Pinball Using a Deep Q-Network

Elias Stenhede, Valter Schütz

Chalmers University of Technology
Electrical Engineering Department

CHALMERS UNIVERSITY OF TECHNOLOGY

The Game

Windows 95's Space Cadet Pinball

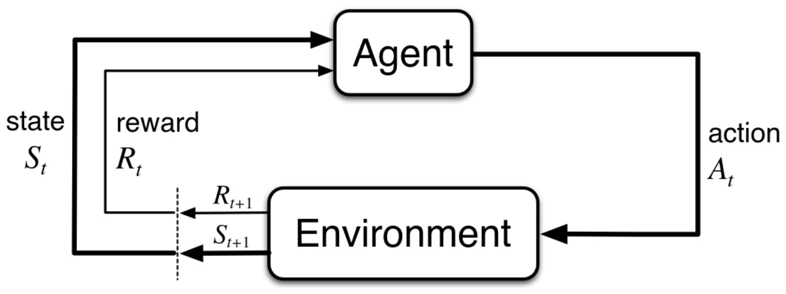
The playing field

- Objective
  - High score
  - Don't waste too much time (holding the ball forever)
- Actions
  - Left (up/down)
  - Right (up/down)
  - Plunger (pull/release)
  - Pause (encouraged by us)
- State
  - Ball position, (x, y)
  - Ball speed, (x, y)
  - Flipper, (right, left)
  - Plunger

7 actions and 7-dimensional state

Reinforcement learning

The purpose of RL is to learn how to act optimally in a Markov decision process [3].



The mapping from states to actions is called a **policy**

$$\pi(a|s) \doteq p(A_t = a | S_t = s) \tag{1}$$

What is Q-learning?

In each step t, we get some **reward**

$$R = \min\left(\frac{\Delta\text{score}}{20000}, 1\right). \tag{2}$$

We want to maximize the **discounted return**

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \qquad \gamma \in (0,1) \tag{3}$$

Given a policy π the expected value of a state/action combination is called the **action-value function**

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G|s,a]. \tag{4}$$

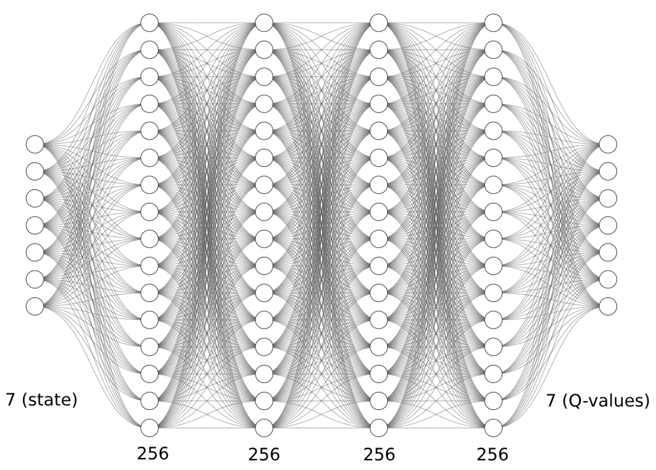
If $\pi(a|s)$ is optimal, it satisfies the **Bellman update equation**

$$q_\pi(S_t, A_t) = R_{t+1} + \gamma \max_a q_\pi(S_{t+1}, a). \tag{5}$$

Thus, we define

$$\textbf{Loss} = \left(q(S_t, A_t) - R_{t+1} + \gamma \max_a q(S_{t+1}, a)\right)^2. \tag{6}$$

Q-Network



A Q-network transforms states to Q-values

The Q-network estimates Q-values for each action, given a state.

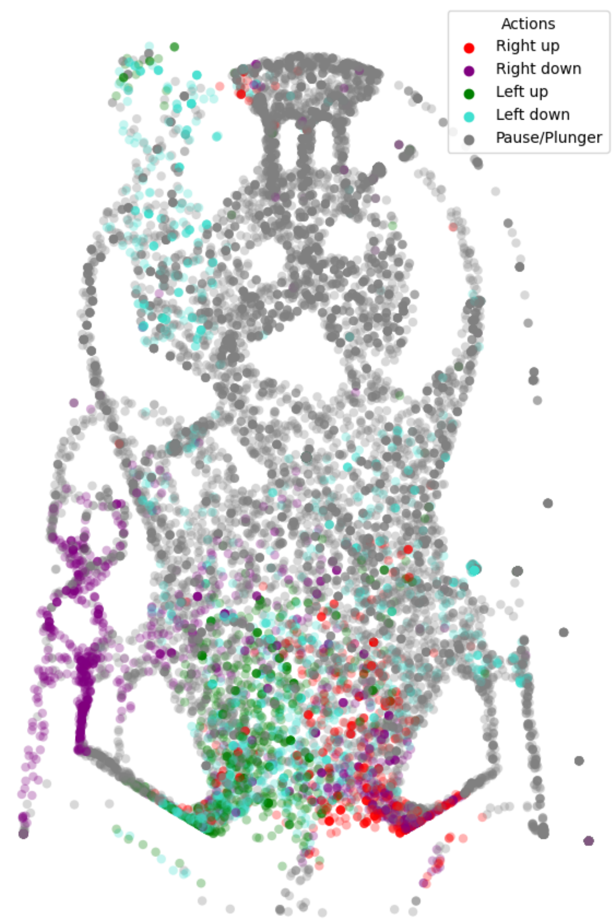
With probability $1-\epsilon$, the agent chooses the action with the highest Q-value, this is called an ϵ-greedy policy.

The Algorithm

Algorithm 1: DQN Training with ε-greedy policy and replay buffer [2]

Initialize Q-network with random values
for *each episode* **do**
  for *each timestep* **do**
    Choose action a with ε-greedy policy
    Execute action a, observe reward r and next state s'
    Store (s, a, r, s') in the replay buffer
    Sample a minibatch of experiences from the replay buffer
    Update Q-network
  end
end

Results



Action with the highest Q-value as a function of position on the board

Approximately optimal actions

When the ball is far from the flippers, the agent chooses to wait!

Since we are using an ϵ-greedy policy, it chooses the "preferred" action with probability $1-\epsilon$. In our case $\epsilon = 0.1$.

Taking $\epsilon = 0$ results in "optimal" but deterministic behaviour.

We approximate the optimal policy, finding a locally optimal solution. With adjustments to DQN, better performance is often achieved

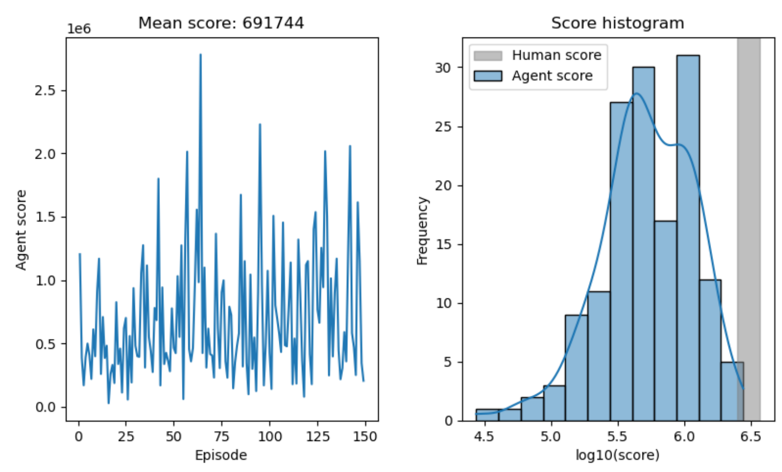
Unusual training loss

MSE loss, according to equation (6).

First, the Q-network approximates the Q-values.

Then, it refines them.

Learning using DQN is usually quite slow, GPU time in the plot is about 72 hours.



Agent performance (blue) vs human (gray)

Human normalized score

We let the former head of *Flipperiet* play the game for 20 minutes.

Only after implementing *improvements* to DQN, results better than human performance is consistently seen, see for example [1].

References

[1] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. 2013.

[3] C. J. C. H. Watkins. Learning from delayed rewards. 1989.

eliasst@student.chalmers.se, schutzv@student.chalmers.se