Playing Atari with Deep Reinforcement Learning

Ha-larm

Abstract

- First deep learning model using reinforcement learning
  -Successfully learn control policies directly
  -From high-dimensional sensory input(pixels)
- CNN model trained on a variant of Q-learning
  -input:raw pixel,output:a value function estimation future reward
- Applied seven Atari 2600 games (no adjustment)
  -Outperforms ALL previous approaches on six games
  -Surpasses a human expert on three games

Problem to Solve

- Learning directly from high dimensional sensory input

  -Long-standing challenges of RL

- Most successful RL relies on hand-crafted features

Solution

- Most DL requires hand labeled training data

  -RL must learn from a scalar reward signal

  -Reward signal is often sparse,noisy,and delayed

  -delay between actions and resulting rewards can be thousand time steps

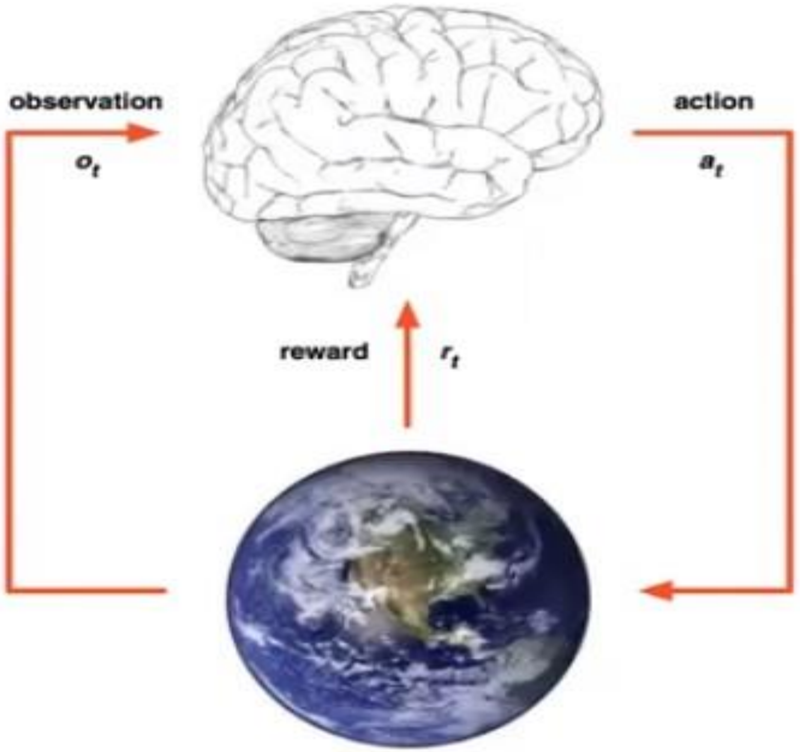
  -CNN with a variant Q-learning

- Mst DL assumes data samples are independent

  -RL encountes sequences of highly correlated states

  -Experience replay

Agent and Environment



- At each step t the agent:
  - Executes action a_t
  - Receives observation o_t
  - Receives scalar reward r_t
- The environment:
  - Receives action a_t
  - Emits observation o_{t+1}
  - Emits scalar reward r_{t+1}

State

- Experience is a sequence of observations, actions, rewards

$$o_1, r_1, a_1, \dots, a_{t-1}, o_t, r_t$$

- The state is a summary of experience

$$s_t = f(o_1, r_1, a_1, \dots, a_{t-1}, o_t, r_t)$$

- In a fully observed environment

$$s_t = f(o_t)$$

Policy

- Policy is the agent’s behaviour

- Deterministic policy: a=$\pi(s)$
- Stochastic policy:$\pi(\alpha|s) = P[\alpha|s]$(확률)

Value Function

- A value function is a prediction of future reward
  - "How much reward will I get from action a in state s?"
- Q-value function gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ

$$Q^{\pi}(s, a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \mid s, a\right]$$

- Value functions decompose into a Bellman equation

$$Q^{\pi}(s, a) = \mathbb{E}_{s', a'}\left[r + \gamma Q^{\pi}(s', a') \mid s, a\right]$$

Optimal Value Functions

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) = Q^{\pi^*}(s,a)$$

- Once we have Q^* we can act optimally,

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

- Optimal value maximises over all decisions. Informally:

$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

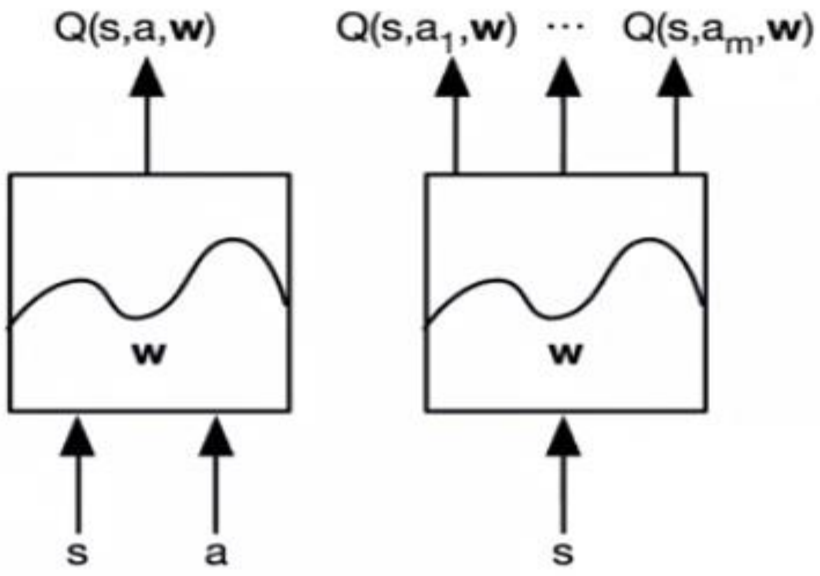
- Formally, optimal values decompose into a Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right]$$

Q-Networks

Represent value function by Q-network with weights **w**

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$



Q-Learning

- Optimal Q-values should obey Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s',a')^* \mid s,a \right]$$

- Treat right-hand side $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as a target
- Minimise MSE loss by stochastic gradient descent

$$l = \Big(\underbrace{r + \gamma \max_{a} Q(s',a',\mathbf{w})} - \underbrace{Q(s,a,\mathbf{w})} \Big)^2$$

Converge

- Converges to max Q using table lookup representation

- But diverges using NN due to:
  -Correlations between samples
  -Non-stationary targets

Deep Q-Networks (DQN): Experience Replay

To remove correlations, build data-set from agent's own experience

$$
\begin{array}{ccc|c|cc}
\cline{4-4}
 & & & s_1, a_1, r_2, s_2 & & \\
\cline{4-4}
 & & & s_2, a_2, r_3, s_3 & \rightarrow & s, a, r, s' \\
\cline{4-4}
 & & & s_3, a_3, r_4, s_4 & & \\
\cline{4-4}
 & & & \ldots & & \\
\cline{4-4}
s_t, a_t, r_{t+1}, s_{t+1} & \rightarrow & & s_t, a_t, r_{t+1}, s_{t+1} & & \\
\cline{4-4}
\end{array}
$$

Sample experiences from data-set and apply update

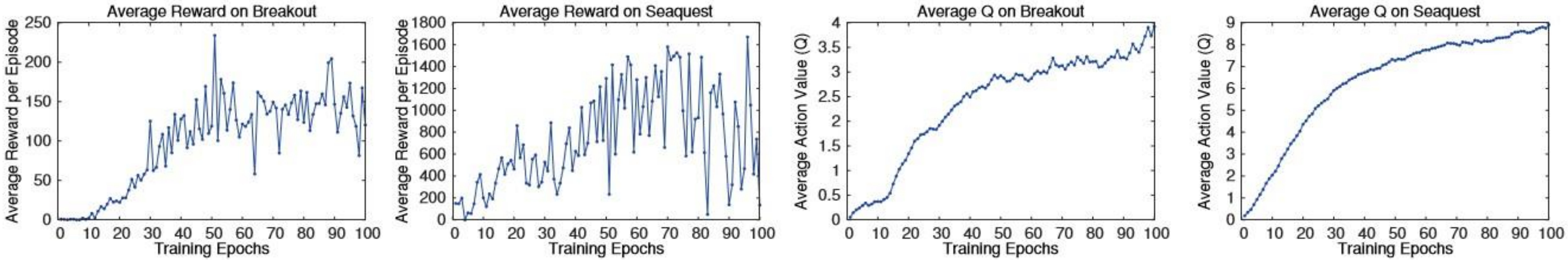


Figure 2: The two plots on the left show average reward per episode on Breakout and Seaquest respectively during training. The statistics were computed by running an ϵ-greedy policy with $\epsilon = 0.05$ for 10000 steps. The two plots on the right show the average maximum predicted action-value of a held out set of states on Breakout and Seaquest respectively. One epoch corresponds to 50000 minibatch weight updates or roughly 30 minutes of training time.

Figure 3: The leftmost plot shows the predicted value function for a 30 frame segment of the game Seaquest. The three screenshots correspond to the frames labeled by A, B, and C respectively.

Q & A

Thank you!

Paper:PlayingAtariwithDeepReinforcementLearning