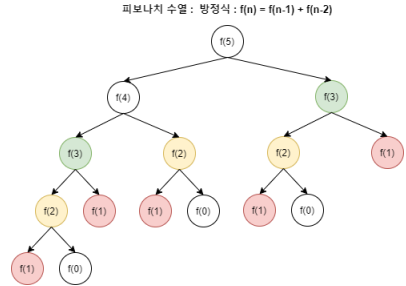
DP & LR

Kihwan Lee

Contents

1. Dynamic Programming

Dynamic Programming

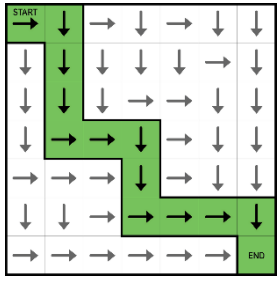


Dynamic Programming : 동적계획법이 무엇인가?

복잡한 문제를 여러개의 간단한 문제로 나누어 효율적으로 푸는 방법으로
특정 알고리즘은 나이고, 하나의 문제 해결 패러다임
+ DP는 planning (계획) 으로 <-> learning (학습) 과 상반

DP는 복잡한 문제를 작은 문제로 나누어 모두 직접 계산을 하는 것이며,
강화학습의 learning은 그것을 직접 계산할 수 없기 때문에 컴퓨터가 학습하는 것을 말합니다.
이런 면에서 서로 대조적입니다.

Dynamic Programming



Dynamic Programming의 세 가지 핵심 요소

1) Substructure : original 복잡한 하나의 큰 문제를 간단한 작은 문제 여러개로 나눕니다.

2) Table Structure : 작은 문제들을 계산한 후에 해답들을 테이블에 기록하는 것 입니다.
이후 이를 반복적으로 사용합니다. => 연산 효율성 증대!

3) Bottom-up Computation : 밑에 있는 작은 문제들을 풀어 위에 있는 큰 문제를 해결한다는 것 입니다.
테이블에 있는 작은 문제들의 해답을을 사용하여 조금 더 큰 문제에 대한 해답을 구하고
결국 최종 큰 문제에 대한 해답을 얻는 것 입니다.

Dynamic Programming

- **Dynamic Programming**

  1) 최적 하부 구조 : Optimal Substructure

  : 여러 개의 작은 문제들로 분할

  2) 중복되는 하위 문제들 : Overlapping subproblems

  : 반복되는 작은 문제들의 솔루션들을 다시 사용

- **MDP**

  1) 최적 하부 구조 : Optimal Substructure

  : Bellman Equation은 이 특성을 활용해 재귀적으로 MDP를 분해

  2) 중복되는 하위 문제들 : Overlapping subproblems

  : 가치함수는 솔루션을 저장하고 재사용

Dynamic Programming

그렇다면 언제 MDP에 DP 방법론을 적용시키는 것이 효과적?

우선 DP를 활용한 Planning 고려

: DP는 모든 state에 대한 계산을 해야 하기에 너무나 방대

=> DP의 planning : value function을 활용하여 계산

=> value functino을 활용한 값을 통해 table에 누적

하지만 **DP의 세 가지 문제점**

1) 계산복잡도 : 작은 문제로 나누는데 한계가 있습니다.

2) 차원의 저주 : 테이블에 저장하는데 state space가 너무 커지면 다 기록을 할 수 없게 됩니다. 그렇기에 state space가 어느정도 한정되어 있어야 한다는 점입니다.

3) 환경에 대한 완벽한 정보가 필요 : transition probability 과 reward를 알고 있어야 합니다.

Dynamic Programming

그렇다면 강화학습은?

강화학습은 model-based로 incomplete information을 주로 갖고 있을 때 고려하며, 정확한 계산이 불가할 때 사용

++ 따라서 데이터를 활용한 learning을 진행

즉, DP와 달리 직접적인 모든 계산이 아닌 학습을 진행하며, 이러한 데이터가 쌓임으로 이를 통해 학습

=> 계속 얻어 지는 rewar에 대해 polic를 평가하게 되고, 이 total rewar를 maximize하는 방향으로
policy를 업데이트 해 나가는 것이 강화학습의 기본 틀

따라서 강화학습은 주어진 데이터에 대해서만 계산을 진행하기에 계산복잡도나 차원의 저주를 해결 가능!

Dynamic Programming

- Dynamic Programming (DP): full backup

$$V(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})]$$

- Monte Carlo (MC): sample multi-step backup

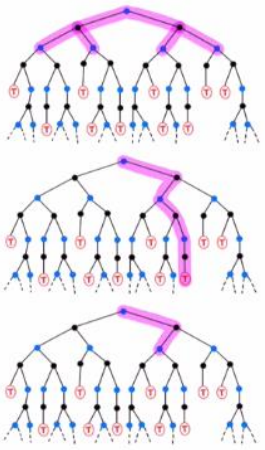
$$V(S_t) \leftarrow V(S_t) + \alpha\,[G_t - V(S_t)]$$

- Temporal Difference (TD): sample backup

$$V(S_t) \leftarrow V(S_t) + \alpha\,[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

RL

Sarsa, Q-learning



2. Reinforcement Learning

Reinforcement Learning

Review

- 목적 : table을 잘 업데이트 하여 최적의 policy를 찾자!
- DP와 RL의 차이 : DP는 planning, RL은 learning

  DP은 model-based에서 bellman equation을 통해 state에 대한 값들을 update – full backup

  RL는 model-free로 transition probability를 모를 때, 특정 state에서 얻어진 action에 대해서 고려 – sample backup

중요하게 고려할 점

DP는 모든 값을 알기에 정확히 bellman optimal eq를 만족하는 값을 찾아가는데, RL의 경우는 샘플들로만 진행하기에 정확히 bellman optimal equation를 만족하지 않습니다.

=> Approximation을 통해 문제를 해결

Reinforcement Learning

=> DP

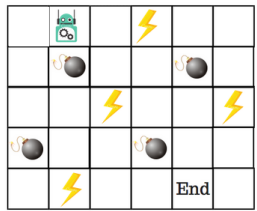
테이블 안의 모든 state에 대해 update를 진행하기에 계산량을 고려하여 state value 값들을 업데이트

이를 통해 업데이트 이후 maximize하는 action을 찾으며, optimal policy를 찾아감을 반복

=> RL

샘플링을 통해 계산하기에 모든 정보를 가지고 있지 않은 점이 존재

즉, 부분적으로 업데이트가 가능하기에 DP와 달리 Q table을 업데이트



Actions :	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

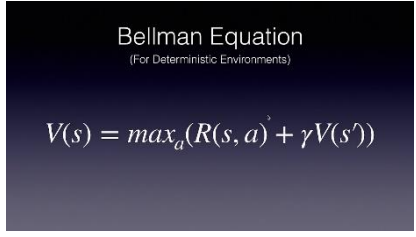
Reinforcement Learning

Reinforcement Learning

강화학습의 구성

1) Monte Carlo method

2) Temporal Difference learnings : Sarsa, Q-learning



Bellman Equation

Bellman Equation은 state에서 가능한 모든 return값들의 기댓값을 immediate reward와 next state를 통해 구한 식

DP는 모든 값을 알고 있기에, Bellman Equation을 사용하지 않고 모든 retur을 계산하기에는 부담

즉, Bellman방정식을 사용하는 것이 필수적

하지만 이에 반해 강화학습의 경우는 transition probability를 모르기에 샘플만을 가지고 계산

즉, Bellman방정식 사용은 선택적

Reinforcement Learning

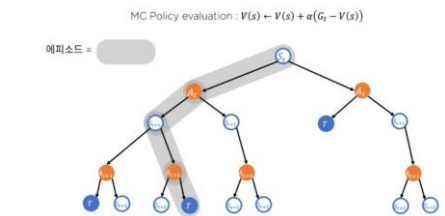
Reinforcement Learning

강화학습의 구성

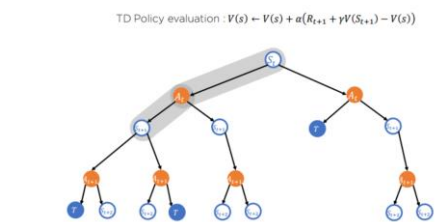
1) Monte Carlo method : 모든 reward들에 대해 total discounted reward를 한 return값을 사용하는 방법

2) Temporal Difference learnings : 바로 한 step만 가서 계산한 값을 이용하는 것 : Sarsa, Q-learning

MC 기법을 활용한 Sample backup



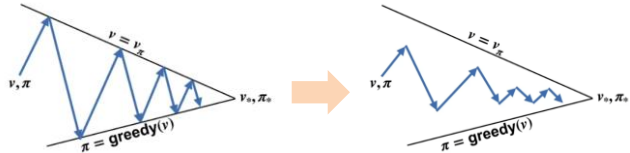
TD 기법을 활용한 Sample backup



Reinforcement Learning

GPI : Generalized Policy Iteration

대부분 모든 강화학습은 GPI를 기본으로 구성



Policy iteration : policy evaluation과 policy improvement를 반복적으로 사용

1) policy evaluation : 현재까지 업데이트 된 current policy가 있다면 이것을 기준으로 value ft값들을 계속 converge 할 때 까지 업데이트해가는 것
2) policy improvement : 1)을 통해 value ft을 업데이트해서 모든 state에 대해 value ft값들이 converge한다면 value ft테이블을 가지고 policy improvement를 진행

중요한 핵심은 만약 value funciton값은 계속 converge 하지만, 이미 optimal policy에 도달했다면?

필요없는 계산을 converge 할 때 까지 계속 진행해야 한다는 비효율성이 존재

=> GPI는 Sample backu을 통해 샘플에 주어진 데이터에 대해서만 value ft 값을 계속 업데이트 해 나가기에 위와 같은 비효율성을 야기시키지 않음