Reinforcement Learning

HOML – chapter18

TAVE Research DL001

Changdae Oh

2021. 02. 21

Topics

1. What is Reinforcement Learning
2. MDP & Value function
3. Value-based RL
4. Policy-based RL

Topics

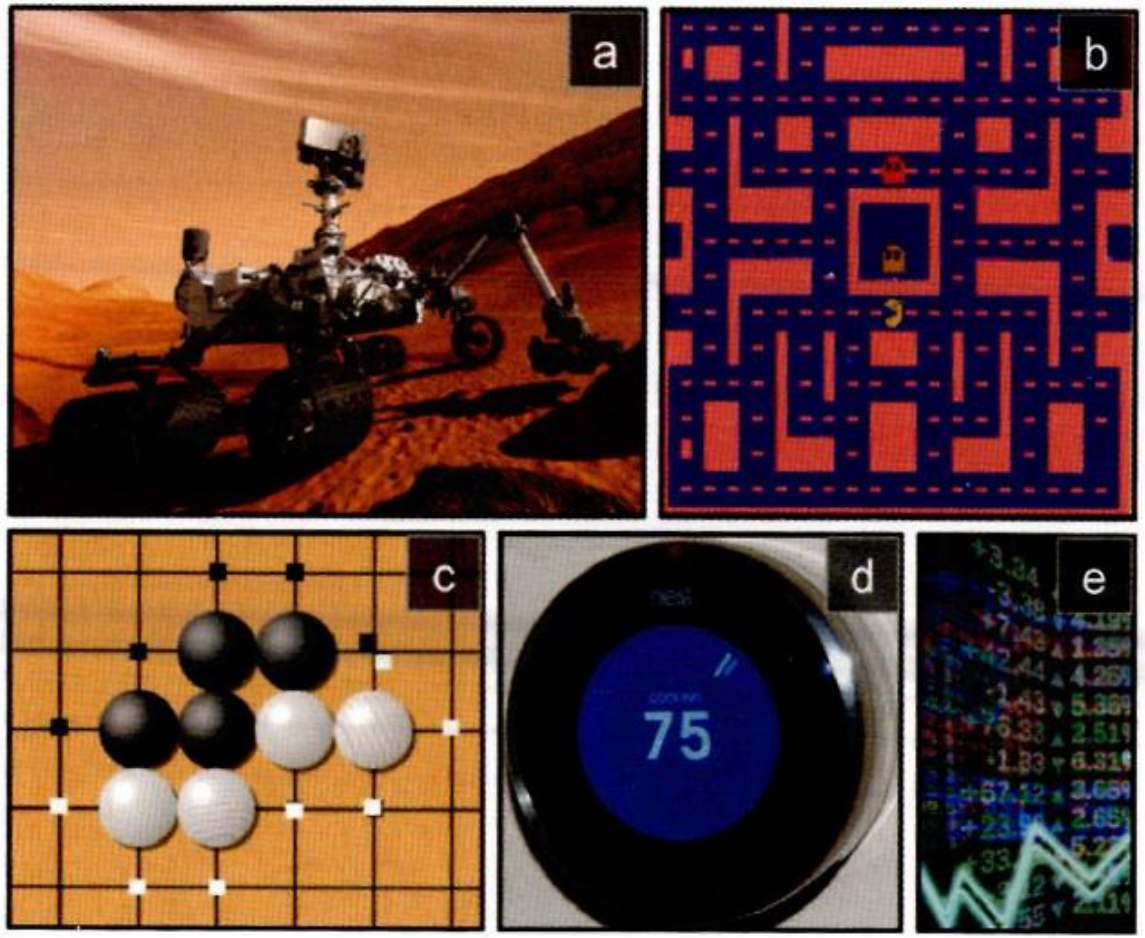
1. What is Reinforcement Learning
2. MDP & Value function
3. Value-based RL
4. Policy-based RL

What is RL?

Reinforcement Learning

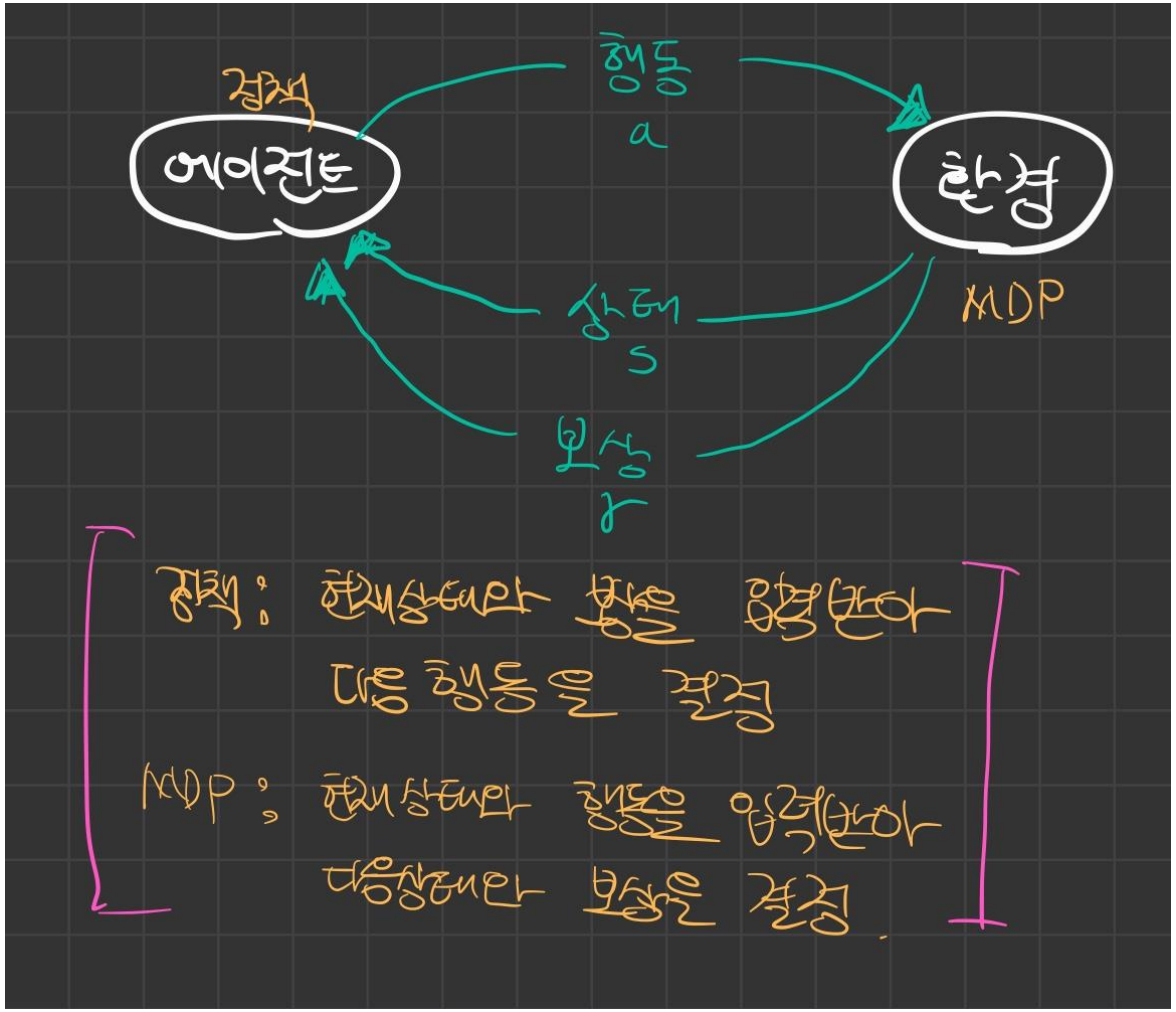
강화학습은 **에이전트와 환경**이 action, state, reward라는 정보를 통해 **상호작용**하면서 목표를 달성하는 학습 과정이다.

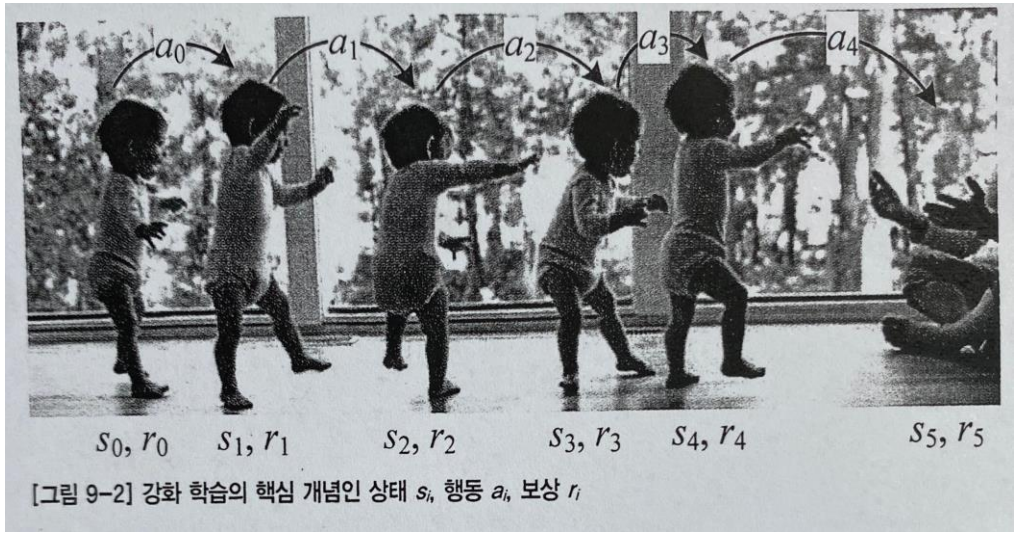
목표는 보상(reward)의 장기간 기대치**를 최대화**하는 것이다.



- There is no supervisor, only a reward signal
- Feedback is delayed. Not instantaneous
- Time really matters

What is RL?





[그림 9-2] 강화 학습의 핵심 개념인 상태 s_i, 행동 a_i, 보상 r_i

핵심연산

$$f : (s_t, a_t) \rightarrow (s_{t+1}, r_{t+1})$$

What is RL?

Policy Search

정책(policy) : 에이전트가 행동을 결정하기 위해 사용하는 알고리즘

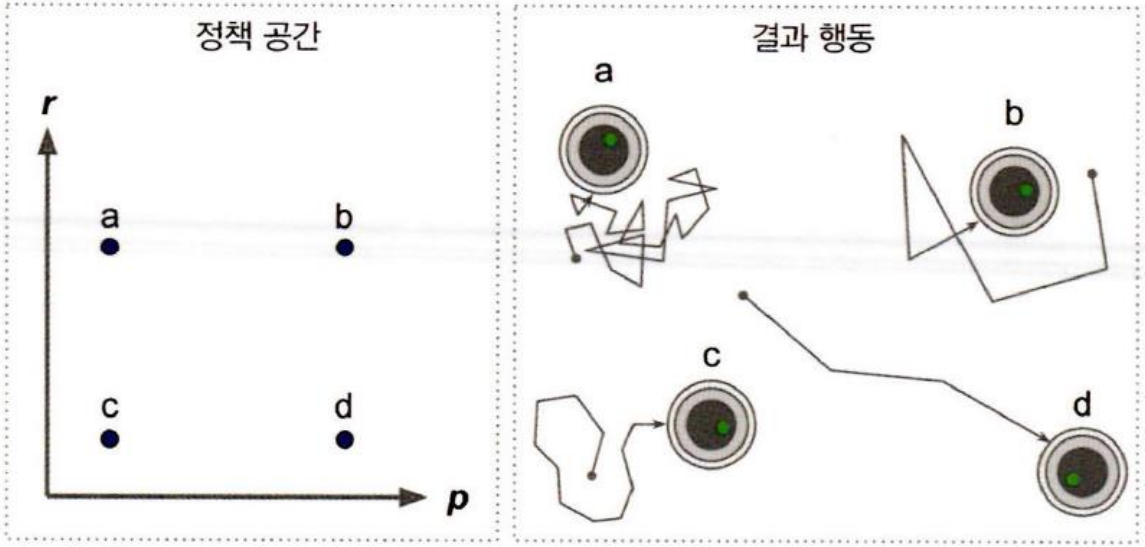
- Deterministic Policy : 어떤 상태에 대해 항상 동일한 action을 반환
- Stochastic Policy : 어떤 상태에 대해 action들의 **확률분포를 반환**

지도학습에서 경사하강법 등을 이용하여 최적해를 찾아가듯,
강화학습도 적절한 최적화 알고리즘을 사용하여 최적의 정책을 찾아야 한다.

Ex) 로봇 진공청소기의 정책

- 매초 p의 확률로 전진, 1-p의 확률로 왼쪽/오른쪽 랜덤 회전.
- 회전각도는 (-r, r) 사이 랜덤하게 결정.

정책 파라미터(policy parameter) : p, r
정책 공간(policy space) : (0 < p < 1, r > 0)



What is RL?

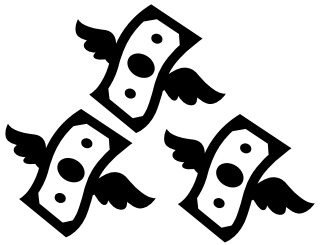
Credit assignment problem

강화학습에서 에이전트가 얻을 수 있는 가이드는 "행동에 따른 보상" 뿐이다.
But, 보상은 일반적으로 드물고, **지연되어(delayed) 나타남**

So, 어떤 시점의 특정 행동을 평가하는 것은 어려운 문제

Discount Factor

현재와 미래의 상대적 중요도를 조율



1

현재 시점 1억의 가치

γ

10년 뒤 1억의 가치

γ^2

20년 뒤 1억의 가치

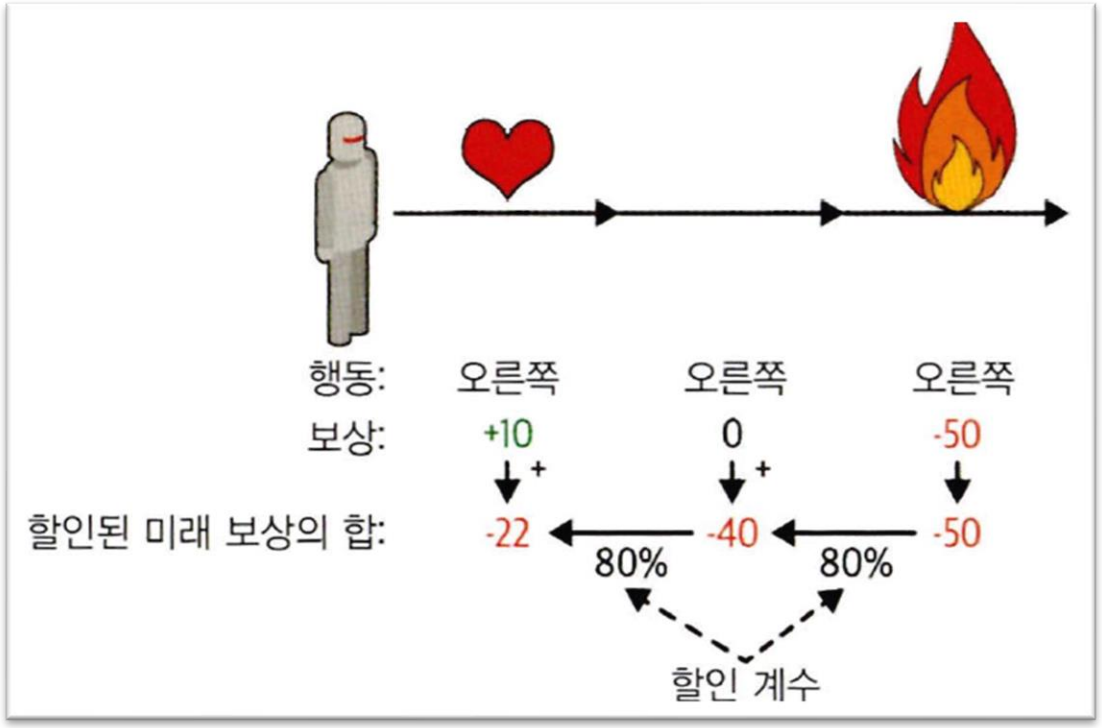
행동 발생 후 각 단계마다 할인 계수를 적용한 보상을 모두 합하여 특정 시점의 행동을 평가하자!

➡ **할인된 미래 보상의 합(Sum of discounted reward)** (혹은 Accumulative reward)

What is RL?

Action advantage

우리는 할인된 미래 보상의 합으로
행동 이익(action advantage)을 계산할 수 있다.



아래와 같은 과정으로 **행동을 평가**할 수 있다..

1. 충분히 많은 에피소드 실행
2. 발생한 모든 행동들의 return을 구하여 평균, 표준편차 구함
3. 각 행동들의 return을 **정규화**한다.
4. 행동이익이 음수인 행동은 나쁘고, 양수인 행동은 좋다고 가정할 수 있다.

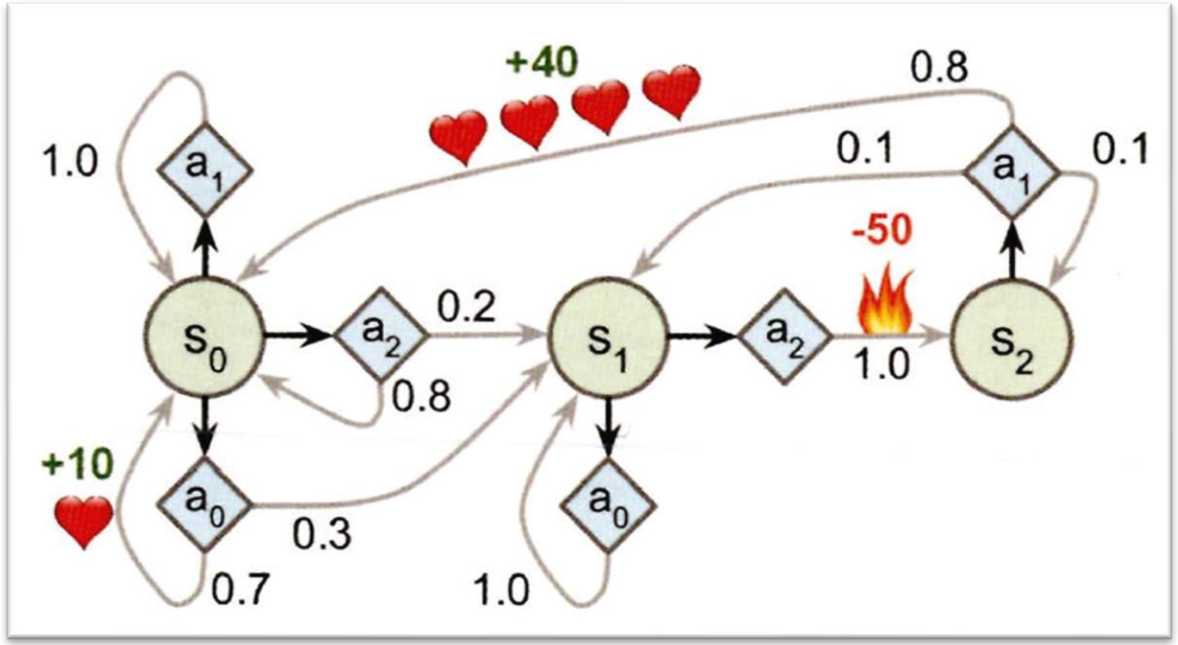
Topics

1. What is Reinforcement Learning
2. MDP & Value function
3. Value-based RL
4. Policy-based RL

MDP & Value function

Markov Decision Process

고정된 전이확률 P, 할인율 gamma, 보상 R이 정의 되어있을 때,
확률적으로 얻어지는 s, a, r의 시퀀스 (stochastic process)




```
# 상태 s에서 행동 a를 하여 상태 s'로 전이할 확률
transition_probabilities = [ # shape=[s, a, s']
        [[0.7, 0.3, 0.0], [1.0, 0.0, 0.0], [0.8, 0.2, 0.0]],
        [[0.0, 1.0, 0.0], None, [0.0, 0.0, 1.0]],
        [None, [0.8, 0.1, 0.1], None]]
rewards = [ # shape=[s, a, s']
        [[+10, 0, 0], [0, 0, 0], [0, 0, 0]],
        [[0, 0, 0], [0, 0, 0], [0, 0, -50]],
        [[0, 0, 0], [+40, 0, 0], [0, 0, 0]]]
possible_actions = [[0, 1, 2], [0, 2], [1]]
```

➡ 환경에 대한 확률모델 $P(s', r|s, a)$

MDP & Value function

State Value

어떤 상태 s의 상태가치 :
에이전트가 상태 s에 도달한 후 종료 상태에 이르기까지 누적 보상액의 추정치

$$V^{\pi}(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots | s_0 = s, \pi]$$

최적 상태가치 : 에이전트가 이후의 step들에서 항상 최적으로 행동한다고 가정

$$V^{*}(s) = \max_{\pi} V^{\pi}(s)$$

Bellman optimality equation

$$V^{*}(s) = \max_{a} \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \cdot V^{*}(s') \right]$$ 모든 s에 대해

$$V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \cdot V_k(s') \right]$$ 모든 s에 대해 (가치 반복 알고리즘)

MDP & Value function

✓ 최적의 상태가치를 아는 것은 정책을 평가할 때 유용하다.
✓ 그러나 최적의 정책을 알려주지는 않는다.

➡ **Q-value** 상태-행동 가치(state-action value)

상태-행동의 튜플 (s, a)쌍에 대한 **최적의 Q-value**는
에이전트가 상태 s에 도달해서 행동 a를 선택한 후
종료상태에 이르기까지 최적의 결정을 반복한다고 가정할 때,
평균적으로 기대할 수 있는 누적 보상액

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \cdot \max_{a'} Q_k(s',a')\right]$$ 모든 (s, a)에 대해

(Q가치 반복 알고리즘)

Optimal policy : $\pi^*(s) = \underset{a}{\operatorname{argmax}} Q^*(s,a)$

MDP & Value function

```python
Q_values = np.full((3, 3), -np.inf) # -np.inf for impossible actions
for state, actions in enumerate(possible_actions):
    Q_values[state, actions] = 0.0  # zero init for all possible actions

print(Q_values,'\n')

# Q-value Iteration
gamma = 0.90

for iteration in range(50):
    Q_prev = Q_values.copy()
    for s in range(3):
        for a in possible_actions[s]:
            Q_values[s, a] = np.sum([
                    transition_probabilities[s][a][sp]
                    * (rewards[s][a][sp] + gamma * np.max(Q_prev[sp]))
                for sp in range(3)])

print(Q_values,'\n')
print(np.argmax(Q_values, axis = 1)) # 각 상태에서 최선의 행동
```

Q-values & action

```
[[18.91891892 17.02702702 13.62162162]
 [ 0.                 -inf -4.87971488]
 [       -inf 50.13365013        -inf]]

[0 0 1]
```

Note

✓ Discount factor를 0.9에서 0.95로 변경(more 미래 지향적)시키면 최적의 행동이 변함

✓ 미래의 보상에 더 가치를 둘수록 미래의 이익을 위해 당장의 고통을 견디도록 행동함.

Topics

1. What is Reinforcement Learning
2. MDP & Value function
3. Value-based RL
4. Policy-based RL

Value-based RL

Dynamic programming

- 정책 반복 알고리즘
- 가치 반복 알고리즘

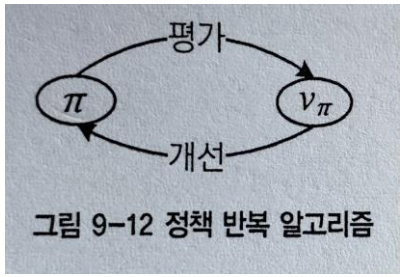


그림 9-12 정책 반복 알고리즘

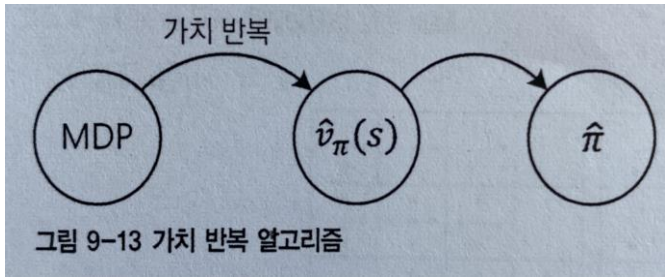


그림 9-13 가치 반복 알고리즘

✓ 초창기 강화학습이 주로 사용하던 고전적인 방법.
✓ 상태와 행동의 개수가 적어 계산 시간과 메모리 요구량이
현실적이여야 한다는 강력한 제약조건이 있음.
✓ 현실에서는 알기 힘들 수 있는 MDP 확률분포에 대한 완전한 정보를 요구함.

Monte carlo method

✓ 에이전트와 환경의 상호작용을 통해 직접 샘플을 수집하여
이를 훈련집합으로써 활용해 최적 정책을 찾아내자!

Value-based RL

Temporal difference learning

- 몬테카를로 방법과 동적 프로그래밍의 중간에 있으며, 둘의 장점을 취하는 알고리즘

  " Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics.
  Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap)"

식 18-4 TD 학습 알고리즘

$$V_{k+1}(s) \leftarrow (1-\alpha)V_k(s) + \alpha\left(r + \gamma \cdot V_k(s')\right)$$

- Sarsa
- Q-learning

Value-based RL

Q-learning

- 전이확률과 보상을 초기에 알지 못한 상황에서 Q-value 반복 알고리즘을 적용.
- 점진적으로 Q-value 추정을 향상시켜 나간다.
- 충분히 좋은 Q-value 추정을 얻게되면, 그것을 최대화하도록 행동을 선택하는 정책을 취함.

식 18-5 Q-러닝 알고리즘

$$Q(s,a) \underset{\alpha}{\leftarrow} r + \gamma \cdot \max_{a'} Q(s',a')$$

```python
def step(state, action): # f(s, a) -> s', r
    probas = transition_probabilities[state][action]
    next_state = np.random.choice([0, 1, 2], p=probas)
    reward = rewards[state][action][next_state]
    return next_state, reward
```

```python
def exploration_policy(state):
    return np.random.choice(possible_actions[state])
```

```python
Q_values = np.full((3, 3), -np.inf)
for state, actions in enumerate(possible_actions):
    Q_values[state][actions] = 0 # init

alpha0 = 0.05 # initial learning rate
decay = 0.005 # learning rate decay
gamma = 0.90 # discount factor
state = 0 # initial state

for iteration in range(10000):
    action = exploration_policy(state)
    next_state, reward = step(state, action)
    next_value = np.max(Q_values[next_state]) # greedy policy at the next step
    alpha = alpha0 / (1 + iteration * decay)
    Q_values[state, action] *= 1 - alpha
    Q_values[state, action] += alpha * (reward + gamma * next_value)
    state = next_state
```

Value-based RL

Exploration and Exploitation trade-off

탐험 : 전체 공간을 골고루 탐색 vs **탐사** : 특정한 곳 주위를 집중적으로 탐색

- 탐험은 시간이 너무 오래 걸리는 문제가 있고,
- 탐사는 다른 곳에 있는 더 좋은 해를 놓칠 가능성이 있다.

(Decaying) epsilon-greedy policy

Use exploration function

$$Q(s,a) \underset{\alpha}{\leftarrow} r+\gamma \cdot \max_{a'} f\left(Q(s',a'),N(s',a')\right)$$

$$f(Q,N)=Q+\kappa/(1+N)$$

- 더 효율적인 path 발견 가능
- 미지의 더 큰 보상 획득 가능

Value-based RL

Approximate Q-learning

Q-learning을 위해 모든 상태와 모든 행동의 확률을 표현하려면
|S| * |A| 크기의 lookup table이 필요한데, 게임이나, 바둑 등의 문제 상황에서는
메모리 자원 측면에서 infeasible하다. -> 모든 Q-value에 대한 추정값을 기록할 수 없음

➡ 어떤 상태-행동(s,a)쌍의 Q-value를 근사하는 함수 $\mathbf{Q_{theta}}(s, a)$를
적절한 수의 파라미터를 사용하여 찾자!

Deep Q-network(DQN)

- Q-value를 근사하기 위한 DNN
- (s, a)를 입력으로 받아 근사 Q-value를 출력

식 18-7 타깃 Q-가치

$$Q_{\text{target}}(s,a) = r + \gamma \cdot \max_{a'} Q_{\theta}(s',a')$$

Topics

1. What is Reinforcement Learning
2. MDP & Value function
3. Value-based RL
4. Policy-based RL

Policy-based RL

Value-based RL
: Q라는 action-value function에 초점을 맞춰 Q function을
먼저 구하고 그것을 토대로 policy를 구하는 방식

Policy-based RL
: policy 자체를 parameterize하여 근사하자!

Advantages:

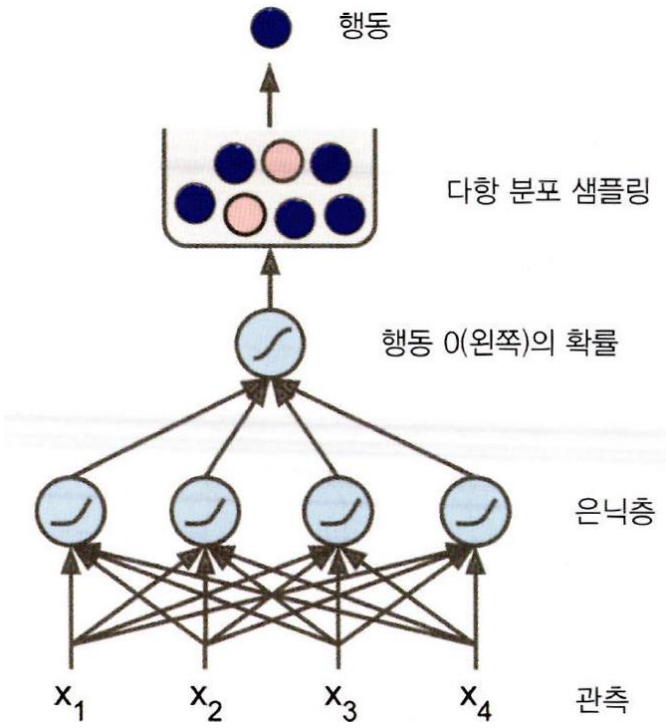
- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages:

- Typically converge to a local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

Policy-based RL

<신경망 정책>



신경망 정책

환경에 대한 관측을 INPUT으로 받고,
그에 따라 실행할 행동에 대한 확률분포를 OUTPUT한다.

Policy Gradient Algorithm

높은 보상을 얻는 방향의 gradient를 따르도록
정책의 파라미터를 최적화하는 알고리즘

ex) REINFORCE 알고리즘

1. 신경망 정책을 이용해 여러 번의 시뮬레이션을 진행하고,
   매 스텝마다 선택된 행동이 더 높은 가능성을 가지도록 하는 그레디언트 계산
2. 에피소드를 몇 번 실행한 다음, 각 행동의 이익 계산
3. (1)에서 계산한 각 행동에 대한 그레디언트 벡터와 (2)에서 계산한 행동의 이익을 곱함
4. 모든 결과 그레디언트 벡터를 평균 내어 경사 하강법 스텝을 수행