RL 스터디

2주차

Value base model free

- Model-Free control 은 MDP 같은 확률적 전이 과정을 모르는 상태에서 문제를 풀어나가는 방식
- Value base는 그 중에서 상태에 따른 가치를 근사하고 이를 기반으로 행동을 결정하는 방식

=> dqn, ddqn, per, noisy_net, C51, rainbow 를 다룰 예정

DQN – Deep Q Learning

Q- Learning 은 Q table을 기반으로 행동을 결정 state => Q

문제점

State가 고차원에 있으면? 예를 들어 이전까지는 상태를 이산화 해서 상태 별 Q를 계산했는데 실제는 가능?

해결

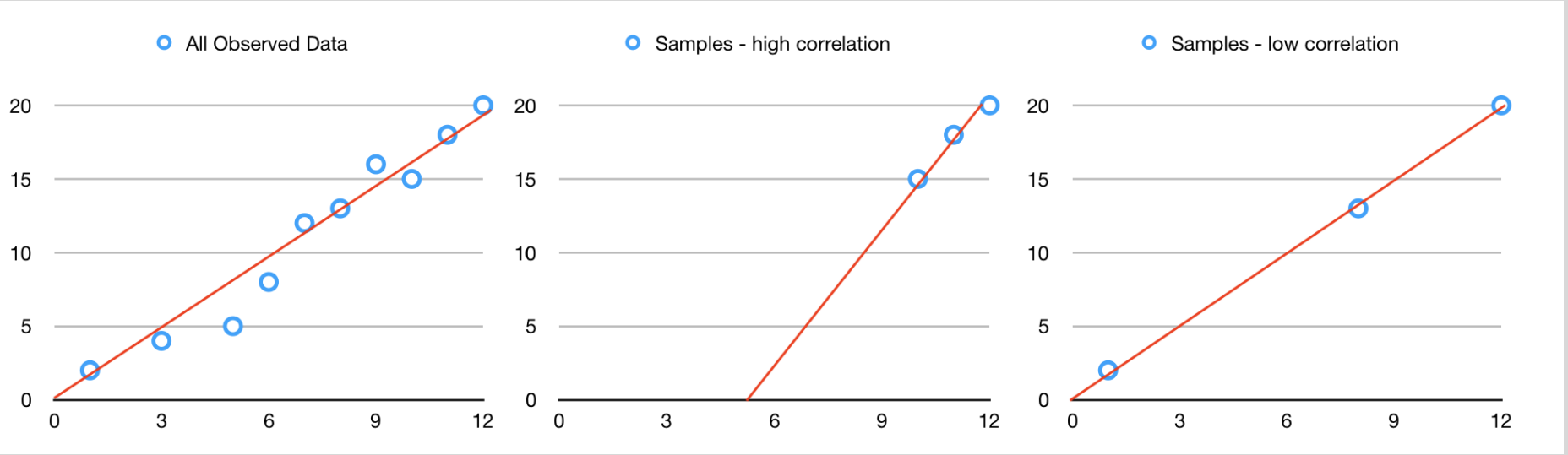
인공신경망 이용 =>

고차원의 데이터를 입력 받을 수 있다. Ex) 사진, 어떤 상태의 벡터 등등

복잡한 함수로 이루어진 상태 = > Q 근사가 가능

DQN – Deep Q Learning

1. 기존 딥러닝(지도학습)은 학습시킬 수많은 데이터를 직접 라벨링을 통해 제작해야 하는데 RL 알고리즘의 경우에는 학습시킬 보상이 노이즈가 끼거나 지체되는 경우가 빈번하다. 여기서 보상이 지체된다는 것은 지금의 행동이 결과에 영향을 주더라도 일정한 상태에 도달하기 전까지는 보상이 나오지 않은 다는 것을 의미한다.
2. 기존 딥러닝 알고리즘은 각 학습의 데이터가 독립적이지만 강화학습의 경우 연속적인 행동과 상태가 매우 중요하다. 또한 학습시킬 데이터가 이전의 행동에 의해 결정된다. 기존의 딥러닝(지도학습)이 고정된 상태 분포를 가진다고 가정하는 것과 상반된다.



replay memory

Time-step 별, 상태, 행동, 보상, 다음 상태 => 저장 => 랜덤 샘플링으로 불러온다

⇒장점

1. 한번 얻은 경험 (experience) 는 여러 번 사용가능 즉 샘플 효율성이 높아진다.
2. 학습 시키는 데이터를 최대한 독립되도록 학습에 편향이 이루어지지 않도록 돕는다. <= 제일 중요

DQN

알고리즘

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise 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 $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

DDQN

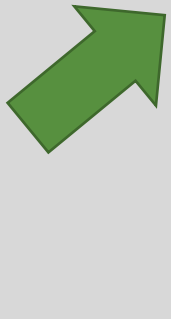
- Q의 overestimate 가 문제야

- 실제 환경은 노이즈가 많다.
- 어떠한 상태는 정확하게 한가지 상태 가치로 표현하기 애매하다.
- Q는 일단 과적합이 매우 쉽다.

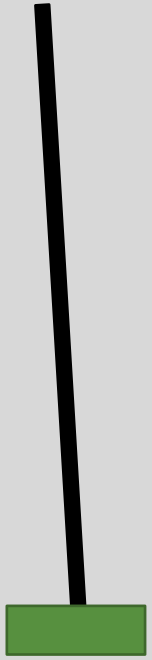
과적합 예시

주의 – DDQN은 이런 과적합을 해결하지 않는다. 그냥 과적합 예시일뿐

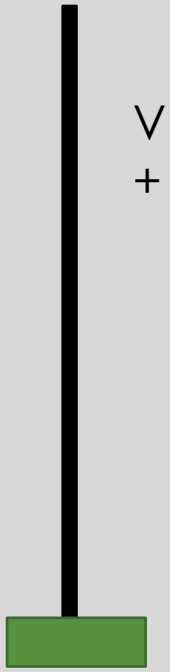
V = 1



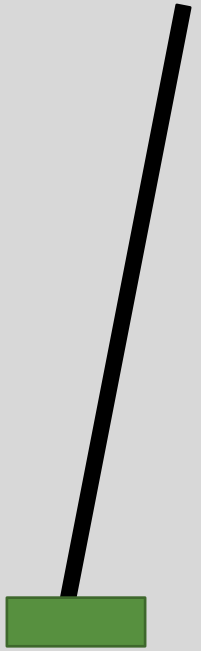
V = 1 + 0.99
=1.99



V = 1.99*0.99
+ 1 =약 3 ????



V = 1 + 0.99
=1.99



DDQN

수식은 논문 참조

실제 환경은 노이즈가 많다 => estimate error로 이어짐

이러한 estimate error는 실제 value 보다 큰 값을 low bound 가 되도록 학습하게 함

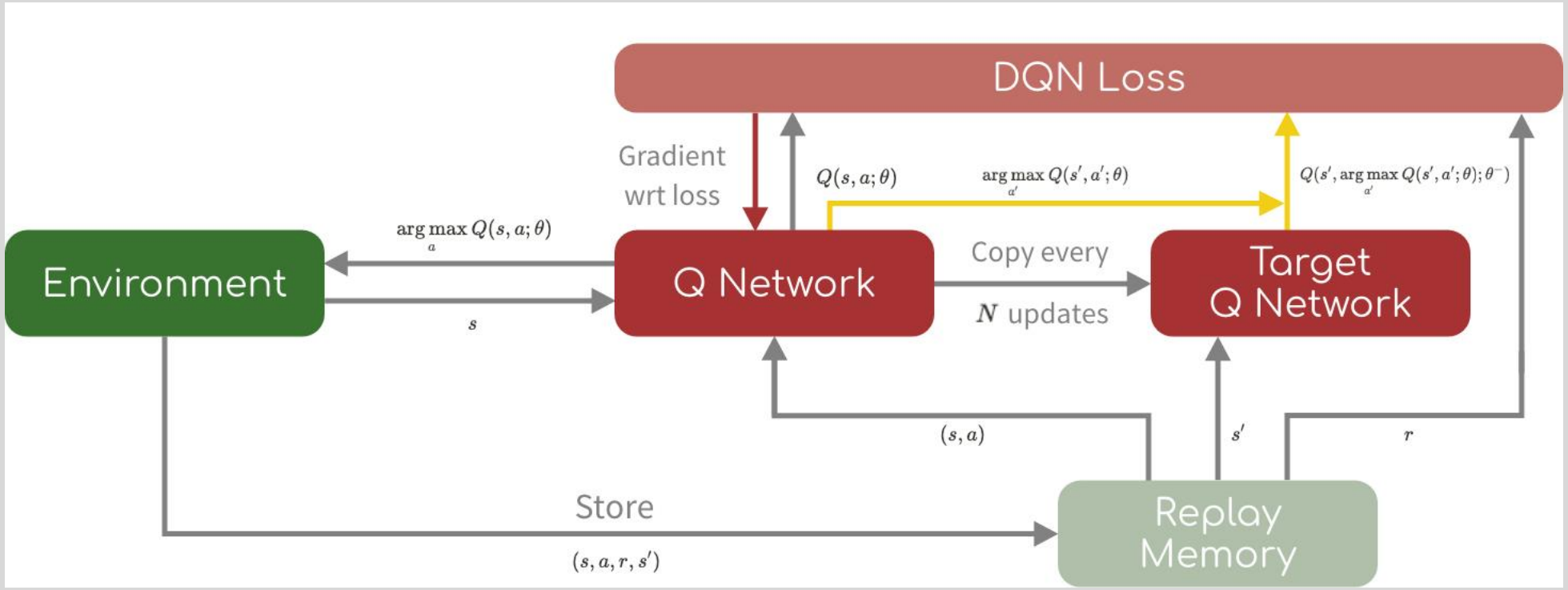
=> Double Q learning에서 아이디어를 획득

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\text{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^-) .$$

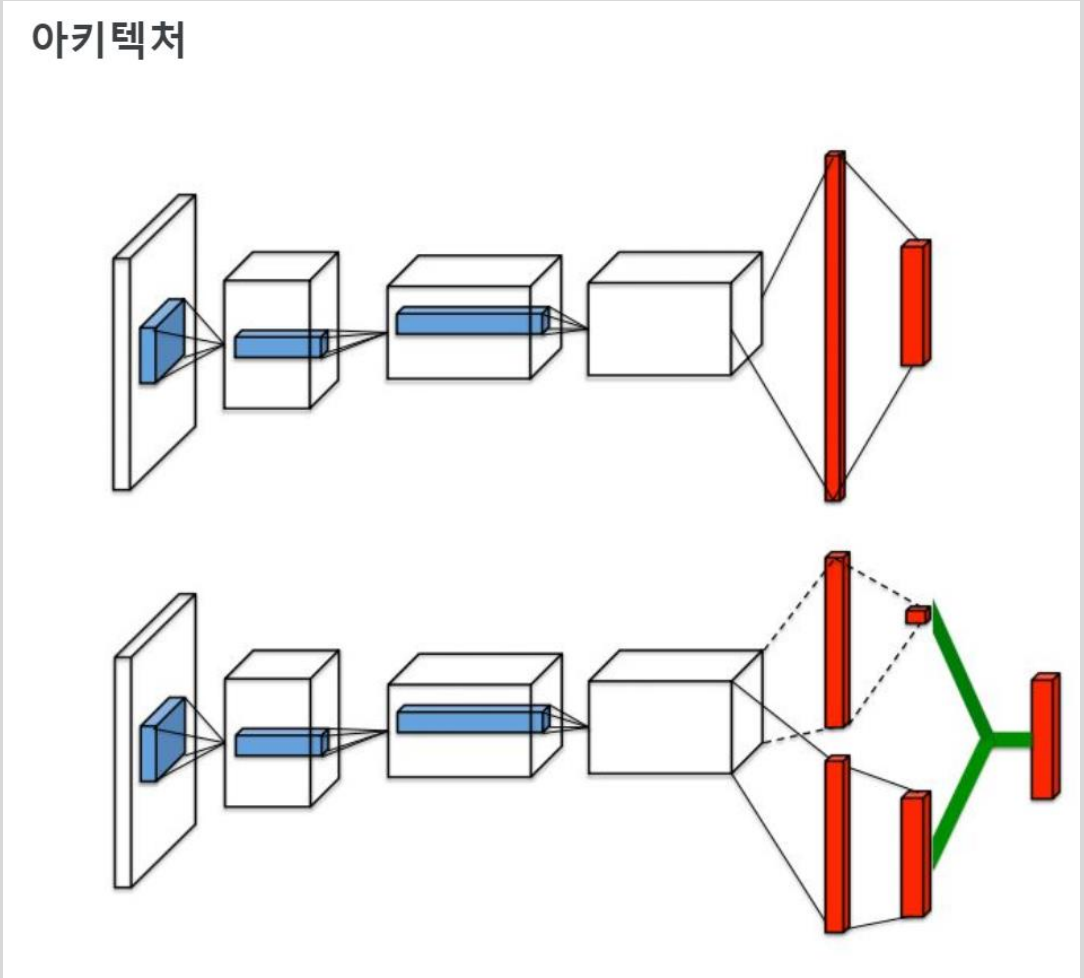
Double – q learning

- Q 함수를 두개 두고 서로가 서로의 타겟이 되어 학습이 된다. <= 서로 독립적이라 편향이 없다.(완전 독립은 아님)
- DDQN은 컴퓨터 오버헤드 감소를 위해 target Net을 일정 학습 단위마다 policy에서 복사하는 방식을 사용
- Policy와 Target을 독립시킴 <= (사실 완전 독립은 아님)

DDQN


DQN Loss
Gradient
wrt loss
$Q(s, a; \theta)$
$\arg\max_{a'} Q(s', a'; \theta)$
$Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-)$
$\arg\max_{a} Q(s, a; \theta)$
Environment
Q Network
Copy every
N updates
Target
Q Network
s
(s, a)
s'
r
Store
(s, a, r, s')
Replay
Memory

Dueling Q network

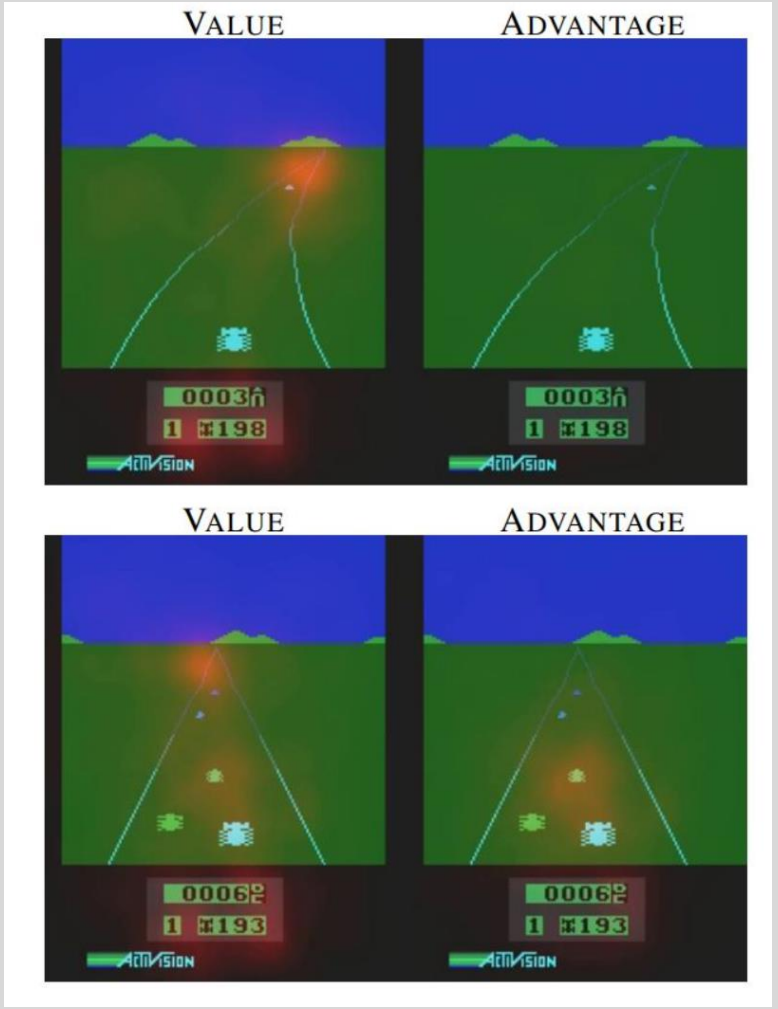


Dueling Q network

- state value와 action advantage 를 명시적으로 표현

Dueling Q network

- 현재의 상태를 원시안 적인 value 와 근시안 적인 advantage로 분리해 판단할 수 있도록 돕는다고 한다.
- Advantage는 현재 당장의 이득이 될 수 있는 행동에 대해 집중적으로 학습이 되고
- Value는 전체적인 이득을 쫓도록 학습이 된다.



Dueling Q network Target Q

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

단순히 더하면 Q가 V에 의존해서 변하는지 A에 의존해서 정해지는지 identity가 떨어진다.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a' \in |\mathcal{A}|} A(s,a';\theta,\alpha) \right)$$

Argmax를 통해 a를 선택하면 Q = V 즉 학습의 타겟에는 V만 영향을 주게
행동을 결정할 때만 A가 영향을 줄 수 있게 한다.

Dueling Q network Target Q

- 실험에는 편의상 다음과 같은 수식을 사용
- 정의와는 다르지만 안정성이 증가 오히려 성능이 오른다고?

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

PER – perioritized experience memory

- Replay 메모리를 보면 좀 쓸데 없는 그런 것들보다 좀 중요한 것들 위주로 학습시키고 싶다.

- 학습에 큰 영향을 주는 데이터 <= 의외성이 높고 현재 문제가 있는 행동을 해서 교정을 해야 하거나 task와 직접적인 연관을 가지는 데이터

- 이중 의외성과 현재 문제가 있는 행동은 td error를 통해 쉽게 우위를 결정할 수 있다.

PER – perioritized experience memory

- 각 샘플링의 우선도? => 무조건 데이터를 의외성이 높은 순으로만 학습하면 학습에 편향이 발생
- 다른 것들도 중요해 그냥 우선일 뿐

- 우선도는 어떻게 정할까?

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

p를 나타내는 방법은 여러가지 있는데 두가지를 에를 보여주자면 다음과 같다.

1. 직접적으로 TD error를 기용 그리고 최소값을 보장하기 위한 항 추가.

$$p_i = |\delta_i| + \epsilon,$$

1. TD error에 따른 우선 순위의 역수를 priority로 취급

$$p_i = \frac{1}{\text{rank}(i)}$$

PER – perioritized experience memory

◦ 확률을 다르게 표본을 추출 => important sampling 이 필요 이는 important weight를 곱해주어 해결

$$\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$$

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^\beta$$

PER 구현 이슈

- Per 특성상 td error 가 커서 -1 ~ 1 범위로 잘라서 사용
- Per 특성상 loss가 크기 때문에 lr 를 좀 작게 해 주는 것이 좋다.

Noisy Net

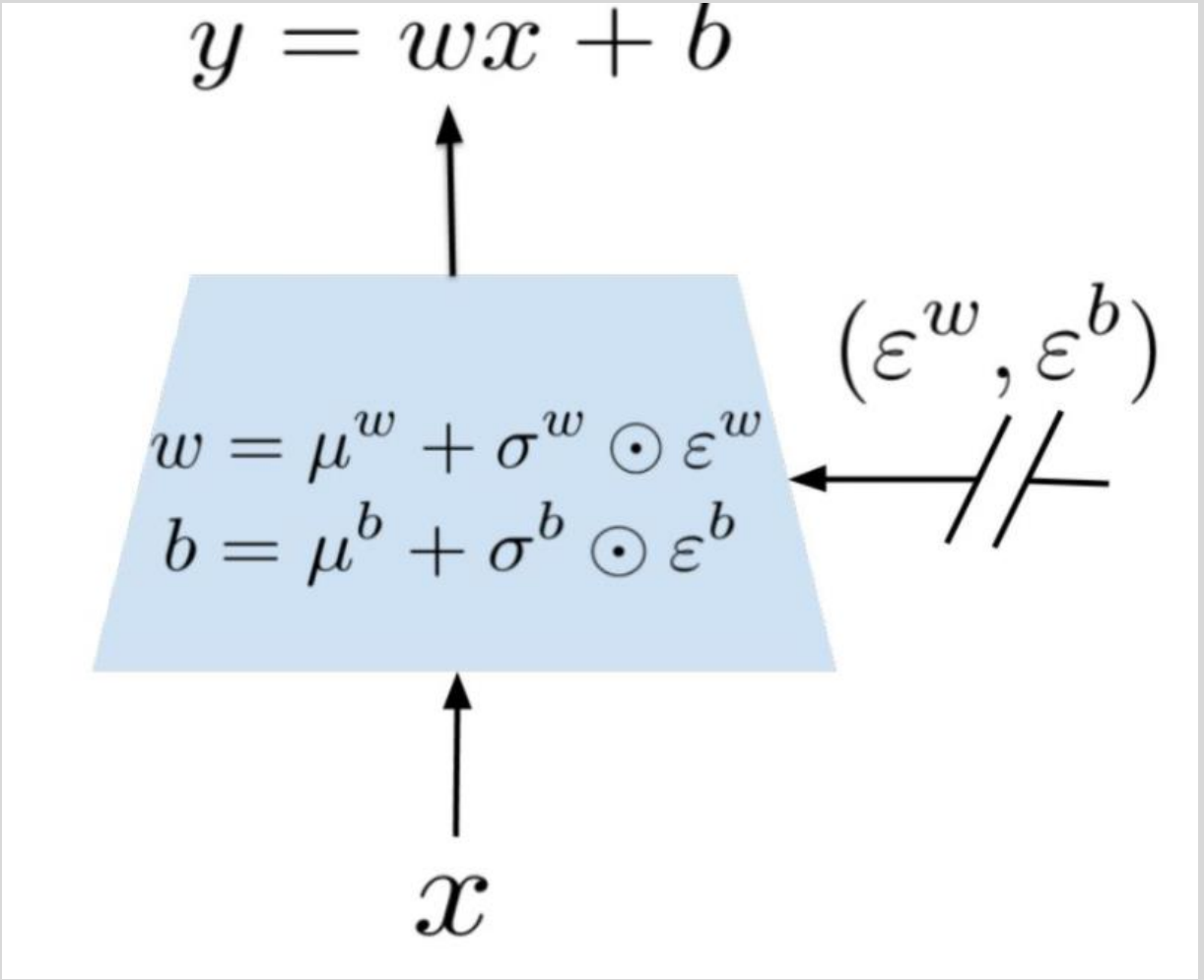
◦ 기존의 탐색 방식은 너무 휴리스틱에 의존 e-greedy
◦ 실제 학습의 진척도에 따라 환경에 따라 최적화된 탐색이 아니다.

◦ 명시적인 내적 동기를 기반으로 한 탐사 방식이 좀 있긴 한데 일반화가 힘들고
=> 직접적으로 탐사의 어떠한 지표를 등록

◦ 정책의 분포를 기반으로 한 탐사 방식은 환경과의 상호작용이 너무 많이 필요함
⇒policy state 분포에 의존

네트워크 가중치의 변화를 기반으로 한 좀더 일반적인 차원의 탐색 방법
환경에 상관없이 매우 일반화가 가능하고 쉽게 적용이 된다.

Noisy Net


$y = wx + b$
$w = \mu^w + \sigma^w \odot \varepsilon^w$
$b = \mu^b + \sigma^b \odot \varepsilon^b$
$(\varepsilon^w, \varepsilon^b)$
x

Noisy Net

파라미터 방식이 두가지인데,
한 개는 모든 파라미터를 독립적으로 사용

또다른 한가지 방법은 다음과 같은
Factorised gaussian noise 를 사용

가우시안 분포를 뽑는 양을 줄여서 컴퓨터
오버헤드를 감소시킨다.

$$\varepsilon_{i,j}^{w} = f(\varepsilon_i) f(\varepsilon_j),$$
$$\varepsilon_{j}^{b} = f(\varepsilon_j),$$

여기서 weight의 경우에 f(x)는

$$f(x) = \mathrm{sgn}(x)\sqrt{|x|}.$$

bias의 경우 f(x) = x 이다.

Noisy Net – loss function

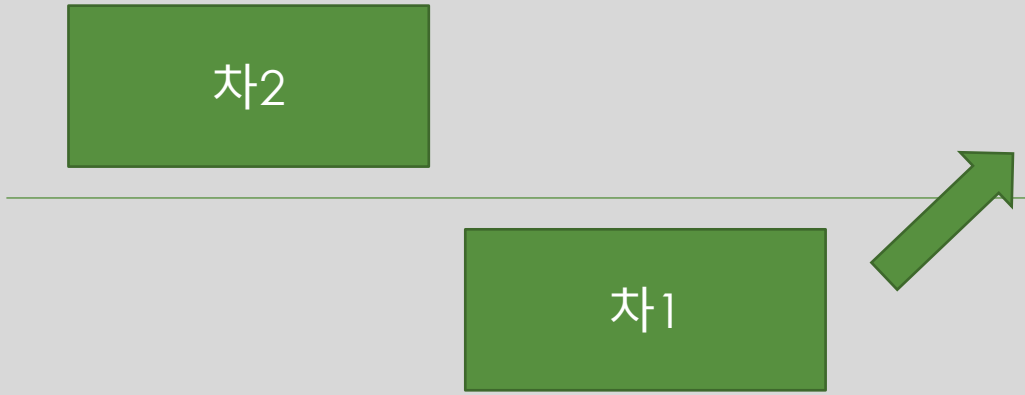
$$\nabla \bar{L}(\zeta) = \nabla \mathbb{E}\left[L(\theta)\right] = \mathbb{E}\left[\nabla_{\mu,\Sigma} L(\mu + \Sigma \odot \varepsilon)\right]$$

Noisy Net 부가 설명

- 자세한 구현은 논문 참조
- 일반적으로 2개 정도의 레이어를 사용
- 최종 레이어는 학습 진척에 따라 일반적으로 노이즈가 감소
- 하지만 결정론적으로 무조건 감소하는 것은 아님 <= 확률적인 상태 분포를 학습하는 경우

C51

- 어떤 상태에서 value는 결정론적으로 선택되지 않는다.
- 상태에 따른 value는 어떠한 distribution을 띌 것이다.



Ex) 차선 변경은 뒤의 차2가 어떤 행동을 하냐에 따라 매우 위험한 행동이 될 수도 그냥 안전한 행동이 될 수도 있다.

단순히 평균으로 땡 치기에는 상황 표현이 안됨

C51

알고리즘

Algorithm 1 Categorical Algorithm

input A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$

$Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$

$a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$

$m_i = 0, \quad i \in 0, \ldots, N-1$

for $j \in 0, \ldots, N-1$ **do**

Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$

$\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$

$b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z \quad \# b_j \in [0, N-1]$

$l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$

Distribute probability of $\hat{\mathcal{T}} z_j$

$m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$

$m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$

end for

output $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross-entropy loss

C51

- 가치를 일정한 간격의 이산 확률 분포를 가진다고 가정
- Q는기대값으로 생각

$$\triangle z := \frac{V_{\text{MAX}} - V_{\text{MIN}}}{N-1}.$$

$$Z_\theta(x,a) = z_i \quad \text{w.p.} \quad p_i(x,a) := \frac{e^{\theta_i(x,a)}}{\sum_j e^{\theta_j(x,a)}}.$$

C51

- 벨만 방정식을 쓰면 일정한 이산 분포 간격이 깨지는데? => projection 사용

- Ex) 가치가 1일 확률이 0.5 2일 확률이 0.5, r = 1일때
- Q = r +0.99*Qnext => 가치가 1.99 일 확률이 0.5 가치가 2.99일 확률이 0.5

$$(\Phi\hat{\mathcal{T}}Z_\theta(x,a))_i = \sum_{j=0}^{N-1}\left[1 - \frac{|[\hat{\mathcal{T}}z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}} - z_i|}{\triangle z}\right]_0^1 p_j(x', \pi(x')),$$

C51- loss

◦ Q 의 분포를 쓰기에 KL divergence를 사용해 loss를 계산 수학적으로는 l2 distance 즉 mse 랑 같긴함

$$D_{\mathrm{KL}}(\Phi\hat{\mathcal{T}}Z_{\tilde{\theta}}(x,a) \,\|\, Z_{\theta}(x,a)),$$

C51

- 특징

- 확률적인 시행이 강할 수록 성능이 높게 나온다
- 보상이 희박한 경우 기존보다 성능이 잘 나온다

rainbow

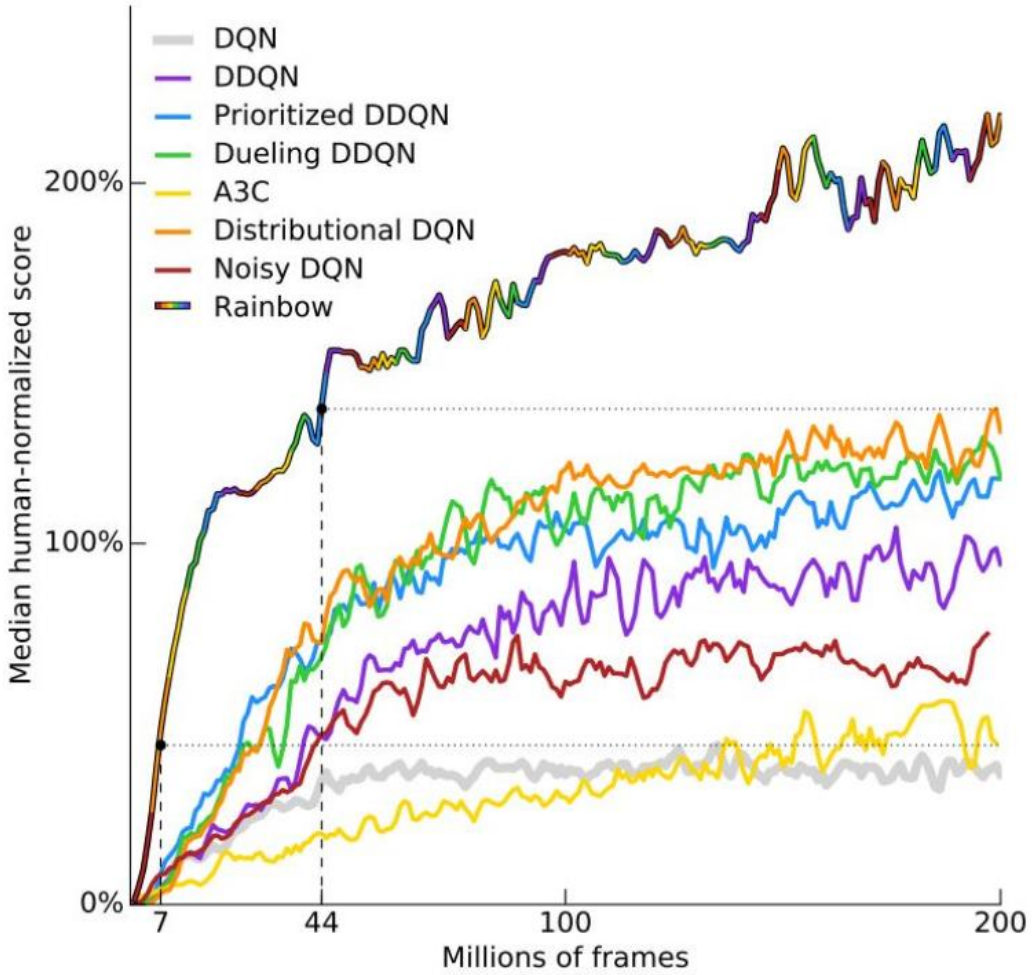
- Rainbow는 이산 행동 공간에서 value base model free 알고리즘 중 서로 겹치지 않으면서 중요하고 시너지를 낼 수 있는 요소 7(6) 가지를 섞은 것이다.

- 앞에 안 나온 A3C의 n-step 학습이 포함되어 있다.

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}$$

rainbow

구현 결과



Rainbow

어떻게 합치는가?

1. 기본 아키텍처 구성을 Dueling DQN + Noisy DQN 으로 구성한다.

   Dueling DQN으로 구성하기 위해 내부에 value와 advantage 부분을 나누고 다시 이를 Q-network에 연결 마지막 부분에 Noisy layer을 넣는다.

2. DDQN 처럼 targat과 policy net을 분리한다.

3. multi-step learning과 distribution DQN을 합치기 위해 target distribution 을 다음과 같이 정의한다.

$$d_t^{(n)} = (R_t^{(n)} + \gamma_t^{(n)} \boldsymbol{z}, \quad \boldsymbol{p}_{\overline{\theta}}(S_{t+n}, a_{t+n}^*))$$

1. PER을 사용하기 위해 loss를 TD error 가 아닌 Kullbeck-Leibler divergence를 사용

$$p_t \propto \left(D_{\mathrm{KL}}(\Phi_{\boldsymbol{z}} d_t^{(n)} || d_t) \right)^{\omega}$$

1. Dueling DQN의 advantage, value 함수를 이용해 target p를 계산할때 다음과 같이 한다.

$$p_\theta^i(s,a) = \frac{\exp(v_\eta^i(\phi) + a_\psi^i(\phi,a) - \overline{a}_\psi^i(s))}{\sum_j \exp(v_\eta^j(\phi) + a_\psi^j(\phi,a) - \overline{a}_\psi^j(s))},$$

where $\phi = f_\xi(s)$ and $\overline{a}_\psi^i(s) = \frac{1}{N_{\text{actions}}} \sum_{a'} a_\psi^i(\phi, a')$.

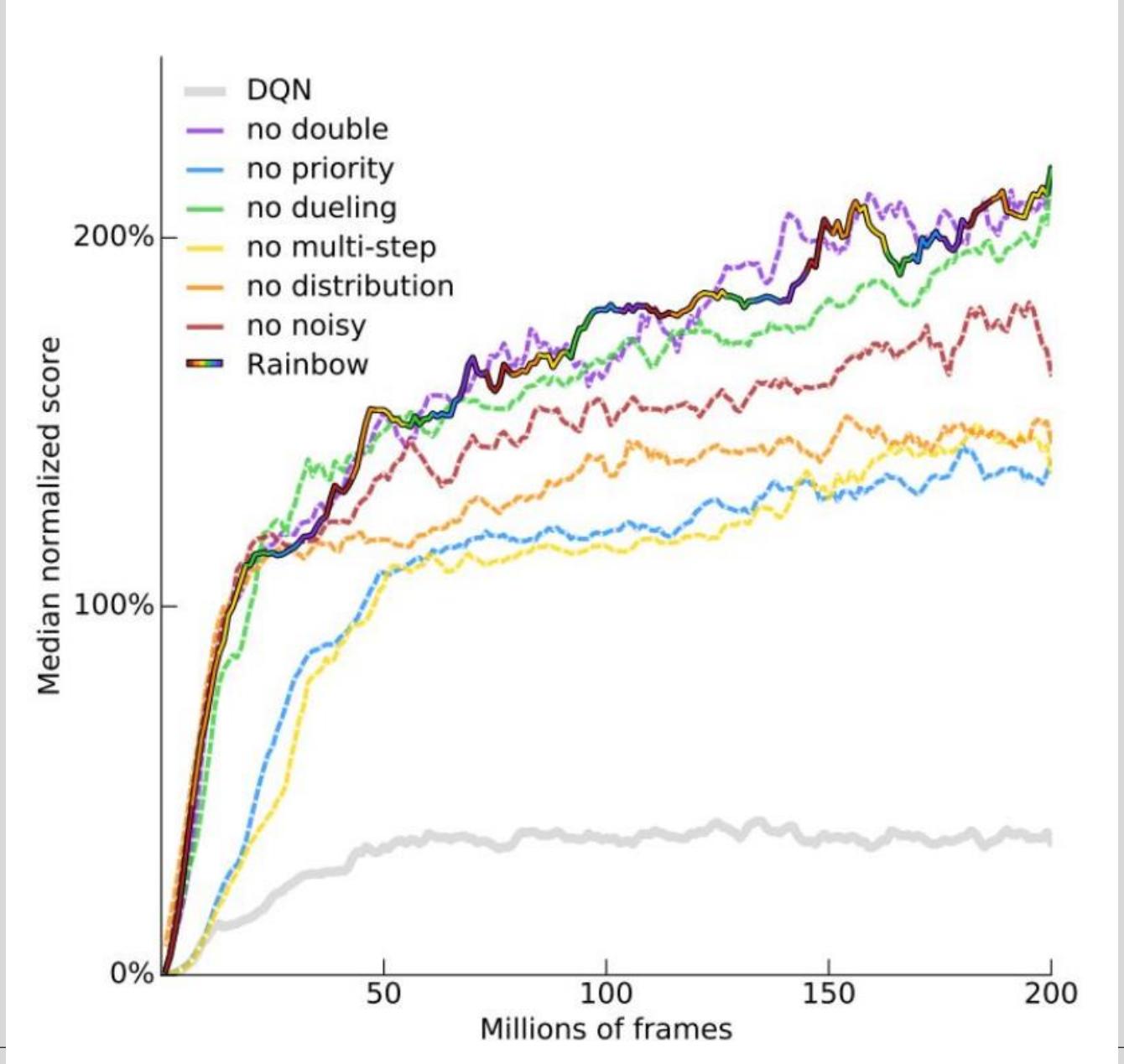
rainbow

DDQN 방식은 별 영향이 없다.
Dueling도 좀 영향이 약함

Noisy는 꽤 중요한 요소

Distribution은 중요한 성능 요인

multi-step과 per은 성능에 필수적인 요소이다.



끝