Reinforcement Learning: Snake

Lu Sang, Michael Haberl,
Ming Yip Cheung, Roberto Pereira
Technical University of Munich

Project Overview

- We used reinforcement learning to train a neural network to play Snake.
- We used different learning methods like DQN and DDQN and compared their performance.

Reinforcement Learning

- An artificial agent learns by itself to achieve successful strategies that lead to the greatest long-term rewards.
- Learning by trial-and-error, rewards or punishments.

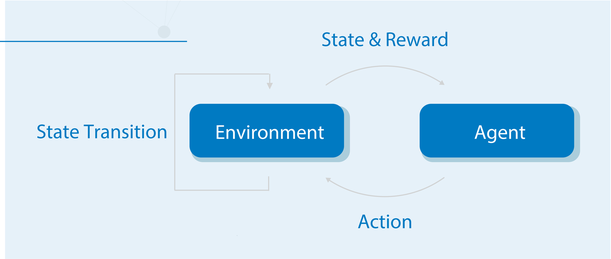

Figure 1: Reinforcement Learning diagram[1]

Network Architecture

Input	Conv1	Conv2	FC1	FC2
4 x 10 x 10	kernel 3x3 strides 1x1 16 x 8 x 8	kernel 3x3 strides 1x1 32 x 6 x 6	Dense 256	Dense 3

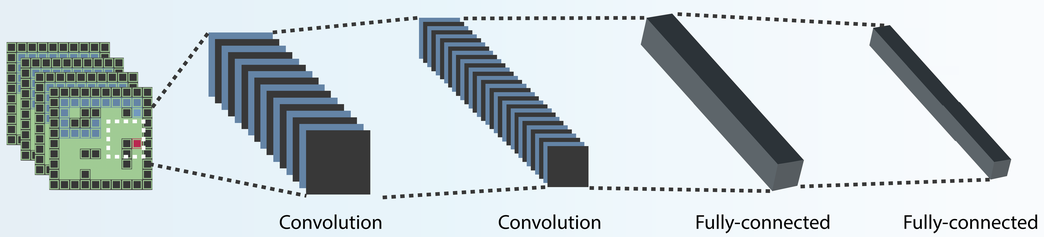

Figure 2: Neural network architecture

Methods

- Basic: Q function
  $loss = (r + \gamma \max_{a'} Q^{\wedge}(s, a') - Q(s, a))^2$
- Double Q function
  $loss = (r + \gamma(Q(s', \arg\max_{a'}(Q^{\wedge}(s, a')) - Q(s, a))^2$
- Multi step rewards
  $$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}$$
- States-action-rewards-states-action
  $Q(s_t, a_t) \rightarrow r_t + \gamma Q(s_{t+1}, a_{t+1})$

s, r → Q(s, a) → a' → s'
model agent env.

- Prioritized replay
  $$p_t \propto |R_{t+1} + \gamma_{t+1} max_{a'} q_\theta(S_t, A_t)|^\omega$$
- Dueling networks
  $$q_\theta(s,a) = v_\eta(f_\xi(s)) + a_\psi(f_\xi(s), a) - \frac{\sum_{a'} a_\psi(f_\xi(s), a')}{N_{actions}}$$

Procedure

- a set of environment and agent states at time t S_t;
- a set of actions A_t of the agent;
- $P_a(s,s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ probability of transition from state s to state s' under action a.
- $R_a(s, s')$ is the immediate reward after the transition from s to s' with action a.

The agent interacts with the environment at each time step t by observing the reward r_t, then choose the action a_t from the action set A. Then the environment responses to the new action a_t by moving to a new state s_{t+1} and gives the reward r_{t+1}.

Method Analysis

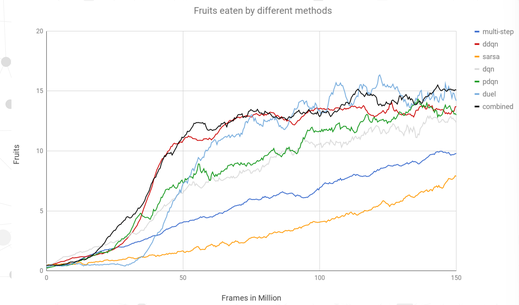

Figure 3: Fruits eaten for different methods during testing

Hyperparameter Tuning - Experience Replay Memory Size

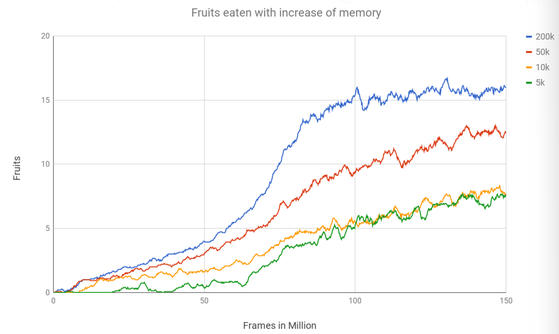

Figure 4: Fruits eaten with increase of memory

Our model is based on one important assumption which is the process is a Markov process, the big memory size, where we sample the batches from, can help us get rid of the dependence of the data.

Hyperparameter Tuning - Multi-step Stepsize

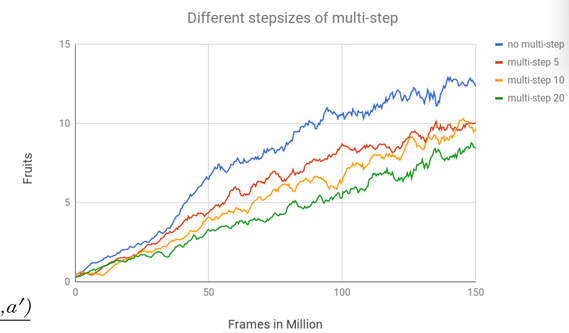

Figure 5: Fruits for different stepsizes

HYPER-PARAMETER	VALUE
BATCH SIZE	64
MEMORY SIZE	100,000
REPLAY PERIOD	EVERY 1 AGENT STEP
DISCOUNT FACTOR	0.9
MAX FRAMES PER EPISODE	1,000
FRUIT, STEP, DEATH REWARD	[1, 0, -1]
MULTI STEP SIZE	10

Snake Game - General Setting

- penalize when the snake runs into the screen or itself.
- reward when the snake eats a fruit.

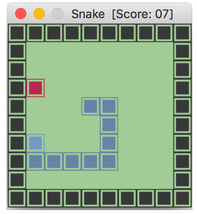

Figure 6:
Enviroment used to train [4]

Visualizing the Value Function

Reward Curve

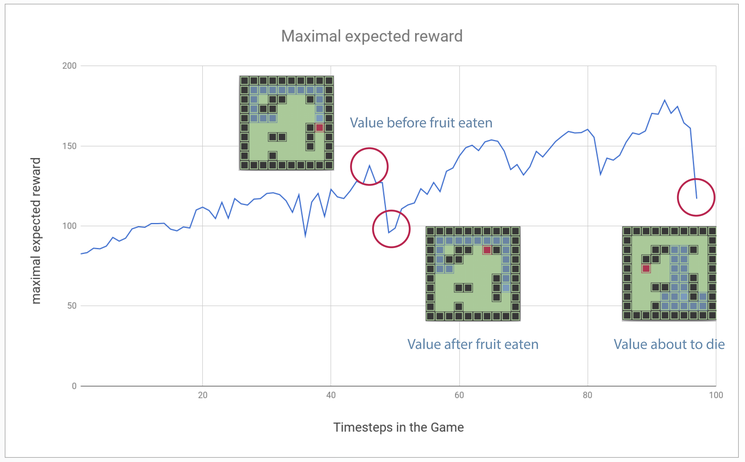

Figure 7: Reward Curve of DQN

The graph shows how the expected maximal reward increases or decreases during one game. Note that every time the expected maximal reward goes down, it means that the snake ate a fruit at the last step. Because the fruit is further away at the next step the expected reward goes down.

Conclusion

- Method
  - The combination of DDQN, dueling networks, prioritized replay and multi-step performed the best.
  - The performance SARSA and multi-step was worse than original DQN.
- Memory
  - The more memory we used for Experience Replay the better the performance.
- Multistep
  - The performance decreases as the stepsize gets bigger.

Reference

[1] Fuertes, R. A (2017). *Machine Learning Agents: Making Video Games Smarter With AI.* Retrieved January 15, 2018 from https://edgylabs.com/machine-learning-agents-making-video-games-smarter-with-ai

[2] Hessel, M., Modayil, J. & Hasselt, H. V. (2017). *Rainbow: Combining Improvements in Deep Reinforcement Learning.* Retrieved January 10, 2018 from https://arxiv.org/pdf/1710.02298.pdf

[3] Mnih, V., Kavukcuoglu, K. & Silver, D. (2013). *1312.5602v1.pdf.* Retrieved January 10, 2018 from https://arxiv.org/pdf/1312.5602v1.pdf

[4] Yuriyguts (2017). *GitHub - YuriyGuts/snake-ai-reinforcement: AI for Snake game trained from pixels using Deep Reinforcement Learning (DQN).* Retrieved January15, 2018 from https://github.com/YuriyGuts/snake-ai-reinforcement