Machine Learning in Practice
#6: Deep Reinforcement Learning

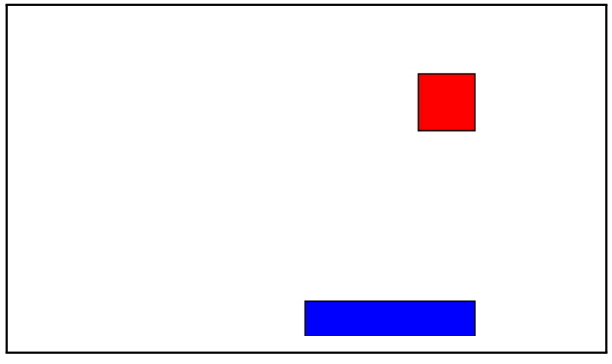
Sang-Hyun Yoon

Summer 2019

Outline

Recall: Q-Learning for Catch (catch_MDP/Q_learning/Q_test.py)

- Environment는 non-random MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$:
  - $\mathcal{S} = \underbrace{\{0, 1, \cdots, 9\}^2}_{\text{과일의 위치}} \times \underbrace{\{0, 1, \cdots, 7\}}_{\text{basket의 위치}}$,
  - $\mathcal{A} = \{\text{RIGHT, LEFT, WAIT}\}$ (basket의 이동 방향)
  - $\mathcal{P}(s, a)$ = state s에서 basket을 방향 a로 움직이고 과일이 한칸 떨어졌을 때의 state
  - $\mathcal{R}(s, a)$ = 과일이 바닥에 있고 basket이 받았으면 1, 놓쳤으면 -1, 과일이 바닥에 있지 않으면 0
  - $\gamma = 0.9$



- $q(s, a)$를 보관하기 위한 table 크기 $= |\mathcal{S}| \cdot |\mathcal{A}| = 1701$

Q-Learning for Non-Trivial Game: Atari Breakout

- $\mathcal{A} = \{\text{RIGHT, LEFT, WAIT}\}$
- $\mathcal{S} = \{\text{consecutive 4 frames}\}$
  - Each frame consists of 210×160 pixels with 128-color
- $|\mathcal{S}| = (128^{210\times160})^4 \approx 1.047 \times 10^{283209}$
- $q(s, a)$를 보관하기 위한 table을 만들 수 없음..
- $q(s, a)$를 CNN으로 approximation할 수 있지 않을까?

Deep Reinforcement Learning

Deep reinforcement learning =

RL with a deep NN as a function approximator for $q(s, a)$

- Supervised learning: approximation할 함수가 미리 주어짐
- Reinforcement learning: $q(s, a)$가 미리 주어지지 않고 강화학습을 하면서 점진적으로 update

- 즉, deep RL은 $q(s, a)$은 기존의 RL(예: Q-Learning)으로 학습하면서 $q(s, a)$값은 CNN으로 학습하는 구조
- 그런데, 고정된 함수를 CNN으로 학습하는 것도 간단하지 않은데, 강화학습으로 계속 변해가는 $q(s, a)$를 CNN 상에서 효율적으로 update하는 것은..

Recall: Bellman Optimality Equations

- $q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') \cdot \left(\max_{a' \in \mathcal{A}} q^*(s',a') \right)$

- $q(s,a) := (1-\alpha) \cdot q(s,a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} q(s',a') \right)$
  $= q(s,a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} q(s',a') - q(s,a) \right)$

Recall: Q-Learning

$q(s,a) :=$ random value for each $s \in \mathcal{S}$, $a \in \mathcal{A}$
$q(s_f,a) := 0$ for each $a \in \mathcal{A}$ and final state s_f
for (each episode)
    $s :=$ start state
    for (each step of the episode)
        $a := \pi(s)$ where $\pi = \epsilon$-greedy(q)
        take action a to get reward r & next state s'
        $q(s,a) := q(s,a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} q(s',a') - q(s,a) \right)$
        $s := s'$

Deep Q-Learning with Experience Replay

- Action-value function을 (C)NN으로 approximate. (C)NN의 weight 값이 θ일때 $q_\theta(s,a)$로 표현
- Target action-value function($r+\gamma\cdot\max_{a'\in\mathcal{A}} q(s',a')$) 역시 NN으로 approximate. Weight 값이 $\overline{\theta}$일때 $\hat{q}_{\overline{\theta}}(s,a)$로 표현

initialize replay memory D to capacity N
$\theta :=$ random value # weights of NN for q
$\overline{\theta} := \theta$ # weights of NN for $\hat{q}$
for (each episode)
    $s_1 :=$ start state
    for (each step t of the episode)
        $a_t := \pi(s_t)$ where $\pi = \epsilon$-greedy(q_θ)
        take action a_t; get reward r_t & next state s_{t+1}
        store (s_t, a_t, r_t, s_{t+1}) in D
        get random batch $\{(s_i, a_i, r_i, s_{i+1}) \mid i \in I\}$ from D
        $y_i := r_i + \gamma\cdot\max_{a'\in\mathcal{A}} \hat{q}_{\overline{\theta}}(s_{i+1}, a')$ for each $i \in I$
        search for θ that minimizes $\sum_{i\in I}(y_i - q_\theta(s_i, a_i))^2$
        $\overline{\theta} := \theta$ for every C steps

Deep Q-Learning with Experience Replay: the Rationale

Use experience replay

Q. (s_t, a_t, r_t, s_{t+1})를 θ의 학습에 이용하지 않고 replay memory에 저장된 (오래된) $\{(s_i, a_i, r_i, s_{i+1}) \mid i \in I\}$를 끄집어 내서 학습에 이용하는 이유는?

A. (s_t, a_t, r_t, s_{t+1})를 순차적으로 바로 학습에 이용하면 이 data 간의 correlation으로 인한 divergence를 막는데 도움이 됨

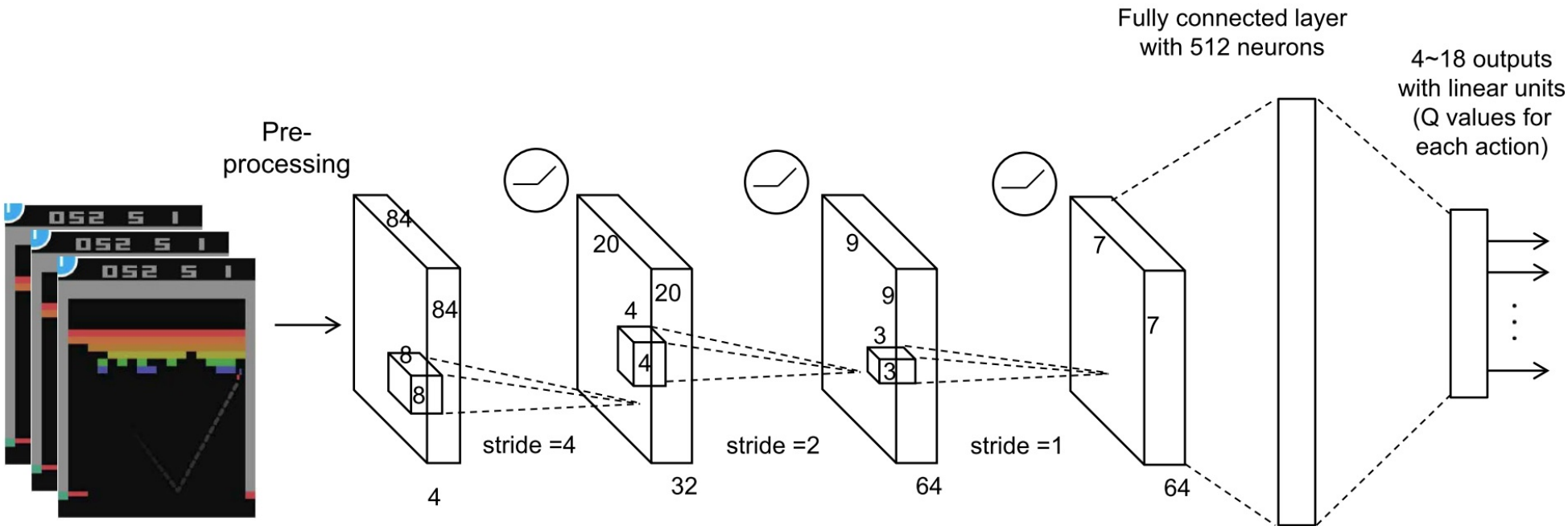
Freeze target Q-network

Q. q_θ 하나만 사용하여 매 step마다 θ를 update하면 될 것 같은데 굳이 $\hat{q}_{\overline{\theta}}$로 분리해서 C step마다 update하는 이유는?

A. Policy oscillation을 막고 q_θ와 target value간의 correlation을 없애는데 도움이 됨

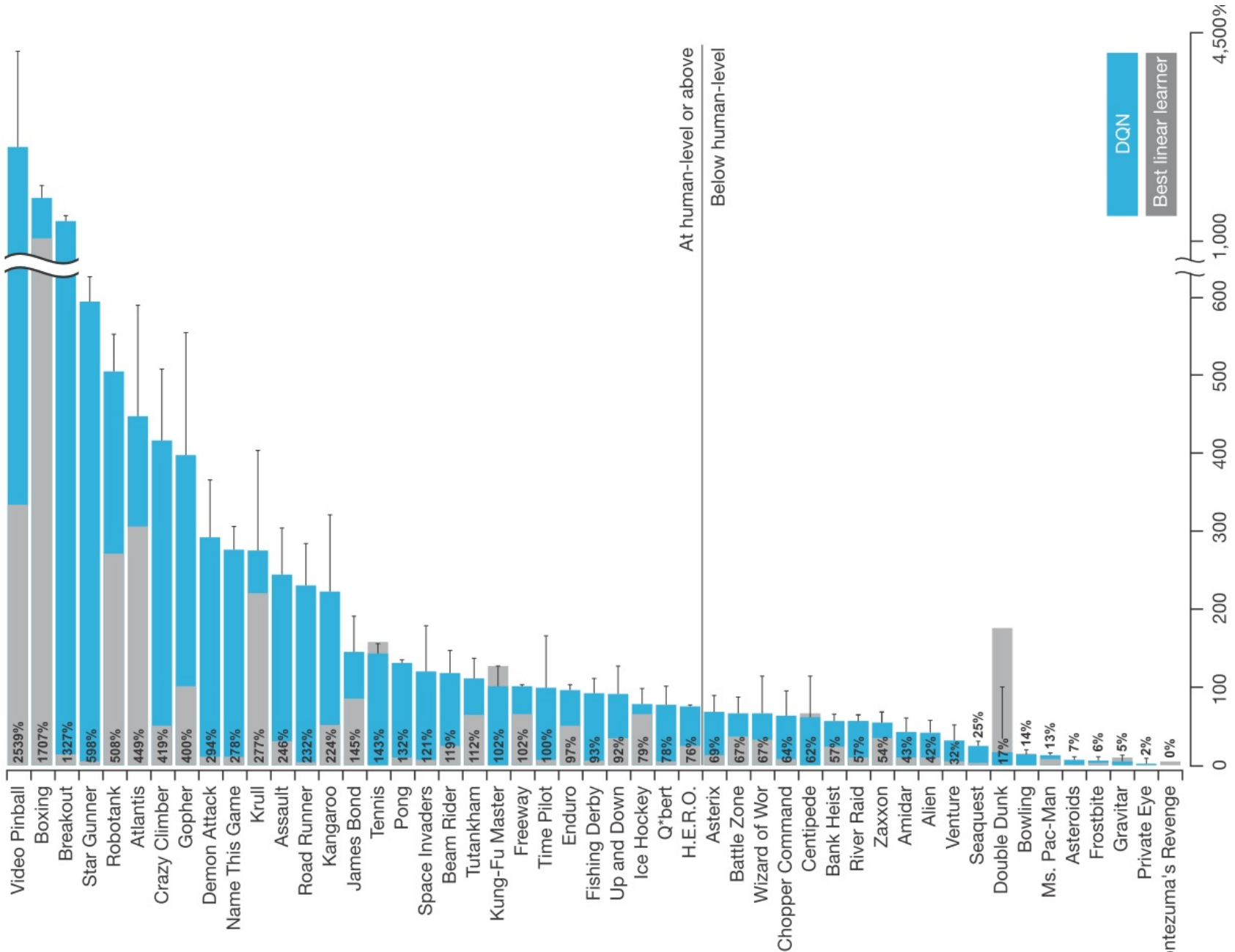
- Policy changes rapidly with slight changes to q values

Deep Q Network (DQN) for Atari Games



- 모든 Atari 게임에 대해 동일한 hyperparameter 사용
- 우선, 128색 210×160 크기를 grayscale 84×84로 줄임
- State는 4개의 연속된 frame들
- Frame skipping을 사용 (4개씩 skip)
- Action은 18가지 (조이스틱의 레버/버튼 조합)

DQN Results in Atari



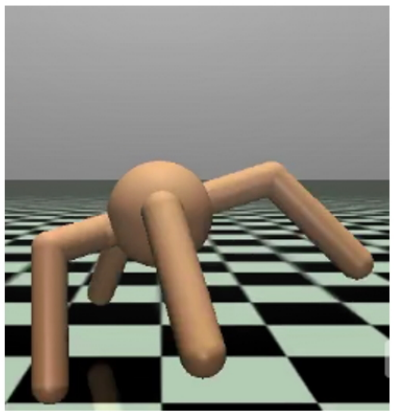
DQN Performance Factors

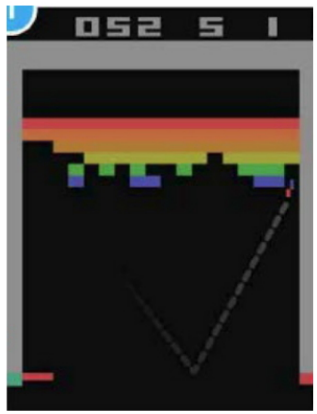
Game	Linear	DQN	DQN with fixed Q	DQN with replay	DQN with replay and fixed Q
Breakout	3	3	10	241	317
Enduro	62	29	141	831	1006
River Raid	2345	1453	2868	4102	7447
Sequest	656	275	1003	823	2894
Space Invaders	301	302	373	826	1089

- Experience replay가 성능에 매우 큰 기여를 함
- Double DQN (actor/critic networks) 등으로 성능을 더욱 높일 수도 있음

Outline

OpenAI gym package



- 강화학습 기법의 연구/개발/평가에 유용한 다양한 environment(MDP)들을 제공
  - 강체 운동 제어, 다관절 운동 제어, Atari 게임 등
- Environment를 만드는 것은 매우 laborous
- gym에서 제공하는 environment를 사용하면 강화학습 기법 그 자체에만 집중할 수 있음

OpenAI gym package: Breakout

```
env = gym.make("Breakout-v0")   # environment for Breakout
while True:
    env.render()        # update screen
    action = ...        # determined by your policy function
    next_state, reward, terminate, info = env.step(action)
    total_rewad += reward
```

- Breakout 게임의 경우 단순한 DQN으로는 학습이 잘 안됨
- Actor/critic network을 사용한 double DQN을 사용