

1. Introduction

1-1. Why Classic Games ?

1-2. AI and RL in Games : State of the Art

2. Game Theory

2-1. Nash Equilibrium

2-1-1. Definition

2-1-2. Three Conditions for Unique NE

2-2. Game Tree Search (Minimax Search)

2-2-1. Minimax Value Function

2-2-2. Minimax Search

2-3. Self-Play Reinforcement Learning

3. Combining Reinforcement Learning and Minimax Search

3-1. Introduction

3-2. TD Root

3-3. TD Leaf

3-4. TreeStrap

3-5. Simulation-Based Search

4. Reinforcement Learning in Imperfect-Information Games

4-1. Smooth UCT Search

5. Conclusion

1. Introduction

1-1. Why Classic Games ?

#. 강화학습의 연구는 Classic Game를 이용하여 이루어진다. 왜냐하면 classic game의 특징과 관련되어 있는데, 첫 번째로 바둑과 같이 규칙은 간단하지만 깊은 개념이 들어있기 때문이다. 두 번째로 오랫동안 사람들이 play하면서 수많은 경험과 지식이 쌓여있기 때문이다. 세 번째로 classic games는 AI의 초파리와 같은 존재이다. 즉 실험하는데 많이 사용한다는 것이다. 마지막으로 현실 세계의 상황들을 반영하는 소우주와 같기 때문이다.

1-2. AI and RL in Games : State of the Art

#. 아래와 같이 일반적인 AI, RL에서 State of the Art를 정리할 수 있다.

AI in Games : State of the Art		
Program	Level of Play	Program to Achieve Level
Checkers	Perfect	Chinook
Chess	Superhuman	Deep Blue
Othello	Superhuman	Logistello
Backgammon	Superhuman	TD-Gammon
Scrabble	Superhuman	Maven
Go	Grandmaster	MoGo ¹ , Crazy Stone ² , Zen ³
Poker ⁴	Superhuman	Polaris
RL in Games : State of the Art		
Program	Level of Play	RL Program to Achieve Level
Checkers	Perfect	Chinook
Chess	International Master	KnightCap / Meep
Othello	Superhuman	Logistello
Backgammon	Superhuman	TD-Gammon
Scrabble	Superhuman	Maven
Go	Grandmaster	MoGo ¹ , Crazy Stone ² , Zen ³
Poker ⁴	Superhuman	SmooCT

a. 'Perfect'는 완전히 풀렸다는 의미이고, 'Superhuman'은 사람 1등을 이겼다는 것이다. 'Grandmaster'는 사람 수준에서의 pro-level이다.

2. Game Theory

#. 다수의 참여자가 어떤 규칙 하에서 reward를 최대화하는 각각의 policy(optimality)를 구하기 위해 게임이론을 도입한다. 그리고 이 균형을 찾기 위한 방법론으로 Game Tree Search와 Self-Play RL가 있다.

2-1. Nash Equilibrium

2-1-1. Definition

#. i 명의 참여자가 있을 때, 나를 제외한 다른 참여자들의 policy가 고정되어 있다고 가정했을 때, 나의 optimal policy는 $\pi_*^i(\pi^{-i})$ 로 정의되며 이를 Best response라고 한다. Nash Equilibrium은 i 명의 참여자 모두 Best response인 경우를 의미하며 아래와 같이 정의한다.

$$\pi^i = \pi_*^i(\pi^{-i})$$

- 이 균형에서는 모두 자신이 생각하는 최선의 전략을 선택하고 있기 때문에, 전략을 바꿀 유인이 없는 것이 특징이다.
- Single-Agent 문제에서는 하나의 agent를 제외하곤 다른 player를 모두 Env.로 묶을 수 있기 때문에, MDP로 표현할 수 있으며, Best Response가 곧 optimal policy가 된다.
- 나와 나 자신이 붙으며 학습하는 Self-Play 문제에서 Nash Equilibrium을 찾기 위해서는 Env.가 동적으로 변하는 과정 속에서 fixed-point를 찾는 것이라 할 수 있다. 왜냐하면 b에서와 같이 한 agent 입장에서는 자신을 제외한 모든 것을 Env.로 취급하기 때문에 지속적으로 다른 agent의 policy가 변하면 그 agent 입장에서 Env.이 변하는 것으로 되기 때문이다.
- 일반적으로 Nash Equilibrium은 Unique하지 않지만 2-1-2.에서 살펴볼 것과 같이 3가지 조건이 충족되면 Nash Equilibrium은 unique하다.

2-1-2. Three Conditions for Unique Nash Equilibrium

#. 3가지 조건은 아래와 같으며 이 조건들이 만족되면 Nash Equilibrium이 Unique하다.

Two-Player Alternative Game	두 명의 player가 번갈아 가면서 한 번씩 action을 하는 게임이다.
Zero-Sum Game	$R^1 + R^2 = 0$ 로 표현되며, 한 player의 이득이 다른 player의 손해가 된다.
Perfect Information	상대가 어느 state인지 알 수 있는 것을 의미한다. 예를 들어 Go, Chess, Checker 등의 게임이 있고, Imperfect Information은 상대의 state를 완전히 모르는 것을 의미한다. 예를 들어 Poker가 있다.

2-2. Game Tree Search (Minimax Search)

2-2-1. Minimax Value Function

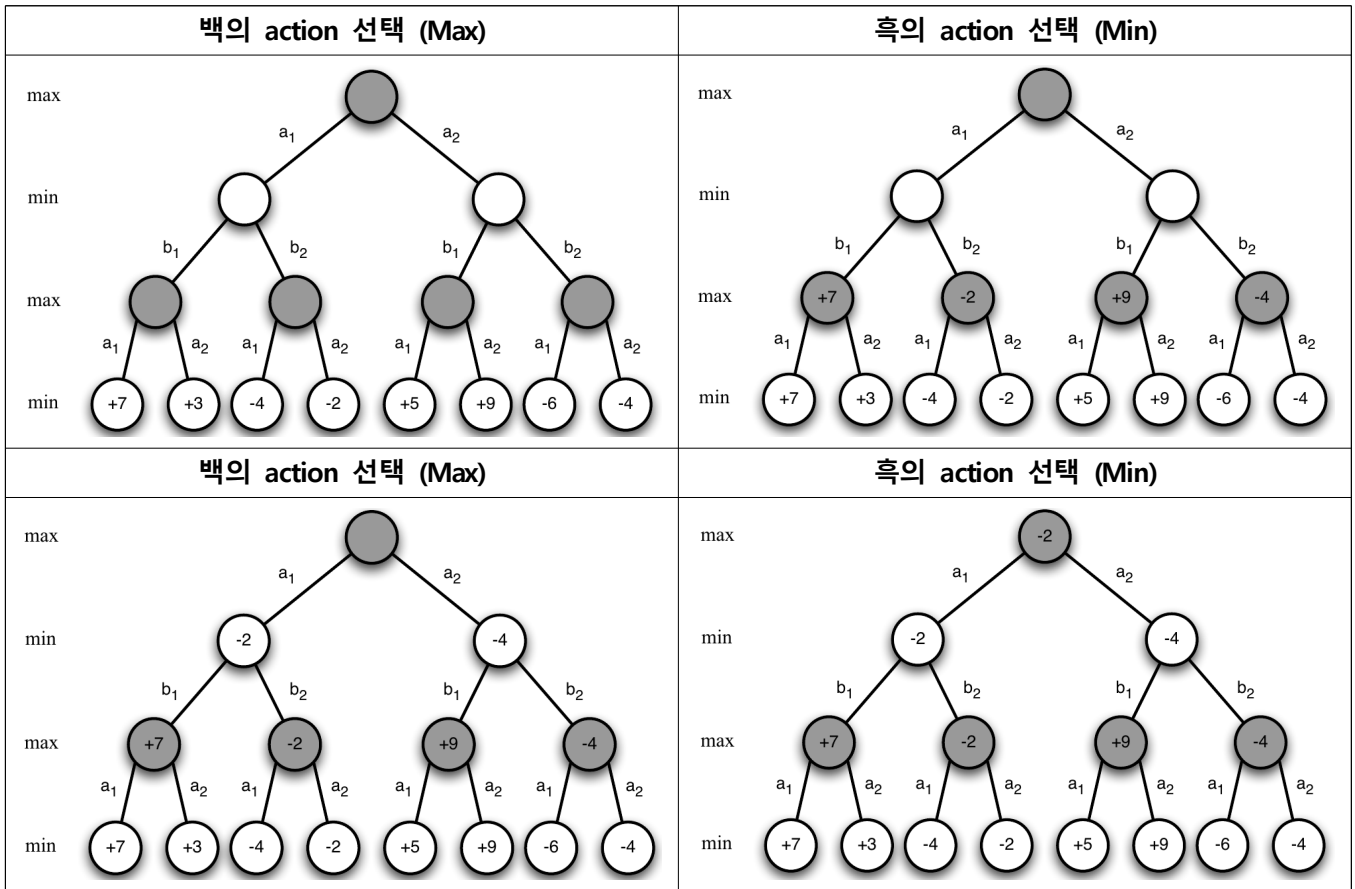
#. 2-Players(백/흑)의 상황에서 어느 한 player를 기준으로, 현재의 state에서 둘 다 최선의 policy로 play를 하였을 때, 누가 이길 것인지를 나타내는 값이며 minimax value라고도 한다. minimax value function의 parameter를 minimax policy라고 하며 이는 Nash Equilibrium이다. 표기는 아래와 같다.

Joint Policies	$\pi = \langle \pi_1, \pi_2 \rangle$
Value Function	$v(s) = E_{\pi}[G_t S_t = s]$
Minimax Value Function	$v_*(s) = \max_{\pi_1} \min_{\pi_2} v_{\pi}(s)$

- a. 백 입장에서 이기는 것을 1, 흑 입장에서는 -1이 된다. Example에 살펴보겠지만, state의 minimax value가 양수이면 백이 이긴다는 의미, 0이면 비긴다는 의미, 음수이면 흑이 이긴다는 의미이다.
- b. 따라서 Minimax Search에서는 현 state에서 누가 이길지를 나타내는 minimax value function을 찾아내는 것이 목적이다.
- c. 각 state마다 unique minimax value function이 존재한다.

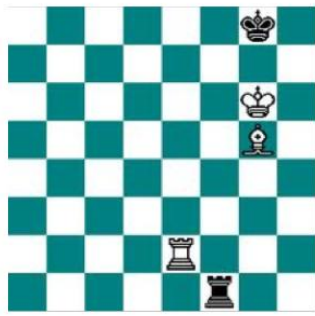
2-2-2. Minimax Search

#. 어떤 state의 Minimax Value Function을 찾기 위한 알고리즘이며 아래의 예를 들어 그 과정을 알아보자. 즉 백과 흑 2명의 players가 있으며 각각 action이 2가지며 zero-sum and alternative game이다. 검은 node는 백을 의미한다.



- 결과를 보면 현재 state의 minimax value function의 값은 -2가 되었으므로 이 state에서는 흑이 무조건 이길 수밖에 없다.
- 이 알고리즘은 전통적 기계학습이며 자체로도 강력한 성능을 발휘하며, Deep Learning과 결합하면 더 높은 성능을 낼 수 있다.
- 위의 예시에서 알 수 있듯이 action이 2가지이고 terminal state가 짧더라도 search 해야 하는 경우의 수는 기하급수적으로 증가함을 알 수 있다. 이를 해결하기 위해 value function approximator¹⁾를 사용하여 fixed depth까지만 search를 진행한 후 fixed depth 해당 state에는 estimated value function 값을 할당하고 현 state의 minimax value를 계산하는 truncate 방법이 있다.
- 이러한 truncate의 방법론으로는 흥미 없는 tree의 부분을 잘라내는 Alpha-beta Search가 있다.
- Value Function Approximator의 간단한 예시인 Binary-Linear value function을 알아보면, feature vector 인 $x(s)$ 을 만들고 weight vector w 을 곱해준 후 더해진 것이 해당 state의 estimated value이며, chess를 예로 들면 아래와 같으며 이 때 weight vector는 handcrafted이다.

1) $v(s, w) \approx v_*(s)$



$$v(s, \mathbf{w}) = \mathbf{x}(s) \cdot \mathbf{w} =$$

$$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \cdot \begin{bmatrix} +5 \\ +3 \\ +1 \\ -5 \\ -3 \\ -1 \\ \vdots \end{bmatrix}$$



$$v(s, \mathbf{w}) = 5 + 3 - 5 = 3$$

f. 위와 같은 방법론을 사용한 것이 Deep-Blue와 Chinook이다.

Deep-Blue	Chinook
<ul style="list-style-type: none"> ■ Knowledge <ul style="list-style-type: none"> ■ 8000 handcrafted chess features ■ Binary-linear value function ■ Weights largely hand-tuned by human experts ■ Search <ul style="list-style-type: none"> ■ High performance parallel alpha-beta search ■ 480 special-purpose VLSI chess processors ■ Searched 200 million positions/second ■ Looked ahead 16-40 ply ■ Results <ul style="list-style-type: none"> ■ Defeated human champion Garry Kasparov 4-2 (1997) ■ Most watched event in internet history 	<ul style="list-style-type: none"> ■ Knowledge <ul style="list-style-type: none"> ■ Binary-linear value function ■ 21 knowledge-based features (position, mobility, ...) ■ x4 phases of the game ■ Search <ul style="list-style-type: none"> ■ High performance alpha-beta search ■ Retrograde analysis <ul style="list-style-type: none"> ■ Search backward from won positions ■ Store all winning positions in lookup tables ■ Plays perfectly from last n checkers ■ Results <ul style="list-style-type: none"> ■ Defeated Marion Tinsley in world championship 1994 <ul style="list-style-type: none"> ■ won 2 games but Tinsley withdrew for health reasons ■ Chinook solved Checkers in 2007 <ul style="list-style-type: none"> ■ perfect play against God

2-3. Self-Play Reinforcement Learning

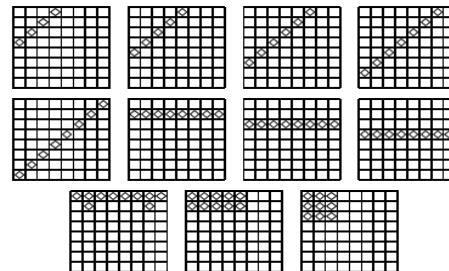
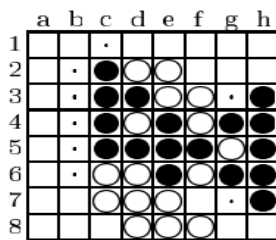
#. Self-Play는 한 player가 두 명의 players 역할을 하는 것을 의미하며, 이러한 방식에서 MC, TD(0), TD(λ)와 같은 value-based RL algorithms를 사용하여 학습하는 것을 의미한다. 하지만 two players game의 상황이므로 value가 minimax value로 바뀌었고²⁾, intermediate reward가 없어서 r, γ 가 사라진 것이 기존 방법과의 차이이다. 아래는 각 algorithm에 따른 학습(update)방식이다.

MC	update minimax value function approximator towards the return G_t $\Delta \mathbf{w} = \alpha (\mathbf{G}_t - v(S_t, \mathbf{w})) \nabla_{\mathbf{w}} v(S_t, \mathbf{w})$
TD(0)	update minimax value function approximator towards successor value $v(S_{t+1})$ $\Delta \mathbf{w} = \alpha (v(S_{t+1}, \mathbf{w}) - v(S_t, \mathbf{w})) \nabla_{\mathbf{w}} v(S_t, \mathbf{w})$
TD(λ)	update minimax value function approximator towards the λ -return G_t^λ $\Delta \mathbf{w} = \alpha (\mathbf{G}_t^\lambda - v(S_t, \mathbf{w})) \nabla_{\mathbf{w}} v(S_t, \mathbf{w})$

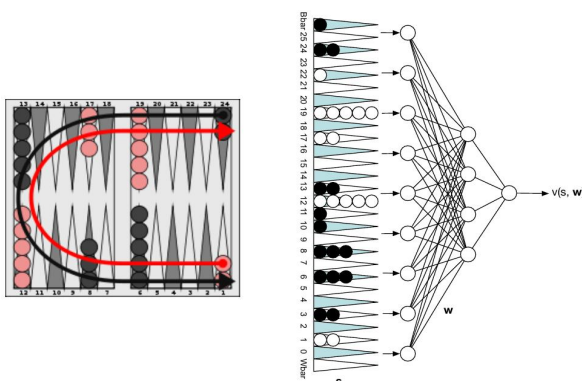
- a. Classic games는 deterministic game이다. 즉 게임 rule에 따라 action을 하면 다음 state가 명확히 정해지는 특징이 있다. 따라서 policy improvement를 할 때, q-value대신 state value만 알아도 충분하다. 왜냐하면 어떤 state의 value만 알면 deterministic 하므로 게임의 rule에 따라 어떤 action을 취하면 되는지 알기 때문이다.
- b. 구체적으로 백의 입장에서 다음 state의 value가 최대가 되는 action을, 흑의 입장에서 다음 state value가 최소가 되는 action을 선택하면 된다. 이렇게 하면 두 players의 joint policy가 improvement된다. 이를 수식으로 표현하면 아래와 같다.

AfterState	$q_*(s, a) = v_*(succ(s, a))$
Action selection for White	$A_t = \operatorname{argmax}_a v_*(succ(S_t, a))$
Action selection for Black	$A_t = \operatorname{argmin}_a v_*(succ(S_t, a))$

- c. 이를 적용한 예시로 Othello 게임에 적용한 Logistello가 있다. 이는 binary-linear value function을 사용하여 돌들의 position의 조합으로 feature를 만들었다. 이를 바탕으로 policy iteration을 사용하여 학습을 하였는데, 현재 policy로 self-play를 통해 game을 만들고 이로 experiences를 생성한 다음, 이를 바탕으로 MC를 사용³⁾하여 현재 policy를 평가한다. 평가를 하고 나면 Greedy policy improvement를 진행하게 된다. 이 때 백은 maximize를 하고 흑은 minimize를 선택하도록 한다.



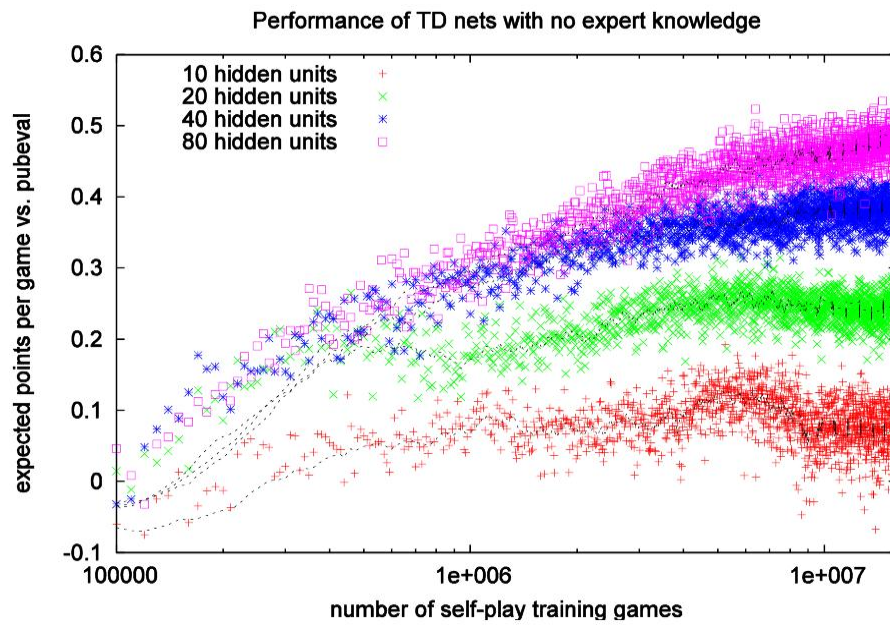
- d. 다른 예시로 Neural Net.을 Backgammon을 적용한 TD Gammon이 있다. Neural Net.을 random으로 초기화한 다음 self-play로 진행하면서 아래의 식과 같이 non-linear TD(λ) algorithm으로 policy를 평가하였다. policy improvement는 exploration없이⁴⁾ Greedy 방식으로만 진행하였다. 이 방식으로 학습하면 항상 수렴하였다.



$$\delta_t = v(S_{t+1}, \mathbf{w}) - v(S_t, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha \delta_t \nabla_{\mathbf{w}} v(S_t, \mathbf{w})$$

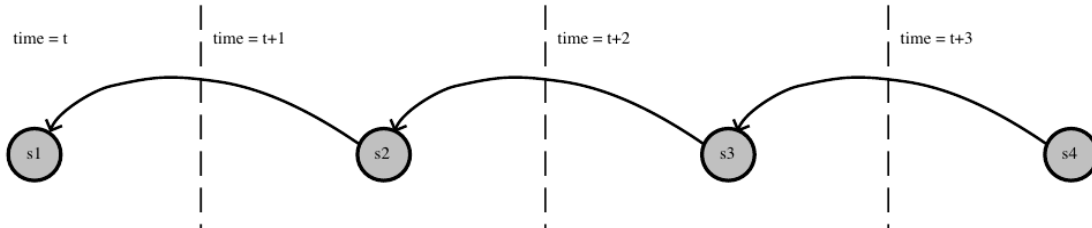
3) Outcome를 regression을 한다는 의미이다. 즉 어떤 state에서 어떤 action을 취하면 어떤 state가 나올지 학습하는 것이다.
 4) Backgammon game 자체에 inherent stochasticity이 있어서 exploration에 불필요하기 때문이다.



3. Combining Reinforcement Learning and Minimax Search

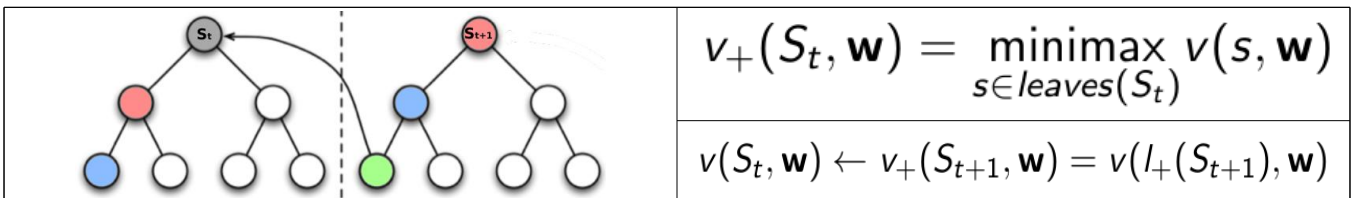
3-1. Introduction

#. 이전까지는 learning과 search가 분리되어 있었다. 즉 TD learning으로 state value를 구한 다음에, $(v(S_t, w) \leftarrow v(S_{t+1}, w))$ 다음에 이를 이용하여 $\text{minimax search}(v_+(S_t, w) = \min_{s \in \text{leaves}(S_t)} \max v(s, w))$ 를 하였다. 이 방법의 한계는 search의 결과가 learning에 영향을 주지 않는다는 것이다. 이를 보완하기 위해 search과정을 learning에 포함시켜 더 좋은 state value를 찾는 3가지 방법론(TD-Root, TD-Leaf, Tree Strap)을 배울 것이다.



3-2. TD-Root

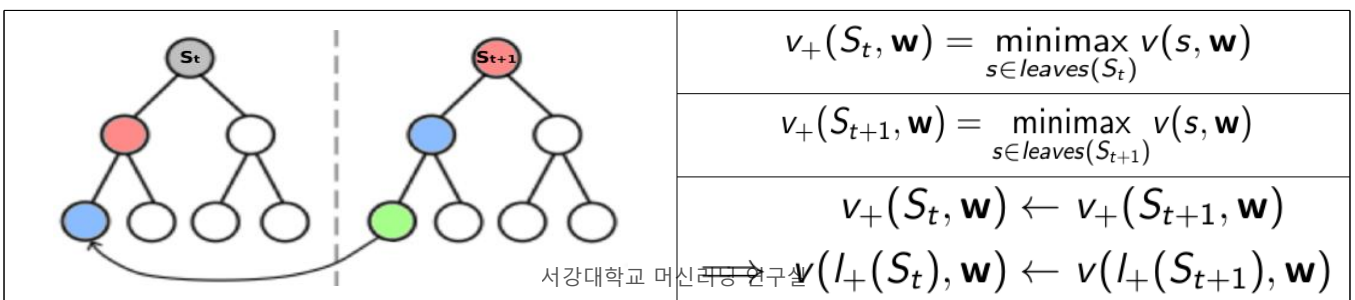
#. TD-Learning은 $v(S_t)$ 을 $v(S_{t+1})$ 을 이용하여 update하는 것이었다. TD-Root는 $v(S_t)$ 을 update할 때, $v(S_{t+1})$ 에서 minimax search의 결과를 이용하는 것이 핵심이다. 왜냐하면 한 step 진행 후에서의 search 결과가 더 정확하기 때문에 이 방향으로 $v(S_t)$ 을 update하는 것은 의미가 있기 때문이다.



a. 위의 notation에서 v_+ 은 minimax search의 결과 값을 의미하며 l_+ 은 state s 에서 minimax search의 결과로 도달하는 leaf node를 의미한다.

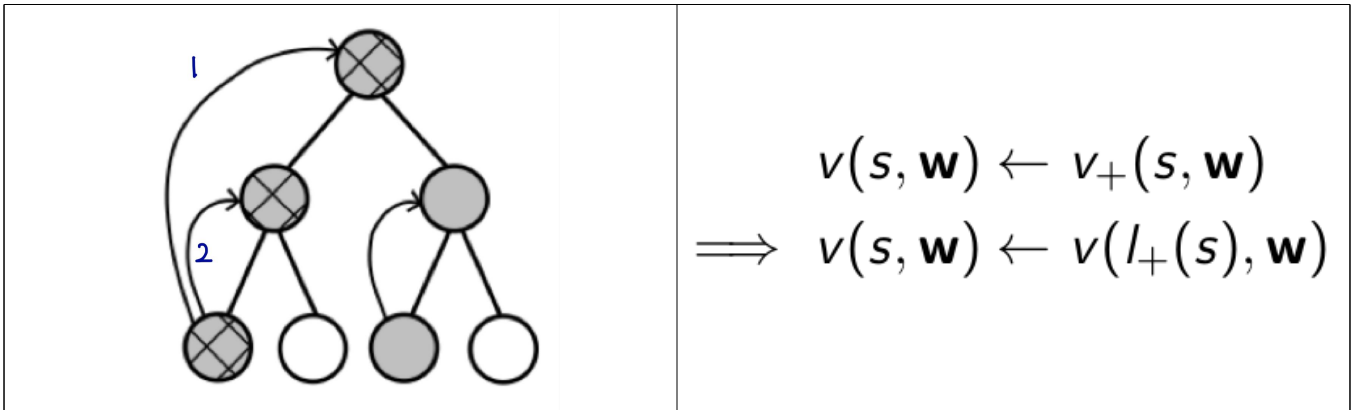
3-3. TD-Leaf

#. TD-Root에선 $v(S_t)$ 을 update해주었다면, TD-Leaf에선 $v(l_+(S_t), w)$ 을 update해주는 것이다. 즉 S_{t+1} 에서 minimax search한 결과 값을 이용하여 S_t 에서 minimax search한 결과 값을 update해주는 것이다.



3-4. Tree Strap

#. 특정 state에서의 minimax search를 통해 terminal state에 도달하게 되면 결과 value를 얻게 된다. 이 때, 지나왔던 각각의 node에서 terminal state까지 minimax search한 결과 value로 각각 update를 해주는 방법이다.



a. TD-Root와 TD-Leaf는 real step과 imaginary step이 섞여있는 상태인데, 차라리 상상 속에서 모든 것을 해 버리자는 배경에서 나온 것이다. 즉 real step을 하지 않고 모두 imaginary step(minimax search)로 구성되어 있다.

3-5. Simulation-Based Search

#. 대표적으로 MCTS이다. 이 방법은 Search과정에서 MC control을 사용하는 방법으로써, root state에서 시작하여 self-play로 simulation을 통해 다양한 experiences를 생성하고 이를 이용하여 Model-Free RL 중 하나인 MC를 쓴 것이다. 즉 action value를 experiences에서 학습하고 좋은 action들만 선택하는 방법이다. 하지만 exploration을 고려하여 UCT algorithm을 사용하여 매 node에서⁵⁾ action을 선택할 때, exploration 또한 진행한다.

5) Bandit 문제와 동일하다.

4. Reinforcement Learning in Imperfect-Information Games

4-1. Smooth UCT Search

#. imperfect information일 때 사용하는 방법이며 average strategy를 생성하여 상대방의 average behaviour를 배우고 UCT를 수정하는 방법이다.

$A \sim \begin{cases} \text{UCT}(S), & \text{with probability } \eta \\ \pi_{avg}(\cdot S), & \text{with probability } 1 - \eta \end{cases}$	$\pi_{avg}(a s) = \frac{N(s,a)}{N(s)}$
--	--

a. 주로 Poker에서 사용하는 방법이다.

5. Conclusion

Program	Input features	Value Fn	RL	Training	Search
Chess <i>Meep</i>	Binary <i>Pieces, pawns, ...</i>	Linear	TreeStrap	Self-Play / Expert	$\alpha\beta$
Checkers <i>Chinook</i>	Binary <i>Pieces, ...</i>	Linear	TD leaf	Self-Play	$\alpha\beta$
Othello <i>Logistello</i>	Binary <i>Disc configs</i>	Linear	MC	Self-Play	$\alpha\beta$
Backgammon <i>TD Gammon</i>	Binary <i>Num checkers</i>	Neural network	TD(λ)	Self-Play	$\alpha\beta$ / MC
Go <i>MoGo</i>	Binary <i>Stone patterns</i>	Linear	TD	Self-Play	MCTS
Scrabble <i>Maven</i>	Binary <i>Letters on rack</i>	Linear	MC	Self-Play	MC search
Limit Hold'em <i>SmooCT</i>	Binary <i>Card abstraction</i>	Linear	MCTS	Self-Play	-

저작자 : 서강대학교 데이터마이닝 연구실 서강현 (010 2572 5665)

서강대학교 머신러닝 연구실

서강현

지도교수 : 양지훈