1. 강화학습 소개

강화학습의 최종 목표는 **환경(Environment)**과 상호작용을 하는 임의의 **Agent**를 학습시키는 것이다. Agent는 **상태(State)**라고 부르는 다양한 상황 안에서 **행동(Action)**을 취하며 조금씩 학습해나간다. Agent가 취한 행동은 그에 대한 응답으로 양(+)이나 음(-), 또는 0의 **보상(Reward)**을 돌려받는다.

여기에서 Agent의 목표는 처음 시작하는 시점부터 종료시점까지 일어나는 모든 에피소드에서 받을 보상값을 최대로 끌어올리는 것이다. 이를 위해 음(-)값의 보상을 받는 행동은 최대한 피하도록 하고, 양(+)값의 보상을 받을 수 있는 행동을 강화시킨다는 의미에서 강화학습이라는 이름이 붙게 되었다. 이렇게 Agent가 학습을 하는 과정에서 점점 발전하게 될 의사결정 전략을 **정책(Policy)**라고 한다.

모든 **상태(State)**는 그 **직전의 상태**와, **그 상태에서 Agent가 선택한 행동**이 만든 직접적인 결과이다. 직전의 상태는 또한 그 이전의 상태와 그 이전의 상태에 한 행동의 결과이고, 이렇게 계속하다보면 처음 시작했던 시점까지 올라갈 수 있겠다. 이렇게 연결되어 있는 모든 단계들과 그 순서는 **현재 상태를 결정짓는 어떤 정보**를 담고 있을 것이고, 그에 따라 지금 순간에 Agent가 어떤 행동을 선택해야 하는지에 대해서도 직접적인 영향을 미친다. 즉 모든 정보들을 다 활용할 수 있다면 Agent가 어떤 선택을 해야 하는 지 결정하는 데에 큰 도움이 될 것이다.

하지만 이런 식이라면 한 가지 문제가 발생한다. 바로 Agent가 한 단계씩 나아갈 때마다, 점점 더 많은 정보들이 저장되어야 하고 그 양은 계속 늘어난다는 점이다. 이렇게 많은 양의 데이터를 다루면서 연산한다는 것은 쉽지 않은 일이다.

이 문제를 해결하기 위해서, 모든 상태가 **Markov State**에 해당한다고 가정한다. **Markov State**란, "모든 상태는 오직 그 직전의 상태와 그 때 한 행동에 대해서만 의존한다"는 가정이다. 즉, 직전 상태 말고 그보다 더 이전의 상태들에 대해서는 고려하지 않아도 되는 것이다.

모든 상태에서 취하는 각각의 행동에 대해 보상을 얼마나 받을지 미리 알고 있다고 가정해보자. 최종적으로 가장 높은 보상을 받을 수 있는 행동들을 연속적으로 취하면 될 것이다. 이렇게 최종적으로 받는 모든 보상의 총합을 **Q-value** (Quality Value)라고 한다.

$$Q(s,a)=r(s,a)+\gamma \max_a Q(s',a)$$

상태 s 에서 행동 a 를 취할 때 받을 수 있는 모든 보상의 총합 Q(s,a) 는, 현재 행동을 취해서 받을 수 있는 **즉각보상**과 미래에 받을 **미래보상의 최대값**의 합으로 계산할 수 있다. 우변의 r(s,a) 는 현재 상태 s 에서 행동 a 를 취했을 때 받을 즉각보상값을 나타낸다. 또한, s′ 은 현재 상태 s 에서 행동 a 를 취해 도달하는 바로 다음의 상태로, maxaQ(s′,a) 는 다음 상태 s′ 에서 받을 수 있는 보상의 최대값이다. 이 값을 최대화 할 수 있는 행동을 선택해내는 것이 Agent의 목표이다.

또한 그 앞에 붙어있는 γ 는 할인율이라고 부르는 값으로, 미래가치에 대한 중요도를 조절한다. γ 값이 커질수록 미래에 받을 보상에 더 큰 가치를 두는 것이고, 작아질수록 즉각보상을 더 중요하게 고려하는 것이다.

이제 **"어떤 상태이든, 가장 높은 누적 보상을 얻을 수 있는 행동을 취한다"** 라는 기본 전략이 생겼다. 이렇게 매 순간 가장 높다고 판단되는 행동을 취한다는 점에서, 알고리즘은 *greedy* (탐욕적) 이라고 부르기도 한다. 이런 전략을 현실 문제에 적용시키는 방법은 모든 가능한 상태-행동 조합을 표로 그리고, 거기에 모든 Q-value를 적어 사용하는 것이다.

$$Q(s,a) := r(s,a) + \gamma \max_a Q(s',a)$$

$$Q(s,a) = r(s,a)$$

탐욕적 알고리즘은 심각한 문제가 하나 있는데, 만약 항상 '최선의 선택'만 고집한다면, 새로운 선택은 해보지 않을 것이고, 그렇게 된다면 아직 받아보지 못한 미지의 보상에 대해서는 앞으로도 그 존재를 알 수 없다는 것이다.

이 문제를 해결하기 위해, ϵ-greedy 전략을 추가한다. 바로 $0<\epsilon<1$ 인 어떤 ϵ 값을 이용해서, 탐험과 활용을 적절히 번갈아가며 사용하는 것이다. $p=1-\epsilon$ 의 확률로는 원래 했던 대로 탐욕적으로 행동을 선택하고, 나머지 $p=\epsilon$ 의 확률로는 랜덤으로 행동을 취하는 것이다. 이렇게 함으로써 Agent는 새로운 공간으로의 탐험을 해보면서 알지 못하는 것들도 놓치지 않고 학습할 수 있는 기회를 가지게 되는 것이다. 이 알고리즘을 바로 Q Learning, 즉 Q-러닝이라고 한다.

Q-러닝과 딥러닝을 합친 것을 바로 Deep Q Networks 라고 부른다. Q-table 대신 신경망을 사용해서, 그 신경망 모델이 Q 가치를 근사해낼 수 있도록 학습시키는 것이다. 그래서 이 모델은 주로 *approximator* (근사기), 또는 *approximating function* (근사 함수) 라고 부르기도 한다. 모델에 대한 표현은 $Q(s,a;\Theta)$ 라고 하고,

여기서 Θ 는 신경망에서 학습할 가중치를 나타낸다.

$$Q(s,a)=r(s,a)+\gamma \max_a Q(s',a)$$

학습해나가는 과정에 있어서 "=" 라는 기호는 '할당'이라는 의미를 가진다. 좌변과 우변이 실제로 같아지는 순간은 바로 Q 가치가 수렴해서 참값에 도달했을 때일 것이다. 그러므로 우리는 두 값의 차이를 최소화하는 방향으로 모델을 학습해나가면 된다. 그럴 경우 비용함수는 다음과 같은 식으로 간단히 정의될 수 있다.

$$Cost = \left[Q(s,a;\theta) - \left(r(s,a) + \gamma \max_a Q(s',a;\theta) \right) \right]^2$$

학습 과정에 대해 살펴보면, 강화학습에서는, Training data set이라는 게 처음에는 따로 없다. Agent가 행동을 취함에 따라 데이터셋은 점점 쌓여나가는 것이다. Agent는 매 순간 그때까지 학습된 네트워크를 통해 최적이라고 판단된 행동을 취하면서, 에피소드가 끝날 때까지 상태, 행동, 보상, 그리고 다음 상태에 대한 데이터를 쌓아나간다.

데이터를 쌓으면서 학습하는 과정은 다음과 같다. 학습에 필요한 batch size를 b라고 한다면, Agent는 행동을 b 번 취하면서 b 개의 데이터 세트들이 기록한다. 여기서 주목해야 할 점은, 저장되어있는 메모리 공간에서 방금 기록된 b 개의 데이터셋으로 학습하는 것이 아니라, 지금까지 쌓인 데이터셋 중 b 개를 **랜덤으로 선택**해서 신경망을 학습시킨다는 점이다.

Q-table에서 봤던 방식에 따른다면, 신경망은 **현재 상태**에 대한 정보와 **현재 취할 행동**, 두 가지를 입력받습니다. 그 후에는 모델 내에서 연산을 거쳐 **Q-value**를 출력하게 된다. 벨만 방정식에서 우리는 가치를 최대화하기 위해 미래에 선택할 수 있는 행동에 따른 가치 중 최댓값을 취해야 하는 부분이 있었다. 최대값을 찾기 위해서는 **모든 행동들을 취해봐야 알 수 있을 것**이다. 모델에 상태만을 입력해줬을 때 각 행동에 대한 모든 Q-value를 한 번에 얻을 수 있다.

2. 강화학습 단점

아직 성능이 만족스럽지 못하다는 것이다. 만약 특정 시스템의 수학적 모델을 정확히 알고 있다면 강화학습 보다는 기존의 제어 이론을 사용하는 것이 훨씬 더 우

수한 성능을 얻을 수 있다. 강화학습이 데이터만을 가지고 학습하기 때문에 수학적 모델 만큼 시스템의 반응을 매우 정교하게 포착하지 못할 수 있다.

강화학습의 보상(reward) 함수에 관한 것이다. 강화학습은 보상 기능이 있거나 환경이 제공한다고 가정한다. 그러나 실제로는 강화학습 설계자가 시스템이 원하는 반응을 보이도록 보상함수를 설계해야 한다. 따라서 보상함수는 설계자가 원하는 것을 정확히 포착하도록 해야 하지만, 함수 자체가 단일 스칼라 함수이기 때문에 복잡한 운동을 하는 시스템에서는 적절한 보상함수를 찾는 것이 쉬운 일이 아니다.

강화학습의 최종 결과가 불안정하고 재현하기 어려울 수 있다는 점이다. 다른 딥러닝(deep learning) 알고리즘과 마찬가지로 강화학습에도 학습(training)에 영향을 주는 하이퍼파라미터(hyperparameter)가 있다. 하이퍼파라미터는 아직 설계자가 수작업으로 결정한다. 하이퍼파라미터에 따라서 학습이 불안정하게 될 수도 있고 안정적으로 진행될 수도 있다. 또한 동일한 하이퍼파라미터를 사용하는 경우에도 동일한 작업에서 서로 다른 학습 성능을 보일 수도 있다.

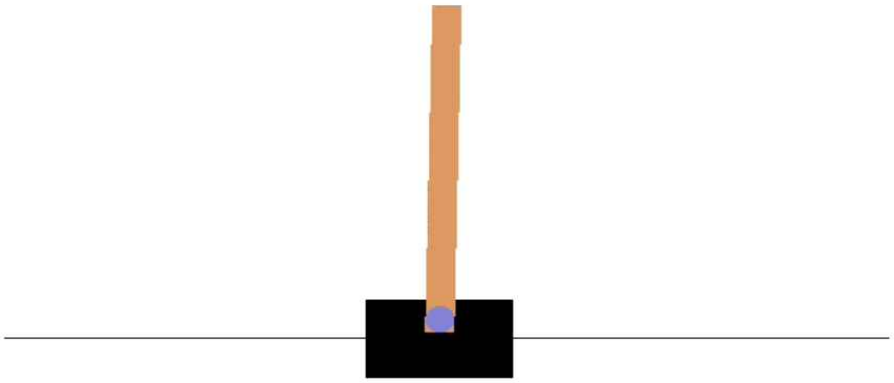
심지어는 학습에 실패하는 경우도 생긴다. 학습 과정에서 사용되는 데이터의 무작위성(randomness) 때문이며 알고리즘 차제가 이에 매우 민감하게 반응하는 것이다. 적절한 초기화와 함께 적절한 하이퍼파라미터를 찾기 위해서는 부단한 노력과 인내가 필요하다. 즉, 강화학습은 아직 기존 제어 알고리즘보다도 체계적이지 못하다고 말할 수 있다.

2. 카트 폴(Cart-Pole) 예제

카트 폴(Cart-Pole) 예제는 REINFORCE 법 (Williams, 1992)을 사용하여 "Simulated Cart Pole" 환경에서 보상을 극대화하기 위한 간단한 신경망의 훈련 방법을 보여준다. 카트 폴 환경은 마찰이 없는 1차원 경로를 따라 이동하는 카트와 힌지(즉 역 진자)로 카트에 장착된 가중 막대기로 구성된다. 카트는 개입이 없으면 막대기가 쓰러져 버리는 초기 속도를 가지고 있다. 에이전트의 목표는 가능한 한 봉을 오랫동안 똑바로 세운 상태를 유지하는 것이다. 이것은 두 가지 가능한 동작(왼쪽으로 움직이는지, 오른쪽으로 움직이는지)의 어느 쪽을 실시할 것인지를 학습함으로써 달성된다.

카트 폴 문제는 cart 위에 막대기를 세워 최대한 오랫동안 버티는 것을 목표로 한

다. 2D 평면 위의 track에 cart가 있고 cart는 track 위에서 좌우로 움직일 수 있다. cart에는 아래와 같이 pole이 360° 회전할 수 있게 연결되어 있어, cart가 움직이지 않으면 pole은 중력에 의해 아래로 늘어뜨려지도록 환경이 구성되어있다. 따라서 cart는 계속 좌우로 움직이며 균형을 맞춰야 목표를 성취할 수 있다.



1. Observation

변수	최솟값	최댓값
카트의 위치	-2.4	+2.4
카트의 속도	-Inf	+Inf
막대의 각도	-41.8°	+41.8°
막대 끝의 속도	-Inf	+Inf

2. Action

숫자	Action
-1	카트를 왼쪽으로
+1	카트를 오른쪽으로

3. Reward

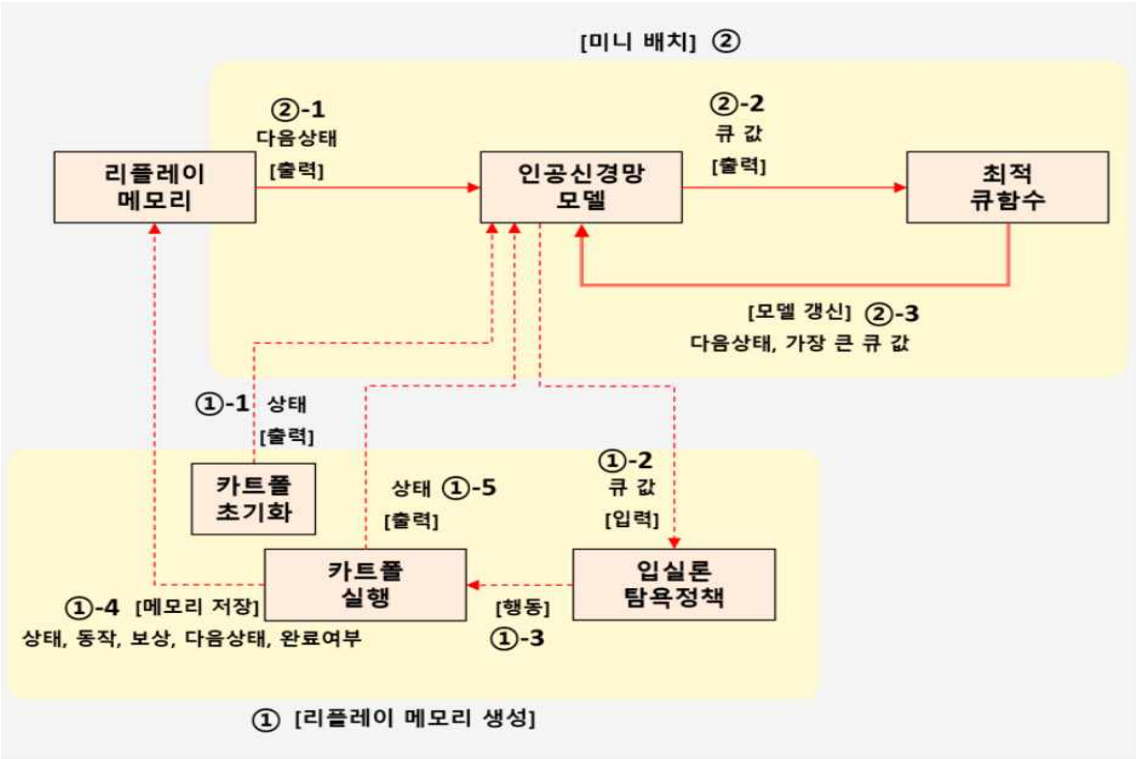
매 스텝마다 카트가 중심을 기준으로 일정 범위 안에 있고, 막대기가 넘어지지 않으면 +1의 보상을 받습니다.

4. Episode 종료

1) 막대 각도가 ±20.9를 초과한다.

2) 카트의 위치가 양쪽 끝에 도달한다.

3) Action 한계 횟수를 초과한다.



1. 리플레이를 저장할 수 있는 리플레이 메모리와 네트워크를 구성한다.

2. state를 초기화하고 전처리를 해준다.(이 코드에서 전처리는 state 값을 2차 배열로 만들어 주는 것이다)

3. 한 개의 에피소드를 시작하고 e-greedy 으로 랜덤하게 action을 선택하거나 Q-value 값(Weight를 이용한)으로 action을 선택하여 실행한다.

4. action을 통한 reward와 다음 state를 받는다.

5. 위 받은 값들(state, action, reward, next_state, done)을 메모리에 저장한다.

6. 저장된 메모리의 값들 중에 몇 개를 가져와서 미니배치로 학습을 한다.

7. 위 과정을 반복한다.