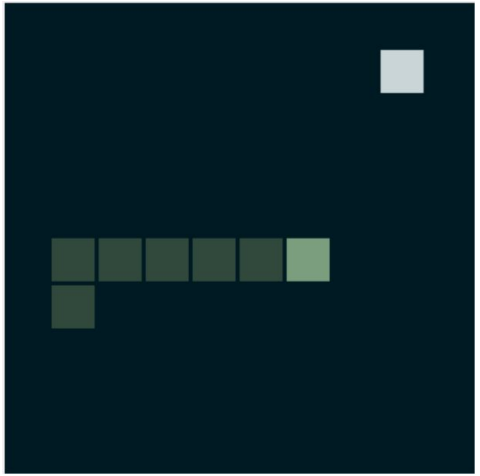


Related Work

Z. Wei, D. Wang, M. Zhang, A. Tan, C. Miao and Y. Zhou, "Autonomous Agents in Snake Game via Deep Reinforcement Learning," 2018 IEEE International Conference on Agents (ICA), Singapore, 2018, pp. 20-25, doi: 10.1109/AGENTS.2018.8460004.

The Environment

- 2D matrix representing a game of Snake.
- 0, 1, 2, 3 representing empty block, tail, head, food, respectively.
- Rewards based on length, distance from food and time.



Dataset & Basic Model

Neural Net:

- Two hidden layers, 256 and 128 rectifiers
- Input of size 55.
- Output layer, 4 actions.
- MSE loss and Adam Optimizer

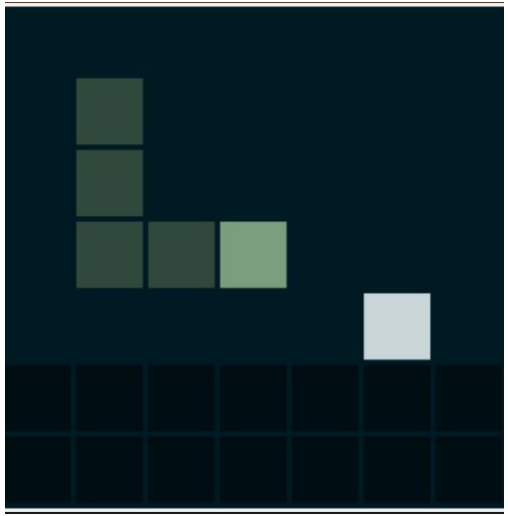
Agent:

- Translated to make snake’s head the center of matrix.
- -1, 0, 1, 2, 3 representing empty block, tail, head, food, and wall, respectively.
- Reward may not fall within the map.

Dataset & Basic Model

Agent:

- Array of size 6 is concatenated to the matrix
- Describes snake direction, food direction, distance etc.



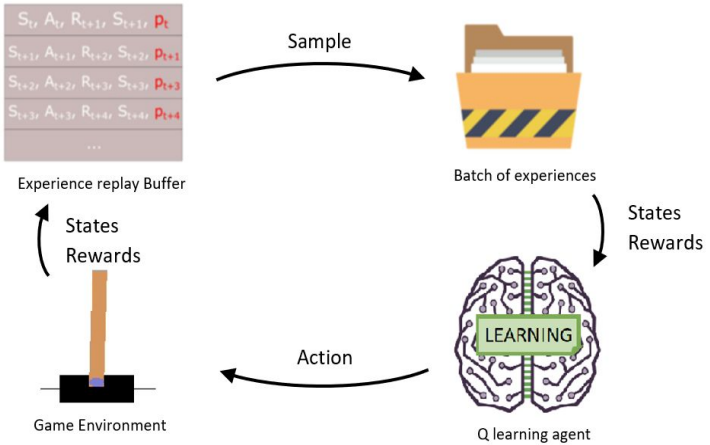
Exploration vs Exploitation

- Necessary for the agent to explore its environment.
- Can double the average number of steps survived.
- However, random predictions can end the game during the later stages.
- Solution: Decrease epsilon as the agent explores its environment.
- With starting eps = .9 and a decay rate of 10%, increases the chance of snake finding its food.

```
Progress: 0.0 %   Current eps= 0.8
Progress: 5.0 %   Current eps= 0.76
Progress: 10.0 %   Current eps= 0.72
Progress: 15.0 %   Current eps= 0.68
Progress: 20.0 %   Current eps= 0.64
Progress: 25.0 %   Current eps= 0.6
Progress: 30.0 %   Current eps= 0.56
Progress: 35.0 %   Current eps= 0.52
Progress: 40.0 %   Current eps= 0.48
Progress: 45.0 %   Current eps= 0.44
Progress: 50.0 %   Current eps= 0.4
Progress: 55.0 %   Current eps= 0.36
Progress: 60.0 %   Current eps= 0.32
Progress: 65.0 %   Current eps= 0.28
Progress: 70.0 %   Current eps= 0.24
Progress: 75.0 %   Current eps= 0.2
Progress: 80.0 %   Current eps= 0.16
Progress: 85.0 %   Current eps= 0.12
Progress: 90.0 %   Current eps= 0.08
Progress: 95.0 %   Current eps= 0.04
```

Experience replay

- Main memory containing 100000 samples.
- Randomly select 64 samples from the main memory to fill the batch.
- Doubles the number of terminated games during the first 100,000 steps.
- Diverse experiences.

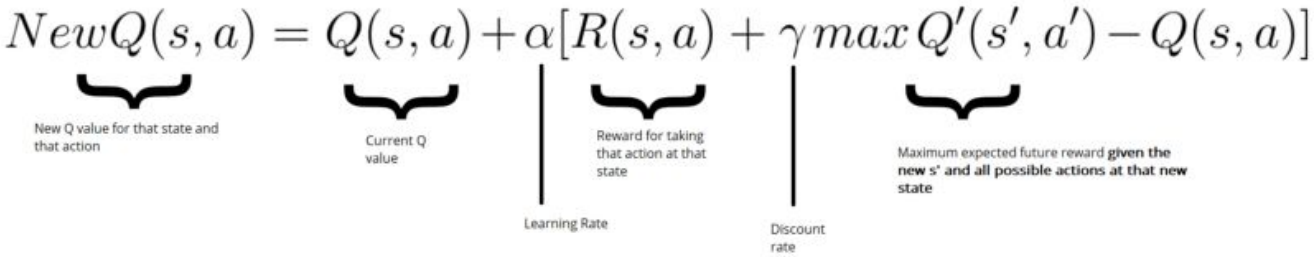


New Q-Learning algorithm with gamma

- In addition to current rewards, factor in the max possible reward in the following state.
- Sample consists of state, action, reward, new state and terminal.
- Snake length 20 (average of terminated games).

Q-Learning algorithm

- 5 memory buffers: states, action, reward, new_state, terminal.
- Solves 6x6 games of Snake.



Prioritized replay

- Assign priorities to samples.
- All samples have priority of at least one.
- prio = 1 + scaling_factor * priority
- P = priority divided by sum of all priorities

```
def get_probabilities(self):
    priorities = t.from_numpy(self.priorities).to(self.device)

    # Raise priority to a scaling factor
    scaled_priorities = t.pow(priorities, self.priority_sampling)

    # Every sample as atleast a priority of 1
    offset_priorities = t.add(scaled_priorities, 1)

    sample_probabilities = t.div(offset_priorities,
                                 t.sum(offset_priorities).item())
    return sample_probabilities.cpu().numpy()

def sample(self):
    probs = self.get_probabilities()
    indx = np.random.choice(self.mem_size, self.batch_size,
                            p=probs, replace=False)

    return (self.states[indx], self.new_states[indx], self.actions[indx],
            self.rewards[indx], self.terminal[indx])
```

Results

Average scores per 50 games steps after 400,000 training steps:

- Basic model: 8
- With exploration: 8
- Exploration + Experience replay: 14
- All above + new Q algorithm: 24
- All above + prioritized learning: 29
- Baseline CNN: 12

Observation:

- Exploration speeds up training but does not increase score
- Experience replay prevents loops
- New algorithm less likely to blindly chase food
- Prioritized learning prone to overfitting

Sources and Conclusion

Z. Wei, D. Wang, M. Zhang, A. Tan, C. Miao and Y. Zhou, "Autonomous Agents in Snake Game via Deep Reinforcement Learning," 2018 IEEE International Conference on Agents (ICA), Singapore, 2018, pp. 20-25, doi: 10.1109/AGENTS.2018.8460004.

Tom Schaul, John Quan, Ioannis Antonoglou, David Silver "Prioritized Experience Replay" arXiv:1511.05952