Reinforcement Learning for Rapid Roll

Bera Shi, Zhecheng Wang, Yang Li
{bshi, zhecheng, yangli95}@stanford.edu

Overview

Our project aimed at implementing classical Reinforcement Learning and state-of-art Deep Reinforcement Learning to train an agent to play “Rapid Roll”, a popular video game.

We expected to develop that how Deep Reinforcement Learning, taking the power of Deep Neural Network (DNN), can outperform the classical Reinforcement Learning on certain tasks. While the classical Reinforcement Learning algorithm learn the hand-crafted backstage features of the game, the Deep Reinforcement Learning directly learn images from the game interface, which is an end-to-end learning case.

Classical Fitted Value Iteration

We regarded playing “Rapid Roll” as a continuous-state Marlov Decision Process (MDP) and implemented Fitted Value Iteration algorithm to approximate the value function in the process. In the deterministic model of continuous-state MDP, the value function was updated as below:

$$V(s) = R(s) + \max_a V(s')ds'$$

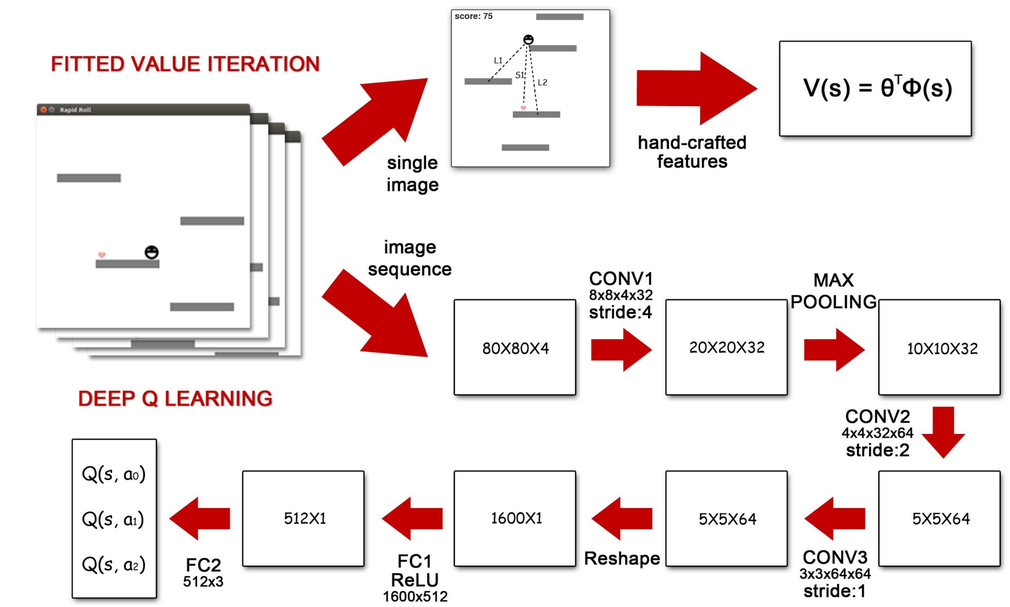
In Fitted Value Iteration, we used the linear regression to approximate the value function as a linear function of the states:

$$V(s) = \theta^T \varphi(s)$$

References

[1].Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

[2].Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).



Result

Game level	Random	human	FVI	DQN
easy	163.7	Inf	577.7	Inf
hard	102.1	314.7	529.8	Inf

Table 1. Average score of Random Action, human player, Fitted Value Iteration (FVI) Agent and Deep Q Learning (DQL) Agent at two different game difficulty level.

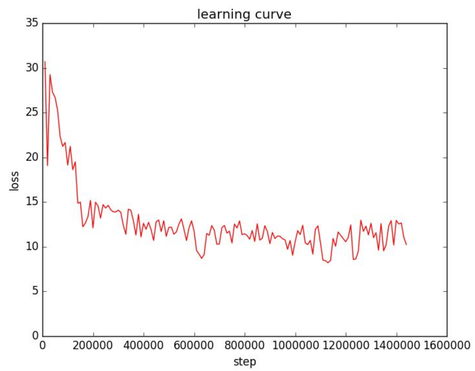


Figure 1. Learning curve of Deep Q Learning

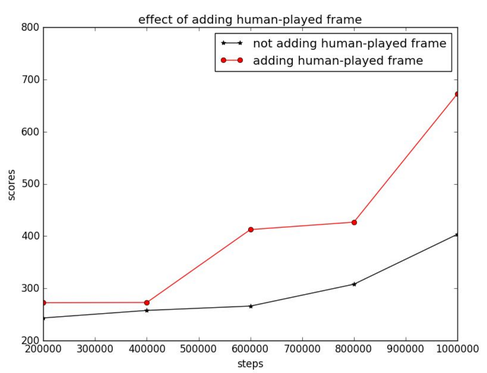


Figure 2. Effect of adding human-played frames

Deep Q-Learning

In deep Q-learning, we expected to build an end-to-end learning algorithm, where the agent could only get information that was feeded to human players during the entire game, including the UI images at each state, for example, the flags of game over and scoring, but nothing else.

The value function was a function of states and actions. It was called the state-action value function Q, and Q=Q(s,a). In this case, the action-choosing decision at each step was made as follow:

$$a_t = \max_a Q(s_t, a ; \theta)$$

The appriximation of Q function was:

$$y_i = r_t + \max_a Q(s_{t+1}, a ; \theta_{i-1})$$

We minimized the loss function:

$$L(\theta_i) = (y_i - Q(s_t, a_t ; \theta_i))^2$$

with Adam Optimizer method.

We also implemented **ε-greedy approach** and constructed a **replay memory** to store (s_j, r_j, a_j, s_{j+1}) for training.

Parameter	Value	Parameter	Value
minibatch size	32	replay memory capacity	200,000
observe steps	50,000	explore steps	2,000,000
reward (alive, scoring, eating bonus, dead)		+0.1, +0.2, +10, -10	

Discussion

- Agent with deep Q-learning agent outperformed humans to a large extend, especially when the game difficulties increased.
- The performance of deep Q-learning was far better than the classical linearly fitted value iteration, though deep Q-learning directly learned the images while classical method learned hand-crafted features.
- Adding human-played information(frames) into the replay memory would speed up the training of deep Q-learning.