Reinforcement Learning

DQN / PerDQN / Dueling DQN

2021.03.18

최영제

Index

가치 함수와 벨만 방정식

- 상태 가치 함수(state value function)란 특정 상태에 도달 후 앞으로 얻을 수 있는 감가 누적 보상을 의미
- 행동 가치 함수(action value function, Q-function)란 특정 상태-행동을 취했을 때 앞으로 얻을 수 있는 감가 누적 보상을 의미

$$V_\pi(s) = E_\pi[G_t|S_t = s] \qquad Q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a]$$

- 상태 가치 함수와 행동 가치 함수는 서로 변환이 가능하며 해당 성질을 활용하여 벨만 방정식을 도출할 수 있음

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s] \qquad Q_\pi(s,a) = E_\pi[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$$

- 벨만 방정식이란 현재 시점과 다음 시점의 가치 함수 간의 관계식을 말하며 가치 함수를 도출할 수 있는 식
- 벨만 최적 방정식은 최적 가치 함수를 도출할 수 있는 식이며 경험적인 방법을 통해서 풀어가야함(모델을 알 때 : DP / 모델을 모를 때 : MC, TD)
- 몬테 카를로 학습과 시간차 학습은 target을 무엇으로 두느냐에 따라서 차이가 존재함

MC와 TD

- 몬테 카를로 학습(MC-learning)은 감가 누적 보상(G_t)을 target으로 삼아 학습을 함

$$V(S_t) = V(S_t) + \alpha(G_t - V(S_t))$$

- 몬테 카를로 학습은 G_t를 사용하기 때문에 low bias / high variance 특징이 존재하며 episode가 끝나는 환경에서만 사용이 가능함(episodic)
- 에피소드가 끝날 때만 사용이 가능한 MC 대신에 시간차 학습(TD-learning)은 다음 시점의 가치 함수를 target으로 삼음

$$V(S_t) = V(S_t) + \alpha(\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD-target}} - V(S_t))$$

TD-error

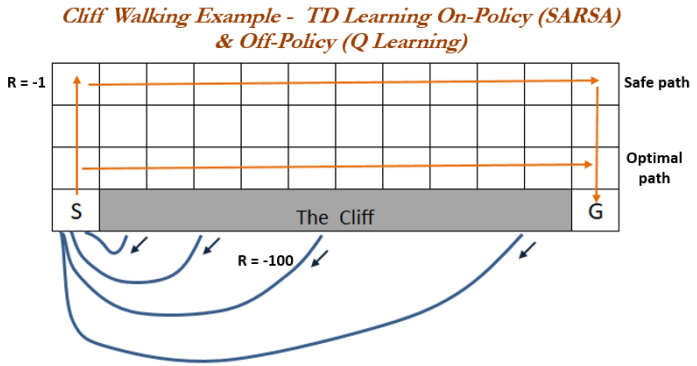
- 시간차 학습은 high bias / low variance 특징이 존재하며 episodic 환경이 아니더라도 활용이 가능함
- 이후 대부분의 알고리즘은 TD-learning을 기반으로 동작함

Q-learning

- Q-learning은 TD-learning 알고리즘의 하나로 행동 가치 함수를 학습하는 알고리즘, 갱신 공식은 다음과 같음

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{\acute{a}} Q(S_{t+1}, \acute{a}) - Q(S_t, A_t)]$$

- 앞의 TD-learning에서 다른 점은 행동 가치 함수를 학습한다는 것과 target으로 다음 시점의 maximum Q-value를 사용한다는 것
- Q-learning은 행동 정책(behavior policy)은 e-greedy를, 갱신 정책(target policy)는 greedy policy를 사용하는 off-policy 알고리즘임



- 다만 Q-learning은 table 기반의 강화학습이기 때문에 state-action space가 크거나 무한한 공간에서는 적용할 수 없음
- 따라서 미분 가능한 함수(e.g., linear combination of features, neural net)를 이용하여 Q-value를 approximation하는 방법이 제안되었음

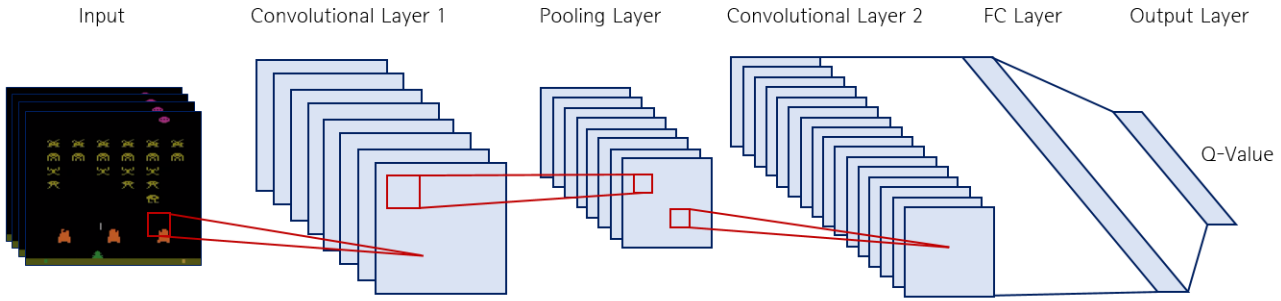
2 SIL Playing Atari with Deep Reinforcement Learning (DQN)

Introduction

- DQN은 2013년 DeepMind에서 발표한 “Playing Atari with Deep Reinforcement Learning” 제목의 논문에서 제안된 방법
- 60HZ의 210x160 RGB video의 state space를 갖는 Atari game 환경은 기존의 table 기반의 강화학습으로는 불가능한 문제였음

Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

- 따라서 특정 함수를 이용하여 Q-value를 근사하는 방법이 등장하였으며 해당 함수의 용량만 차지하기 때문에 효율적인 방법임



$$Q(S_t, A_t) \approx Q(S_t, A_t; \theta)$$

- DQN 이전에 neural network를 활용하여 Q-value를 근사하는 Q-network 방법론이 존재하였음

2 SIL Playing Atari with Deep Reinforcement Learning (DQN)

Introduction

- Q-network 알고리즘은 다음과 같음

Initialize action-value function Q with random weights
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  For t = 1, T **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

$$\text{Set } y_t = \begin{cases} r_t & \text{for terminal } \phi_{t+1} \\ r_t + \gamma \max_{a'} Q(\phi_{t+1}, a'; \theta)) & \text{for non-terminal } \phi_{t+1} \end{cases}$$

    Perform a gradient descent step on $(y_t - Q(\phi_t, a_t; \theta))^2$
  end for
end for

---

2 SIL Playing Atari with Deep Reinforcement Learning (DQN)

Introduction

- Q-network 알고리즘은 다음과 같은 단점이 존재하였음

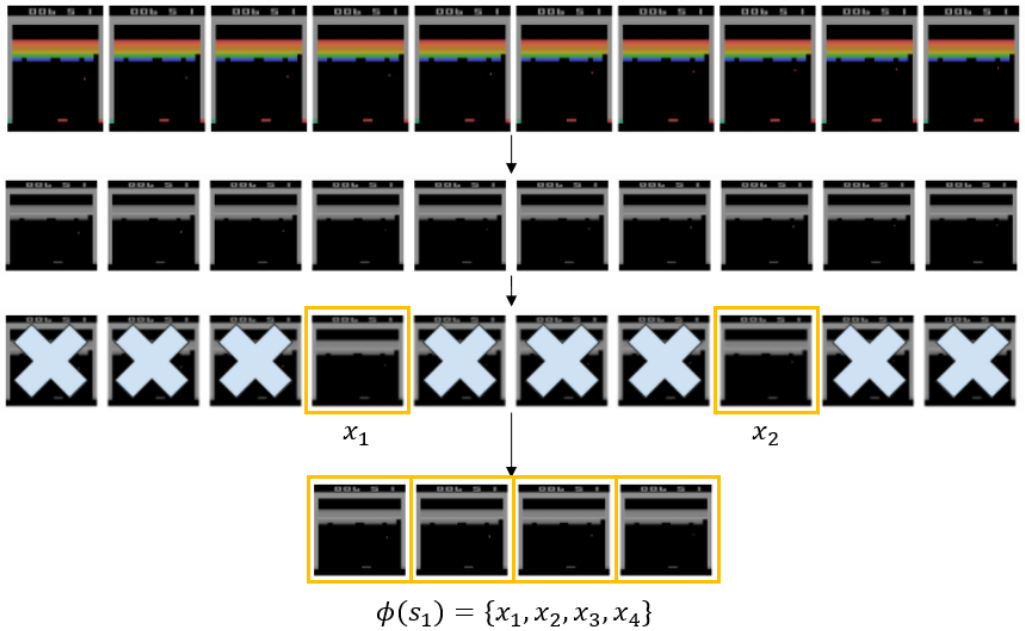
  1) Small volum of training data
     - 일반적으로 neural network는 학습 데이터가 많은 supervised/unsupervised learning에 효율적인 방법임
     - 강화학습은 한번 갱신에 사용한 transition은 다시 사용하지 않기 때문에 neural network를 학습시키기에 충분한 데이터를 확보하기 힘듦
  2) High correlation between samples
     - 또한 강화학습의 학습 데이터인 transition은 correlation이 굉장히 높음(직전 transition에 종속되기 때문)
     - 따라서 global optimum에 도달하지 못하고 local minimum에 빠질 위험이 발생
  3) Non-stationary target problem
     - Q-network의 target 값인 $r_t + \gamma \max_{a'} Q(\phi_{t+1}, a'; \theta))$ 값 또한 network의 추정값임, 즉 동일한 network로 target과 predict Q-value를 계산함
     - 따라서 Q-network의 갱신 주기(=batch size)에 따라서 target 값이 변하게 되어 안정적인 학습이 어려움

- 이는 Q-network 알고리즘의 한계가 아닌 강화학습에서 nueral network 기반의 approach의 한계로 받아들여지고 있었음
- Deepmind에서 제시한 DQN을 통해서 위와 같은 문제를 해결하여 강화학습이 각광을 받게 됨(Nature에도 등재..)

Playing Atari with Deep Reinforcement Learning (DQN)

Proposed methods : data preprocessing

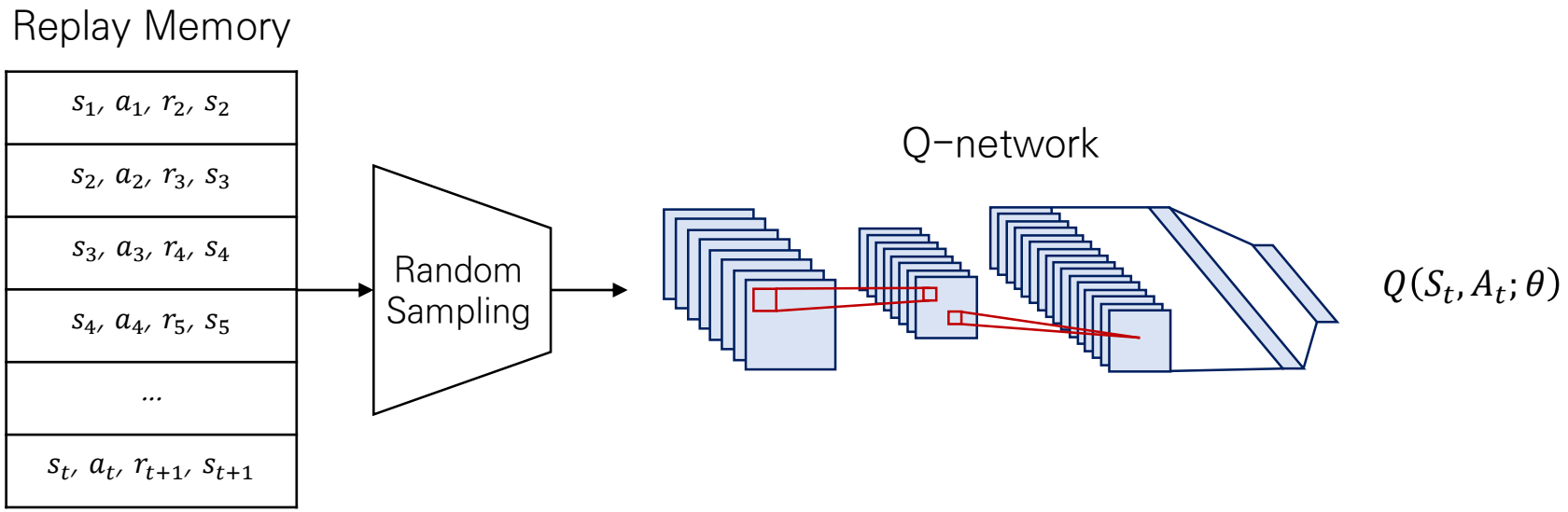
- DQN은 Atari game 스냅샷의 sequence를 state로 사용함. 단일 스냅샷만으로는 상황 정보를 제대로 반영하지 못해 history 정보를 제공하기 위함
- 또한 저자들은 210x160x3의 고해상도 스냅샷을 84x84x1의 사이즈로 변경 후 연속된 스냅샷 중 K번째 위치한 이미지(x_t)들만을 사용
  - 연속된 스냅샷을 그대로 사용할 경우 입력 데이터 간 correlation을 높이기 때문



2 SIL Playing Atari with Deep Reinforcement Learning (DQN)

Proposed method

- DQN은 Q-network의 단점 중 small volum of training data/high correlation between samples 문제를 해결하기 위해 replay memory를 도입함
- 이를 experience replay라고 부르며 에이전트가 매 time-step마다 획득한 <state, action, reward, next state> 튜플을 replay memory에 저장함
- Replay memory는 first in first out 형태로 기록되며 memory에 저장된 샘플 중 n개를 임의로 추출하여 CNN을 학습함
- Experience replay를 통해 transition을 지속적으로 사용할 수 있고 샘플을 추출하여 학습하기에 high correlation 문제도 해결 가능함



2 SIL Playing Atari with Deep Reinforcement Learning (DQN)

Proposed method

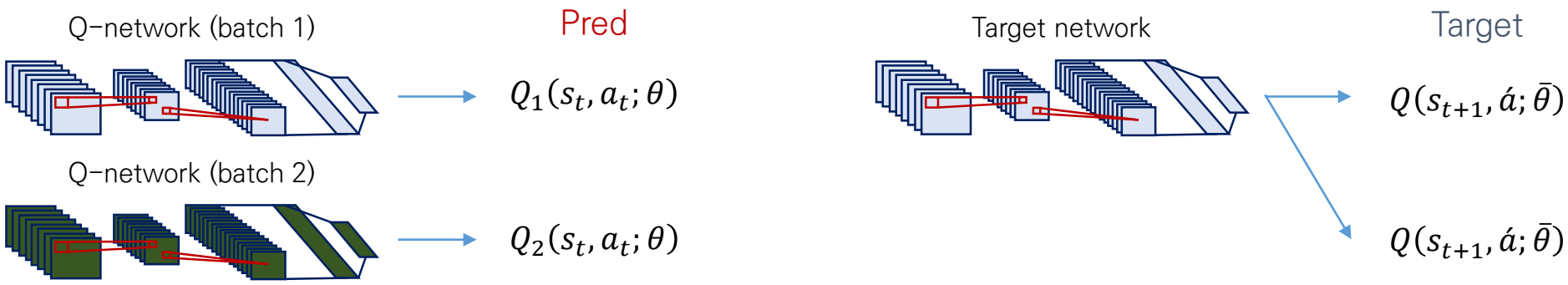
- Non-stationary target problem은 Q-target을 추정하는 network와 Q-value를 추정하는 network를 분리하여 해결하였음
- Q-target을 추정하는 network를 target network($\bar{\theta}$)라고 부르며 일정 주기마다 Q-network(θ)의 가중치를 복사하여 사용함

$$\min_{\theta} \sum_{t=0}^{T} [r + \gamma \max_{á} Q(s_{t+1}, á; \theta) - Q(s_t, a_t; \theta)]^2 \quad \Rightarrow \quad \min_{\theta} \sum_{t=0}^{T} [r + \gamma \max_{á} Q(s_{t+1}, á; \bar{\theta}) - Q(s_t, a_t; \theta)]^2$$

독립된 가중치

Target Pred

- 즉 Q-target을 추정하는 network가 별도로 존재하기 때문에 Q-network를 갱신하여도 target 값은 변하지 않음



2 SIL Playing Atari with Deep Reinforcement Learning (DQN)

Proposed method

- DQN 알고리즘은 다음과 같음

Initialize action-value function Q with random weights
For episode = 1, M do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  For t = 1, T do
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $S_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\boldsymbol{\phi_t, a_t, r_t, \phi_{t+1}})$ $\boldsymbol{in\ D}$
    Sample random minibatch of trainsitions $(\boldsymbol{\phi_t, a_t, r_t, \phi_{t+1}})$ $\boldsymbol{from\ D}$

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

    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameter $\boldsymbol{\theta}$
    Every C steps reset $\bar{\boldsymbol{\theta}} = \boldsymbol{\theta}$
  end for
end for

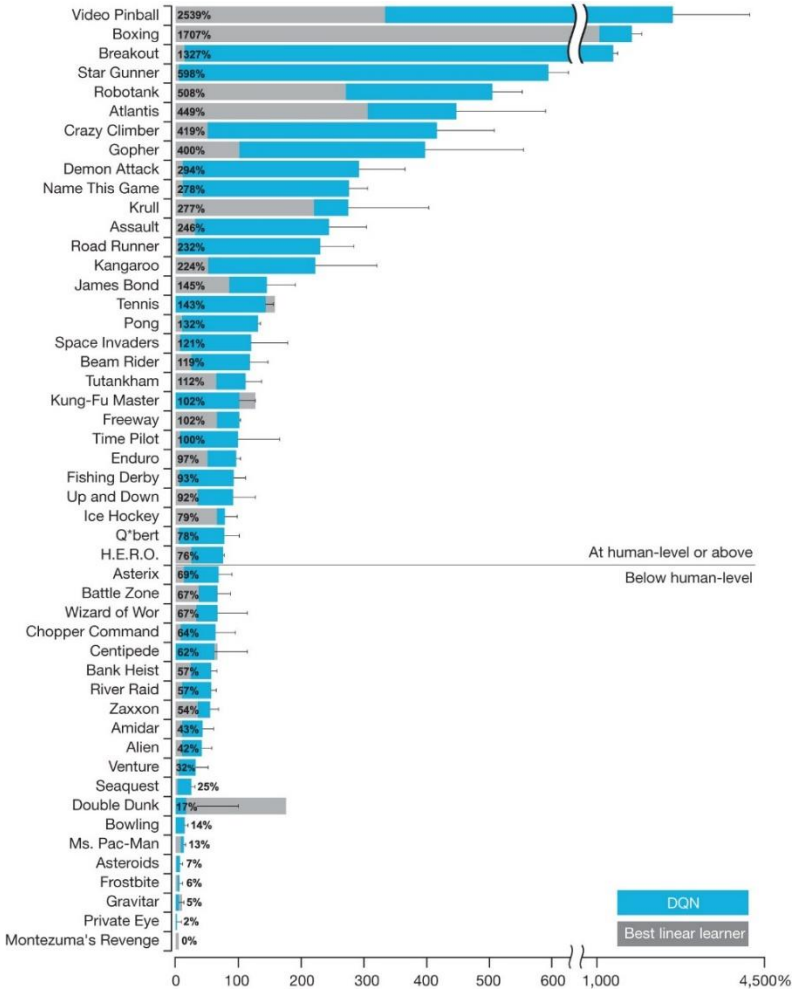
---

2 SIL Playing Atari with Deep Reinforcement Learning (DQN)

Results

- DQN의 ablation study 및 성능은 다음과 같음

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99



Prioritized Experience Replay

PerDQN

- Replay memory에는 학습에 효율적인 transition과 그렇지 않은 transition이 혼재해 있으나 DQN은 이에 상관없이 n개를 추출하여 학습함
- PerDQN은 이를 개선하여 replay memory에서 좋은 transition에 우선 순위(priority, p)를 부여하여 추출하여 학습하는 알고리즘임
- Priority는 TD-error(δ)의 절대값으로 정해지며 추출 확률(sampling probability, P)은 priority를 토대로 구해짐

$$\delta_j = R_j + \gamma \underset{a}{max} Q(S_j, a) - Q(S_{j-1}, A_{j-1})$$

$$p_j \leftarrow |\delta_j| + \varepsilon$$

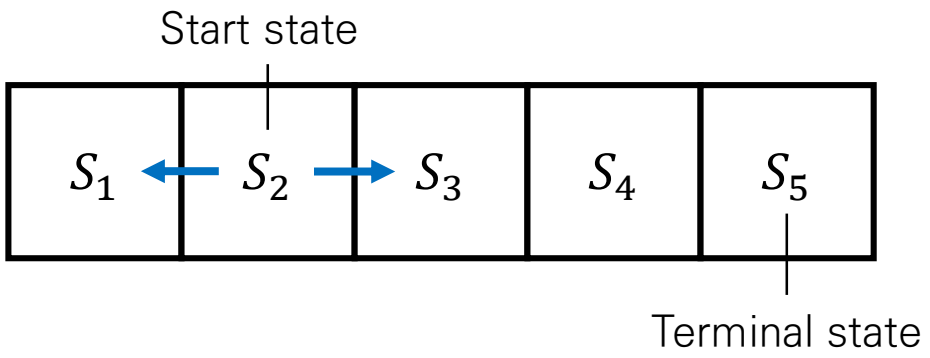
$$P(j) = \frac{p_j^\alpha}{\sum_k p_k^\alpha}$$

- Priority의 ε은 매우 작은 상수값으로 해당 샘플의 priority가 0이 되어 아예 뽑히지 않는 경우를 방지하기 위함
- TD-error가 큰 transition에 보다 높은 확률을 두어 학습하기 때문에 갱신해야할 양이 많은 샘플 위주로 학습을 하게 됨

Prioritized Experience Replay

PerDQN : simple example

- 다음과 같은 환경과 replay memory가 존재할 때 3개의 transition을 뽑아 DQN과 PerDQN 갱신을 비교하고자 함
- 해당 환경은 terminal state에 도달 시 +1의 reward를 받고 에피소드가 종료되며 step size는 1, discount factor는 0.9로 설정
- 초기 state value function은 $V(S) = (0,0,0,0,0)$으로 초기화



Replay memory

T_1	—	$< S_2, L, 0, S_1 >$
T_2	—	$< S_1, R, 0, S_2 >$
T_3	—	$< S_2, L, 0, S_1 >$
T_4	—	$< S_1, R, 0, S_2 >$
T_5	—	$< S_2, R, 0, S_3 >$
T_6	—	$< S_3, R, 0, S_4 >$
T_7	—	$< S_4, R, 1, S_5 >$

Prioritized Experience Replay

PerDQN : simple example

- 우선 DQN의 경우를 살펴보면 replay memory로 부터 랜덤하게 3개의 샘플을 추출하여 학습에 사용함
- 다음은 3개의 transition T_2, T_1, T_5이 뽑혔다고 가정하고 갱신을 진행하면 다음과 같음

Replay Memory

T_1	—	$< S_2, L, 0, S_1 >$
T_2	—	$< S_1, R, 0, S_2 >$
T_3	—	$< S_2, L, 0, S_1 >$
T_4	—	$< S_1, R, 0, S_2 >$
T_5	—	$< S_2, R, 0, S_3 >$
T_6	—	$< S_3, R, 0, S_4 >$
T_7	—	$< S_4, R, 1, S_5 >$

TD 갱신 공식

$$V(S_t) = V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- T_2에 의한 갱신 $V(S_2) = 0 + 1 \times (0 + 0.9 \times 0 - 0)$
  $= 0$
- T_1에 의한 갱신 $V(S_1) = 0 + 1 \times (0 + 0.9 \times 0 - 0)$
  $= 0$
- T_5에 의한 갱신 $V(S_2) = 0 + 1 \times (0 + 0.9 \times 0 - 0)$
  $= 0$
- 3번의 갱신 후 State Value Function

$$V(S) = (0,0,0,0,0)$$

Prioritized Experience Replay

PerDQN : simple example

- PerDQN을 위해서 각 transition 별 sampling probability를 구하면 다음과 같음(ε=0.01)

$$V(S) = (0,0,0,0,0)$$

$$\delta_7 = 1 + 0.9 \times 0 - 0$$
$$= 1$$

Replay Memory	TD-Error (δ)	Priority ($|\delta| + \varepsilon$)	Sampling Probability
T_1 — $< S_2, L, 0, S_1 >$	$\delta_1 = 0$	$p_1 = 0.01$	$P(1) = 0.0093$
T_2 — $< S_1, R, 0, S_2 >$	$\delta_2 = 0$	$p_2 = 0.01$	$P(2) = 0.0093$
T_3 — $< S_2, L, 0, S_1 >$	$\delta_3 = 0$	$p_3 = 0.01$	$P(3) = 0.0093$
T_4 — $< S_1, R, 0, S_2 >$	$\delta_4 = 0$	$p_4 = 0.01$	$P(4) = 0.0093$
T_5 — $< S_2, R, 0, S_3 >$	$\delta_5 = 0$	$p_5 = 0.01$	$P(5) = 0.0093$
T_6 — $< S_3, R, 0, S_4 >$	$\delta_6 = 0$	$p_6 = 0.01$	$P(6) = 0.0093$
T_7 — $< S_4, R, 1, S_5 >$	$\delta_7 = 1$	$p_7 = 1.01$	$P(7) = 0.9439$

- 위 확률을 따라 T_7이 뽑혔다고 가정 후 첫 번째 갱신을 진행하면 다음과 같음

$$V(S_4) = 0 + 1 \times (1 + 0.9 \times 0 - 0)$$
$$= 1$$

$$V(S) = (0,0,0,1,0)$$

Prioritized Experience Replay

PerDQN : simple example

- 첫 번째 갱신이 끝나고 각 transition 별 sampling probability를 다시 구하면 다음과 같음

$$V(S) = (0,0,0,1,0)$$

$$\delta_6 = 0 + 0.9 \times 1 - 0$$
$$= 0.9$$

Replay Memory	TD-Error (δ)	Priority ($|\delta| + \varepsilon$)	Sampling Probability
T_1 — $< S_2, L, 0, S_1 >$	$\delta_1 = 0$	$p_1 = 0.01$	$P(1) = 0.0103$
T_2 — $< S_1, R, 0, S_2 >$	$\delta_2 = 0$	$p_2 = 0.01$	$P(2) = 0.0103$
T_3 — $< S_2, L, 0, S_1 >$	$\delta_3 = 0$	$p_3 = 0.01$	$P(3) = 0.0103$
T_4 — $< S_1, R, 0, S_2 >$	$\delta_4 = 0$	$p_4 = 0.01$	$P(4) = 0.0103$
T_5 — $< S_2, R, 0, S_3 >$	$\delta_5 = 0$	$p_5 = 0.01$	$P(5) = 0.0103$
T_6 — $< S_3, R, 0, S_4 >$	$\delta_6 = 0.9$	$p_6 = 0.91$	$P(6) = 0.9381$
T_7 — $< S_4, R, 1, S_5 >$	$\delta_7 = 0$	$p_7 = 0.01$	$P(7) = 0.0103$

- 위 확률을 따라 T_6이 뽑혔다고 가정 후 두 번째 갱신을 진행하면 다음과 같음

$$V(S_3) = 0 + 1 \times (0 + 0.9 \times 1 - 0)$$
$$= 0.9$$

$$V(S) = (0,0,0.9,1,0)$$

Prioritized Experience Replay

PerDQN : simple example

- 두 번째 갱신이 끝나고 각 transition 별 sampling probability를 다시 구하면 다음과 같음

$$V(S) = (0,0,0.9,1,0)$$

$$\delta_5 = 0 + 0.9 \times 0.9 - 0$$
$$= 0.81$$

Replay Memory	TD-Error (δ)	Priority ($|\delta| + \varepsilon$)	Sampling Probability
T_1 — $< S_2, L, 0, S_1 >$	$\delta_1 = 0$	$p_1 = 0.01$	$P(1) = 0.0114$
T_2 — $< S_1, R, 0, S_2 >$	$\delta_2 = 0$	$p_2 = 0.01$	$P(2) = 0.0114$
T_3 — $< S_2, L, 0, S_1 >$	$\delta_3 = 0$	$p_3 = 0.01$	$P(3) = 0.0114$
T_4 — $< S_1, R, 0, S_2 >$	$\delta_4 = 0$	$p_4 = 0.01$	$P(4) = 0.0114$
T_5 — $< S_2, R, 0, S_3 >$	$\delta_5 = 0.81$	$p_5 = 0.82$	$P(5) = 0.9318$
T_6 — $< S_3, R, 0, S_4 >$	$\delta_6 = 0$	$p_6 = 0.01$	$P(6) = 0.0114$
T_7 — $< S_4, R, 1, S_5 >$	$\delta_7 = 0$	$p_7 = 0.01$	$P(7) = 0.0114$

- 위 확률을 따라 T_5이 뽑혔다고 가정 후 세 번째 갱신을 진행하면 다음과 같음

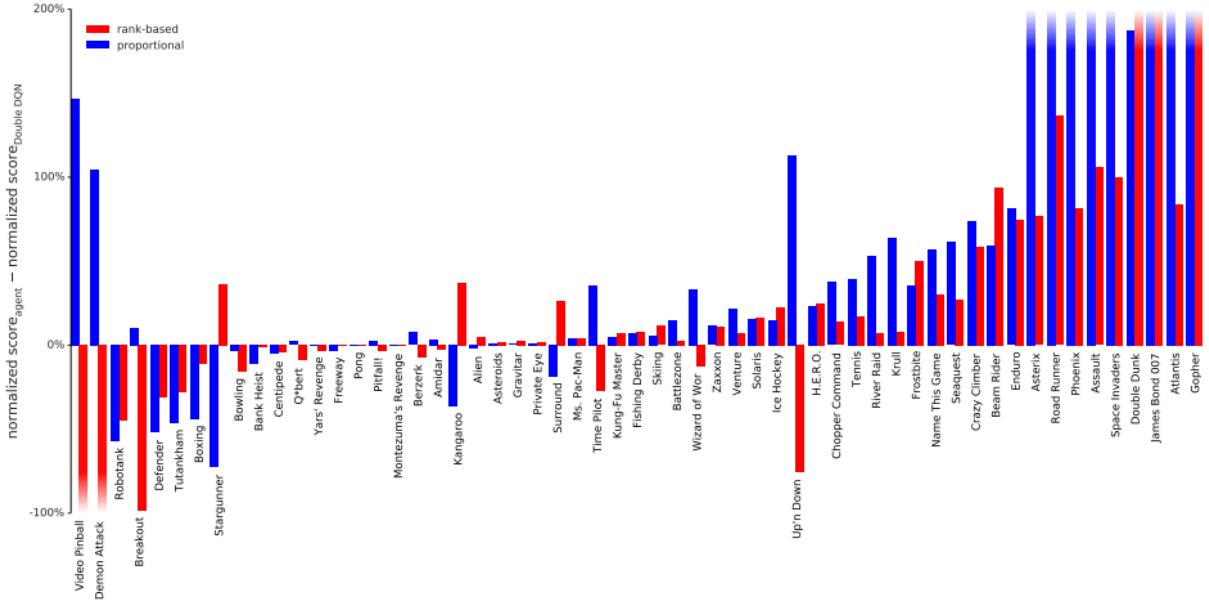
$$V(S_3) = 0 + 1 \times (0 + 0.81 \times 1 - 0)$$
$$= 0.81$$

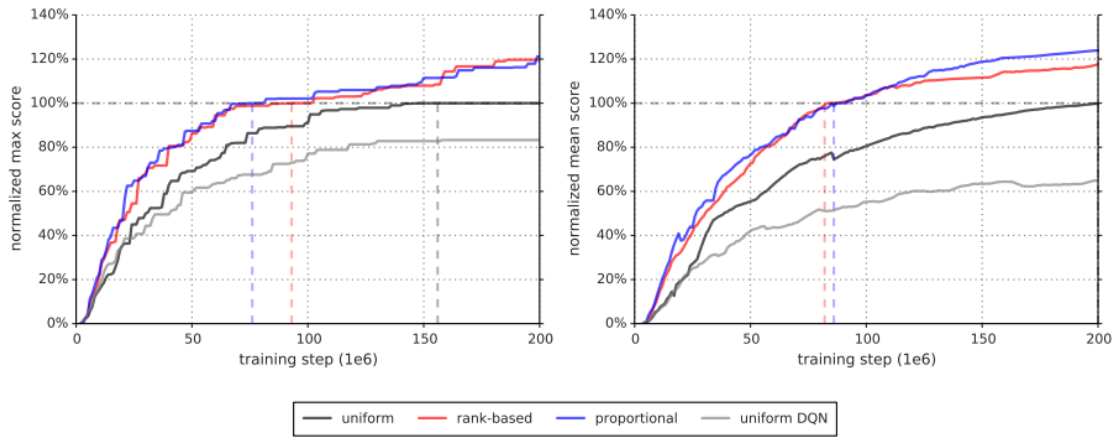
$$V(S) = (0,0.81,0.9,1,0)$$

Prioritized Experience Replay

Results

- 즉 PerDQN은 중요도가 높은 sample을 우선적으로 학습하기 때문에 보다 효율적인 방법임
- 실험 결과 PER을 사용하면 성능 향상과 더불어 학습 속도의 향상도 존재함

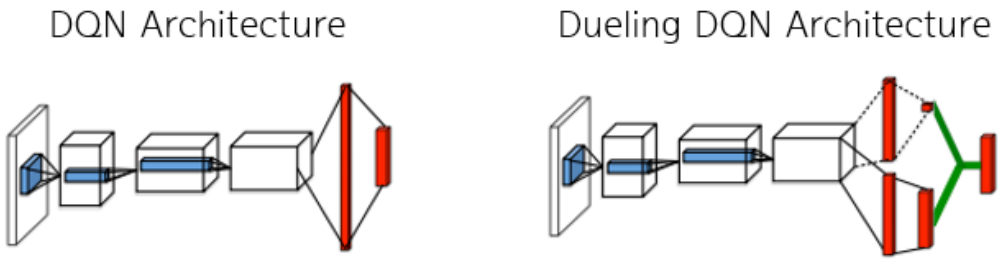




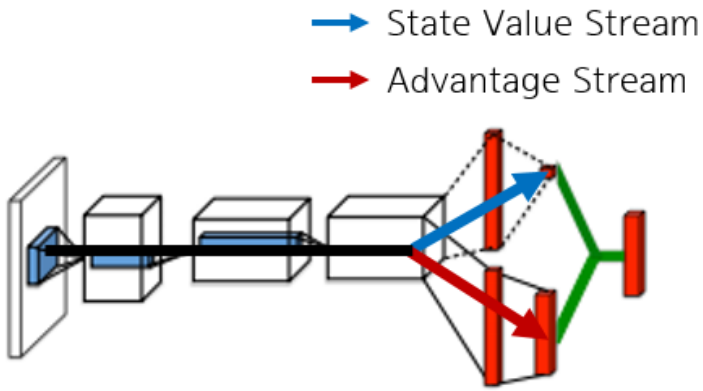
Dueling Network Architectures for Deep Reinforcement Learning

Introduction

- DQN을 개선한 또 다른 모델로 Dueling DQN이 존재하며 구조는 다음과 같음



- 즉 Q-value를 추정할 때 하나의 layer에서 구하는가, 두개로 분리하여 합치는가의 차이가 존재함
- CNN으로 input state의 feature를 추출하는 것은 동일하나 Dueling DQN은 state value와 advantage를 동시에 추정 후 이를 합쳐서 Q-value를 추정함



Dueling Network Architectures for Deep Reinforcement Learning

Proposed method

- Advantage란 action value function과 state value function의 차이로 정의되며 notation은 다음과 같음

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

- Advantage의 목적은 variance를 줄이는 것에 있으며 보다 자세한 사항은 policy gradient > actor-critic > advantage actor-critic에서 자세히 다룸
- Dueling DQN의 최종 아웃풋인 Q-value는 다음과 같이 state value (scalar)와 advantage function (action space)의 합으로 표기함

$$\mathrm{Q}(\mathrm{s},\mathrm{a};\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha)$$

- 이 때 scalar인 state value를 advantage vector만큼 복사하여 크기를 맞춘 후 더해줌
- 즉 Dueling DQN의 핵심 아이디어는 state value와 advantage를 먼저 추정 후 이들을 합산하여 Q-value를 추정하는 것임

Dueling Network Architectures for Deep Reinforcement Learning

Proposed method

- 다만 위 식을 그대로 사용할 경우 성능이 좋지 않으며 논문은 다음과 같은 사항을 경고
  1) 도출된 Q-value가 state value와 advantage 중 어디서 유도된 값인지 알 수 없음
  2) State value와 advantage function 또한 추정값이기에 이를 토대로 구해지는 Q-value를 신뢰하기에 어려움이 있음
- 첫 번째 문제를 해결하기 위해 저자들은 advantage의 max 값을 감하도록 식을 변경함

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + [A(s,a;\theta,\alpha) - \max_{a\in\mathcal{A}} A(s,\acute{a};\theta,\alpha)]$$

- 즉 에이전트가 maximum Q-value를 갖는 행동을 택했을 때의 값이 곧 state value와 동일해지도록 식을 변경
- 두 번째 문제를 해결하기 위해서 저자들은 advantage의 평균을 감하여 advantage의 평균값이 0이 되도록 식을 변경함

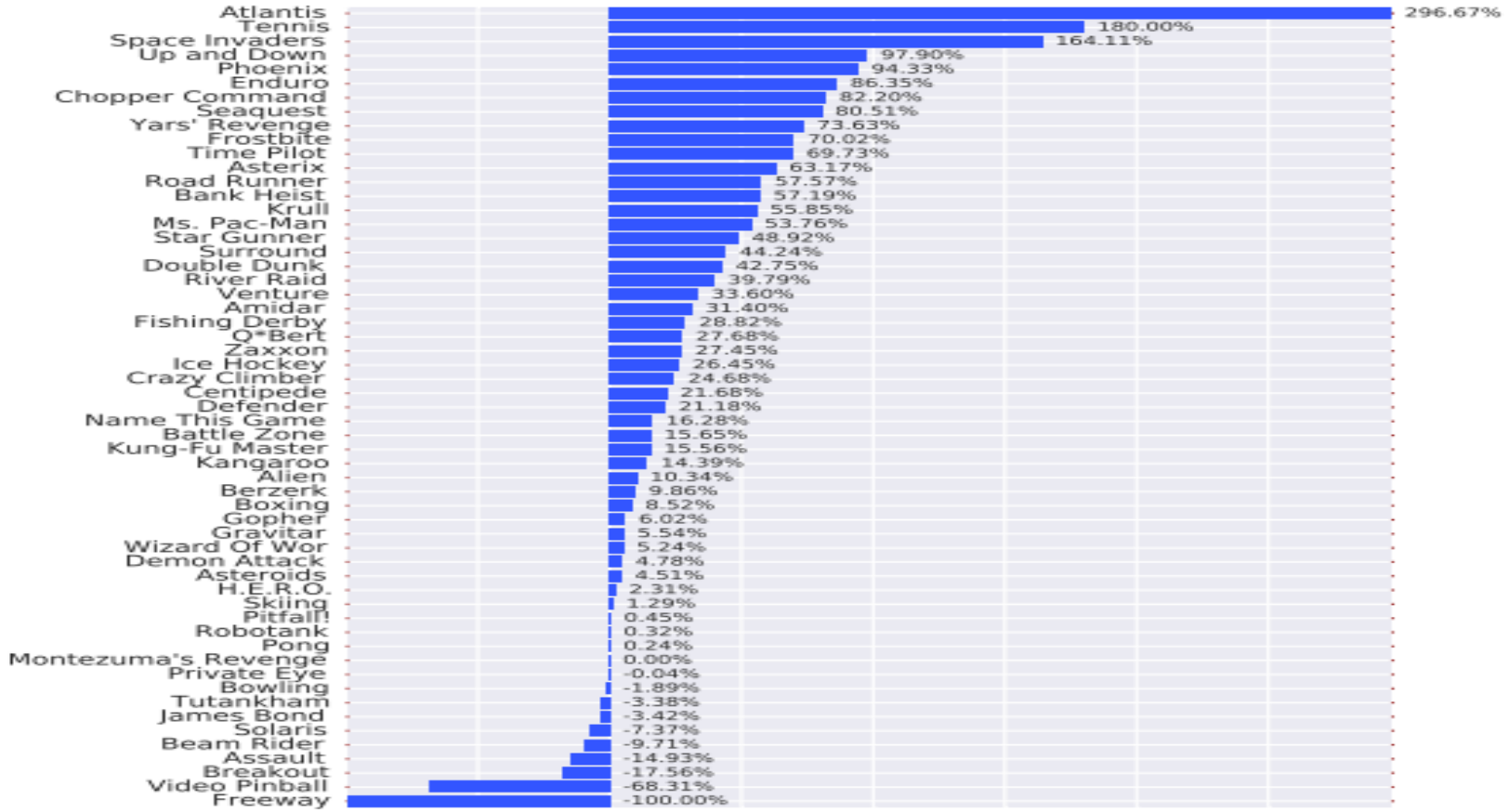
$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + [A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{\acute{a}} A(s,\acute{a};\theta,\alpha)]$$

- Advantage function의 기대값은 0이라는 성질을 활용하여 true advantage function의 성질을 따르도록 강제한 식이며 본 논문의 실험 결과는 위 식을 사용했을 때의 성능임

Dueling Network Architectures for Deep Reinforcement Learning

Results

- Dueling DQN은 기존의 DQN 알고리즘(Double DQN)과 비교하여 높은 성능을 기록하였음



Q&A