8. 러닝과 플래이닝의 통합

김호철

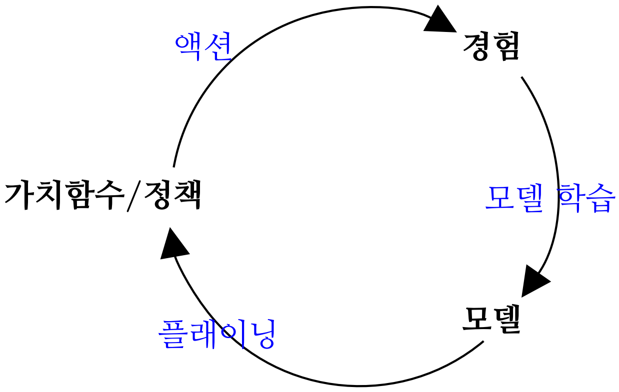
Contents

1 소개

- **모델 기반 강화학습**
  - 06. 벨류 펑션 근사 : 경험으로부터 직접 **가치함수(value function)**를 학습
  - 07. 폴리시 그레디언트 : 경험으로부터 직접 **정책(Policy)**을 학습
  - 이번 장은 경험으로부터 **모델**을 학습
  - 그리고 가치함수나 정책을 구성하기 위해 **플래이닝(planning)**을 사용한다.
  - 학습과 플래이닝을 하나의 아키텍처로 통합한다.

- **모델 기반과 모델 프리 강화학습**
  - 모델 프리(Model-free) 강화학습
    - 모델이 없음
    - 경험으로 부터 가치함수나 정책을 학습
  - 모델 기반(Model-Based) 강화학습
    - 경험으로부터 모델을 학습하고,
    - 그 모델로부터 가치함수나 정책을 플랜(Plan)한다.

2 모델기반 강화학습



- **모델기반 RL 장단점**
  - **장점**
    - ∗ 모델이 간단한 경우(체스 같은) 지도학습으로 효율적 모델 학습이 가능
    - ∗ 불확실한 모델에 대해서도 추론을 할 수 있음
  - **단점**
    - ∗ 먼저 모델을 학습한 후 가치함수를 구성하므로,
    - ∗ 근사 오류가 두 단계에서 발생한다.

2.1 모델 학습하기

- **모델이 무엇인가?**
  - 모델 $\mathcal{M}$은 파라메터 η로 동작되는 MDP $<S, A, P, R>$를 나타낸다.(S:State, A:Action, P:Transation Probability, R:Reward)
  - 상태 공간 $\mathcal{S}$와 액션 공간 $\mathcal{A}$를 안다고 가정하면,
  - 모델 $\mathcal{M}$=P_η, R_η 은 상태 전이 $\mathcal{P}_\eta \approx \mathcal{P}$와 보상 $\mathcal{R}_\eta \approx \mathcal{R}$를 나타낸다.

$$\begin{aligned} S_{t+1} &\sim \mathcal{P}_\eta(S_{t+1}|S_t, A_t) \\ R_{t+1} &= \mathcal{R}_\eta(R_{t+1}|S_t, A_t) \end{aligned} \tag{1}$$

  - 일반적으로 상태 전환과 보상은 서로 조건부 독립으로 가정한다.

$$\mathbb{P}[S_{t+1}, R_{t+1}|S_t, A_t] = \mathbb{P}[S_{t+1}|S_t, A_t]\mathbb{P}[R_{t+1}|S_t, A_t] \tag{2}$$

- **모델 학습하기**
  - 모델 학습의 목표는 경험 $\{S_1, A_1, R_2, ..., S_T\}$에서 모델 M_η 를 추정하는 것이다.
  - 이는 지도학습 문제이다.

$$\begin{aligned} S_1, A_1 &\rightarrow R_2, S_2 \\ S_2, A_2 &\rightarrow R_3, S_3 \\ &\vdots \\ S_{T-1}, A_{T-1} &\rightarrow R_T, S_T \end{aligned} \tag{3}$$

- 학습 $s, a \to r$ 은 회귀(regression) 문제
- 학습 $s, a \to s'$ 는 밀도 추정(density estimation) 문제
- 손실함수(loss function)를 선택, 예 : 평균 제곱 오차, KL 발산, ...
- 손실을 최소화하는 파라메터 찾기

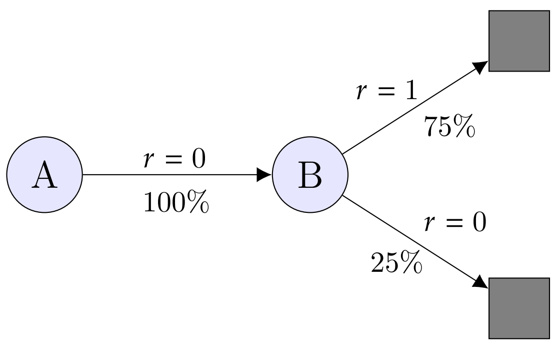
- **모델의 예**
  - 테이블 룩업 모델(Table Lookup Model)
  - 선형 기대 모델(Linear Expectation Model)
  - 선형 가우시안 모델(Linear Gaussian Model)
  - 가우시안 프로세스 모델(Gaussian Process Model)
  - 심층 신뢰망 모델(Deep Belief Network Model)
  - ...

- **테이블 룩업 모델**
  - 모델은 MDP $\hat{\mathcal{P}}, \hat{\mathcal{R}}$ 이다.
  - 각 상태, 액션 쌍들을 방문할 때 $N(s, a)$ 계산

$$
\begin{aligned}
\hat{\mathcal{P}}^a_{s,s'} &= \frac{1}{N(s,a)} \sum_{t=1}^{T} 1(S_t, A_t, S_{t+1} = s, a, s') \\
\hat{\mathcal{R}}^a_{s,s'} &= \frac{1}{N(s,a)} \sum_{t=1}^{T} 1(S_t, A_t = s, a) R_t
\end{aligned}
\tag{4}
$$

  - 또는, 각 시간 단계(time-step) t에서, 경험 튜플 $<S_t, A_t, R_{t+1}, S_{t+1}>$을 기록한 후에,
  - $<s, a, \cdot, \cdot>$ 와 매치하는 튜플들에서 모델을 랜덤 샘플링

- **AB 예제**
  - A,B 2개의 상태, 할인없이, 8번의 에피소드 경험
  - {A, 0, B, 0}, {B, 1}, {B, 1}, {B, 1}, {B, 1}, {B, 1}, {B, 1}, {B, 0}



  - 경험으로부터 테이블 룩업 모델을 구성하였다.

2.2 모델로 플래이닝하기

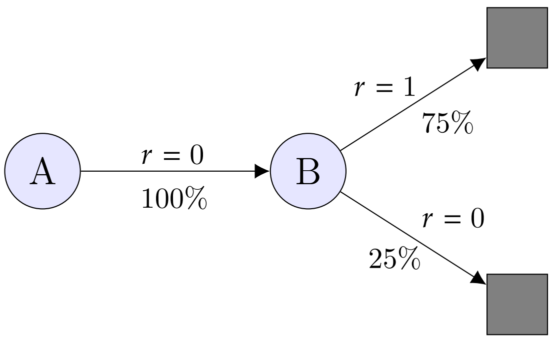
- 모델 $M_\eta = <P_\eta, R_\eta>$이 주어지면,
- MDP $<S, A, P_\eta, R_\eta>$ 풀기
- 선호하는 플래이닝 알고리즘 사용
  - 가치 이터레이션(Value iteration)
  - 정책 이터레이션(Policy iteration)
  - 트리 서치(Tree search)
  - ...

- **샘플 기반 플래이닝**
  - 플래이닝하기 위해, 단순하지만 매우 효율적인 접근
  - 모델은 샘플링 하는데에만 쓰임
  - 모델로부터 경험을 샘플링

$$\begin{aligned} S_{t+1} &\sim P_\eta(S_{t+1}|S_t, A_t) \\ R_{t+1} &= R_\eta(R_{t+1}|S_t, A_t) \end{aligned} \tag{5}$$

  - 샘플링 하는데 모델-프리(model-free) RL을 적용 : 몬테 카를로 콘트롤, 살사, Q-러닝
  - 샘플 기반 플래이닝이 효율적일 때가 많다.
  - 경험으로부터 생성된 한정된 샘플데이터로부터, 무한한 샘플 데이터를 생성할 수 있다.

- **샘플 기반 플래이닝 AB 예제**
  - 실제 경험으로부터 테이블 룩업 모델을 구성
  - 실제 경험 : {A, 0, B, 0}, {B, 1}, {B, 1}, {B, 1}, {B, 1}, {B, 1}, {B, 1}, {B, 0}



  - 샘플된 경험 : {B, 1}, {B, 0}, {B, 1}, {A, 0, B, 1}, {B, 1}, {A, 0, B, 1}, {B, 1}, {B, 0}
  - 예를 들어 몬테카를로 학습: V(A) = 1, V(B) = 0.75

- **부정확한(Inaccurate) 데이터로 부터 플래이닝**
  - 불완전한 모델$<P_\eta, R_\eta> \neq <P, R>$이 주어지면,
  - 모델 기반 RL은 MDP $<S, A, P_\eta, R_\eta>$에 근사하여 제한된다.
  - 즉, 모델 기반 RL은 추정된 모델 만큼만 우수하다.
  - 모델이 부정확하면 플래이닝 처리는 최적이 아닌 정책으로 수행된다.
  - 솔루션 1 : 모델이 나쁘면 모델-프리 RL을 사용한다.
  - 솔루션 2 : 모델 불확실한 모델에 대해서 명시적으로 추론(예를 들어 25~30 구간이 아닌, 0~100구간으로 한다든가)

3 아키텍처 통합

3.1 Dyna

- **실제와 시뮬레이트된 경험**
  - 두 가지 경험의 소스들을 모두 고려
  - 첫번째, 실제 경험에서 샘플링(실제 MDP)

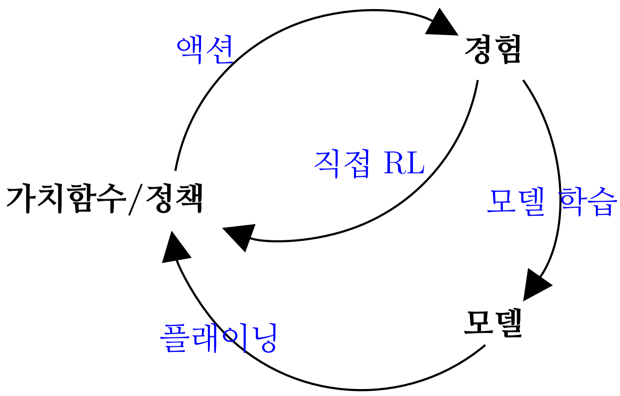
$$\begin{aligned} S' &\sim P^a_{s,s'} \\ R &= R^a_s \end{aligned} \tag{6}$$

  - 두번째, 모델에서 시뮬레이트된 경험으로 샘플링(근사 MDP)

$$\begin{aligned} S' &\sim P_\eta(S'|S, A) \\ R &= R_\eta(R|S, A) \end{aligned} \tag{7}$$

- **학습과 플래이닝의 통합**
  - 모델-프리 RL
    * 모델이 없다.
    * 실제 경험에서 가치함수나 정책을 **학습**한다.
  - 모델-기반 RL(샘플-기반 플래이닝 사용)
    * 실제 경험에서 모델을 학습한다.
    * 시뮬레이트된 경험에서 가치함수나 정책을 **플래이닝**한다.
  - Dyna
    * 실제 경험에서 모델을 학습한다.
    * 시뮬레이트된 경험에서 가치함수나 정책을 **학습하거나 플래이닝**한다.

- **Dyna 아키텍처**



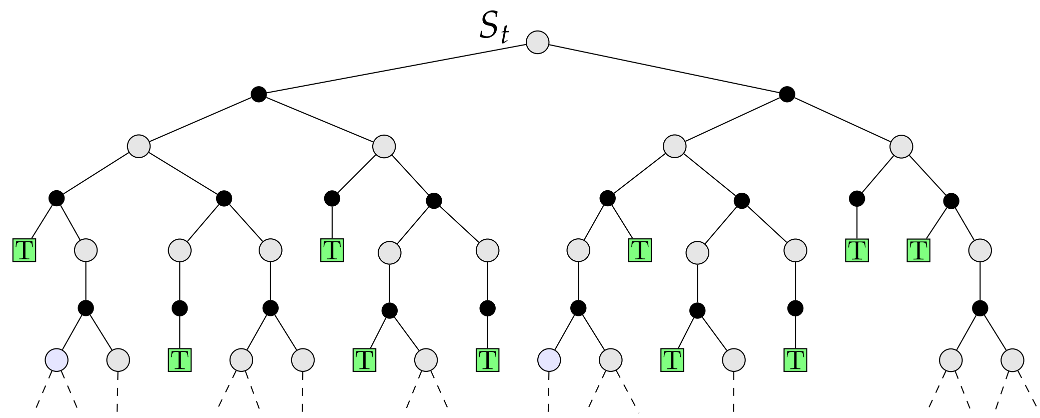
- **Dyna-Q 알고리즘**

모든 $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$에 대해 $Q(s,a)$와 $Model(s,a)$를 초기화
계속 반복
(a) 현재 상태 **S** 선택
(b) S에서 ϵ-greedy로 액션 **A** 선택
(c) 액션 A를 실행하고, 보상 R과 상태 S'를 받음
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma max_a\ Q(S',a) - Q(S,A)]$ (**학습 단계**)
(e) $Model(S,A) \leftarrow R, S'$ (결정적 환경이라 가정)
(f) n 번 반복 (**플래이닝 단계**)
이전에 관측된 상태에서 랜덤 **S** 선택
S에서 이전에 받은 랜덤 액션 **A** 선택
$R, S' \leftarrow\ Model(S,A)$ S에서 A를 하고 R과 S'를 모델에서 받음
$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma max_a\ Q(S',a) - Q(S,A)\right]$ Q 업데이트

4 시뮬레이션 기반 서치

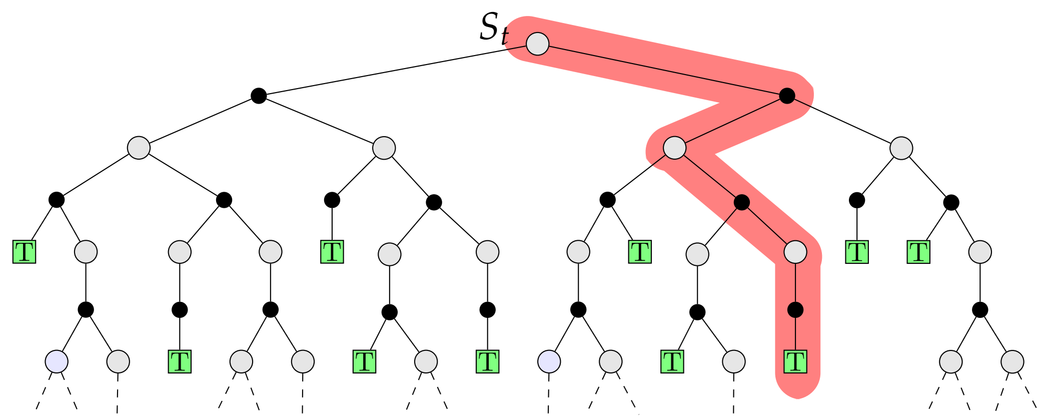
- 플래이닝을 얼마나 더 효율적으로 하느냐의 문제
- 서치(search)는 실제로 해보면서 모델을 만드는 일

- **포워드 서치(Forward Search)**
  - 과거는 필요없고, 현재 이후가 중요하다.
  - 포워드 서치 알고리즘은 미리보기(lookahead)를 통해 최고의 액션을 선택한다.
  - 미리보기를 위해 MDP의 모델을 사용해서, 현재 상태 s_t를 루트로 하는 서치 트리를 만든다.



– 전체 MDP를 풀 필요없이, 지금부터 자식(하위) MDP만 풀면 된다.

- **시뮬레이션 기반 서치**
  – 포워드 서치 패러다임은 샘플 기반 플래이닝을 사용한다.
  – 모델을 이용하여 지금부터 여러 에피소드들을 시뮬레이트해서 생성해 낸다.
  – 시뮬레이트된 에피소드들에 모델-프리 RL을 적용한다.



– 모델을 이용하여 지금부터의 경험의 에피소드들을 시뮬레이션한다.

$$\{s_t^k, A_t^k R_{t+1}^k, ..., S_T^K\}_{k=1}^K \sim M_v \tag{8}$$

– 시뮬레이트된 에피소드들에 모델-프리 RL을 적용한다
  * 몬테카를로 콘트롤 → 몬테카를로 서치
  * 살사 → TD 서치

4.1 몬테카를로 서치

- **심플 몬테카를로 서치**
  – 모델 M_v와 시뮬레이션 정책π가 주어지면,
  – 각 액션 $a \in \mathcal{A}$ 마다 수행 :

        - 현재 실제(real) 상태 s_t에서 K번 만큼 시뮬레이트 한다.

$$\{s_t, a, R_{t+1}^k, S_{t+1}^k, A_{t+1}^k, ..., S_T^K\}_{k=1}^K \sim M_v, \pi \tag{9}$$

        - 몬테카를로 반환 평균으로 액션들을 평가하여, Q를 업데이트 한다.

$$Q(s, a) = \frac{1}{K} \sum_{k=1}^{K} G_t \to^P q_\pi(s_t, a) \tag{10}$$

    - Q에서, 최대값으로 현재 실제 액션을 선택한다.

$$a_t = \underset{a \in A}{\operatorname{argmax}} Q(s_t, a) \tag{11}$$

- **몬테카를로 트리 서치(평가)**
    - 모델 M_v이 주어지면,
    - 현재 시뮬레이션 정책 π를 사용해서, 현재 상태 S_t로부터 K개의 에피소드를 시뮬레이트 (생성, 뽑아내다)

$$\{s_t, A_t^k, R_{t+1}^k, S_{t+1}^k, ..., S_T^K\}_{k=1}^K \sim M_v, \pi \tag{12}$$

    - 방문한 상태들과 액션들이 포함된 서치 트리를 만든다.
    - s, a에서 생성된 에피소드들의 평균 리턴으로 상태들의 $Q(s, a)$ 를 평가(Evaluation)

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{k=1}^{K} \sum_{u=t}^{T} 1(S_u, A_u = s, a) G_u \to^p q_\pi(s, a) \tag{13}$$

    - 서치가 끝난 후, 서치 트리에서 최대값의 현재(실제) 액션을 선택한다.

$$a_t = \underset{a \in A}{\operatorname{argmax}} Q(s_t, a) \tag{14}$$

- **몬테카를로 트리 서치(시뮬레이션)**
    - MCTS에서는 시뮬레이션 정책 π도 개선시킨다(심플 몬테카를로 서치에서는 고정)
    - 각 시뮬레이션은 두 단계(phase)로 구성됨(in-tree와 out-of-tree)
        - 트리 정책(Tree policy)(개선) : $Q(S, A)$를 최대화하는 액션을 선택
        - 디폴트 정책(Default policy)(고정) : 무작위 액션 선택
    - 반복(각 시뮬레이션)
        - 몬테카를로 평가로 상태 $Q(S, A)$를 평가한다.
        - ϵ-greedy(Q) 같은 방법으로, 트리 정책을 개선시킨다.
    - 시뮬레이트된 경험에 몬테카를로 콘트롤을 적용
    - 최적 서치 트리로 수렴한다. $Q(S, A) \to q_*(S, A)$

- **몬테카를로 트리 서치(MCTS)의 장점**
    - 높은 선택적 최선 우선 서치
    - 상태를 동적으로 평가(DP와 다름)
    - 차원의 저주를 깨기위해 샘플링을 사용한다.
    - 모델이 블랙박스(모델의 구성을 모름)여도 된다(샘플만 되면 된다).
    - 계산하기에 효율이 좋고, 병렬도 가능하며, 어느 시점이든 가능하다.

4.2 TD 서치(Temporal-Difference Search)

- **TD 서치**
  - 시뮬레이션 기반 서치
  - MC 대신 TD를 사용(부트스트래핑)
  - MC 트리 서치는 지금부터 하위 MDP에 MC 콘트롤을 적용하고,
  - TD 서치는 현재부터 하위 MDP에 살사를 적용한다.

- **MC vs. TD 서치**
  - 모델 프리 RL에서는 부트 스트래핑이 더 유용하다.
    - TD 러닝은 분산을 줄이고, 편향을 증가 시킨다.
    - TD 러닝은 일반적으로 MC보다 효율적이다.
    - TD (λ)는 MC보다 훨씬 효율적일 수 있다.
  - 시뮬레이션 기반 서치의 경우에도 부트 스트래핑이 도움이 된다.
    - TD 서치는 분산을 줄이고, 편향을 증가 시킨다.
    - TD 서치는 일반적으로 MC 서치보다 효율적이다.
    - TD (λ) 서치는 MC 서치보다 훨씬 효율적일 수 있다.

- **TD 서치**
  - 현재 (실제) 상태 s_t에서 에피소드들을 시뮬레이트 한다
  - 액션 가치 함수 $Q(s, a)$를 추정
  - 시뮬레이션 각 스텝에서, 살사로 액션-가치(action-value)들을 업데이트

$$\Delta Q(S, A) = \alpha(R + \gamma Q(S', A') - Q(S, A)) \tag{15}$$

  - 액션-가치(action-value) $Q(s, a)$에서 액션 기반으로 선택(ϵ-greedy 같은)
  - Q에 함수 근사(approximation)를 사용할 수도 있다.

- **Dyna-2**
  - Dyna-2에서 에이전트는 특성(feature) 2개의 가중치를 저장한다.
    - 장기 메모리(Long-term memory)
    - 단기 메모리(Short-term (working) memory)
  - 장기 메모리는 TD 러닝을 사용해서 실제 경험으로부터 업데이트 된다.
    - 어떤 에피소드에도 적용되는 일반적인 도메인 지식
  - 단기 메모리는 TD 서치를 사용해서 시뮬레이트된 경험으로부터 업데이트 된다.
    - 현재 상황에 관련된 특정 로컬 지식
  - 전체 가치함수(value-function)는 장기, 단기 메모리들의 합이다.