

2019 OSS 개발자 포럼 여름 캠프

강화 학습 기초 - Day 2

옥찬호

Nexon Korea, Microsoft MVP

utilForever@gmail.com

- 오늘 강의 내용은 ‘파이썬과 케라스로 배우는 강화학습’ (위키북스, 2017)을 기반으로 만들었습니다. 허락해주신 이웅원님께 감사드립니다.
또한 위 도서에서 일부 이미지를 참고했음을 알려드립니다.
- 발표 준비를 도와준 조교 김현수/박준영 학생에게 감사의 말씀을 드립니다.

- 근사함수
- 인공신경망
 - 노드와 활성화함수, 딥러닝, 신경망의 학습
- 딥살사(Deep-SARSA)
- 폴리시 그레이디언트
 - 정책 기반 강화학습, REINFORCE 알고리즘
- DQN(Deep Q-Network)

다이나믹 프로그래밍의 한계

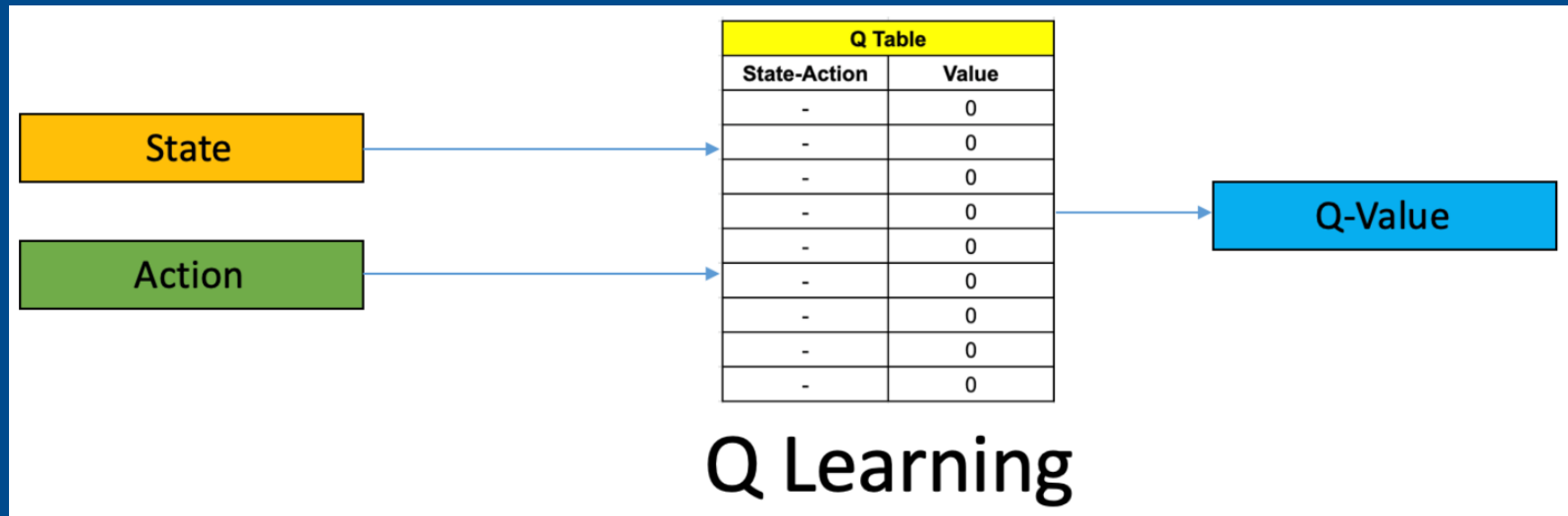
- 계산 복잡도
- 차원의 저주
- 환경에 대한 완벽한 정보가 필요

몬테카를로, 살사, 큐러닝은 이 세 가지 문제를 다 해결했을까?

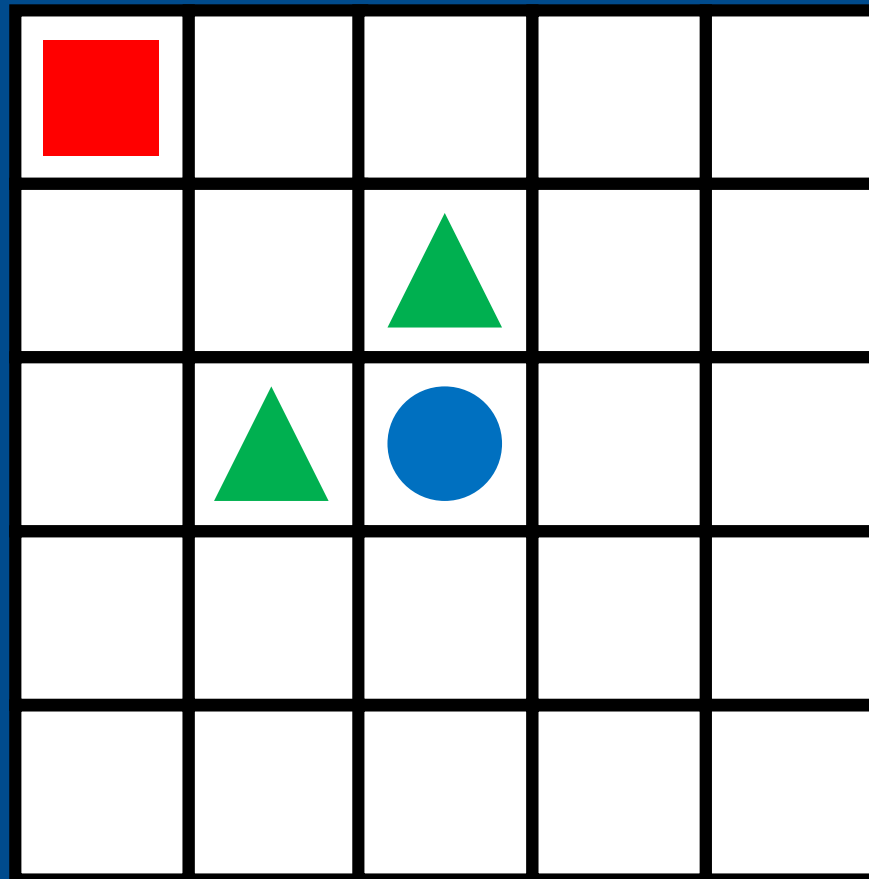
다이나믹 프로그래밍의 한계

- 계산 복잡도 \rightarrow ???
- 차원의 저주 \rightarrow ???
- 환경에 대한 완벽한 정보가 필요
 \rightarrow 모델 프리(환경에 대한 모델 없이 샘플링을 통해 학습)

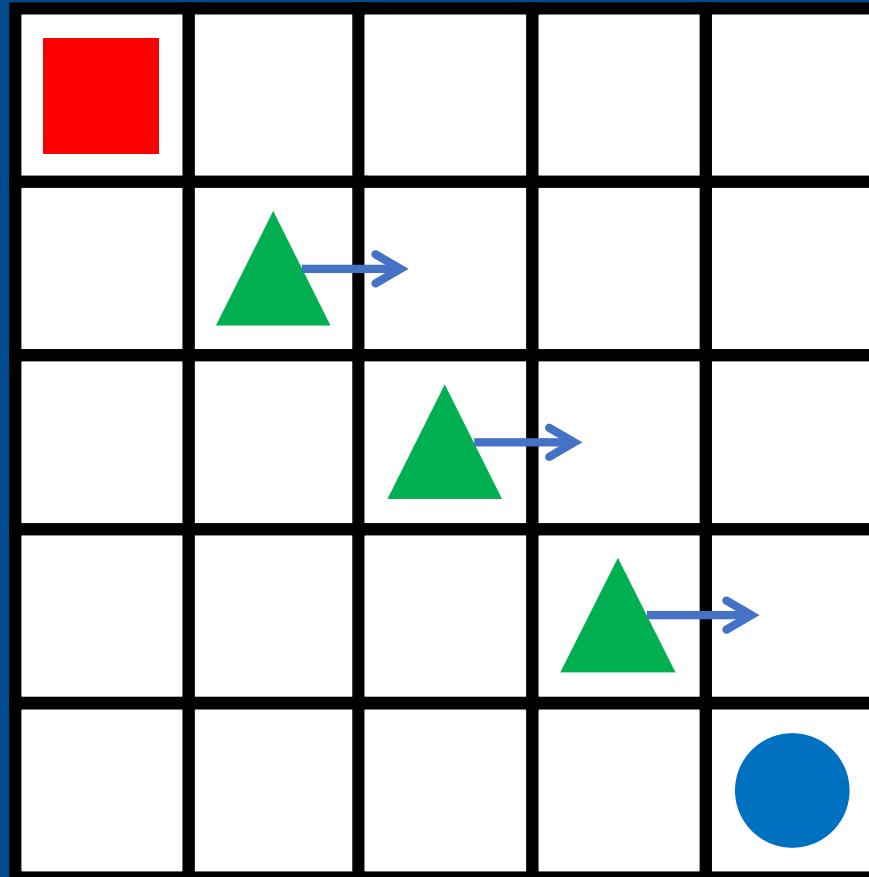
- 그리드월드의 경우 상태는 (x, y) 좌표로 2차원이었고 전체 상태의 수는 25개였다.
- 에이전트가 선택 가능한 행동이 5개였으므로 행동 상태는 125개 → 테이블 형태로 풀 수 있다.
- 환경의 모델을 안다는 장점을 빼면 다이내믹 프로그래밍이 훨씬 빠른 속도로 답을 찾아낸다.



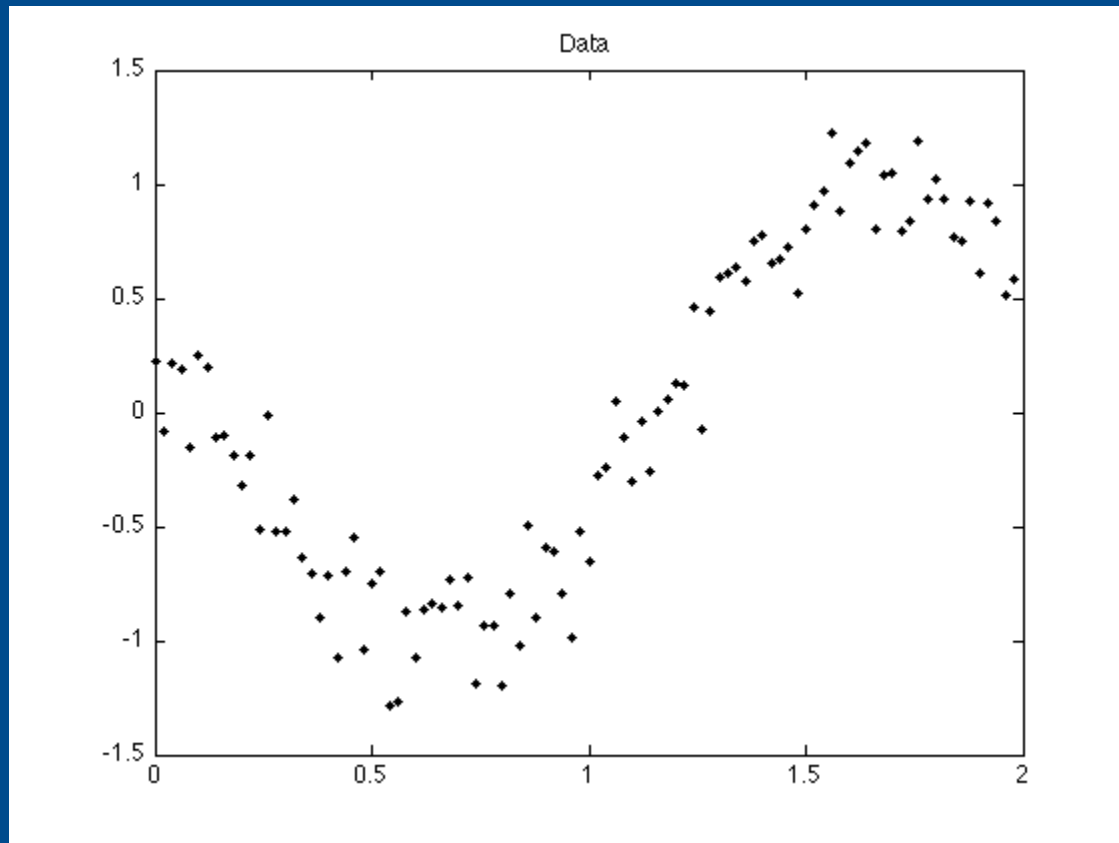
하지만 우리가 풀고 싶은 문제는 이런 게 아니다.



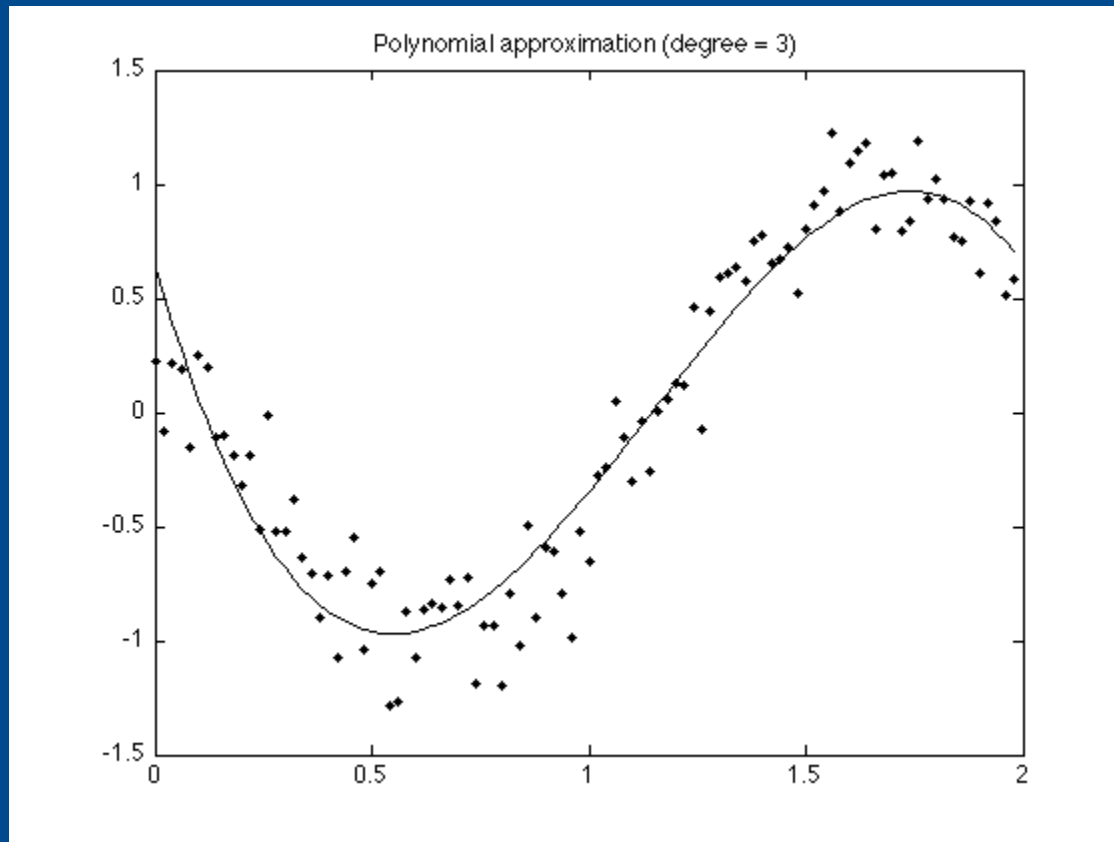
하지만 우리가 풀고 싶은 문제는 이런 게 아니다.



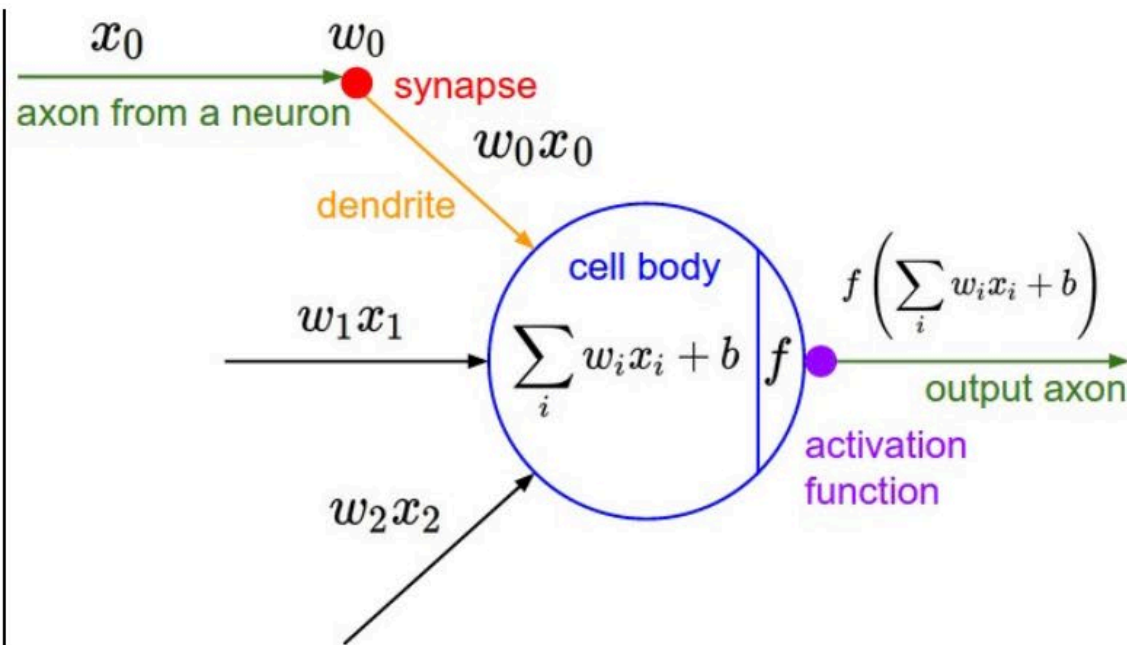
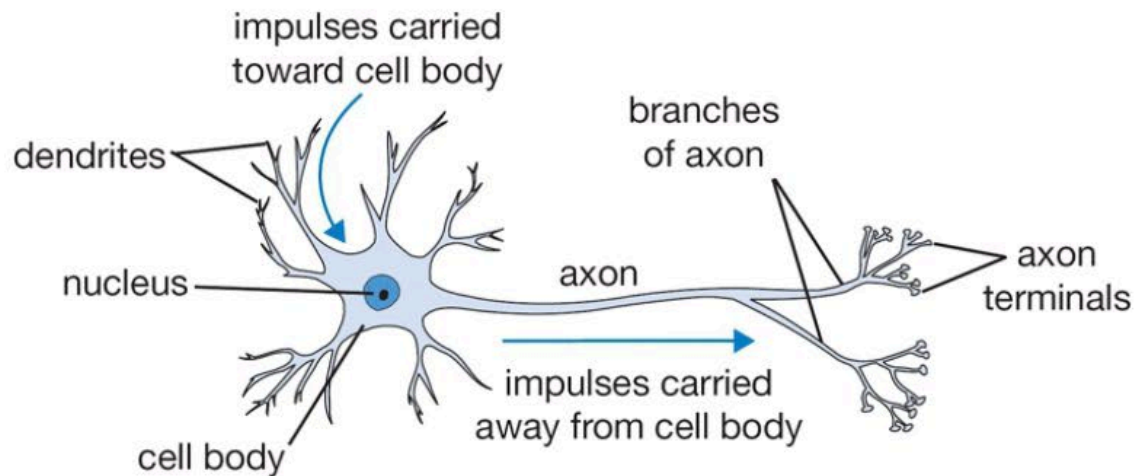
근사함수를 통한 가치함수의 매개변수화



근사함수를 통한 가치함수의 매개변수화

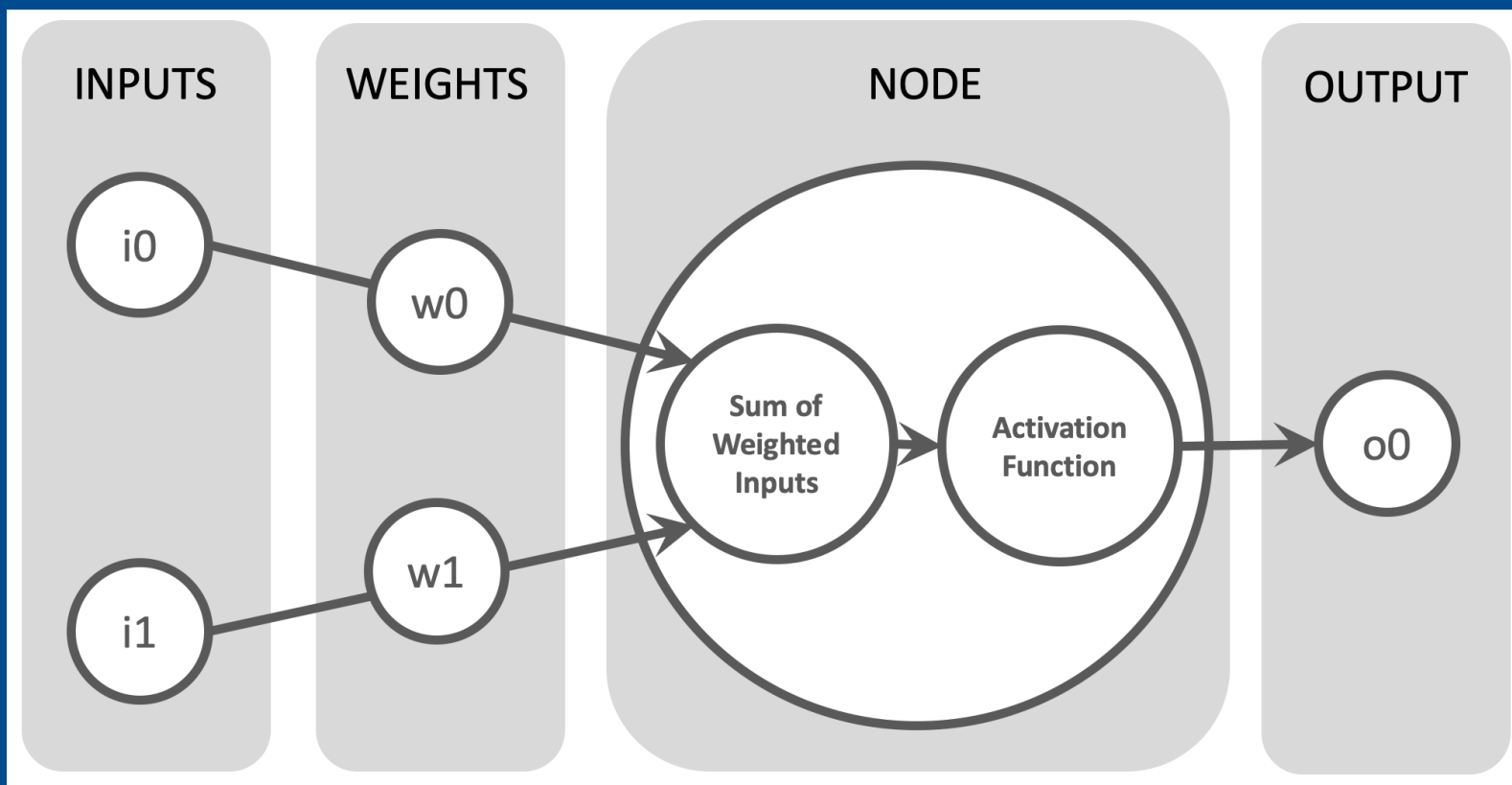


인간의 뇌를 구성하는 신경세포에서 영감을 받아 만든 수학적 모델

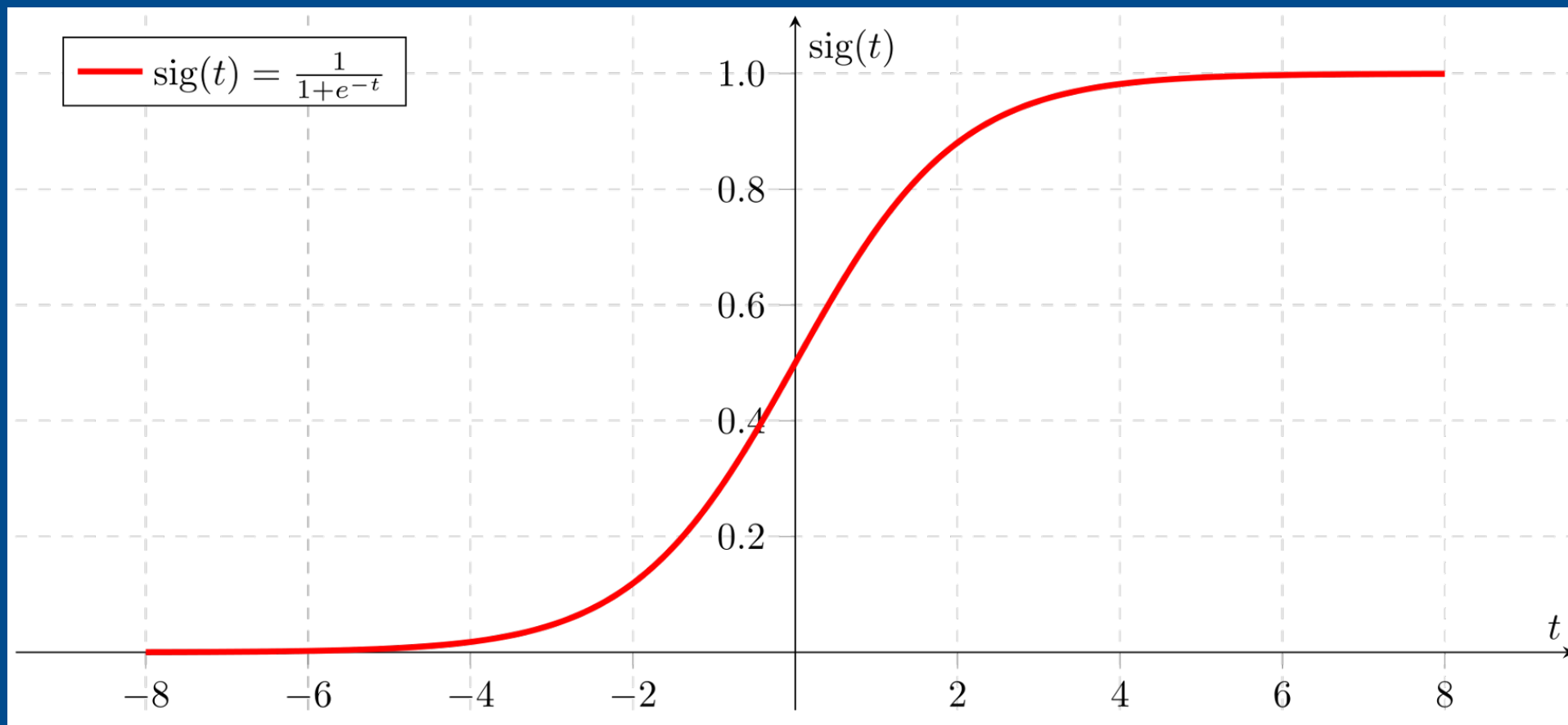


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

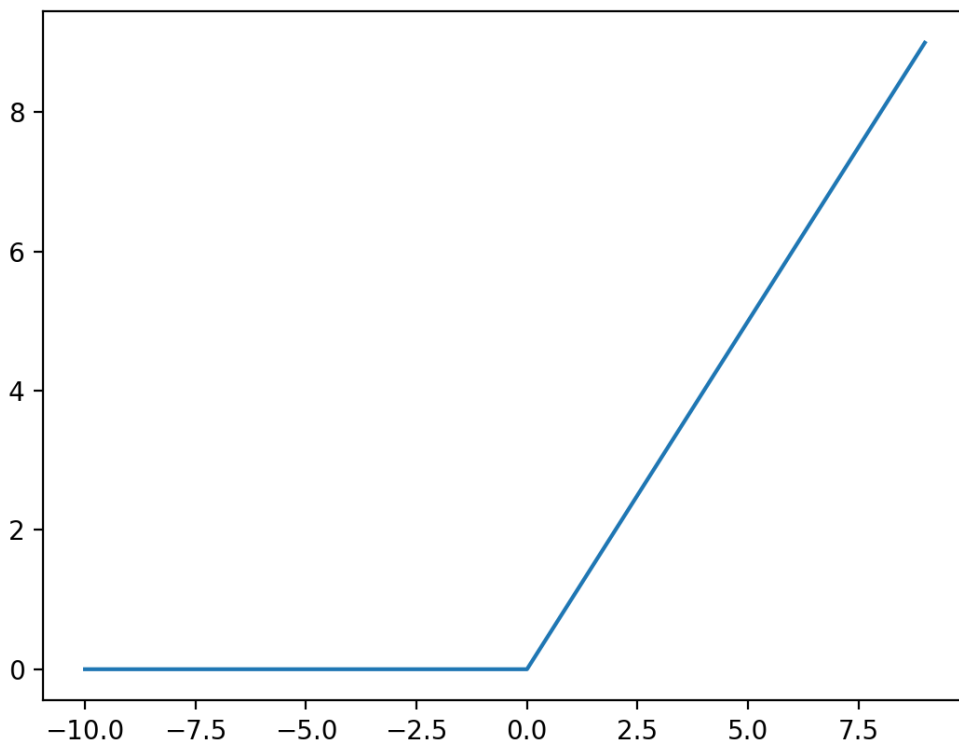
$$\text{활성함수의 입력} = \sum (\text{입력}) \times (\text{가중치}) + \text{편향}$$



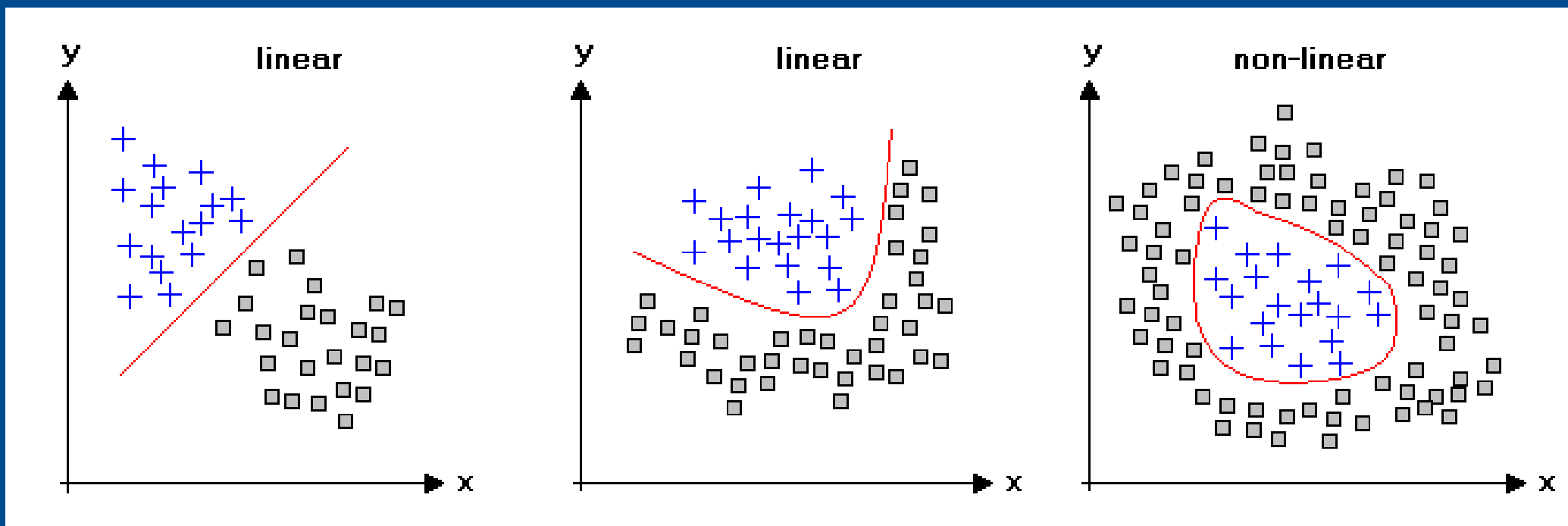
활성함수 - Sigmoid 함수



활성함수 - ReLU 함수

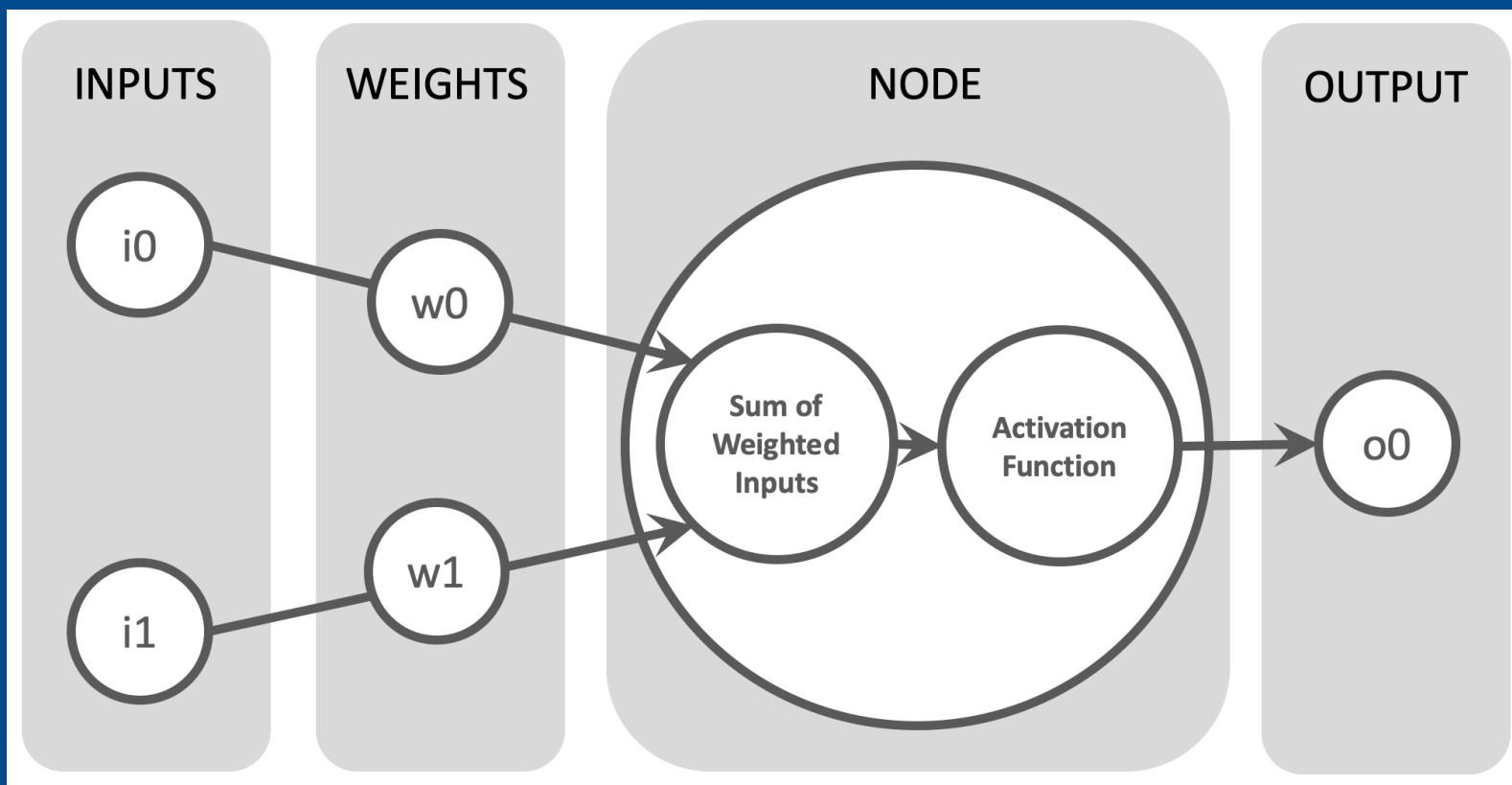


Sigmoid 함수와 ReLU 함수의 공통점 : 비선형(Nonlinear) 함수
→ 비선형 함수를 노드의 활성화함수로 사용하는 이유?



예측 : 입력이 심층신경망을 통과해서 나온 출력

타깃 : 학습 데이터의 정답



오차함수(Loss Function) : 타깃과 예측의 오차를 계산하는 함수

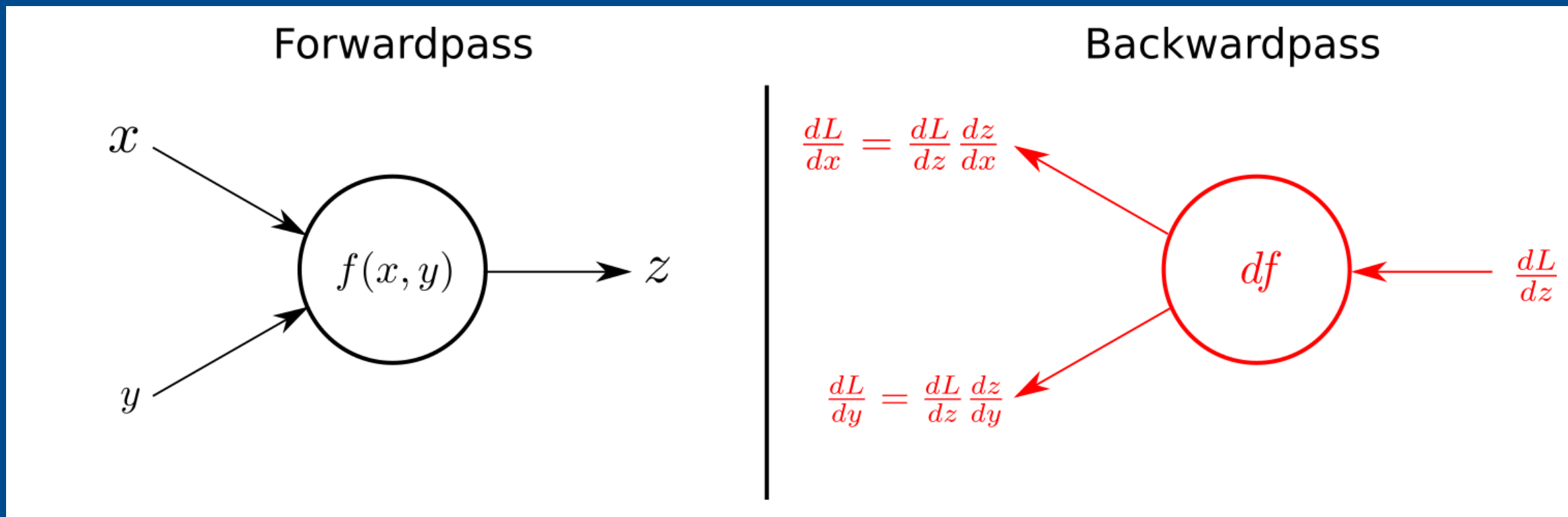
→ 대표적인 오차함수 : MSE(Mean Squared Error)!

$$\text{오차} = (\text{타깃} - \text{예측})^2$$

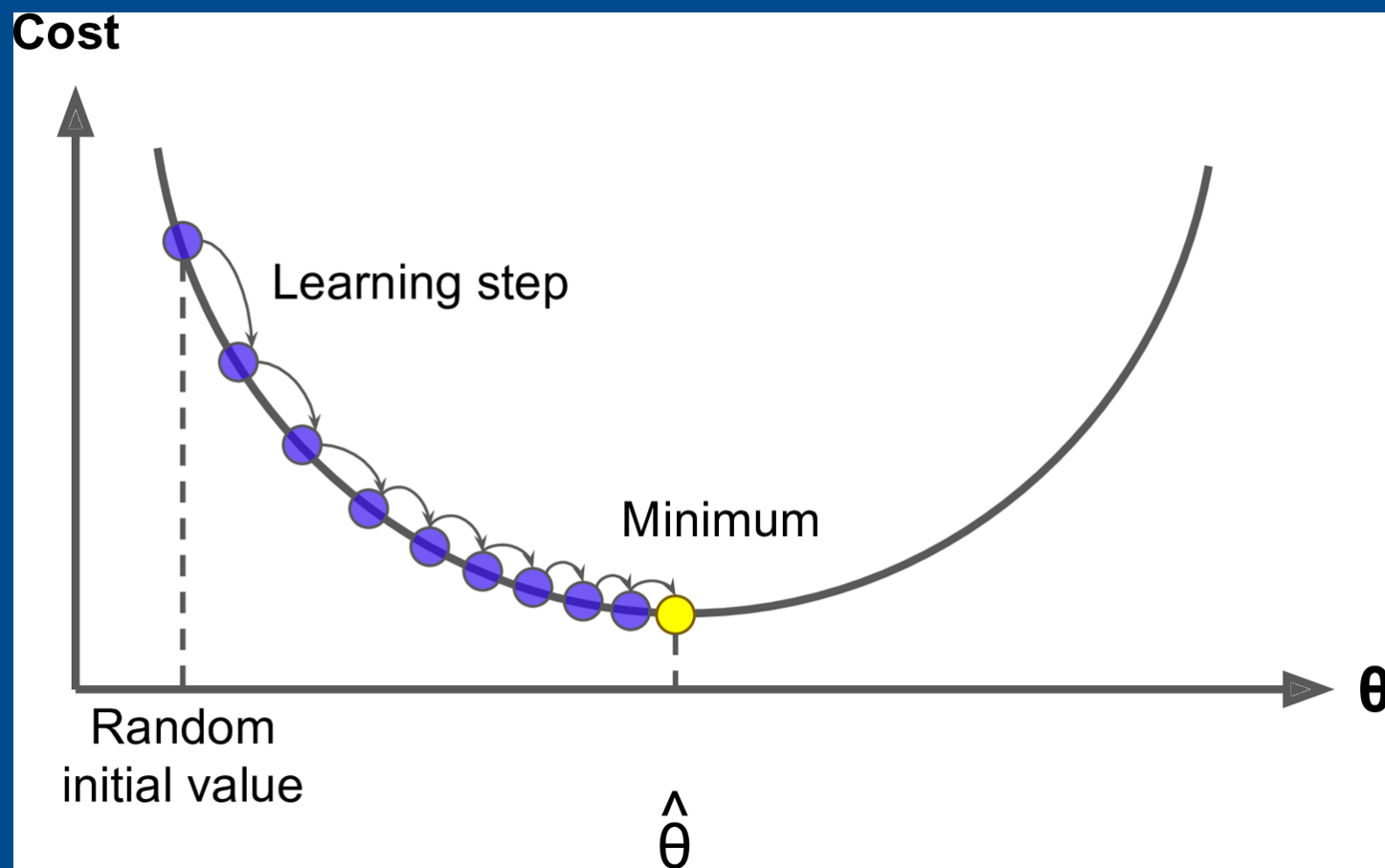
오차함수를 통해 계산한 오차를 최소화하도록
심층신경망의 가중치와 편향을 업데이트한다.

→ 어떻게 업데이트할까?

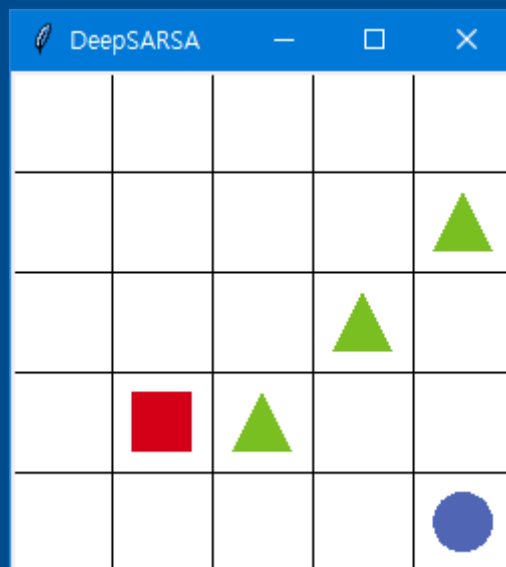
역전파(Back-propagation) 알고리즘



경사 하강법(Gradient Descent)



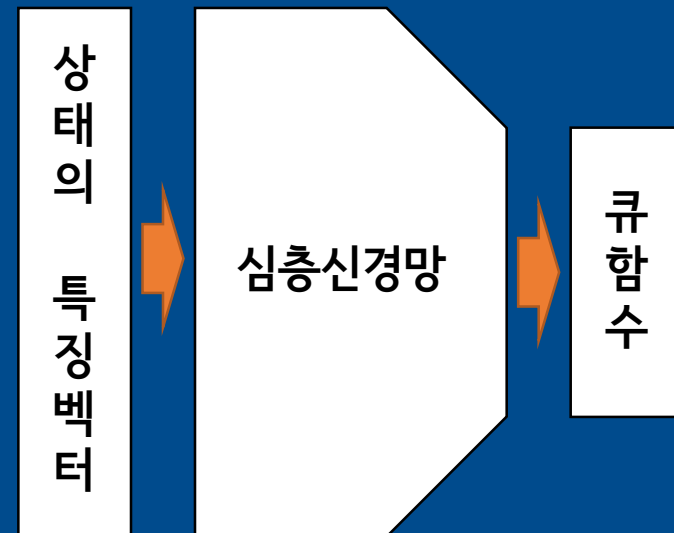
환경이 복잡하면 기존에 사용했던 살사 알고리즘으로 풀기는 어렵다.



- 움직이는 세 삼각형의 경우의 수 5
- 빨간 사각형이 있을 위치의 수 25
→ 총 상태의 개수 125가지

하지만 큐함수를 인공지능망으로 근사할 수 있다.

→ 딥살사 = 살사 알고리즘 + 인공지능망



우선 MDP를 정의해야 한다.

- 상태 이외의 다른 요소는 이전의 그리드월드 예제와 유사하다.
- 따라서 상태에 대해 정의해 보자.

상태를 정의하기 위해 어떤 정보가 필요할까?

→ 장애물을 피하려면 장애물의 상대적인 거리와 방향이 필요

따라서 상태를 다음과 같이 정의할 수 있음

- 에이전트에 대한 도착지점의 상대 위치 x, y
- 도착지점의 라벨
- 에이전트에 대한 장애물의 상대 위치 x, y
- 장애물의 라벨
- 장애물의 속도

입력이 준비됐으니 이제 정답을 알아보자.

살사의 큐함수 업데이트 식은

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

위 수식에서 큐함수의 업데이트에서 정답의 역할을 하는 것은

$$R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$$

예측에 해당하는 것은

$$Q(S_t, A_t)$$

인공신경망의 출력은 값이므로 선형 함수를 사용한다.

오차함수로는 가장 많이 쓰이는 MSE를 이용해 큐함수를 업데이트한다.

$$\begin{aligned} \text{MSE} &= (\text{정답} - \text{예측})^2 \\ &= (R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))^2 \end{aligned}$$

이제 위 오차함수를 이용해 큐함수를 업데이트 할 수 있다.

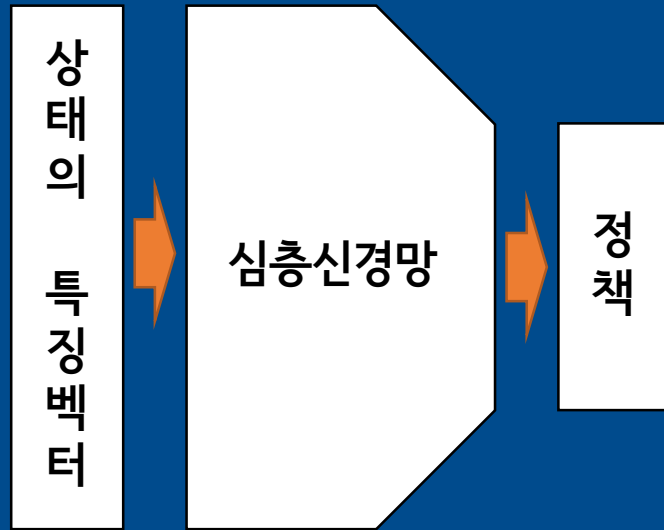
지금까지의 강화학습 알고리즘은 가치 함수를 바탕으로 동작

→ 가치 기반 강화학습(Value-based Reinforcement Learning)

한편, 정책을 기반으로 한 강화학습 알고리즘도 생각해볼 수 있다.

→ 정책 기반 강화학습(Policy-based Reinforcement Learning)

- 상태에 따라 바로 행동을 선택한다.
→ 가치함수를 토대로 행동을 선택하지 않는다.
- 정책을 직접적으로 근사한다.



- 인공신경망으로 정책을 근사하고, 인공신경망의 출력은 정책이 된다.

정책을 근사하는 인공신경망의 출력층 활성화함수는 Softmax를 사용한다.

→ 가장 최적의 행동을 선택하는 분류 문제로 생각할 수 있다.

Softmax 함수의 식은 다음과 같다.

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

위 수식에서 $s(y_i)$ 는 에이전트가 i 번째 행동을 할 확률이 된다.

정책을 인공신경망으로 근사했기 때문에 정책을 다음과 같이 표현할 수 있다.

$$\pi_{\theta}(a|s)$$

- 정책을 인공신경망으로 근사했기 때문에 θ 는 정책 신경망의 가중치가 된다.
- 목표함수는 $J(\theta)$ 로 표현할 수 있다.

강화학습의 목표는 누적 보상을 최대로 하는 최적 정책을 찾는 것이다.

따라서 정책 기반 강화학습의 목표를 수식으로 표현하면 다음과 같다.

$$\text{Maximize } J(\theta)$$

목표함수 $J(\theta)$ 의 최대화는 미분을 통해 미분한 값에 따라 업데이트 하면 된다.

→ 경사를 따라 올라가는 것이므로 “경사상승법”이라고 한다.

미분을 통해 정책 신경망을 업데이트 해보자.

어느 시간 $t + 1$ 에서 정책 신경망의 계수 θ_{t+1} 은 다음과 같이 구할 수 있다.

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta)$$

목표함수는 $J(\theta) = v_{\pi}(s_0)$ 이므로 목표함수의 미분은 다음과 같다.

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} v_{\pi}(s_0)$$

계속하면...

$$\nabla_{\theta} J(\theta) = \sum_s d_{\pi_{\theta}}(s) \sum_a \nabla_{\theta} \pi_{\theta}(a|s) q_{\pi}(s, a)$$

- $d_{\pi_{\theta}}(s)$ 는 s 라는 상태에 에이전트가 있을 확률
- 정책에 따라 각 상태에 에이전트가 있을 확률이 달라진다.
- 위 함수는 가능한 모든 상태에 대해 각 상태에서 특정 행동을 했을 때 큐함수의 기댓값
→ 에이전트의 선택에 대한 좋고 나쁨의 지표

계속하면...

$$\nabla_{\theta} J(\theta) = \sum_s d_{\pi_{\theta}}(s) \sum_a \pi_{\theta}(a|s) \frac{\nabla_{\theta} \pi_{\theta}(a|s)}{\pi_{\theta}(a|s)} q_{\pi}(s, a)$$

log 미분으로 표현하면

$$\nabla_{\theta} J(\theta) = \sum_s d_{\pi}(s) \sum_a \pi_{\theta}(a|s) \times \nabla_{\theta} \log \pi_{\theta}(a|s) q_{\pi}(s, a)$$

이를 기댓값의 형태로 표현할 수 있다.

$$\nabla_{\theta} J(\theta) = E_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) q_{\pi}(s, a)]$$

최종적으로 폴리시 그레이디언트의 업데이트 식은

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta) \approx \theta_t + \alpha [\nabla_{\theta} \log \pi_{\theta}(a|s) q_{\pi}(s,a)]$$

하지만 에이전트에 가치함수나 큐함수가 없기 때문에 $q_{\pi}(s,a)$ 를 구할 수 없다.



목표함수의 미분값 $\nabla_{\theta} J(\theta)$ 를 잘 근사하는 게 중요하다.

→ 가장 고전적인 방법으로는 큐함수를 반환값 r_t 로 대체하는 방법이 있다.
이를 REINFORCE 알고리즘이라고 한다.

REINFORCE 알고리즘의 업데이트 식은 다음과 같다.

$$\theta_{t+1} \approx \theta_t + \alpha [\nabla_{\theta} \log \pi_{\theta}(a|s) r_t]$$

- REINFORCE 알고리즘에선 실제로 얻은 보상으로 학습한다.
→ 몬테카를로 폴리시 그레이디언트라고도 한다.

오차함수의 관점에서 보자.

분류 문제에서 가장 많이 쓰이는 오류함수인 크로스 엔트로피는 다음과 같다.

$$-\sum_i y_i \log p_i$$

- y_i 와 p_i 가 얼마나 가까운지를 나타낸다.
- y_i 와 p_i 가 같아지면 식의 값은 최소가 된다.
- 지도학습에선 y_i 는 정답, p_i 는 예측값을 사용한다.

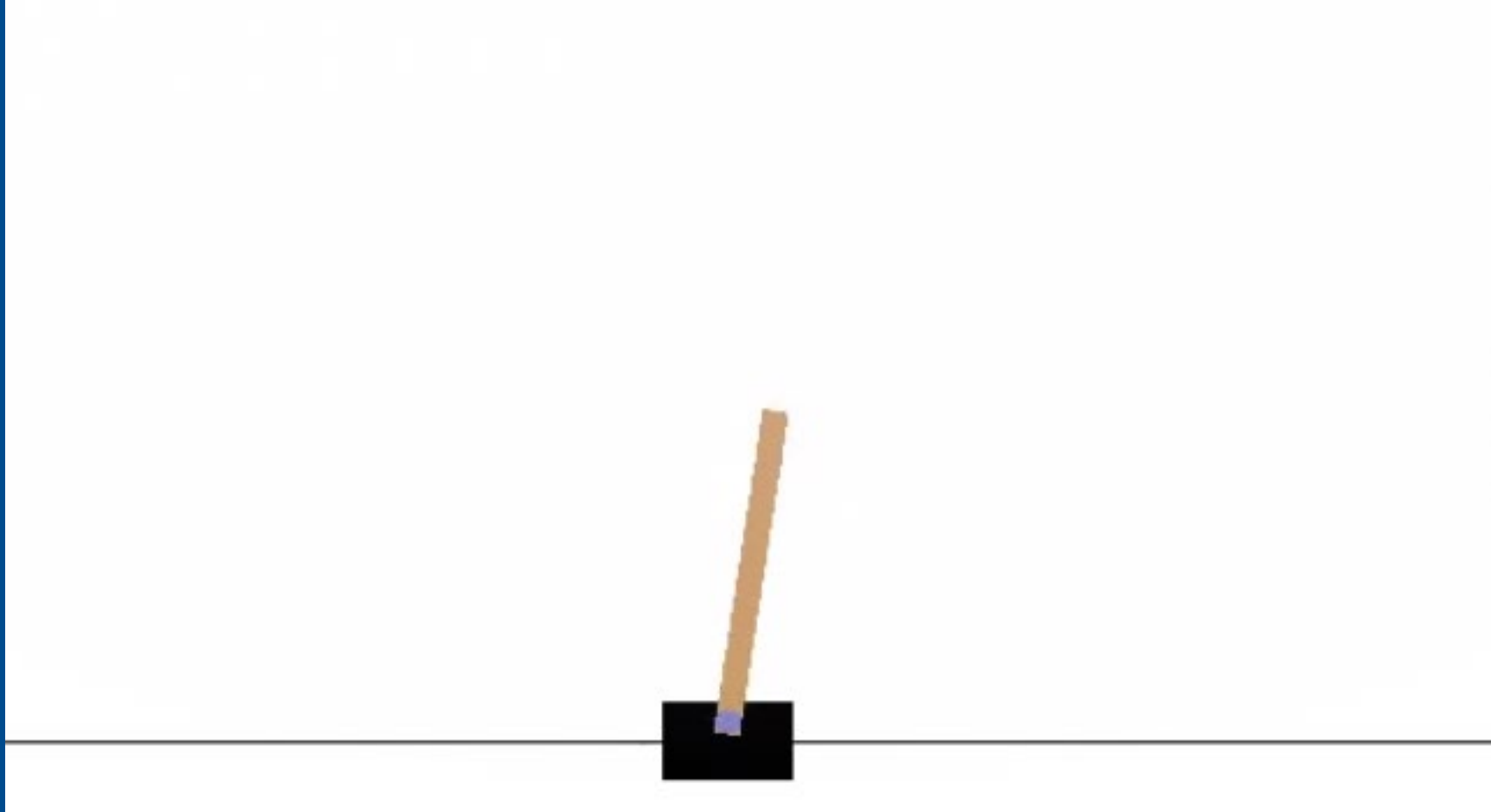
REINFORCE의 오류함수는 다음과 같다.

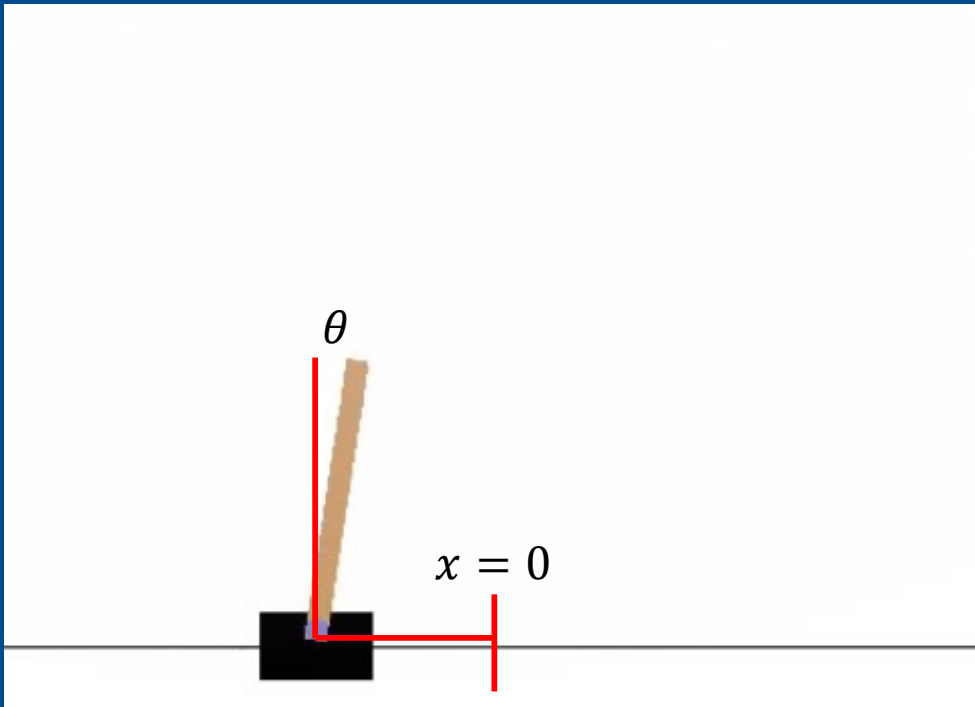
$$\log \pi_{\theta}(a|s) - J_{\theta}$$

- 크로스 엔트로피와 위 수식은 비슷하게 생겼다.
- $\log \pi_{\theta}(a|s)$ 는 실제로 한 행동을 정답으로 둔 것임
- 하지만 잘못된 선택을 할 수도 있어 반환값을 곱해준다.
→ 부정적인 보상을 받게 됐다면 그 행동을 선택할 확률을 낮춘다.

카트폴(Cartpole)

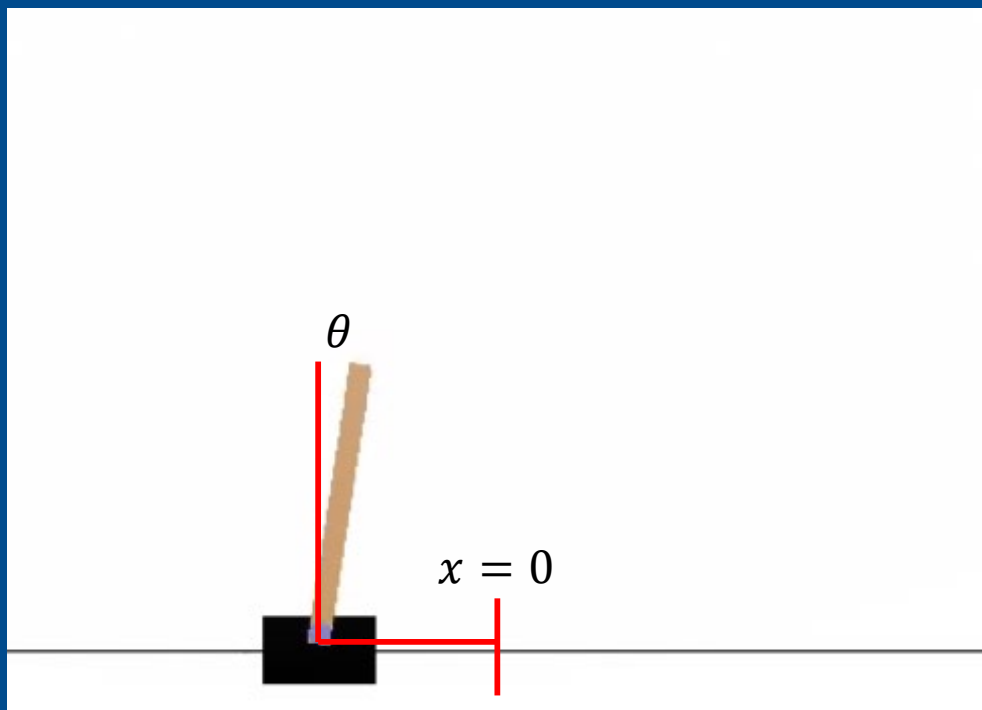
2019 OSS Summer
강화 학습 기초 - Day 2





검은색 상자(카트)를 +1, -1의 힘으로 밀어
갈색 막대(폴)이 넘어지지 않게 하는 문제

카트의 위치 x 가 ± 2.5 밖으로 넘어가거나
 θ 가 $\pm 15^\circ$ 밖으로 넘어가면 에피소드가 끝난다.



상태 s

$$s = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} \begin{array}{l} \text{위치} \\ \text{속도} \\ \text{각도} \\ \text{각속도} \end{array}$$

행동 a

$$a = \begin{bmatrix} +1 \\ -1 \end{bmatrix}$$

상태의 위치, 속도, 각도, 각속도 모두 연속적인 실수다.

그러므로 카트폴의 경우에는 테이블을 만들어 풀 수 없고,

큐함수를 근사하는 함수를 만들어야 한다. → 인공신경망을 이용

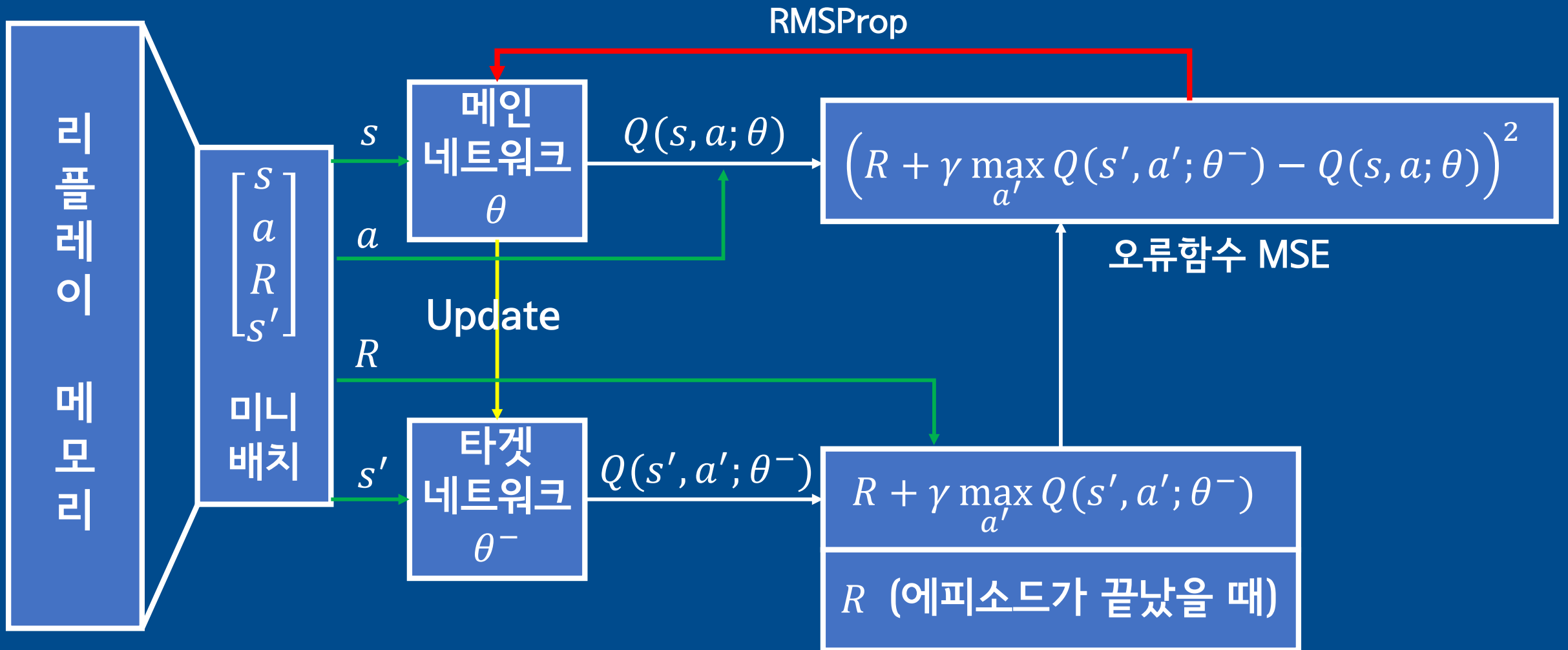
딥살사 : 온폴리시(On-Policy)인 살사 기반

오프폴리시(Off-Policy)인 큐러닝 + 인공신경망? → 딥-큐러닝(DQN)



DQN 구조(맞보기)

2019 OSS Summer
강화 학습 기초 - Day 2



DQN의 특징

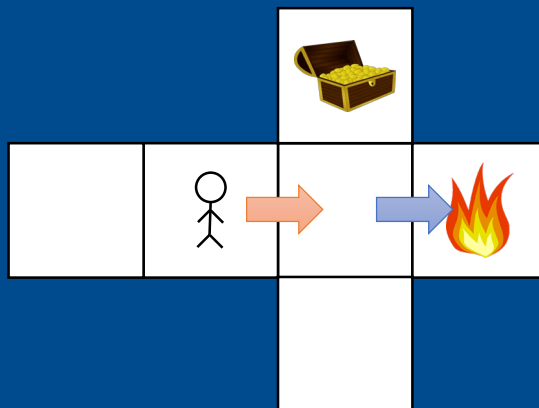
1. 오프폴리시(Off-Policy)
2. 리플레이 메모리 + 미니배치
3. 타겟 신경망

온폴리시 (예 : 살사)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

학습하는 정책과 행동을 고르는 정책이 항상 같아야 한다.

- 다음 상태에서 한 안 좋은 행동이 현재에 큰 영향을 미친다.
- 정책을 업데이트하면 과거의 경험들을 학습에 이용 불가능해 비효율적이다.



오프폴리시 (예 : 큐러닝)

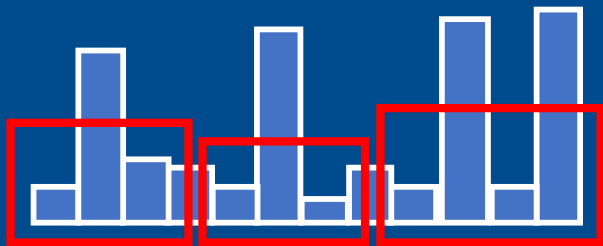
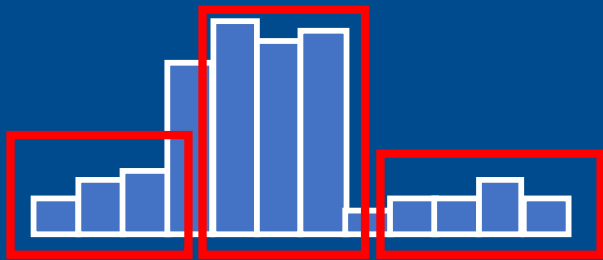
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t))$$

학습하는 정책과 행동을 고르는 정책이 달라도 된다.

과거에 경험한 에피소드들도 학습에 계속해서 이용 가능하다. → 효율적

리플레이 메모리 + 미니배치

2019 OSS Summer
강화 학습 기초 - Day 2



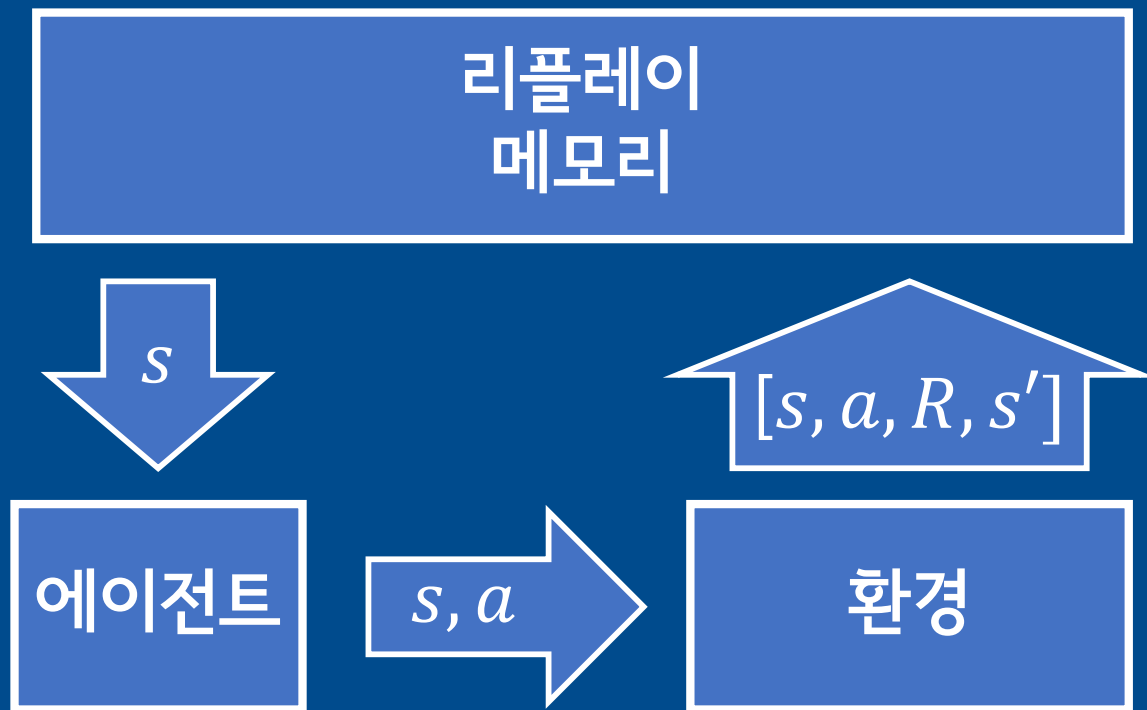
강화학습이 잘 일어나기 위해서는 에이전트가
알 수 없는 데이터들 사이의 연관성이 없어야 한다.

하지만 게임과 같은 환경에서는 데이터들이 행동에
대한 연속적인 결과이기 때문에 연관성이 심하다.

그래서 데이터들을 리플레이 메모리에 저장하고,
이 중 샘플을 고르는 미니배치 방법을 사용한다.

리플레이 메모리 + 미니배치

2019 OSS Summer
강화 학습 기초 - Day 2



리플레이 메모리는 환경에서 받은 $[s, a, R, s']$ 을 저장한다.

받은 s' 를 새로운 s 로 에이전트에게 전달해 에피소드를 계속 진행시킨다.

리플레이 메모리 + 미니배치

2019 OSS Summer
강화 학습 기초 - Day 2

리플레이
메모리



샘플

$[s, a, R, s']$

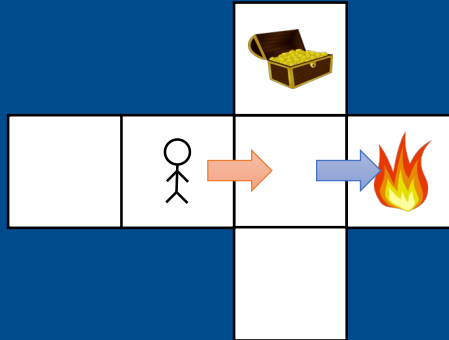
$[s, a, R, s']$

$[s, a, R, s']$

...

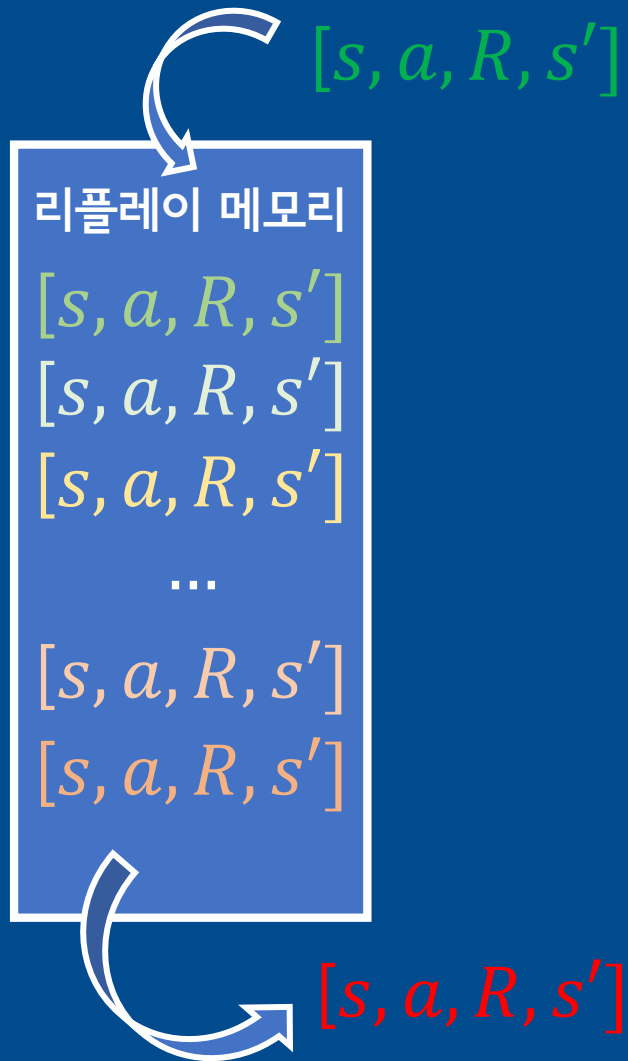
리플레이 메모리에 저장되어 있는 $[s, a, R, s']$ 집합에서 일부를 무작위로 샘플링하고, 이것으로 에이전트를 학습시킨다.

이렇게 하면 샘플 사이의 시간적 상관관계가 줄어들고, 살사에서 발생했던 문제를 줄일 수 있다.



리플레이 메모리 + 미니배치

2019 OSS Summer
강화 학습 기초 - Day 2



사용할 수 있는 메모리의 크기는 유한하므로,
리플레이 메모리의 크기에 제한을 두어야 한다.

제한 이상으로 경험이 추가되면 오래된 순서대로
리플레이 메모리에서 지워준다.

그러면 좋지 않은 과거의 경험이 줄어들면서,
더 좋은 경험이 샘플로 뽑힐 확률이 증가한다.

DQN에서도 딥살사와 비슷하게 오류함수로 MSE를 사용한다.

DQN 에이전트가 학습할 때 사용하는 오류함수는 아래와 같았다. (2013년)

$$MSE = (\text{정답} - \text{예측})^2 = \left(R_{t+1} + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta) \right)^2$$

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

하지만 위 식에는 문제점이 있다. 인공신경망이 스스로 목표를 만들어 내기 때문에 인공신경망이 업데이트될 때 목표가 되는 정답 부분이 계속 변하고, 그 결과 학습이 굉장히 불안하게 이루어진다.

그래서 θ^- 를 매개변수로 갖는 타겟 신경망을 추가한다. (2015년)

$$MSE = (\text{정답} - \text{예측})^2 = \left(R_{t+1} + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta^-) \right)^2$$

$$\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$$

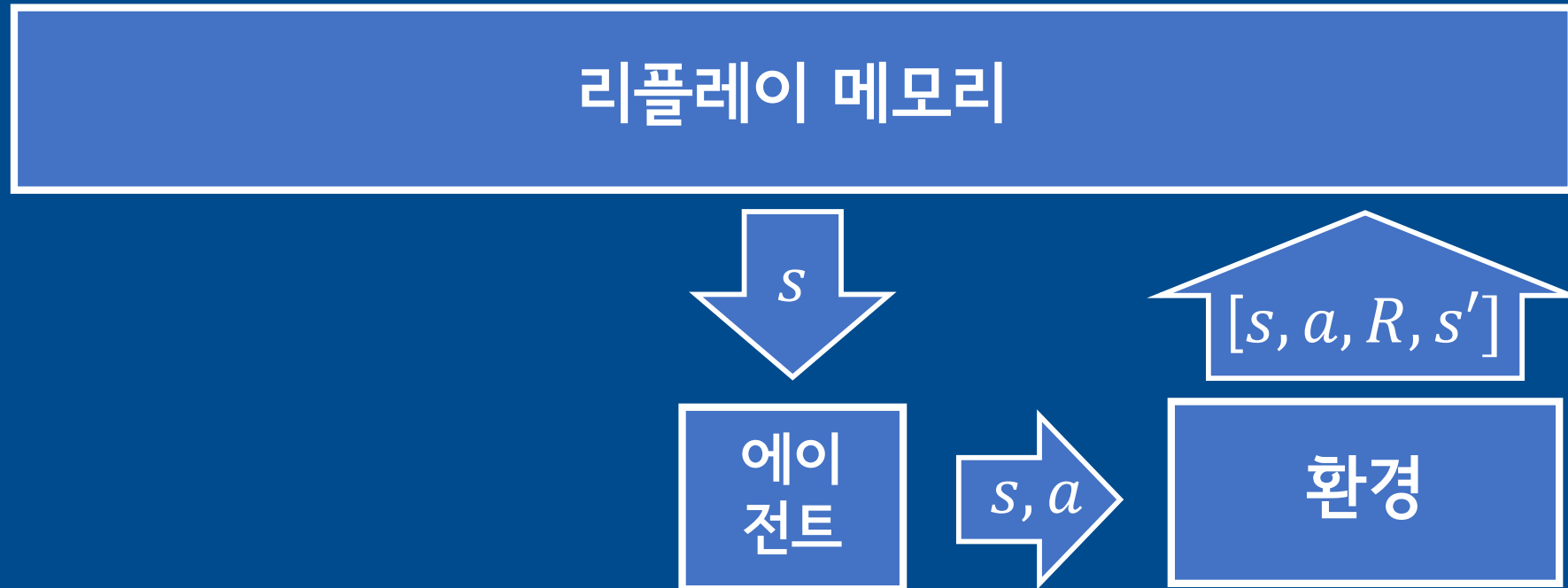
타겟 신경망은 일정 시간동안 그대로 유지되며 정답을 만들어내다가, 에피소드가 끝날 때마다 업데이트된다.

→ 이를 통해 학습의 불안정성을 줄일 수 있다.

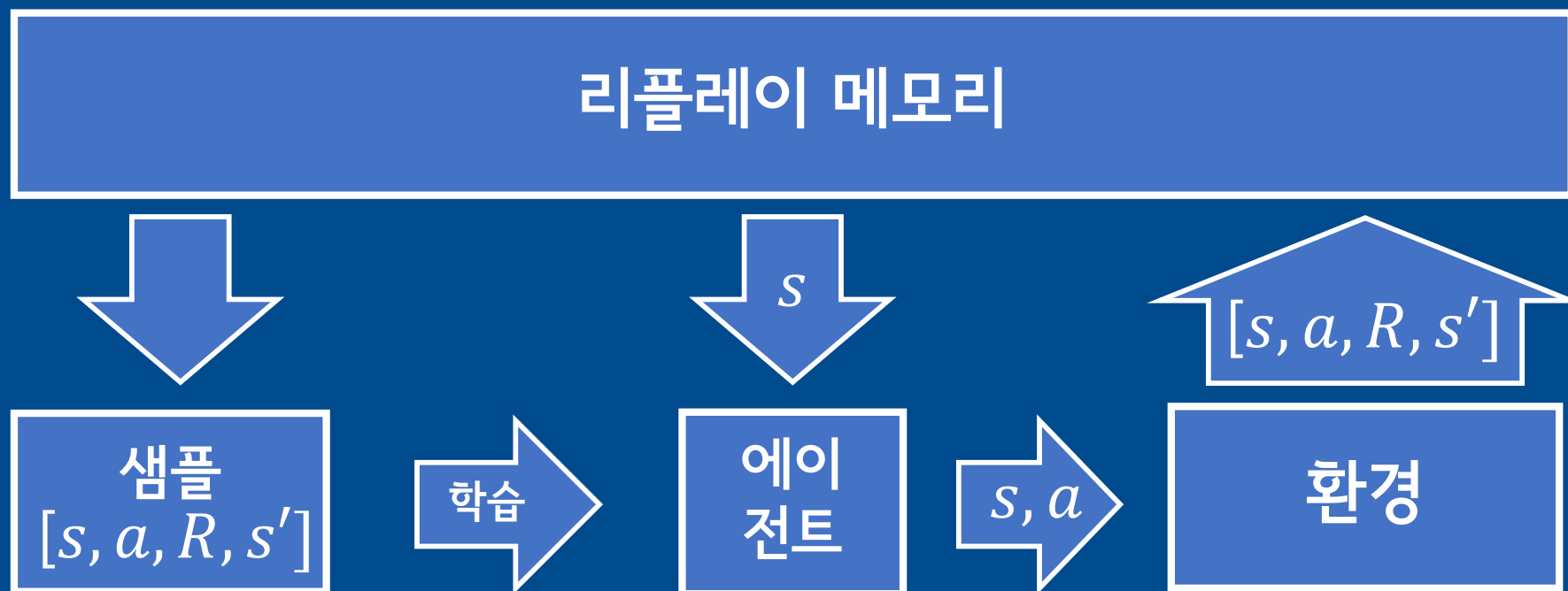
1. 상태에 따라 행동을 선택한다.



2. 선택한 행동으로 환경에서 한 스텝을 진행하고,
샘플을 리플레이 메모리에 저장한다.



3. 리플레이 메모리에서 추출한 샘플로 에이전트를 학습시킨다.
4. 에피소드마다 타겟 모델을 업데이트한다.



감사합니다

<http://github.com/utilForever>

질문 환영합니다!