Reinforce Learning

Deep Q-Learniing (DQN)

소프트웨어 꼰대 강의

노기섭 교수

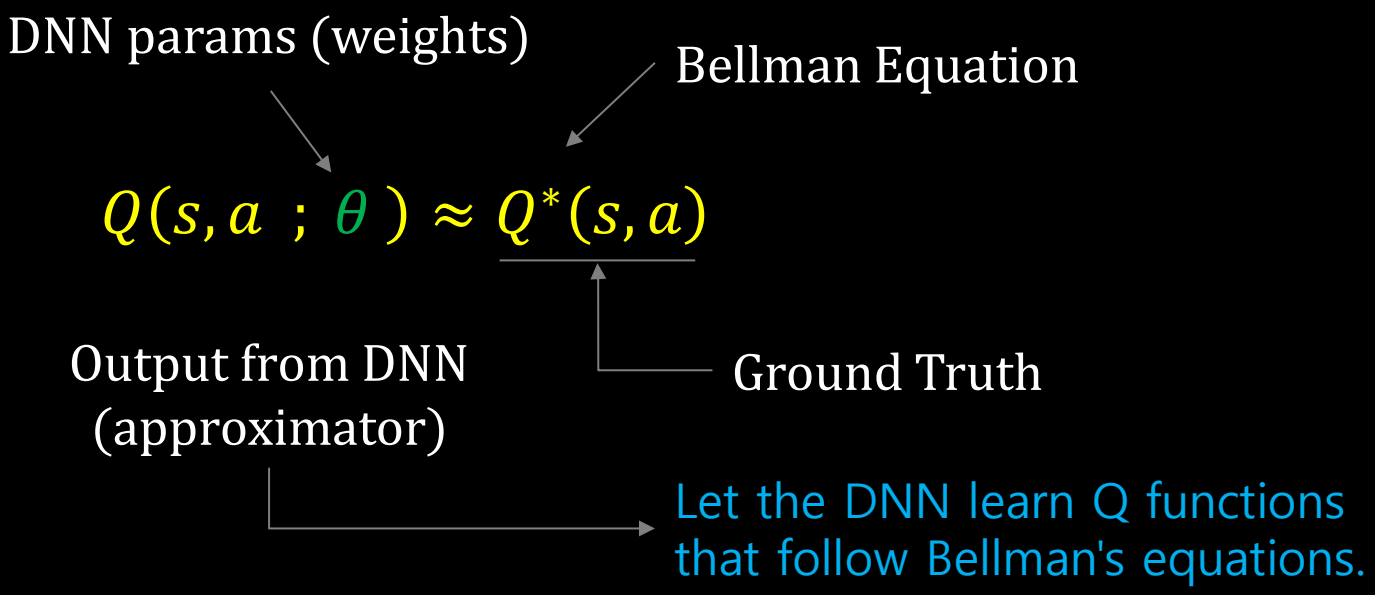
(kafa46@hongik.ac.kr)

Tackle the limitation of Dynamic Programming

Solution

- Create an approximator that simulate $Q(s, a)$
  - ✓ Q-learning

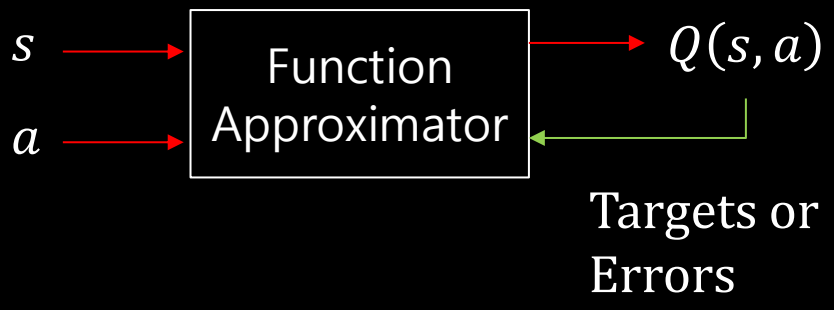
- We can build the approximator using DNN ⇨ Deep Q Network



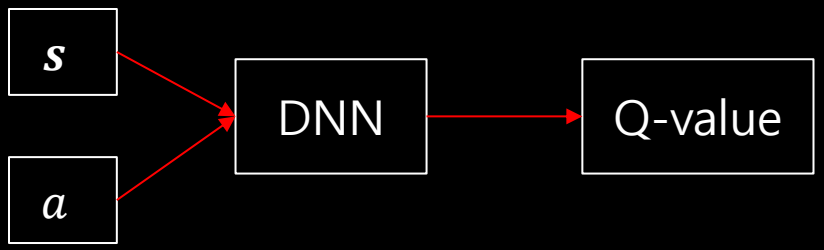
How to Implement DQN?

2 Solutions are possible!

$$Q(s, a\,; \theta) \approx Q^*(s, a)$$

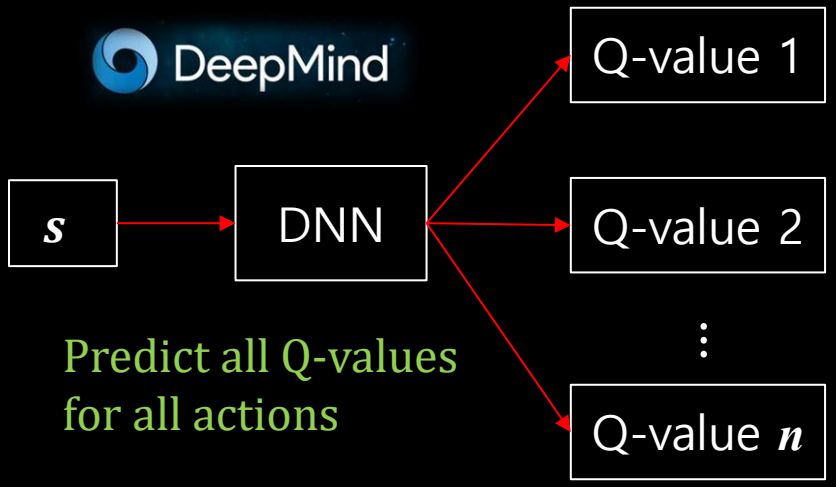


Solution 1



https://arxiv.org/abs/1509.02971

Solution 2



Predict all Q-values for all actions

https://arxiv.org/abs/1312.5602

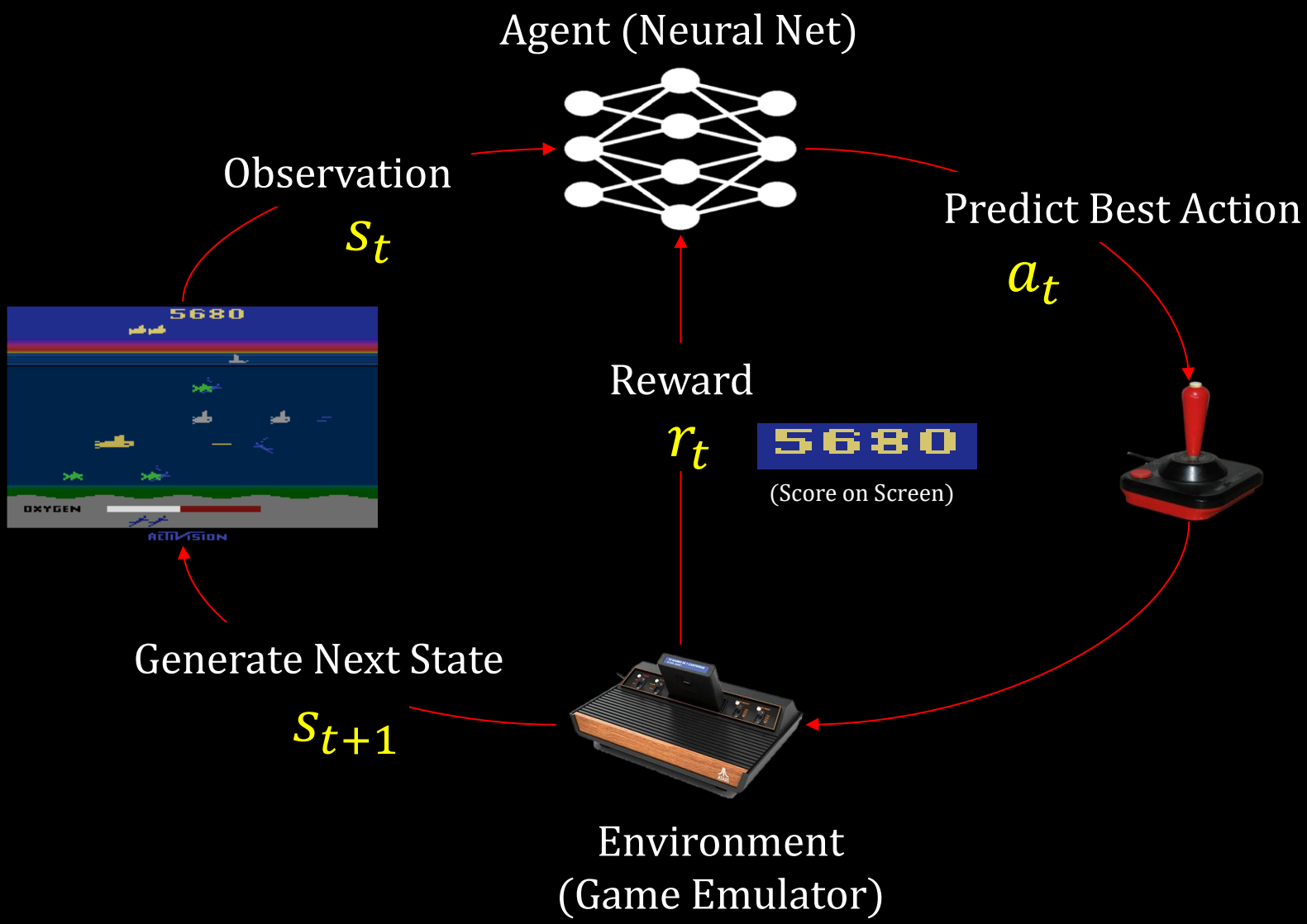
DeepMind DQN: playing Atari games

Atari 2600 Games

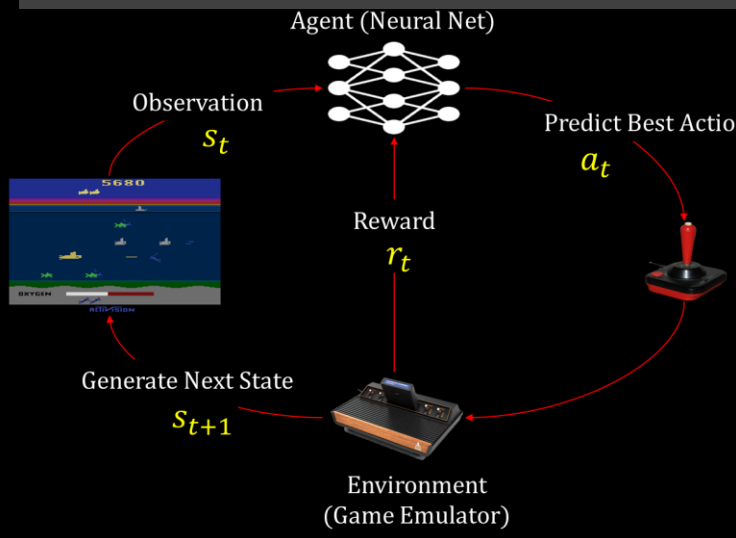
Pong Breakout Space Invaders Seaquest Beam Rider

- Objective: End game with Max Score
- State: Raw pixel distribution of screen
- Action: Game control (left, right, up, down)
- Reward: Score value on screen for each time step

DeepMind DQN: Algorithm Schetch



Detailed DQN Architecture



Choose Max Q-value Action

$Q(s_t, a_1)$ (Action 1)

$Q(s_t, a_2)$ (Action 2)

$Q(s_t, a_3)$ (Action 3)

$Q(s_t, a_4)$ (Action 4)

FC-4 (Q-values)

FC-256

32 4 ×4 Conv, Stride 2

16 8 × 8 Conv, Stride 4

Current state S_t: $84 \times 84 \times 4$ stack of last 4 frames

Hidden Problems - Tricks in DQN (1/2)

Experience Replay Buffer

- Issue: strong correlation between samples

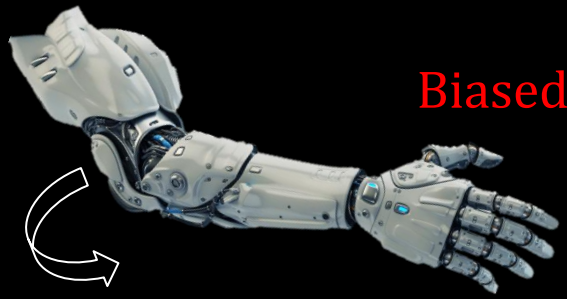
$$s_0 a_1 r_0 s_1 \;\; s_1 a_1 r_1 s_2 \;\; \cdots \;\; s_{t-1} a_{t-1} r_{t-1} s_t$$

sampling

$$(s_0 a_1 r_0 s_1), \;\; (s_1 a_1 r_1 s_2), \;\; (s_2 a_2 r_2 s_3), \;\; \cdots, (s_{t-1} a_{t-1} r_{t-1} s_t)$$

✓ Current params determine the next training samples

✓ Training samples that move the robot arm to the left and then sample can only be seen around the left



Biased sample ➔ Broken i.i.d

Hidden Problems - Tricks in DQN (1/2) i.i.d ?

Independent & Identically Distributed (i.i.d)

Independent (독립)	Identically Distributed (동일분포)
Each data sample does not affect each other. [동전 던지기] 앞면이 나왔다고 다음에 앞면이 나올 확률이 바뀌지 않음	All data came from the same probability distribution All samples are collected from the same sensor, same conditions, same environment, etc.

$$P(X_1, X_2, \cdots, X_n) = \prod_{i=1}^{n} P(X_i)$$

$$X_1 \sim X_2 \sim \cdots \sim X_n$$

$$X_1, X_2, \cdots, X_n \sim \mathcal{D}$$

$$X_i \sim \mathcal{D}$$

$\mu, \sigma, distribution$ are all same!

Why is i.i.d. important in deep learning?

SGD has only one assumption!!!

The gradients calculated from the mini-batch are "unbiased estimators" of the true gradients

	Not Independent	Not Identically Distributed
Cause	• Samples look like each other • Gradient over-biased in certain directions • Increased variance, wobbly orientation	• Domain shift • Federated Learning • Changed Process Environment
Effects	• Oscillation (진동) • Learning rate sensitivity • Unconverged loss	• Different optimal point for each batch • Fail to converge to one minimum

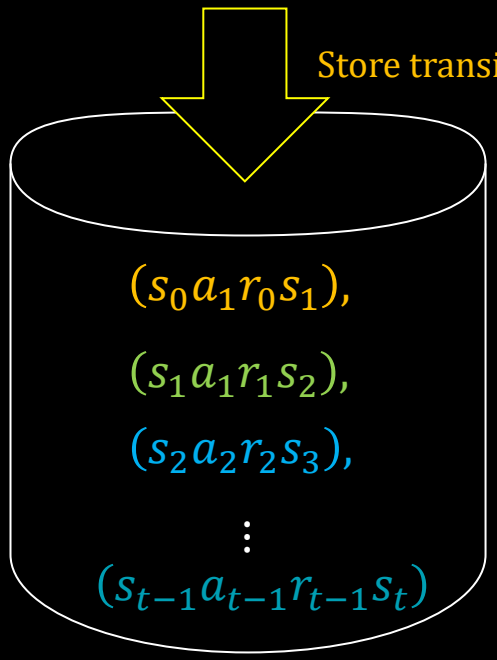
결국에는 중재자 없는 정치 싸움판으로 변질 것임

Hidden Problems - Tricks in DQN (2/2)

Experience Replay Buffer

- Continuously save transition to replay memory while episode play

$Transitions$: s_t, a_t, r_t, s_{t+1}



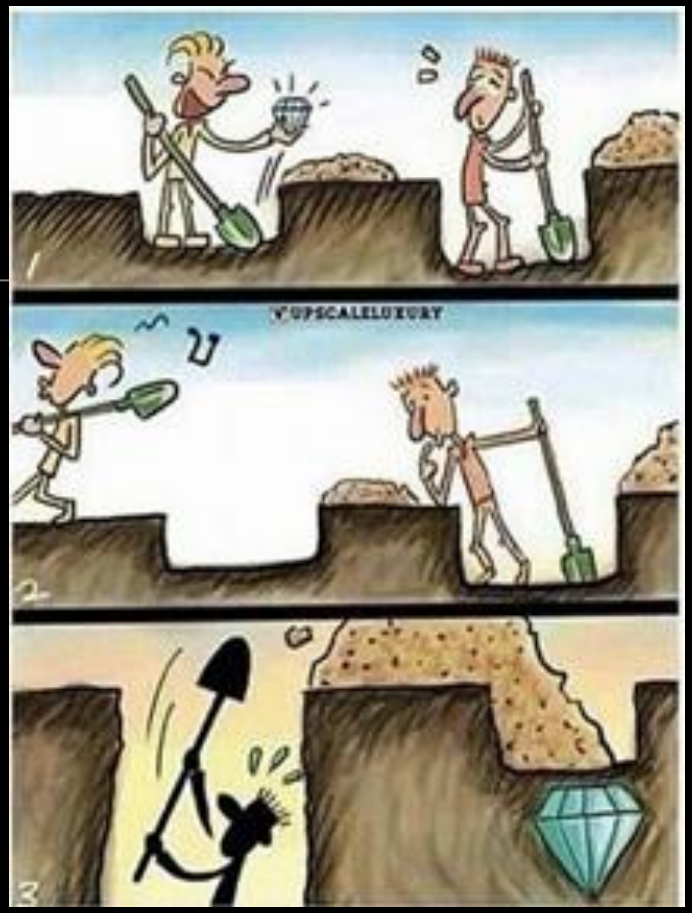
✓ When Replay memory is full enough,

- Make a random minibatch of transition
- Sample it there to learn Q-network

[Ex.]
$(s_7 a_7 r_7 s_8)$,
$(s_{51} a_{51} r_{51} s_{52})$,
$(s_2 a_2 r_2 s_3)$,
⋮

Exploitation vs. Exploration: $\epsilon - greedy$ Policy

Exploitation (활용)	Exploration (탐험)
Perform • optimal action • greedy action	Random action



$\epsilon - greedy$ Policy

$$a_t = \begin{cases} random\ action\ \ with\ prob\ \epsilon \\ \arg\max_a Q(s_t, a)\ with\ prob\ 1-\epsilon \end{cases}$$

- Early: $\epsilon \approx 1.0$ (almost exploration)
- Slowly decrease ϵ (exploration → exploitation)
- End: $\epsilon \approx 0.0$ (almost exploitation)

https://ai-ml-analytics.com/reinforcement-learning-exploration-vs-exploitation-tradeoff/

Fixed Target Network

More ingredient

The calculation of the error contains a target function that varies according to network params.

A dragon chases its own tail ^^

- ✓ Unstable error
- ✓ Not converge

Target (Bellman Eq.)

Prediction

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p Q(s_{t+1}, p) - Q(s_t, a) \right]$$

Ground Truth

DQN Trick: update target every n (for example, 1,000) steps!

DQN Algorithm in pseudo code

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights

for episode=1 to M do

Initialize sequence $s_1 = \{x_1\}$ & preprocessed sequenced $\phi_1 = \phi(s_1)$

for $t = 1$ to T do

With probability ϵ select random action a_t ← exploration

otherwise select $a_t = \max_a Q^*(\phi(s_t), a\,; \theta)$ ← exploitation

Execute action a_t in emulator and observe reward r_t & image x_{t+1}

Set $s_{t+1} = s_t,\ a_t, x_{t+1}$ & preprocess $\phi_{t+1} = \phi(s_{1+1})$

Store transition $(\phi_t,\ a_t,\ r_t,\ \phi_{t+1})$ in D

Training Procedure

Sample random minibatch of transition $(\phi_j,\ a_j,\ r_j,\ \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{for teminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'\,; \theta^-) & \text{for non-teminal } \phi_{j+1} \end{cases}$

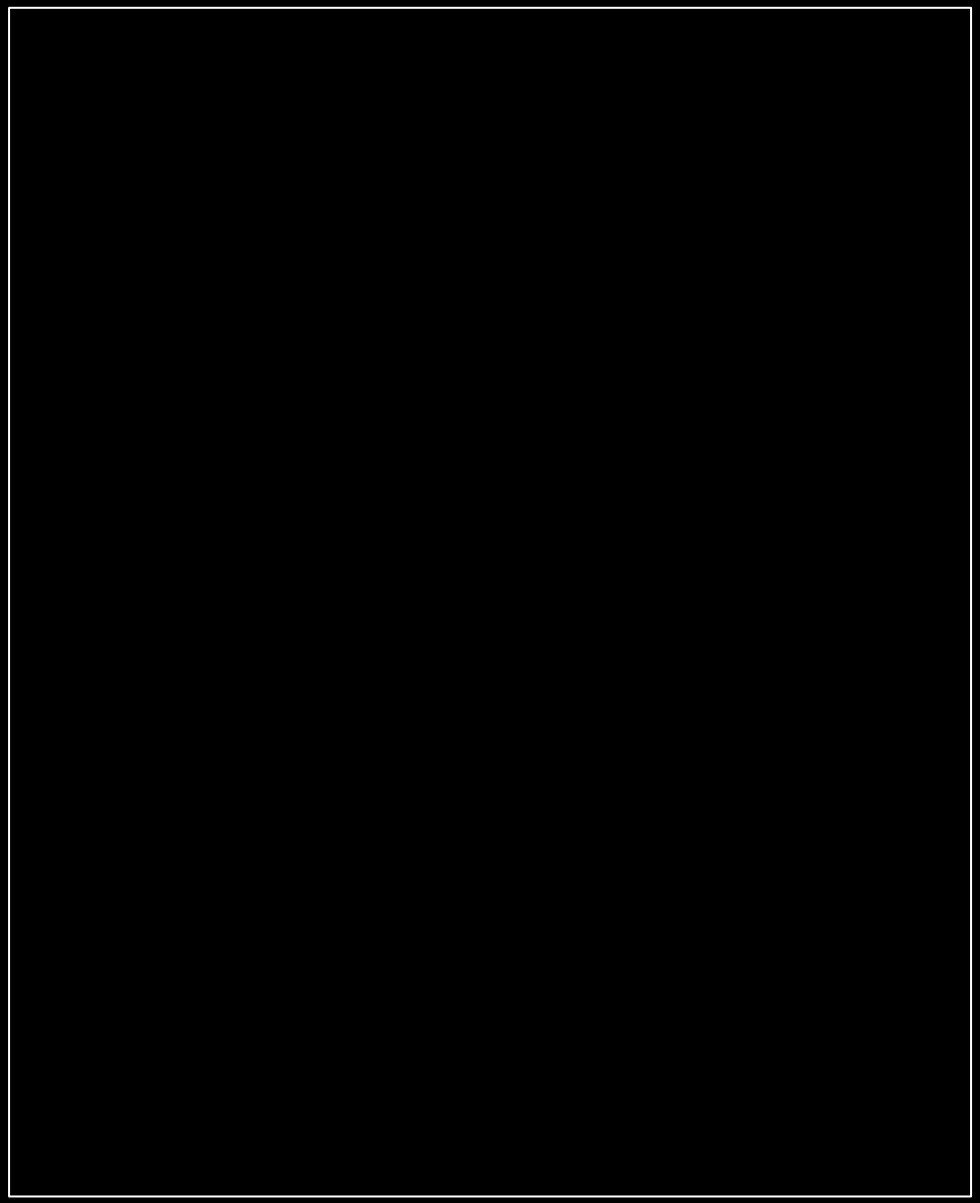
Perform a gradient descent step on $\left(y_i - Q(\phi_j,\ a_j\,; \theta)\right)^2$ according to Eq. 3

Target (Bellman)

Prediction (output) from NN

Google DeepMind DQN - Atari Breakout

Applying
DQN into
Atari Game



영상 출처:
https://youtu.be/V1eYniJ0Rnk?si=WdQARJq09UYM

Summary

- **Q-learning**
  - Idea: Learn a function that approximates $Q(s, a)$ using the Bellman equation.
  - Pros: Highly sample-efficient compared to policy optimization methods due to effective data reuse.
  - Cons: Optimizes the policy indirectly, which can be less stable and less reliable.
  - The learned Q-function is only an approximation and cannot be guaranteed to be perfect.
  - Main challenge: Exploration.

- **DQN (Deep Q-Network)**

- **Next Step: Policy Gradient**
  - Directly solves an optimization problem and converges to a local minimum of the cost function.
  - Requires more samples than DQN.

수고하셨습니다 ..^^..