DQN Variants

Kihwan Lee

Contents

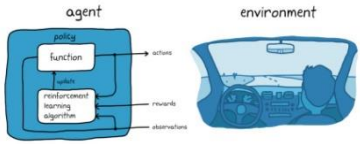
1. Multi-step Learning

Multi-step Learning

- **DQN의 loss function**

$$(Q(s_t, a_t; \theta) - (r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)))^2$$

: target에 rt+1이 존재



=> 게임을 할 때 바로 다음 step의 reward도 중요하지만 자율주행과 같이
10step이후까지 볼 수 있다면 유리할 때가 존재

=> 이를 접목시킨 것이 Multi-step Learning !

n-step return

$$G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \quad \leftarrow \text{TD target}$$

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

$$G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T \quad \leftarrow \text{MC target}$$

Multi-step Learning

- **Multi-step Learning의 loss function**

$$\text{Loss} = \frac{1}{N}\sum_i \left(G_{t:t+n} - Q(s_t, a_t; \theta)\right)^2$$

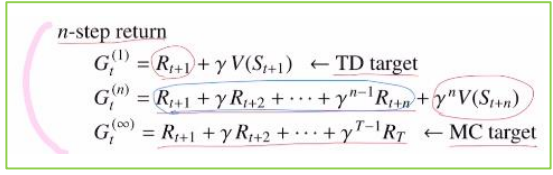
$$G_{t:t+n} = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n \max_a Q(s_{t+n}, a; \theta^-)$$

DQN : $\left(Q(s_t, a_t; \theta) - (r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-))\right)^2$

DQN과 비교했을 때, TD target이 아닌 n-step return을 사용

Q. 활용되는 곳?

* 로봇제어
* 자율주행
* 게임 => 바둑?

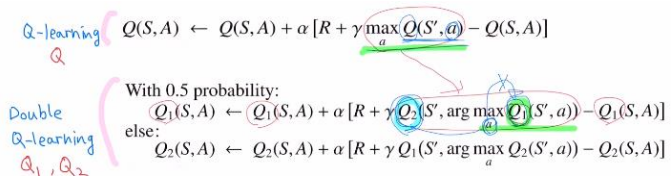


2. Double DQN

Double DQN

- Q leanrnig vs Double Q learning

  Q learning에서는 maximum opreration이 존재하여 실제값보다 높게 측정이 되어 over estimate됩니다. 이를 해결하기 위해 double Q를 사용합니다.



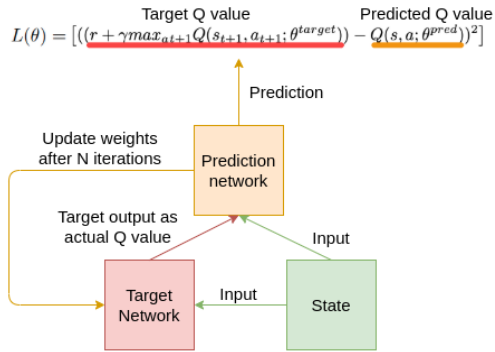
Double DQN

- Double DQN

앞의 방법을 활용하여 DQN에 적용

=> 이미 두 개의 네트워크가 존재! (behavior, target)

- DQN: $L(\theta) = [r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a; \hat{\theta}) - Q(s_t, a_t; \theta)]^2$
- Double DQN: $L(\theta) = [r_{t+1} + \gamma \hat{Q}(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta); \hat{\theta}) - Q(s_t, a_t; \theta)]^2$



3. Prioritized Replay

Prioritized Replay

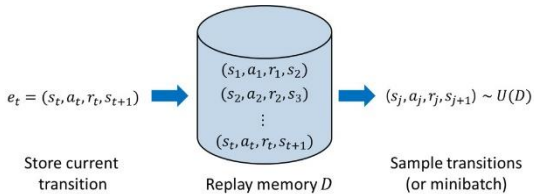
- **Prioritized Replay**

  : Experience replay에서 미니배치만큼 랜덤 샘플링을 할 때 priority를 부여

  (기존 문제점)

  온라인으로 바로바로 업데이트 => strongly temporally-correlated updates => iid 깨지고 성능 저하

  / 해결 / Replay buffer로 mini batch => uniform하게 말고 중요한 데이터에 가중치를 주자!



Prioritized Replay

Q) 중요하다는 판단 기준은?

: Temporal difference error (TD error) 를 통해 결정 (로봇 예시)

$$\mathrm{TD}_1 = r_t + \gamma Q(s_{t+1}, a^*) - Q(s_t, a_t) \quad \Rightarrow \quad P(\mathrm{i}) = \frac{p_i}{\Sigma_k p_k}$$

=> 이 에러가 큰 값을 중요한 것이라 생각하고, 미니배치로 뽑을 때 더 많이 뽑힐 수 있게 확률을 올려줍니다.

- **문제점 발생** : 특정한 쏠림 현상으로 다양성과 편향에 관한 문제가 생깁니다.

  1) loss of diversity

  : stochastic sampling prioritization으로 해결합니다.

  2) bias

  : importance sampling weights로 해결합니다.

Prioritized Replay

- **문제점 발생** : 특정한 쏠림 현상으로 다양성과 편향에 관한 문제가 생깁니다.

1) loss of diversity : 초반에는 TD error는 성능이 좋지 않아, 한 번 높은 값을 가지면 계속 선정
: stochastic sampling prioritization 으로 해결

=> $P(\mathrm{i}) = \frac{p_i^{\alpha}}{\Sigma_k p_k^{\alpha}}$ α : (uniform) 0 ~ 1 (priority)

2) bias
: importance sampling weights로 해결

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

학습이 거의 끝날 때는 unbiased update를 고려하여 β를 점점 1에 가깝게 조절

Ex)