An AI for Game *Tokkun'99*

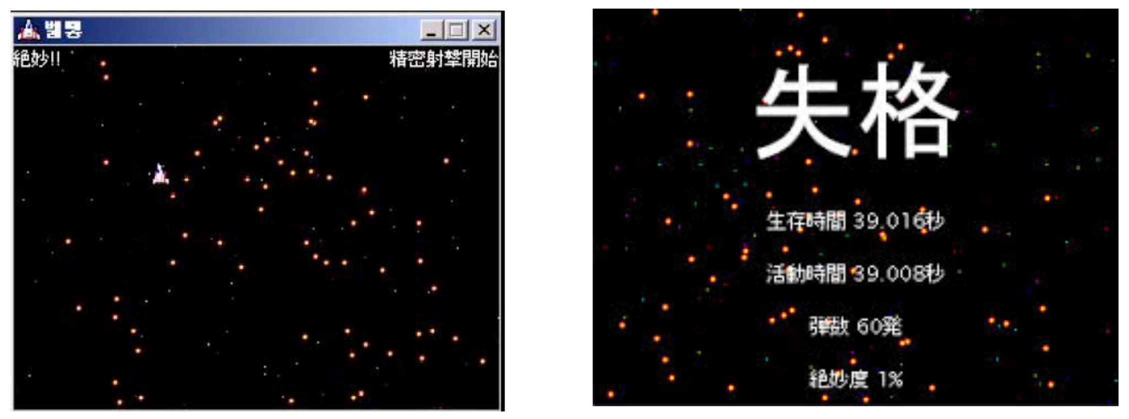
Yilong Li[1], Shiyu Liu[2], Zhefan Wang[2]

[1] Department of Computer Science, Stanford University [2] Department of Electrical Engineering, Stanford University

Introduction

The Game

- Tokkun'99 is a game which requires quick reactions and thus challenging for human players.
- The objective for the player is to control the airplane to go along eight directions on a 2D map in order to dodge bullets.
- Most bullets move in a straight line with constant velocity, but some special ones can follow the agent (player), move along a parabola, or move at high speed.
- Once player is hit by any of the bullets, the game ends.



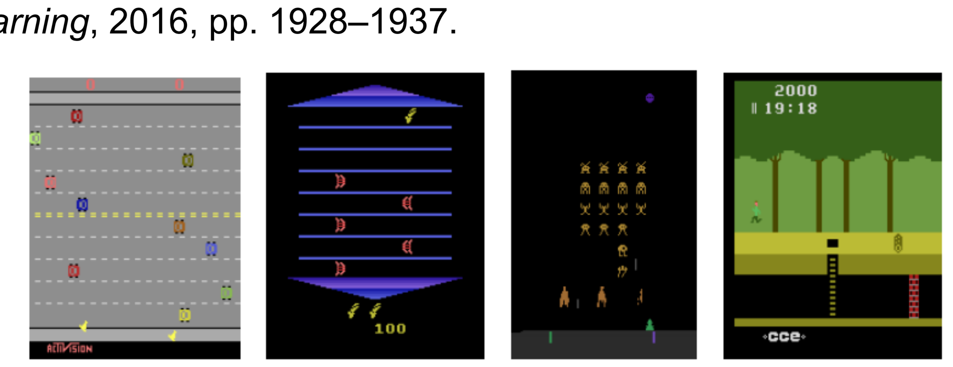
Screenshots of the Tokkun'99 Game

Challenges

- To speed up the training, need to implement a **game simulator** to run in headless mode and return useful information
- Need to experiment and figure out which **features** are most useful to make the AI as good as or even better than an average human player

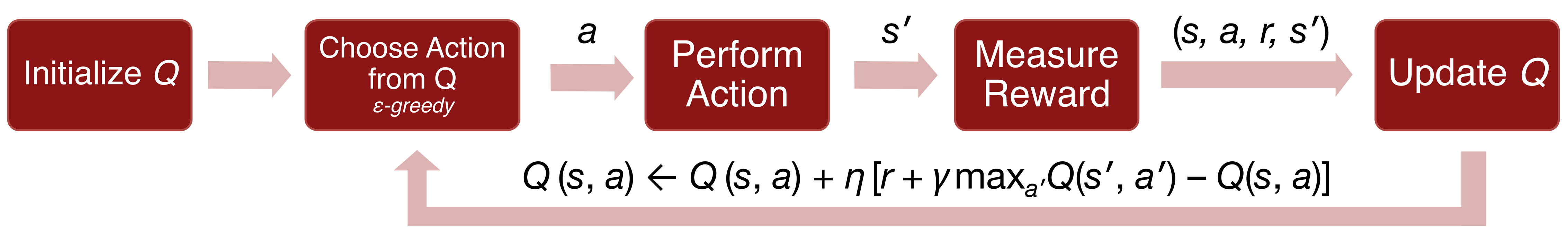
Related Work

- M. G. Bellemare, Y. Naddaf, et al. "***The arcade learning environment: An evaluation platform for general agents***," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, 2013.
- V. Mnih, K. Kavukcuoglu, et al. , "***Playing atari with deep reinforcement learning***," in *NIPS Deep Learning Workshop*, 2013.
- V. Mnih, K. Kavukcuoglu, et al. "***Human-level control through deep reinforcement learning***," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- V. Mnih, A. P. Badia, et al. "***Asynchronous methods for deep reinforcement learning***," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.



Methods

Framework – Q Learning



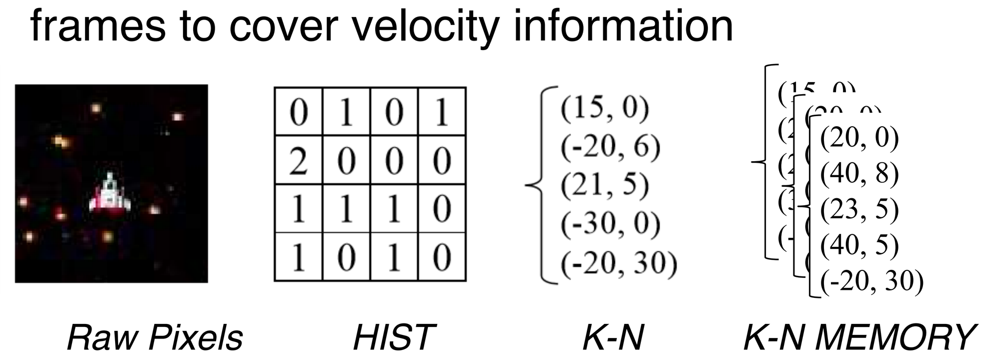
Q-Function Estimation

A. Feature Extraction + Linear / Neural Network Approximation

Features

Hand-labeled Feature

- **HIST:** Histogram of neighboring bullets of R*R region in B*B bins
- **K-N:** Relative offsets of *K* nearest bullets
- **Variations of K-N:**
  - **K-N INVERSE**: K-N features including squared distance to player to highlight nearer bullets
  - **K-N MEMORY**: Include K-N features of *R* recent frames to cover velocity information



General Feature from Raw Pixels

- Methods from (Bellamare et al. , 2012) on Atari games
- **BASS:** Remove the background and use a **8-color palette** to represent a downsampled image.
- **LSH:** Map raw game screens into a small set of binary features using a hashing algorithm such that similar screens have similar hashes.

Approximation Methods

Linear Function Approximation

- Use linear combination of features to approximate Q function

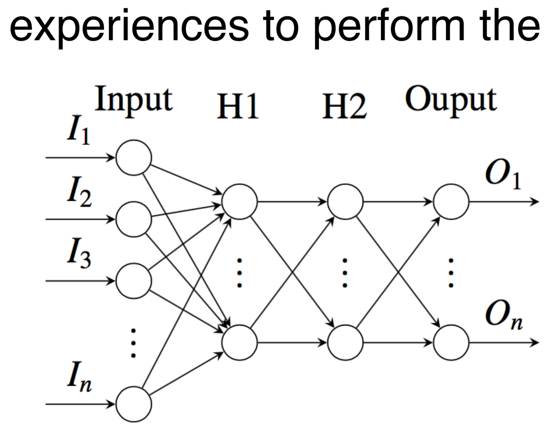
$$Q^{\pi}(s, a; \boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \phi(s, a)$$

- Train $\boldsymbol{\theta}$ using SGD:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\,[r + \gamma \max_{a'} Q^{\pi}(s', a'; \boldsymbol{\theta}) - \boldsymbol{\theta} \cdot \phi(s, a)]$$

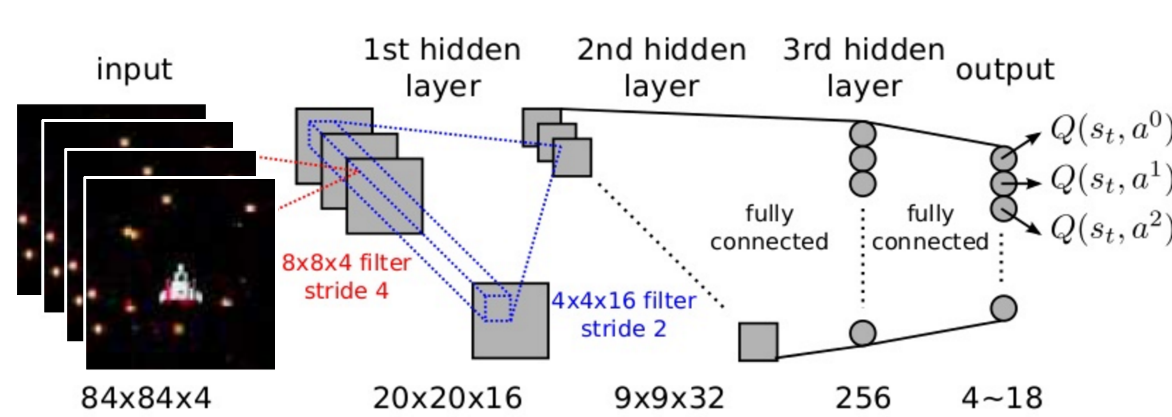
Neural Network Function Approx.

- Train a 3-layer fully-connected neural network (multi-layer perceptron) to approximate Q function
- Memorize *N* most recent (*s*, *a*, *r*, *s'*) tuples, which are previous experiences.
- After each action taken, draw a mini-batch of previous experiences to perform the update step.



B. Deep Q Learning

- Methods from (Mnih et al, 2013)
- Use **Convolution Neural Network** to estimate the Q function.
- Apart from the local memory features, we also keep a large replay memory of size ***D*** to save previous plays
- For each step we draw a minimatch of size ***B*** with (*s*, *a*, *r*, *s'*) tuples to train the network
- In our experiments, *D* = 2000, *B* = 32

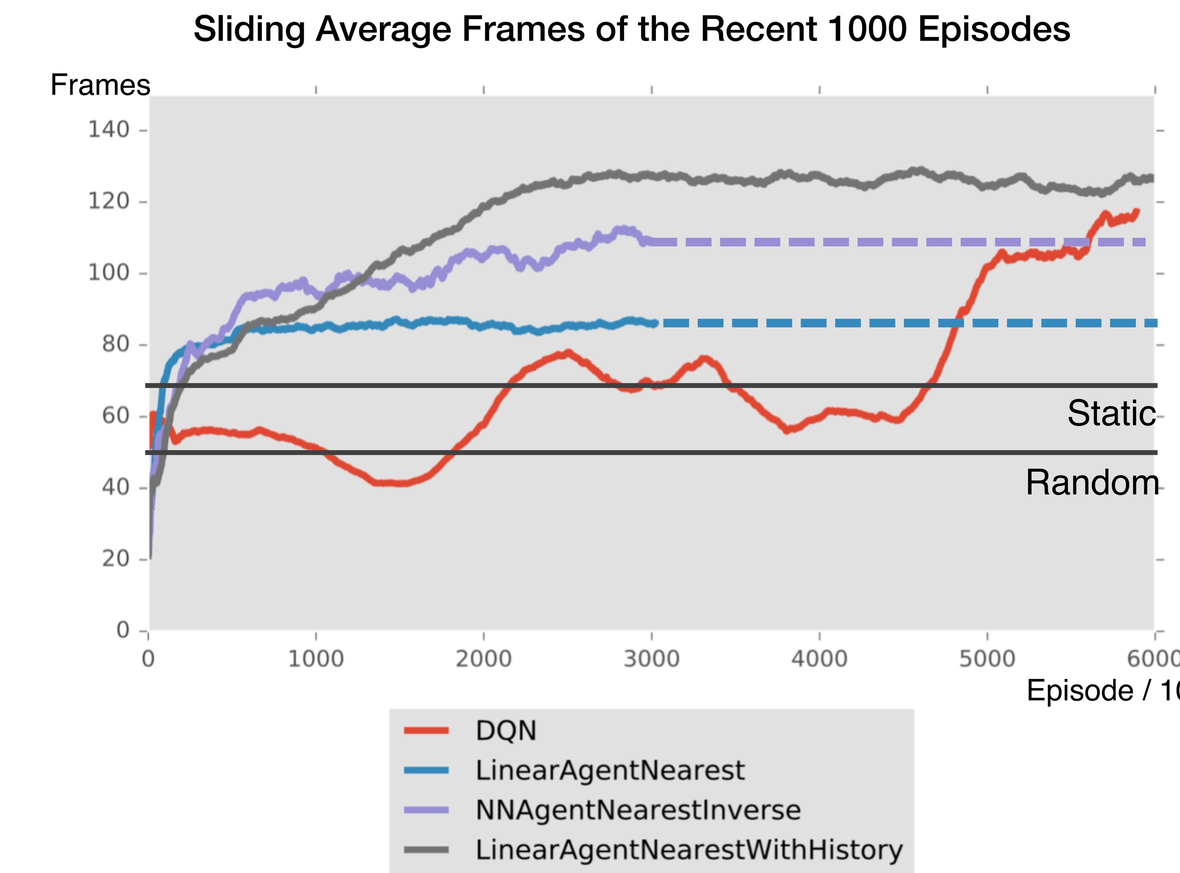


Results

Average # of Frames Using Different Methods

Feature	Params	Linear Approx.	3-Layer MLP
HIST	R = 25, B = 5	89.37	61.28
	R = 30, B = 6	89.34	80.50
K-N	K = 5	85.76	90.13
	INVERSE, K = 5	85.37	**108.40**
	MEMORY, K = 5	**126.17**	100.12
DQN	**115.81**		
Random	52.00		
Static	63.56		

Comparison of the Top Methods



Memory matters; DQN is not satisfactory

Future Work

- Implement some **general feature extracting methods** to compare with current features
- Find the reason for Deep Q Learning's low performance
  - Tune the network structure, network parameters (training η, decaying rate, etc.) and the memory parameters (D, B)
  - Try Double DQN and Dueling Double DQN which have achieved better results on Atari games
  - Now training and tuning is slow, try to implement methods like single-step SARSA / Q Learning and actor-critic based methods like A3C