

Reinforcement Learning Paper Review
**Mastering the game of Go
with Deep Neural Networks and Tree Search**

D.Silver et al., Nature, 2016

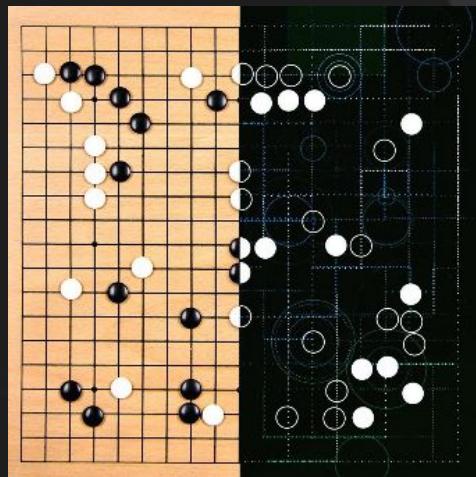
2020.06.01

성민석(Minsuk Sung)
minsuksung@korea.ac.kr

A L P H A G O

발표 목차

- 서론
- 알파고 핵심 키워드
 - 롤아웃 정책(Rollout policy)
 - SL 정책 네트워크(SL policy network)
 - RL 정책 네트워크(RL policy network)
 - 가치 네트워크(Value network)
 - 몬테 카를로 트리 탐색 (MCTS)
 - 대량 CPU / GPU
- 실험
- 결론



A L P H A G O

서론

ALPHAGO

최강 바둑 인공지능의 등장

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver^{1*}, Aja Huang^{1*}, Chris J. Maddison¹, Arthur Guez¹, Laurent Sifre¹, George van den Driessche¹, Julian Schrittwieser¹, Ioannis Antonoglou¹, Veda Panneershelvam¹, Marc Lanctot¹, Sander Dieleman¹, Dominik Grewe¹, John Nham², Nal Kalchbrenner¹, Ilya Sutskever², Timothy Lillicrap¹, Madeleine Leach¹, Koray Kavukcuoglu¹, Thore Graepel¹ & Demis Hassabis¹

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses ‘value networks’ to evaluate board positions and ‘policy networks’ to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s , under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game’s breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 35$, $d \approx 80$) and especially Go ($b \approx 250$, $d \approx 150$), exhaustive search is infeasible^{2–4}, but the effective search space can be reduced by two general principles. policies^{13–15} or value functions¹⁶ based on a linear combination of input features.

Recently, deep convolutional neural networks have achieved unprecedented performance in visual domains: for example, image classification¹⁷, face recognition¹⁸, and playing Atari games¹⁹. They use many layers of neurons, each arranged in overlapping tiles, to construct increasingly abstract, localized representations of an image²⁰. We employ a similar architecture for the game of Go. We pass in the board position as a 19×19 image and use convolutional layers to construct a

최강 바둑 인공지능의 등장

nature research

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver^{1*}, Aja Huang^{1†}, Chris J. Maddison¹, Arthur Guez¹, Laurent Sifre¹, George van den Driessche¹, Julian Schrittwieser¹, Ioannis Antonoglou¹, Veda Panneershelvam¹, Marc Lanctot¹, Sander Dieleman¹, Dominik Grewe¹, John Nham², Nal Kalchbrenner¹, Ilya Sutskever², Timothy Lillicrap¹, Madeleine Leach¹, Koray Kavukcuoglu¹, Thore Graepel¹ & Demis Hassabis¹

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses ‘value networks’ to evaluate board positions and ‘policy networks’ to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s , under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game’s breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 35$, $d \approx 80$) and especially Go ($b \approx 250$, $d \approx 150$), exhaustive search is infeasible^{2–4}, but the effective search space can be reduced by two general principles.

policies^{13–15} or value functions¹⁶ based on a linear combination of input features.

Recently, deep convolutional neural networks have achieved unprecedented performance in visual domains: for example, image classification¹⁷, face recognition¹⁸, and playing Atari games¹⁹. They use many layers of neurons, each arranged in overlapping tiles, to construct increasingly abstract, localized representations of an image²⁰. We employ a similar architecture for the game of Go. We pass in the board position as a 19×19 image and use convolutional layers to construct a

최강 바둑 인공지능의 등장

nature research



David Silver

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver^{1*}, Aja Huang^{1†}, Chris J. Maddison¹, Arthur Guez¹, Laurent Sifre¹, George van den Driessche¹, Julian Schrittwieser¹, Ioannis Antonoglou¹, Veda Panneershelvam¹, Marc Lanctot¹, Sander Dieleman¹, Dominik Grewe¹, John Nham², Nal Kalchbrenner¹, Ilya Sutskever², Timothy Lillicrap¹, Madeleine Leach¹, Koray Kavukcuoglu¹, Thore Graepel¹ & Demis Hassabis¹

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses ‘value networks’ to evaluate board positions and ‘policy networks’ to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s , under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game’s breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 35, d \approx 80$) and especially Go ($b \approx 250, d \approx 150$), exhaustive search is infeasible^{2–4}, but the effective search space can be reduced by two general principles.

policies^{13–15} or value functions¹⁶ based on a linear combination of input features.

Recently, deep convolutional neural networks have achieved unprecedented performance in visual domains: for example, image classification¹⁷, face recognition¹⁸, and playing Atari games¹⁹. They use many layers of neurons, each arranged in overlapping tiles, to construct increasingly abstract, localized representations of an image²⁰. We employ a similar architecture for the game of Go. We pass in the board position as a 19×19 image and use convolutional layers to construct a

최강 바둑 인공지능의 등장

nature research



David Silver

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver^{1*}, Aja Huang^{1†}, Chris J. Maddison¹, Arthur Guez¹, Laurent Sifre¹, George van den Driessche¹, Julian Schrittwieser¹, Ioannis Antonoglou¹, Veda Panneershelvam¹, Marc Lanctot¹, Sander Dieleman¹, Dominik Grewe¹, John Nham², Nal Kalchbrenner¹, Ilya Sutskever², Timothy Lillicrap¹, Madeleine Leach¹, Koray Kavukcuoglu¹, Thore Graepel¹ & Demis Hassabis¹

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses ‘value networks’ to evaluate board positions and ‘policy networks’ to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s , under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game’s breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 35$, $d \approx 80$) and especially Go ($b \approx 250$, $d \approx 150$), exhaustive search is infeasible^{2–3}, but the effective search space can be reduced by two general principles. policies^{13–15} or value functions¹⁶ based on a linear combination of input features.

Recently, deep convolutional neural networks have achieved unprecedented performance in visual domains: for example, image classification¹⁷, face recognition¹⁸, and playing Atari games¹⁹. They use many layers of neurons, each arranged in overlapping tiles, to construct increasingly abstract, localized representations of an image²⁰. We employ a similar architecture for the game of Go. We pass in the board position as a 19×19 image and use convolutional layers to construct a

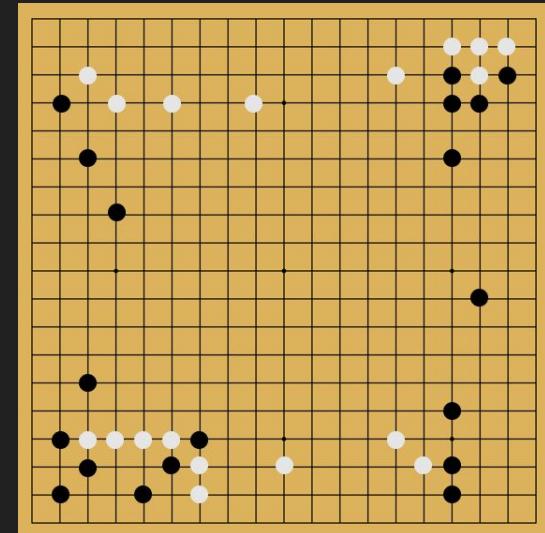


Demis Hassabis

바둑의 기본 규칙(1/4)

바둑에 있어서 기본이 되는 4가지 기본 규칙

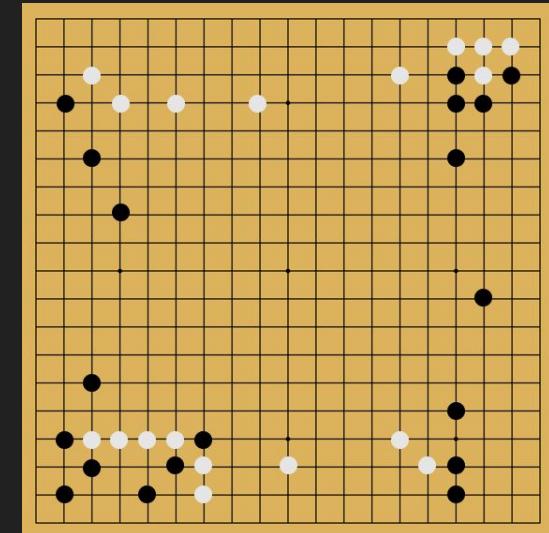
1. 바둑판의 가로세로의 노선이 겹치는 교차점 위에 흑돌, 백돌의 순서로 돌을 둔다
2. 상대의 돌을 둘러싸면 잡을 수 있다.
3. 상대의 돌에 둘러싸인 점은 둘 수 없다.
4. 최종적으로 땅이 큰 쪽이 이긴다.
 - a. 이 때 땅의 계산은 흑의 수 혹은 백의 수 + 둘러싸인 집의 수
 - b. 백을 잡은 사람은 덤 7.5집(중국 규칙)을 더한다.



바둑의 기본 규칙(2/4)

바둑에 있어서 기본이 되는 4가지 기본 규칙

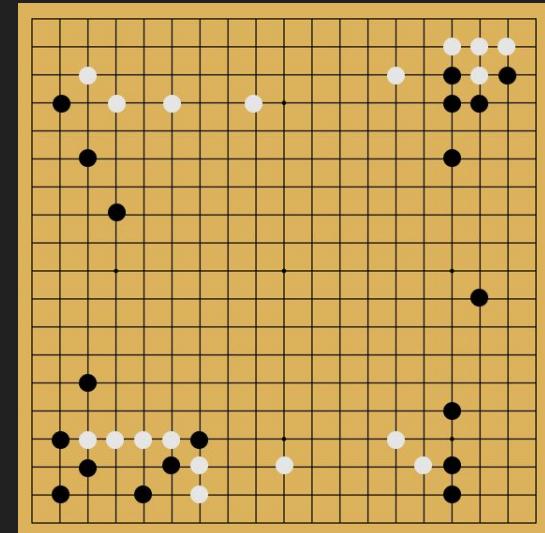
1. 바둑판의 가로세로의 노선이 겹치는 교차점 위에 흑돌, 백돌의 순서로 돌을 둔다
2. 상대의 돌을 둘러싸면 잡을 수 있다.
3. 상대의 돌에 둘러싸인 점은 둘 수 없다.
4. 최종적으로 땅이 큰 쪽이 이긴다.
 - a. 이 때 땅의 계산은 흑의 수 혹은 백의 수 + 둘러싸인 집의 수
 - b. 백을 잡은 사람은 덤 7.5집(중국 규칙)을 더한다.



바둑의 기본 규칙(3/4)

바둑에 있어서 기본이 되는 4가지 기본 규칙

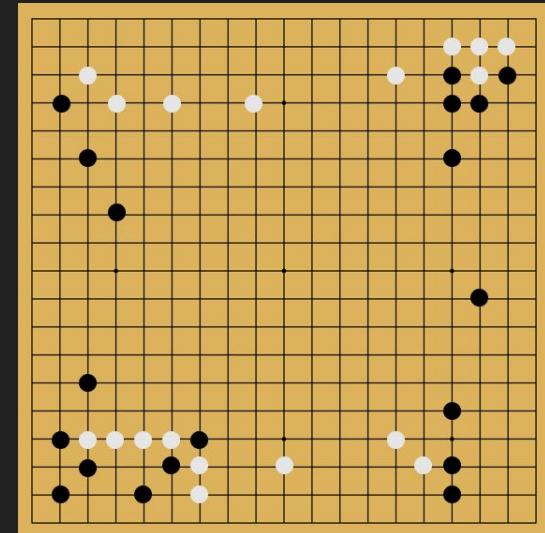
1. 바둑판의 가로세로의 노선이 겹치는 교차점 위에 흑돌, 백돌의 순서로 돌을 둔다
2. 상대의 돌을 둘러싸면 잡을 수 있다.
3. 상대의 돌에 둘러싸인 점은 둘 수 없다.
4. 최종적으로 땅이 큰 쪽이 이긴다.
 - a. 이 때 땅의 계산은 흑의 수 혹은 백의 수 + 둘러싸인 집의 수
 - b. 백을 잡은 사람은 덤 7.5집(중국 규칙)을 더한다.



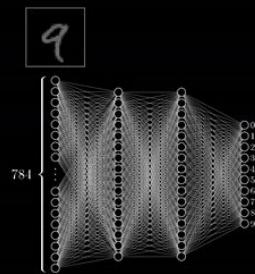
바둑의 기본 규칙(4/4)

바둑에 있어서 기본이 되는 4가지 기본 규칙

1. 바둑판의 가로세로의 노선이 겹치는 교차점 위에 흑돌, 백돌의 순서로 돌을 둔다
2. 상대의 돌을 둘러싸면 잡을 수 있다.
3. 상대의 돌에 둘러싸인 점은 둘 수 없다.
4. 최종적으로 땅이 큰 쪽이 이긴다.
 - a. 이 때 땅의 계산은 흑의 수 혹은 백의 수 + 둘러싸인 집의 수
 - b. 백을 잡은 사람은 덤 7.5집(중국 규칙)을 더한다.



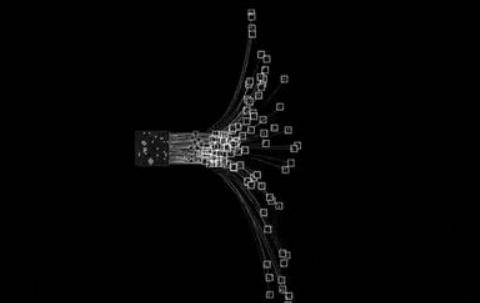
알파고에 적용된 기술들(1/3)



심층 학습
(Deep Learning)



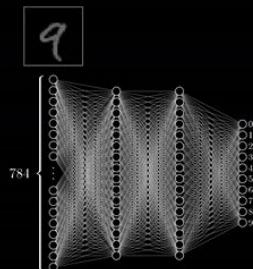
강화 학습
(Reinforcement Learning)



트리 탐색
(Tree Search)

알파고에 적용된 기술들(2/3)

정책 네트워크 / 가치 네트워크



심층 학습
(Deep Learning)

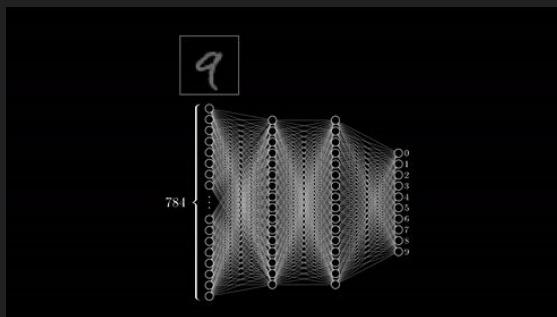


강화 학습
(Reinforcement Learning)



트리 탐색
(Tree Search)

알파고에 적용된 기술들(3/3)



심층 학습
(Deep Learning)



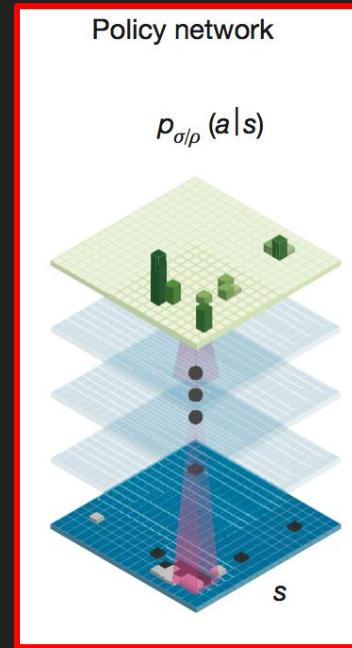
강화 학습
(Reinforcement Learning)



몬테 카를로 트리 탐색
(Monte Carlo Tree Search)

알파고의 3가지 파트

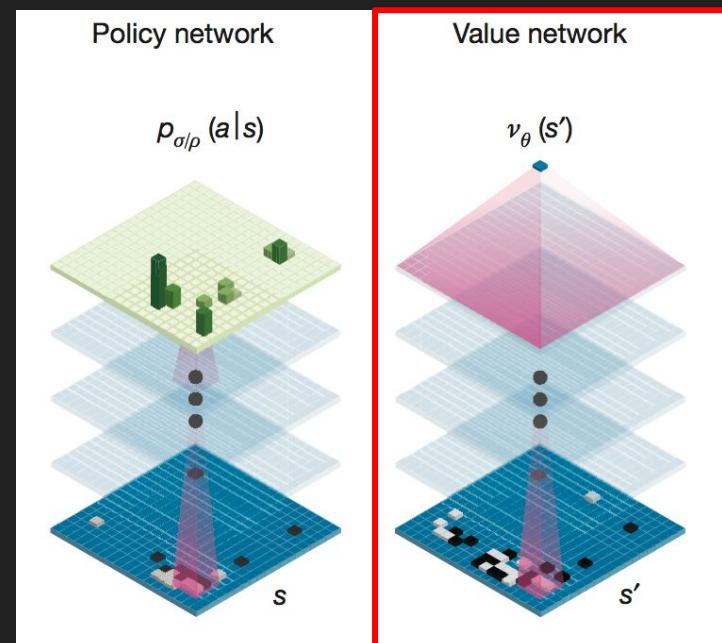
- 정책 네트워크(Policy network)
 - 바둑 고수들의 경기를 모방하여 학습하여 다음 수를 계산



정책 네트워크와 가치 네트워크

알파고의 3가지 파트

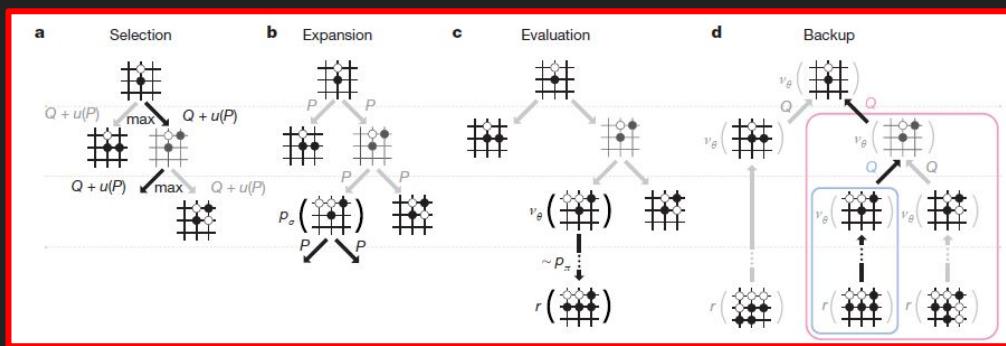
- 정책 네트워크(Policy network)
 - 바둑 고수들의 경기를 모방하여 학습하여 다음 수를 계산
- 가치 네트워크(Value network)
 - 바둑 판의 각 부분을 평가하고 특정 부분의 승리 확률을 계산



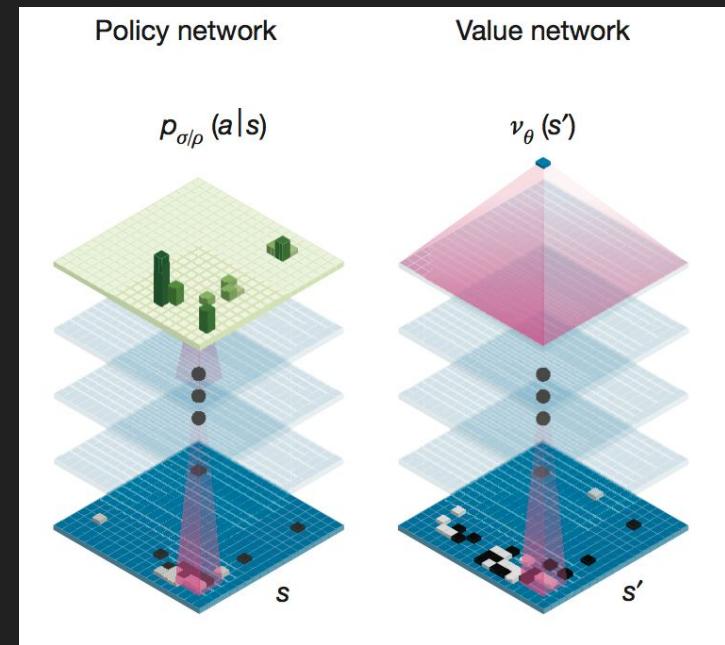
정책 네트워크와 가치 네트워크

알파고의 3가지 파트

- 정책 네트워크(Policy network)
 - 바둑 고수들의 경기를 모방하여 학습하여 다음 수를 계산
- 가치 네트워크(Value network)
 - 바둑 판의 각 부분을 평가하고 특정 부분의 승리 확률을 계산
- 몬테 카를로 트리 탐색(Monte Carlo Tree Search)
 - 여러 변화도를 그려내면서 다음 두어질 수를 예측



알파고에서 적용된 몬테카를로 트리 탐색



정책 네트워크와 가치 네트워크

알파고 핵심 키워드

A L P H A G O



알파고 학습의 핵심 알고리즘

- 방대한 게임 트리 탐색 범위를 효과적으로 줄이기 위해 아래와 같이 2가지 방향으로 학습
 - 넓이 탐색 범위 줄이기: 현재 상태에서 다음 **착수 위치** 찾기
 - 단순 패턴 파악
 - 바둑 기보 패턴 학습
 - 자가 대국
 - 깊이 탐색 범위 줄이기: 해당 착점의 **승률** 계산
 - 판의 부분 승률 계산

롤아웃 정책 (Rollout policy)

A L P H A G O



알파고 학습의 핵심 알고리즘

- 방대한 게임 트리 탐색 범위를 효과적으로 줄이기 위해 아래와 같이 2가지 방향으로 학습
 - 넓이 탐색 범위 줄이기: 현재 상태에서 다음 착수 위치 찾기
 - 단순 패턴 파악 → **롤아웃 정책(Rollout policy)**
 - 바둑 기보 패턴 학습
 - 자가 대국
 - 깊이 탐색 범위 줄이기: 해당 착점의 승률 계산
 - 판의 부분 승률 계산

넓이 탐색 줄이기 - 롤아웃 정책

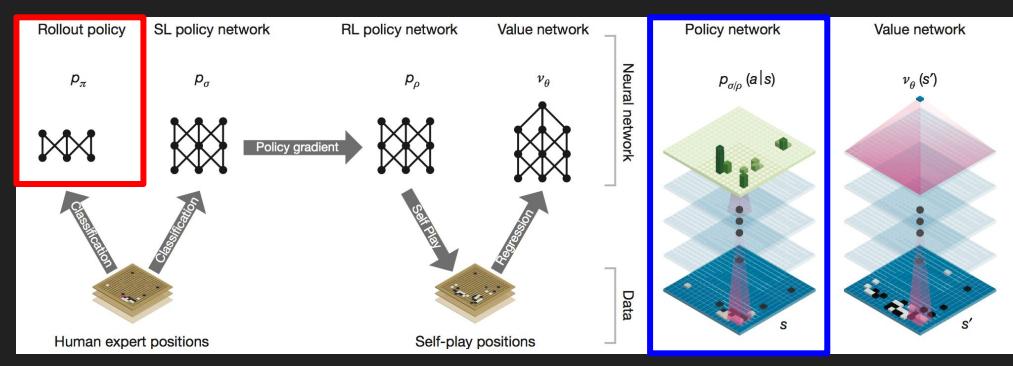
- 단순 패턴 인식 → **롤아웃 정책(Rollout policy)**
 - 알파고에서 수의 표면적인 특징을 바탕으로 그 수가 놓일 예측 확률을 출력하는 수학적 모델
 - 바둑 규칙에 따른 단순 확률 계산하므로 빠르게 평가할 수 있지만, 수의 예측 확률은 다소 떨어짐 (24%)
 - 추후 플레이아웃(Playout)에서 이를 활용하여 몬테카를로 트리 탐색을 진행

Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Feature planes used by the policy network (all but last feature) and value network (all features).

롤아웃 정책에서 사용되는 특징들



알파고 학습 과정 요약

SL 정책 네트워크 (Supervised Learning policy network)

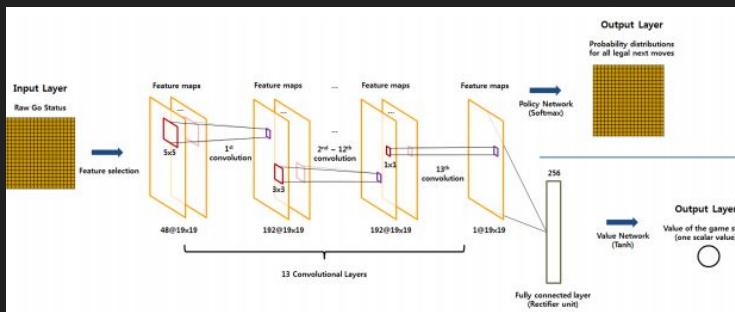
A L P H A G O

알파고 학습의 핵심 알고리즘

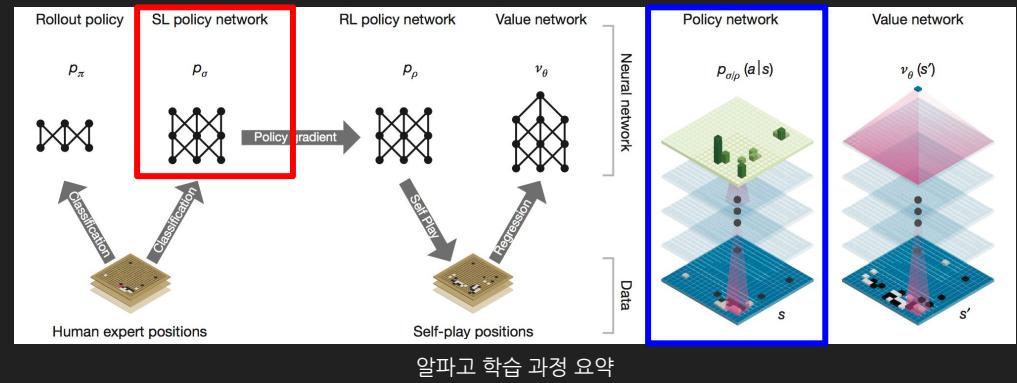
- 방대한 게임 트리 탐색 범위를 효과적으로 줄이기 위해 아래와 같이 2가지 방향으로 학습
 - 넓이 탐색 범위 줄이기: 현재 상태에서 다음 착수 위치 찾기
 - 단순 패턴 파악 → 롤아웃 정책(Rollout policy)
 - 바둑 기보 패턴 학습 → SL 정책 네트워크(SL policy network)
 - 자가 대국
 - 깊이 탐색 범위 줄이기: 해당 착점의 승률 계산
 - 판의 부분 승률 계산

넓이 탐색 줄이기 - SL 정책 네트워크

- 바둑 기보 패턴 학습 → **SL 정책 네트워크(SL policy network)**
 - 알파고가 한 수 한 수 착수할 때 사용하는 네트워크
 - 합성곱 신경망(Convolutional Neural Network)를 지도학습(Supervised Learning)



알파고의 정책 / 가치 네트워크 구조(소프트웨어정책연구소)

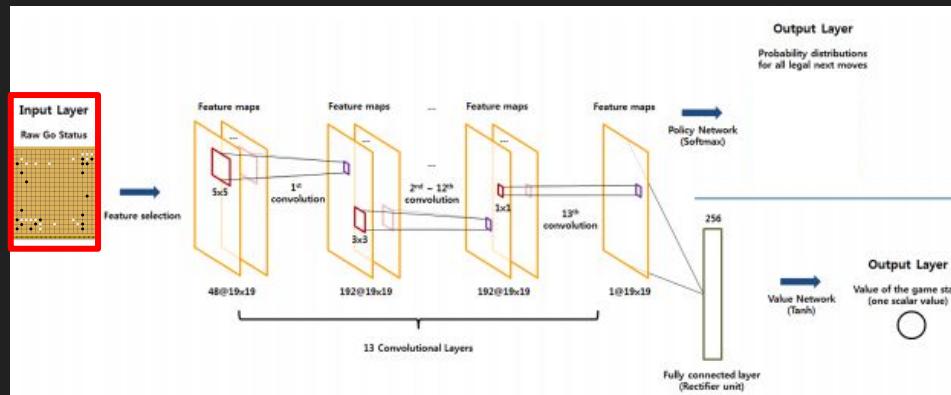


알파고 학습 과정 요약

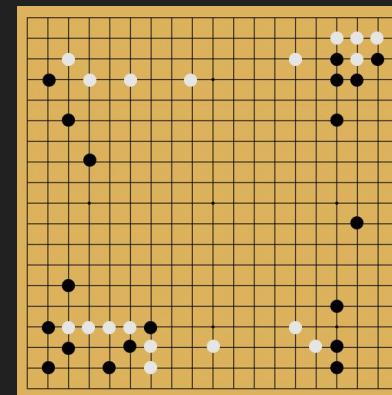
넓이 탐색 줄이기 - SL 정책 네트워크

- SL 정책 네트워크의 구조(1/3)

- 입력층: 48채널
- 제1층: 5×5 의 192종류의 필터와 ReLU 함수
- 제2~12층: 3×3 의 192종류의 필터와 ReLU 함수
- 제13층: 1×1 의 1종류의 필터와 위치에 의존하는 항, 마지막으로 소프트맥스(Softmax) 함수

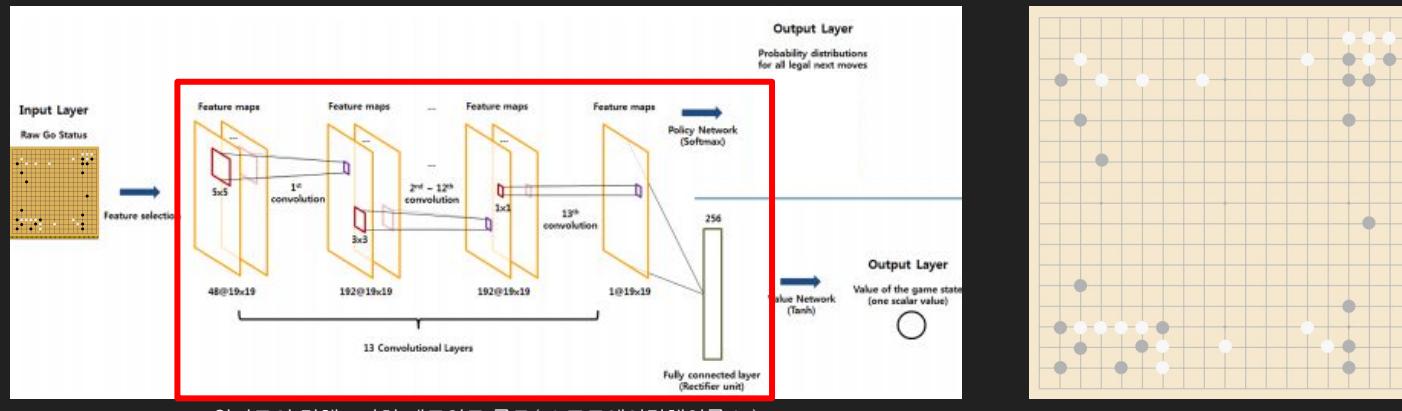


알파고의 정책 / 가치 네트워크 구조(소프트웨어정책연구소)



넓이 탐색 줄이기 - SL 정책 네트워크

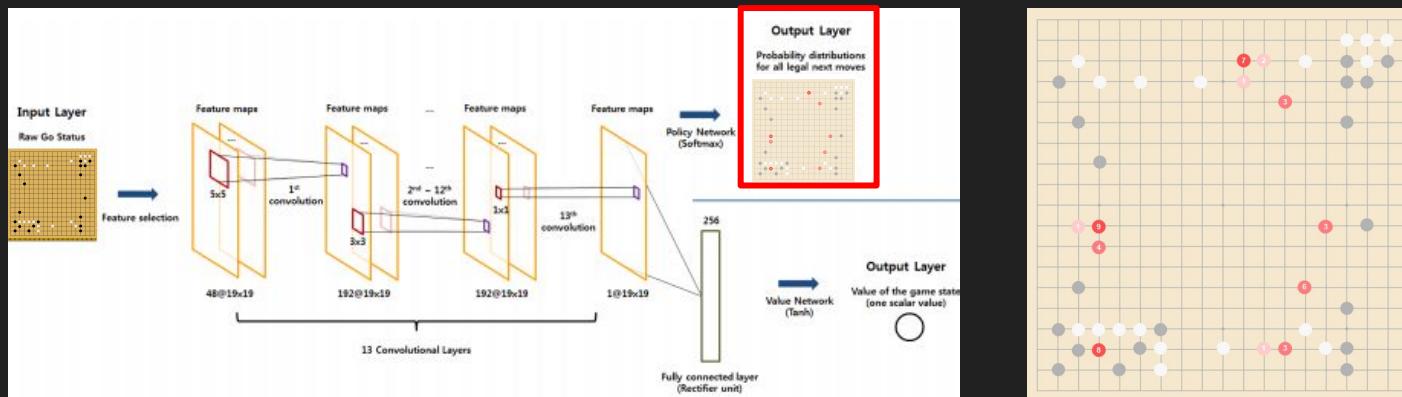
- SL 정책 네트워크의 구조(2/3)
 - 입력층: 48채널
 - 제1층: 5x5의 192종류의 필터와 ReLU 함수
 - 제2~12층: 3x3의 192종류의 필터와 ReLU 함수
 - 제13층: 1x1의 1종류의 필터와 위치에 의존하는 항, 마지막으로 소프트맥스(Softmax) 함수



알파고의 정책 / 가치 네트워크 구조(소프트웨어정책연구소)

넓이 탐색 줄이기 - SL 정책 네트워크

- SL 정책 네트워크의 구조(3/3)
 - 입력층: 48채널
 - 제1층: 5x5의 192종류의 필터와 ReLU 함수
 - 제2~12층: 3x3의 192종류의 필터와 ReLU 함수
 - 제13층: 1x1의 1종류의 필터와 위치에 의존하는 항, 마지막으로 소프트맥스(Softmax) 함수



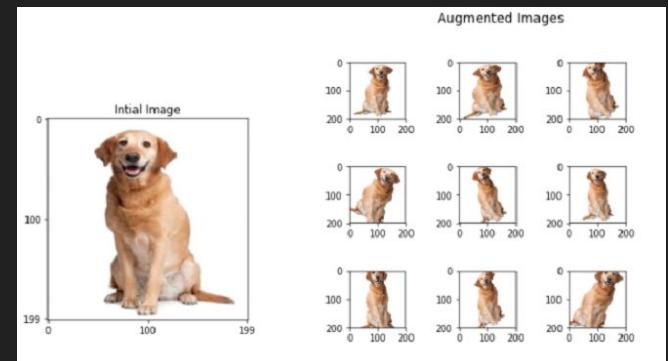
알파고의 정책 / 가치 네트워크 구조(소프트웨어정책연구소)

넓이 탐색 줄이기 - SL 정책 네트워크

- SL 정책 네트워크의 학습데이터
 - SL 정책 네트워크의 약 400만 개의 필터 가중치 파라미터를 학습하기 위해서 대량의 고품질 학습 데이터가 필요
→ 인터넷 대국 사이트 Kiseido Go Service (KGS)에서 확보
 - KGS에서 6단 이상 플레이어 16만 대국 기보를 확보하여 데이터 학습
 - 1국의 기보는 보통 200수 정도이므로 1국에서 200개 정도의 학습데이터 확보시 대략 3,000만 개의 학습자료 생성
 - 앞서 생성된 데이터를 90도씩 회전(4) 및 반전(2) → 8배 증폭 가능
 - 결과적으로 약 2억 4000만개의 학습 데이터를 사용



인터넷 바둑 대국 사이트 KGS

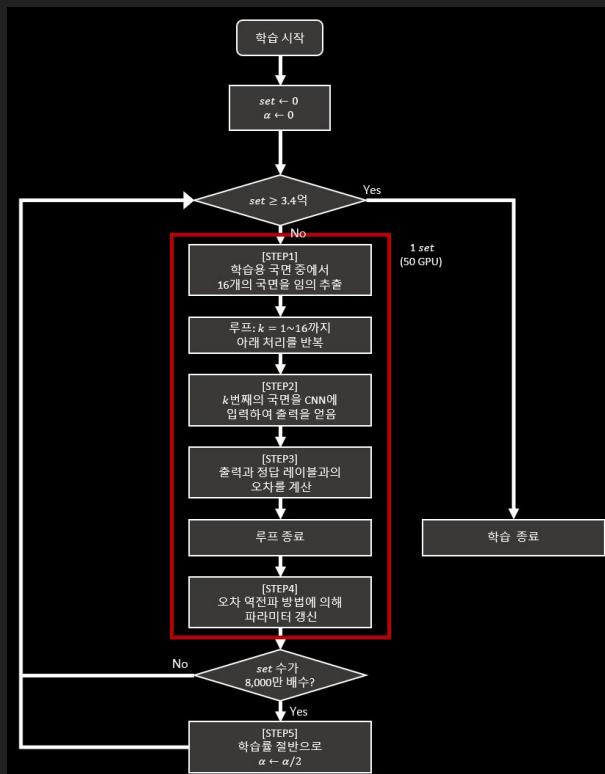


데이터 증폭(Data Augmentation)의 예시

넓이 탐색 줄이기 - SL 정책 네트워크

- SL policy network 의 학습과정
 - STEP1: 국면을 무작위로 추출
 - 전체 3,000만 국면 중에서 16개 국면으로 무작위로 추출 1세트
 - STEP2: 정책 네트워크 경유의 출력을 계산
 - 각 국면을 정책 네트워크를 통해서 출력을 계산
 - STEP3: 오차를 계산
 - STEP2의 출력과 정답 레이블과의 오차를 계산
 - STEP4: 확률적 경사 하강법(SGD)에 의한 파라미터 갱신
 - STEP1부터 STEP3까지를 각 16개의 각 국면에 대해 계산한 후, 오차를 역전파하여 파라미터 갱신
- 8,000만 세트 반복할 때마다 학습률을 절반으로 하고, 전체 3.4억 세트 반복했을 때 학습 종료
 - CPU 1개 사용시, 약 60년 걸리는 계산량
 - GPU 1개 사용시, 약 3년이 걸리는 계산량

넓이 탐색 줄이기 - SL 정책 네트워크



총 학습시간: 약 3주(50GPU 필요)

1GPU: 약 3년 → 고속화 필요

1CPU: 약 60년 → 고속화 필요

- 이렇게 학습된 SL 정책 네트워크가 유럽 챔피언 2단 판 후이와의 대결에서 사용된 정책 네트워크

알파고를 분석하며 배우는 인공지능, 제이펍 p85~86

RL 정책 네트워크

(Reinforcement Learning policy network)

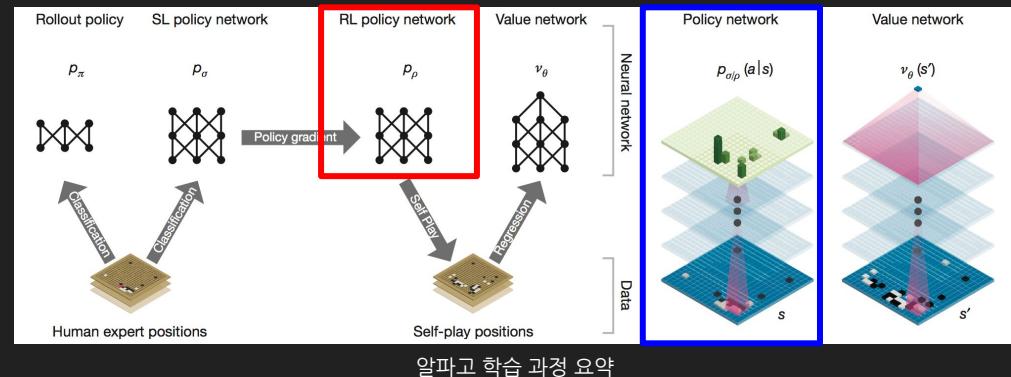
A L P H A G O

알파고 학습의 핵심 알고리즘

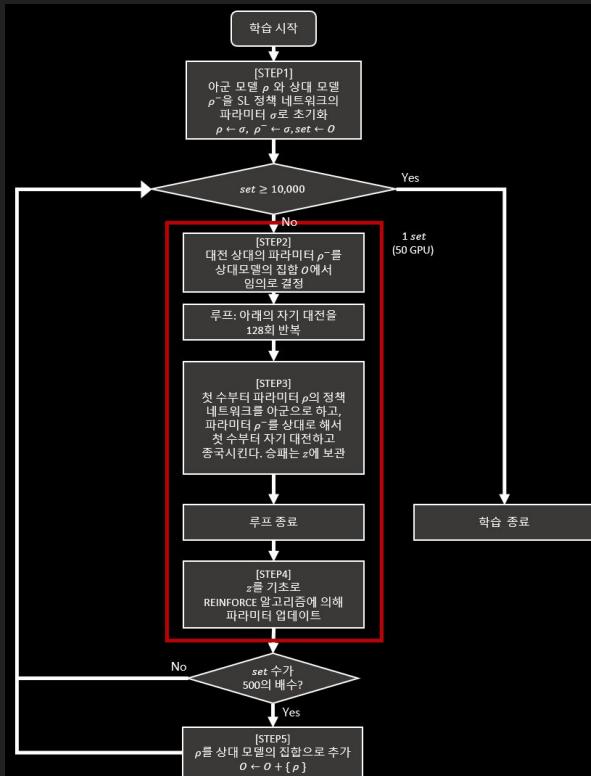
- 방대한 게임 트리 탐색 범위를 효과적으로 줄이기 위해 아래와 같이 2가지 방향으로 학습
 - 넓이 탐색 범위 줄이기: 현재 상태에서 다음 착수 위치 찾기
 - 단순 패턴 파악 → 롤아웃 정책(Rollout policy)
 - 바둑 기보 패턴 학습 → SL 정책 네트워크(SL policy network)
 - 자가 대국 → RL 정책 네트워크(RL policy network)
 - 깊이 탐색 범위 줄이기: 해당 착점의 승률 계산
 - 판의 부분 승률 계산

넓이 탐색 줄이기 - RL 정책 네트워크

- 자가 대국 → RL 정책 네트워크(RL policy network)
 - 앞서 SL 정책 네트워크에서 학습한 기보에만 최적화되는 것을 방지하기 위해 자가 대국(Self-play)을 실행
 - 기보 패턴을 학습한 CNN들(SL 정책 네트워크)간 대국으로 승패에 따른 강화학습(RL)을 실행
 - REINFORCE와 같은 Policy Gradient 알고리즘 적용



넓이 탐색 줄이기 - RL 정책 네트워크



총 학습시간: 약 1일(50GPU 병렬)
1GPU: 약 30일 → 고속화 필요

- 기존의 SL 정책 네트워크와 비교시
→ 직접 대결에서 80% 이상 승률
- 최종적으로 몬테카를로 트리 탐색과 조합할 때 성능이 더 강해지는 것으로 보임
- RL 정책 네트워크는 가치 네트워크의 학습을 위한 학습 데이터 생성에 사용됨

알파고를 분석하며 배우는 인공지능, 제이펍 p135~136

가치 네트워크 (Value network)

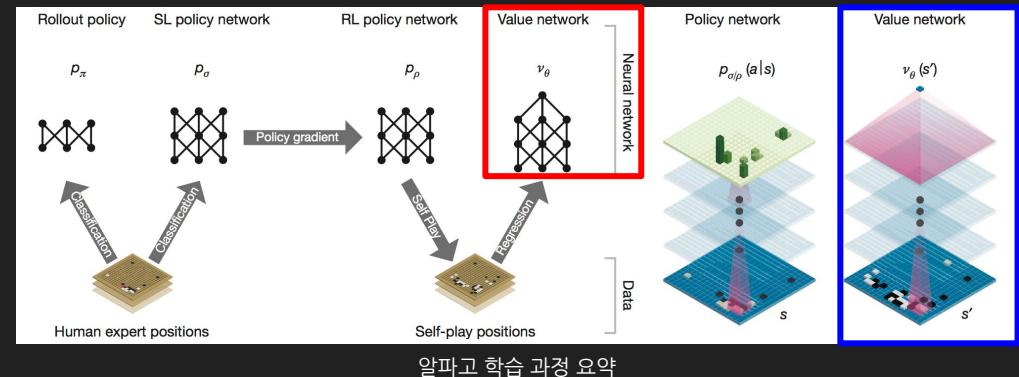
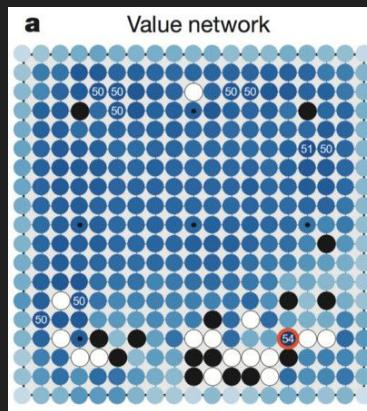
A L P H A G O

알파고 학습의 핵심 알고리즘

- 방대한 게임 트리 탐색 범위를 효과적으로 줄이기 위해 아래와 같이 2가지 방향으로 학습
 - 넓이 탐색 범위 줄이기: 현재 상태에서 다음 착수 위치 찾기
 - 단순 패턴 파악 → 롤아웃 정책(Rollout policy)
 - 바둑 기보 패턴 학습 → SL 정책 네트워크(SL policy network)
 - 자가 대국 → RL 정책 네트워크(RL policy network)
 - 깊이 탐색 범위 줄이기: 해당 착점의 승률 계산
 - 판의 부분 승률 계산 → 가치 네트워크(Value network)

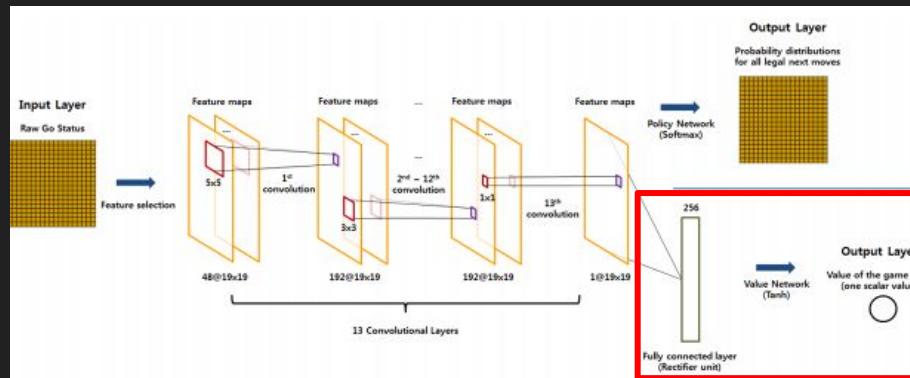
깊이 탐색 줄이기 - 가치 네트워크

- RL 정책 네트워크를 통해서 판의 부분에 해당하는 승률을 계산 → **가치 네트워크(Value network)**
 - RL 정책 네트워크에서 확률 함수 대신 보상 함수를 정의하여 보상의 예측이 잘 맞는 방향으로 학습을 진행
 - DQN과 유사한 형태지만 RL 정책 네트워크의 Policy Gradient와 다름
 - 가치 네트워크는 결국 바둑에서 현재 국면의 평가함수 역할
 - SL 정책 네트워크에서 학습한 KGS 기보 데이터에 과적합되는 걸 방지하기 위해 자가 대국(Self-play) 실시
 - RL 정책 네트워크간 자가 대국 후 생성된 기보로 가치 네트워크 학습



깊이 탐색 줄이기 - 가치 네트워크

- **가치 네트워크**의 구조 → 앞서 설명한 SL 정책 네트워크의 구조와 유사함
 - 입력층: 48채널
 - 제1층: 5×5 의 192종류의 필터와 ReLU 함수
 - 제2~12층: 3×3 의 192종류의 필터와 ReLU 함수
 - 제14,15층: 전체 결합 네트워크를 채용하여 최종적으로 하이퍼볼릭탄젠트(tanh) 함수를 통과시켜 승률 예측치 계산
 - 1에 가까울수록 입력 국면의 두는 쪽이 승률 예측값이 크고
-1에 가까울수록 상대방의 승률이 높음을 의미



알파고의 정책 / 가치 네트워크 구조(소프트웨어정책연구소)

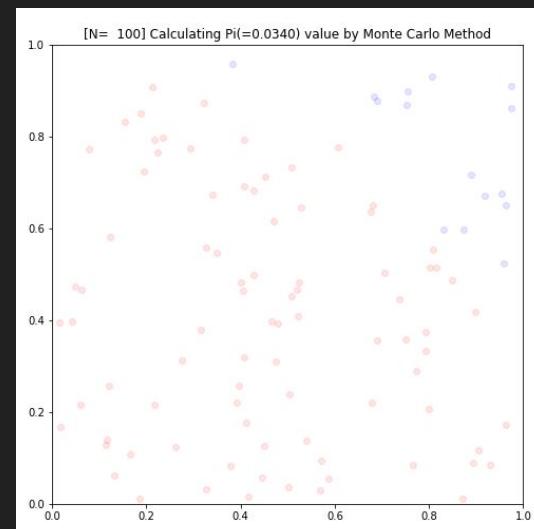
몬테카를로 트리 탐색 (Monte-Carlo Tree Search)

A L P H A G O



몬테 카를로 방법

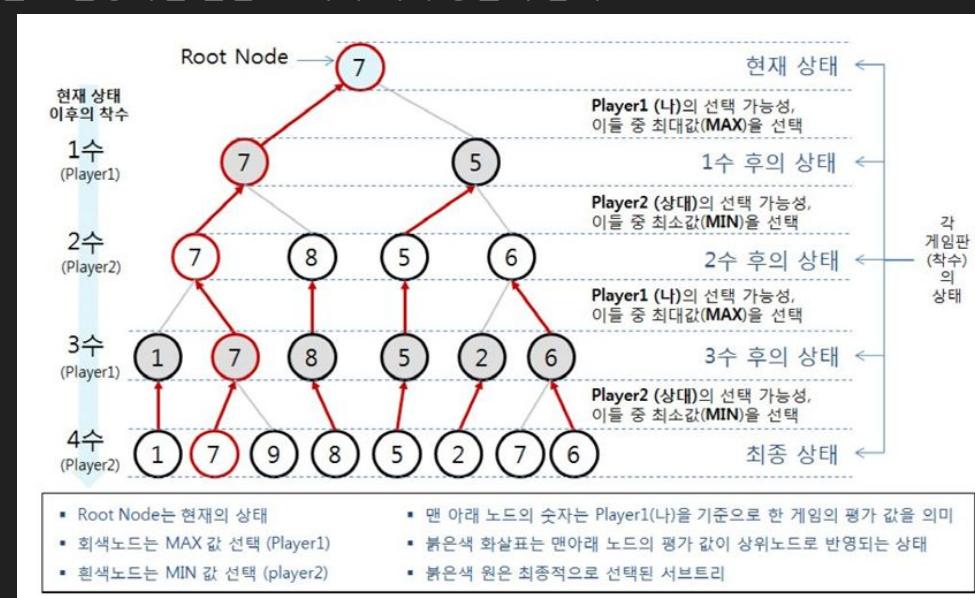
- 몬테 카를로 방법(Monte Carlo method)
 - 난수를 이용하여 함수의 값을 확률적으로 계산하는 알고리즘
 - 계산하려는 값이 닫힌 형식으로 표현되지 않거나 복잡한 경우에 근사적으로 계산할 때 사용
- 예제: 원주율 구하기
 - $[0,1]$ 사이의 점을 표집
 - 표집한 점과 원점 사이의 거리가 반지름 1인 원 안에 속하는지 확인
 - 위 두 과정을 충분히 반복하여 원에 속한 점들의 개수를 계산



몬테 카를로 방법을 통해 원주율 구하기

최소최대 트리

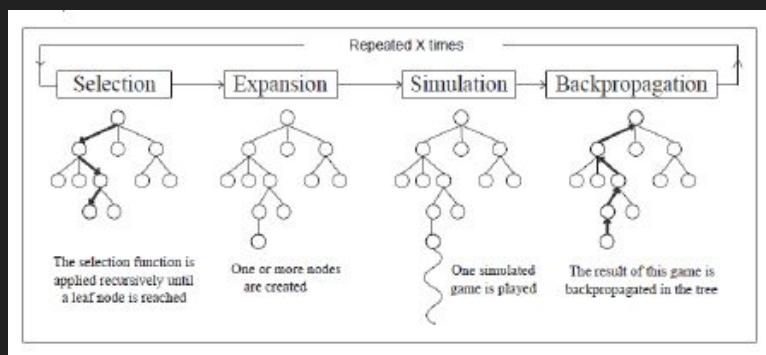
- 바둑은 2인 제로섬 유한 확정 완정 정보 게임
 - 게임 트리에서의 리프 노드를 단순히 한쪽의 승률로 단정하면 안됨 → 흑과 백의 승률의 합이 100%
 - 다시 말해서 바둑의 게임트리를 만들어 갈 때, 한쪽의 승률만 고려해서는 안됨
- 최소최대트리(Min-Max Tree)
 - 나의 차례라면 승률을 **극대화(Max)**하고 상대방의 차례라면 승률을 **최소화(Min)**시키자
 - 모든 리프 노드에 흑의 승률이 부여되어 있으면 흑의 차례에서 자식 노드의 승률의 최댓값, 백의 차례에서 자식 노드의 승률의 최솟값을 취하는 것을 반복하여 진행함
 - 효율적인 탐색을 위해 알파 베타법 사용하여 탐색 영역을 줄이기도 함



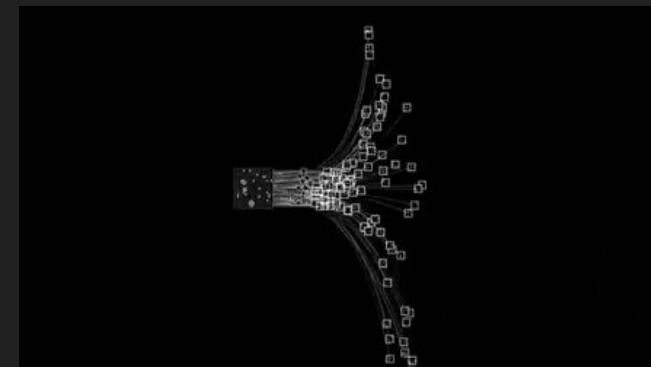
최소최대 트리의 예시

몬테카를로 트리 탐색

- 플레이아웃(Play-out)
 - 특정 국면에서 흑의 순서와 백의 순서가 짧은 시행 시간으로 교대로 수를 선택하고 종국까지 진행하는 기법
- 몬테카를로 트리 탐색(Monte-Carlo Tree Search, MCTS)
 - 바둑과 같이 게임 트리 범위가 상당히 넓은 모든 경로를 탐색하는 것은 불가능할 때 효율적
 - 주어진 시간 내 상대방이 둘 것 같은 수를 두어 가장 효과적으로 탐색하는 과정
 - 현재 상태(Selection)에서 한 단계 예측(Expansion) 후 시뮬레이션(Simulation)한 최종 결과에 따른 트리 상태 정보를 업데이트(Backpropagation)하는 과정을 여러번 반복하는 과정



[Chaslot et al., 2018]



게임 트리 탐색의 예시

몬테카를로 트리 탐색의 과정

- 선택(Selection)

- 현재 바둑판 상태에서 특정 시점까지 착수 선택 → 리프노드가 선택될 때까지 선택
- UCB1 알고리즘을 통해 승률과 바이어스가 가장 큰 값 선택
- 정책 네트워크 사용

- 확장(Expansion)

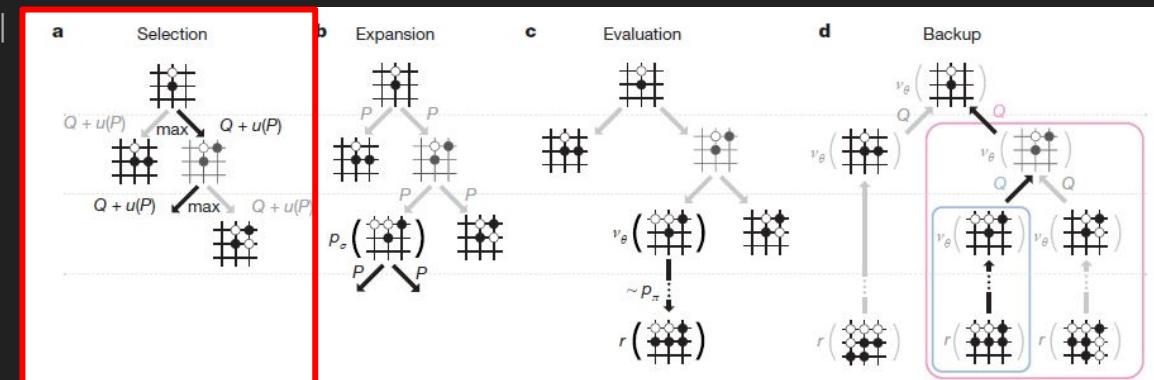
- 탐색 경로의 마지막 노드까지 확장
- 현재 상태에서 다음 착수 위치 찾기
- 넓이 탐색 범위 줄이기

- 평가(Evaluation)

- 게임이 끝날 때까지 시뮬레이션
- 해당 착점의 승률 계산
- 깊이 탐색 범위 줄이기
- 가치 네트워크 사용

- 갱신(Backup)

- 결과가 얼마나 좋은지 파악



알파고에서의 몬테카를로 트리 탐색

몬테카를로 트리 탐색의 과정

- 선택(Selection)

- 현재 바둑판 상태에서 특정 시점까지 착수 선택 → 리프노드가 선택될 때까지 선택
- UCB1 알고리즘을 통해 승률과 바이어스가 가장 큰 값 선택
- 정책 네트워크 사용

- 확장(Expansion)

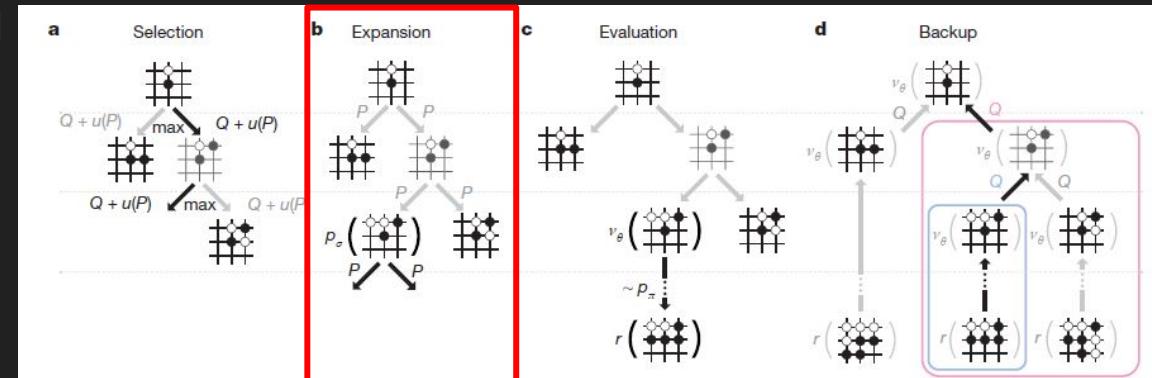
- 탐색 경로의 마지막 노드까지 확장
- 현재 상태에서 다음 착수 위치 찾기
- 깊이 탐색 범위 줄이기

- 평가(Evaluation)

- 게임이 끝날 때까지 시뮬레이션
- 해당 착점의 승률 계산
- 깊이 탐색 범위 줄이기
- 가치 네트워크 사용

- 갱신(Backup)

- 결과가 얼마나 좋은지 파악



몬테카를로 트리 탐색의 과정

- 선택(Selection)

- 현재 바둑판 상태에서 특정 시점까지 착수 선택 → 리프노드가 선택될 때까지 선택
- UCB1 알고리즘을 통해 승률과 바이어스가 가장 큰 값 선택
- 정책 네트워크 사용

- 확장(Expansion)

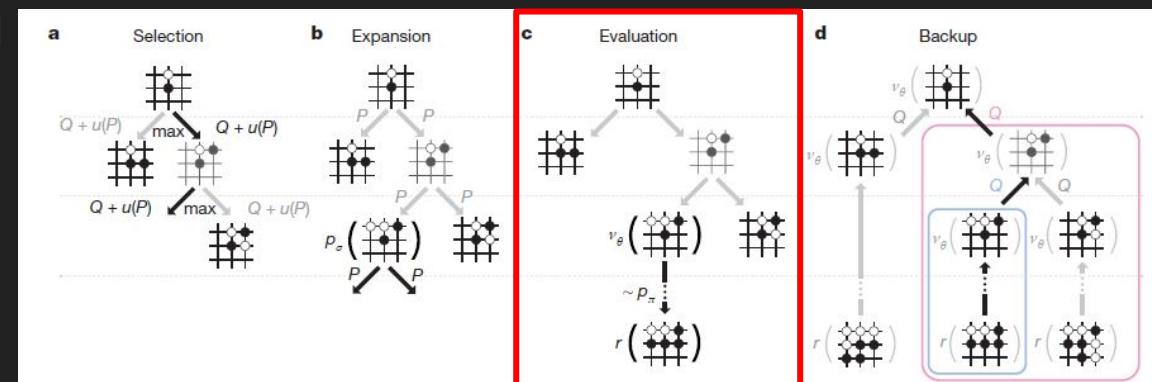
- 탐색 경로의 마지막 노드까지 확장
- 현재 상태에서 다음 착수 위치 찾기
- 넓이 탐색 범위 줄이기

- 평가(Evaluation)

- 게임이 끝날 때까지 시뮬레이션
- 해당 착점의 승률 계산
- 깊이 탐색 범위 줄이기
- 가치 네트워크 사용

- 갱신(Backup)

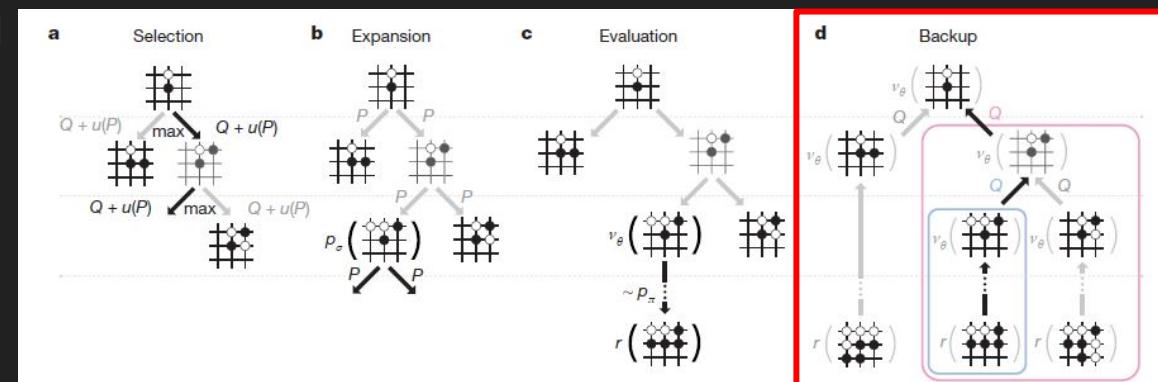
- 결과가 얼마나 좋은지 파악



알파고에서의 몬테카를로 트리 탐색

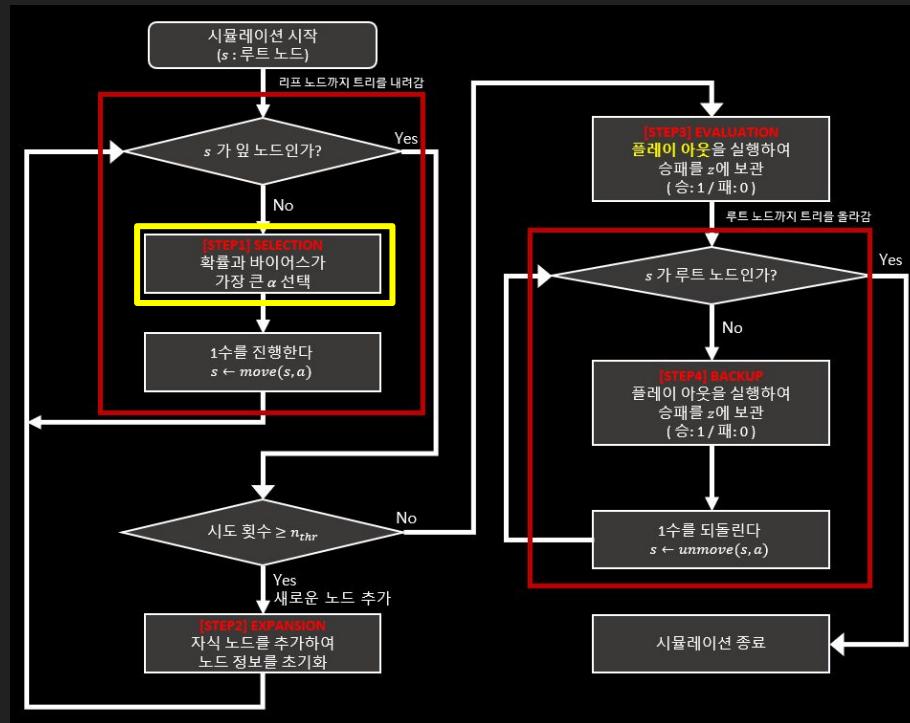
몬테카를로 트리 탐색의 과정

- 선택(Selection)
 - 현재 바둑판 상태에서 특정 시점까지 착수 선택 → 리프노드가 선택될 때까지 선택
 - UCB1 알고리즘을 통해 승률과 바이어스가 가장 큰 값 선택
 - 정책 네트워크 사용
- 확장(Expansion)
 - 탐색 경로의 마지막 노드까지 확장
 - 현재 상태에서 다음 착수 위치 찾기
 - 넓이 탐색 범위 줄이기
- 평가(Evaluation)
 - 게임이 끝날 때까지 시뮬레이션
 - 해당 착점의 승률 계산
 - 깊이 탐색 범위 줄이기
 - 가치 네트워크 사용
- 갱신(Backup)
 - 결과가 얼마나 좋은지 파악



알파고에서의 몬테카를로 트리 탐색

몬테카를로 트리 탐색의 구체적인 과정(1/4)

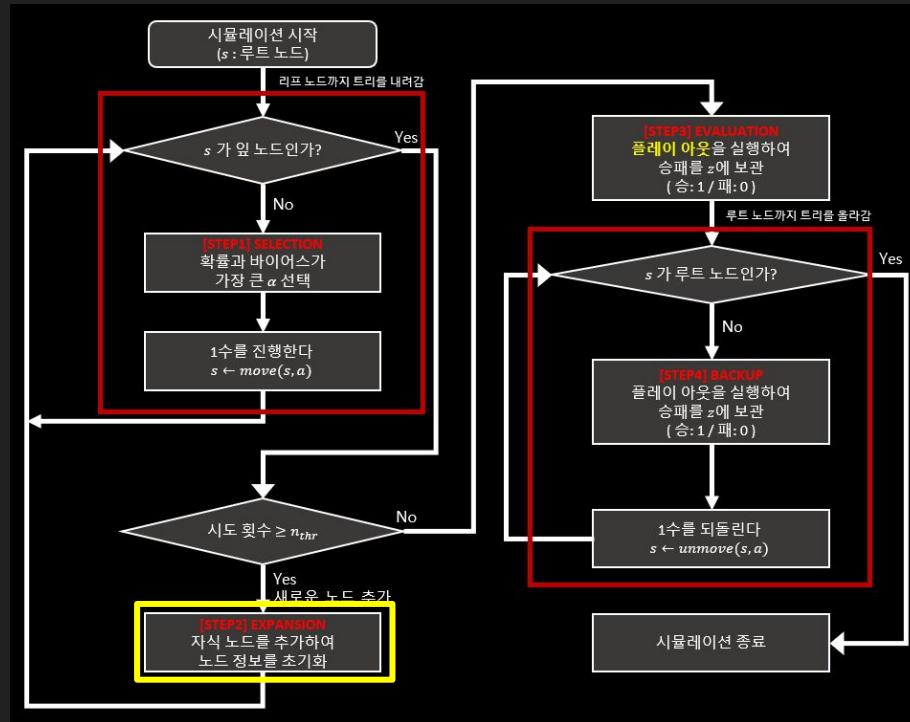


STEP1: 자식 노드 선택 처리

- UCB1을 최대화하는 방향으로 트리를 깊게 전개하는 방법
 - 승률: 이 국면 다음의 플레이 이웃에 의한 승률
 - 바이어스: 승률의 신뢰 구간의 폭
- 기본적으로 승률이 높은 수를 선택하지만, 시도 횟수가 적은 동안에는 다른 수도 시도해봄

알파고를 분석하며 배우는 인공지능, 제이펍 p171~178

몬테카를로 트리 탐색의 구체적인 과정(2/4)

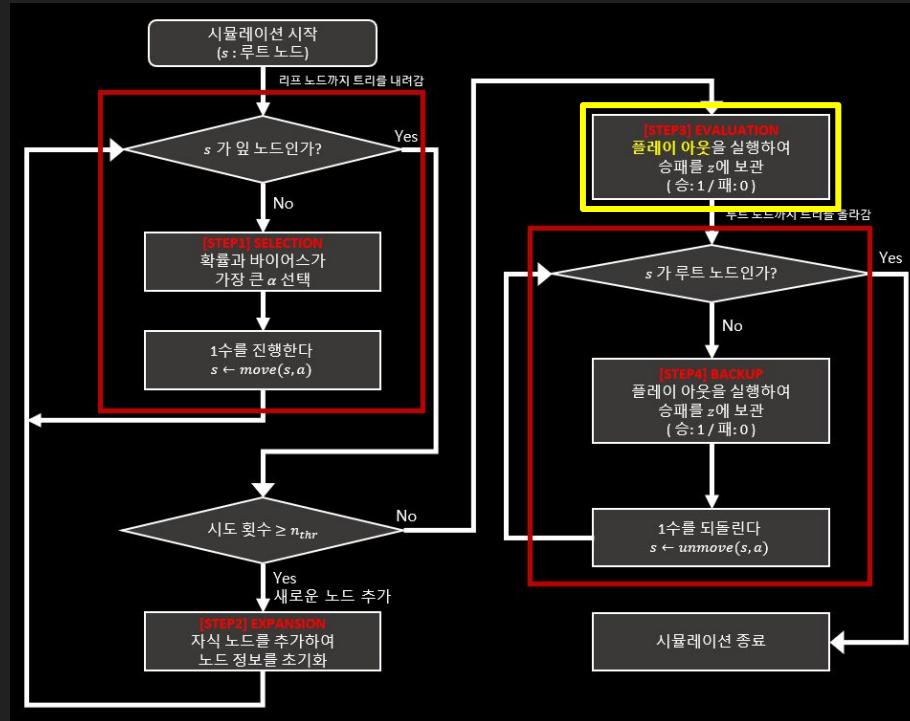


STEP2: 새로운 노드의 전개 처리

- 결과적으로 UCB1이 큰 유망한 자식 노드가 보다 더 깊게 전개
- 한번도 탐색되지 않은 자식 노드를 포함하여 모든 자식 노드를 전개
- 중요한 것 같은 수를 중심으로 점점 노드를 깊게 전개하고 성장해나가는 동작

알파고를 분석하며 배우는 인공지능, 제이펍 p171~178

몬테카를로 트리 탐색의 구체적인 과정(3/4)

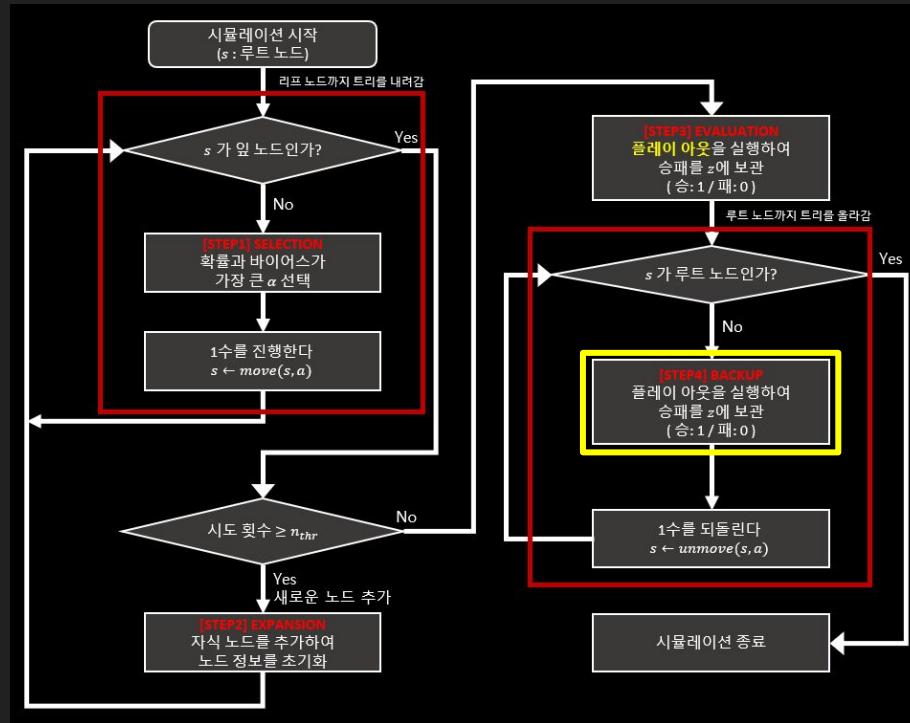


STEP3: 자식 노드의 평가 처리

- 자식 노드의 평가 처리에서는 전개된 마지막 노드에서 플레이 아웃을 실시

알파고를 분석하며 배우는 인공지능, 제이펍 p171~178

몬테카를로 트리 탐색의 구체적인 과정(4/4)



STEP4: 탐색 결과의 갱신 처리

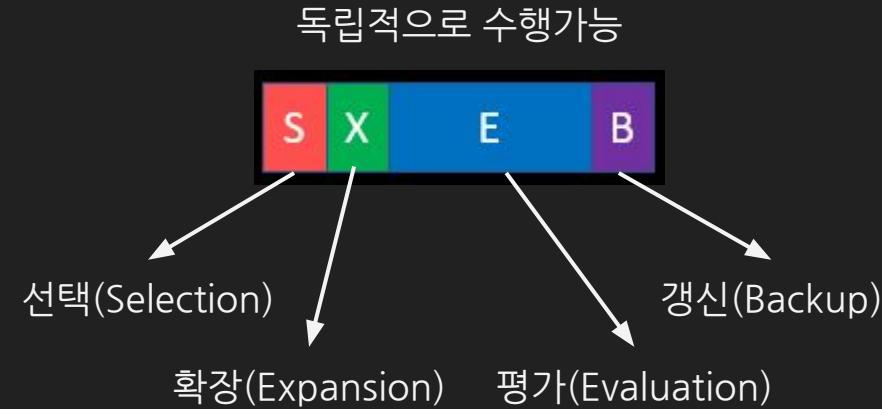
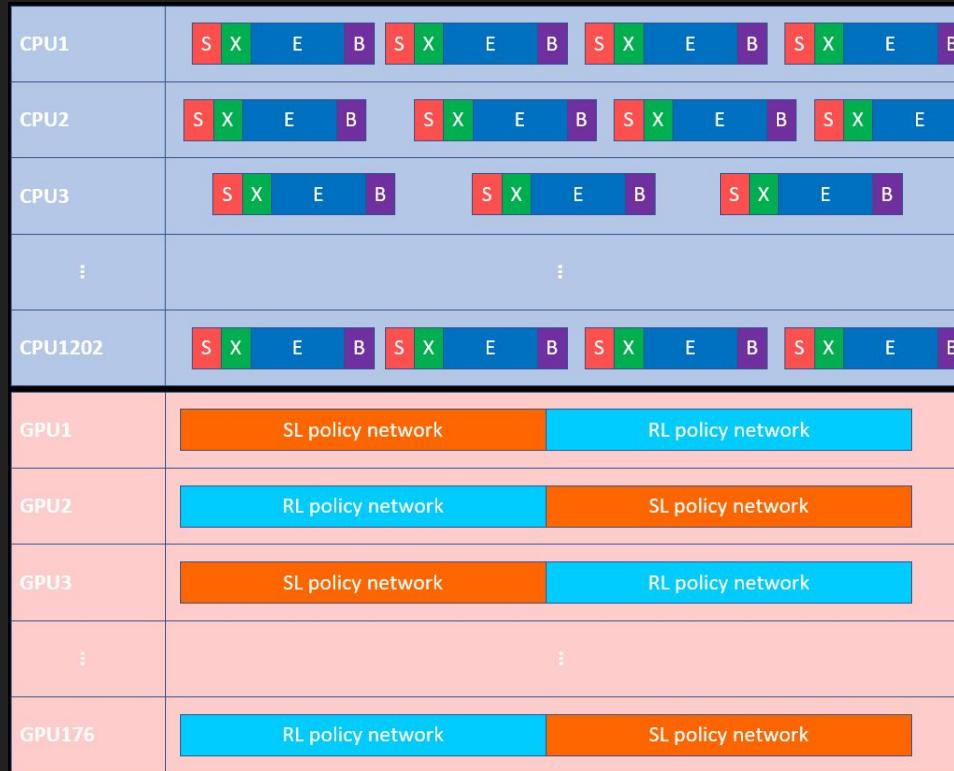
- 탐색 정보의 갱신 처리에서는 플레이 아웃의 결과를 게임 트리에 기록
- 말단의 리프 노드뿐만 아니라 노드를 하나씩 추적해 루트 노드에 이르기까지의 각 노드에 대해서도 승수와 시도 횟수를 갱신

→ 몬테카를로 트리 탐색에서는 탐색 결과를 바탕으로 승률에 따라서 다음의 한 수를 선택 (엄밀히 말하면, 승수와 패수의 합이 가장 많은 수를 선택)

→ 몬테카를로 트리 탐색을 한줄로 정리하자면 랜덤 시뮬레이션을 반복하여 최종적으로 루트 국면에서 가장 시뮬레이션 횟수가 큰 수를 선택하는 방법

알파고를 분석하며 배우는 인공지능, 제이펍 p171~178

대량 CPU / GPU 사용



**CPU 1202
GPU 176**

대량 CPU / GPU 사용

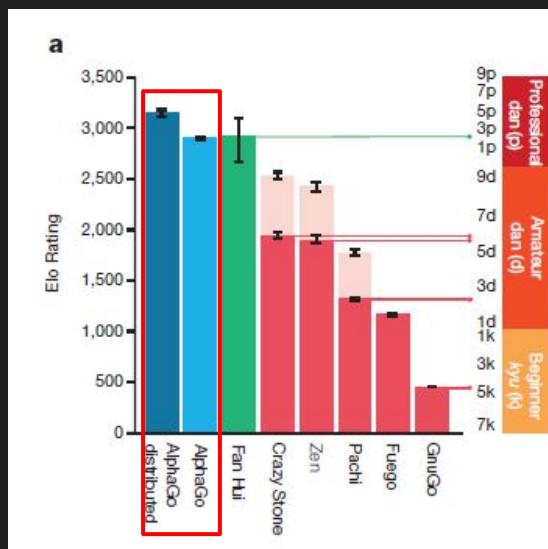
알파고 2년간 운영 비용(최대)			
항목	스펙	총 비용	
1. 개발자	20명 (연봉) 3억6000만원/인당		
2. 프로젝트 기간	2년		
3. 투입리소스	100MM	30억원	
4. 인프라 비용	인프라 CPU(가격/시) GPU(가격/시)	CPU 1920개, GPU 280개 (분산서버 시간당 운영 비용) 282.8\$ $c3.8xlarge * 60\text{대} = \text{CPU}(1920\text{개}) * 1.68\$(\text{가격}/\text{시}) = 100.8\$$ $g2.8xlarge * 70\text{대} = \text{GPU}(280\text{개}) * 2.6\$(\text{가격}/\text{시}) = 182\$$	58억2000만원
합계		88억2000만원	

<그래픽=홍종현 미술기자>

실험

A L P H A G O

알파고와 관련된 몇가지 실험들



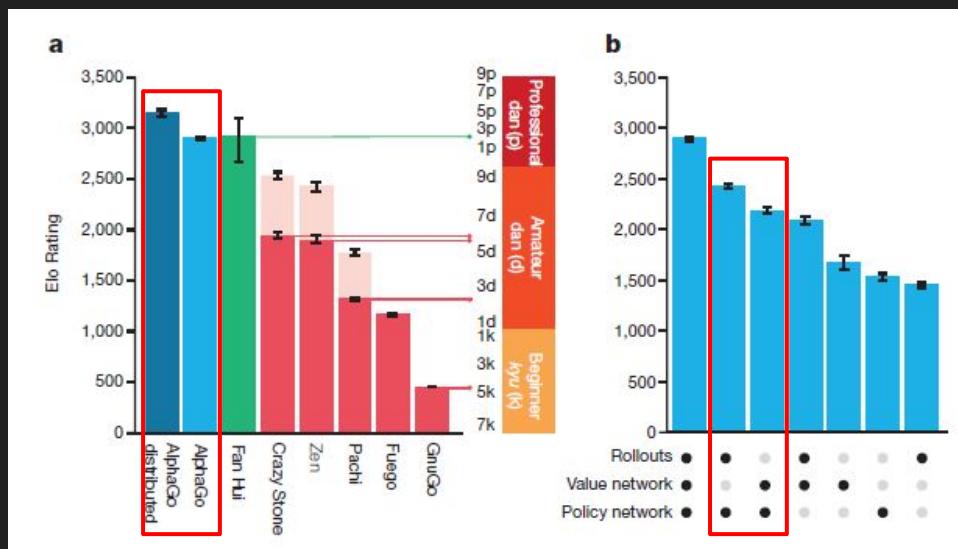
엘로 평점(Elo rating): 여러 플레이어의 승패를 바탕으로 각 플레이어의 실력을 점수로 나타내는 지표

*엘로 평점이 100점 차의 경우에 강한 쪽이 64% 이긴다는 식

**이세돌이나 커제와 같은 기사의 경우 3,000~3,500점 사이

***이세돌 9단과 대진한 2016년 3월 시점, 이미 알파고는 4,000점 육박

알파고와 관련된 몇가지 실험들



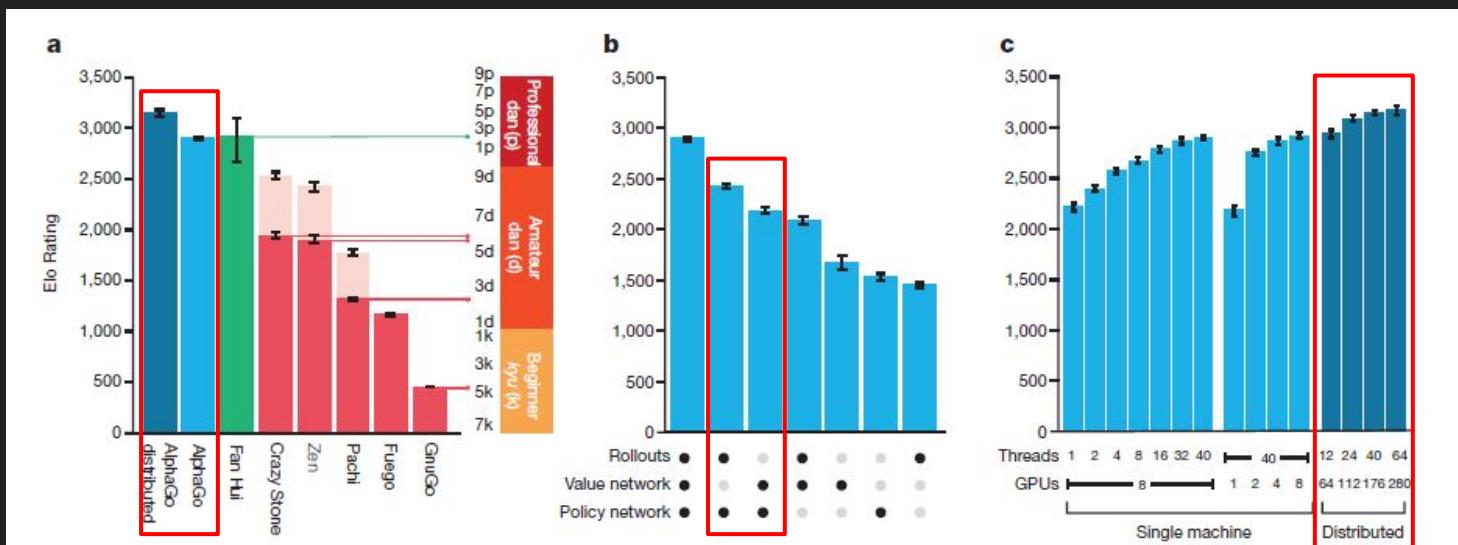
엘로 평점(Elo rating): 여러 플레이어의 승패를 바탕으로 각 플레이어의 실력을 점수로 나타내는 지표

*엘로 평점이 100점 차의 경우에 강한 쪽이 64% 이긴다는 식

**이세돌이나 커제와 같은 기사의 경우 3,000~3,500점 사이

***이세돌 9단과 대전한 2016년 3월 시점, 이미 알파고는 4,000점 육박

알파고와 관련된 몇가지 실험들



엘로 평점(Elo rating): 여러 플레이어의 승패를 바탕으로 각 플레이어의 실력을 점수로 나타내는 지표

*엘로 평점이 100점 차의 경우에 강한 쪽이 64% 이긴다는 식

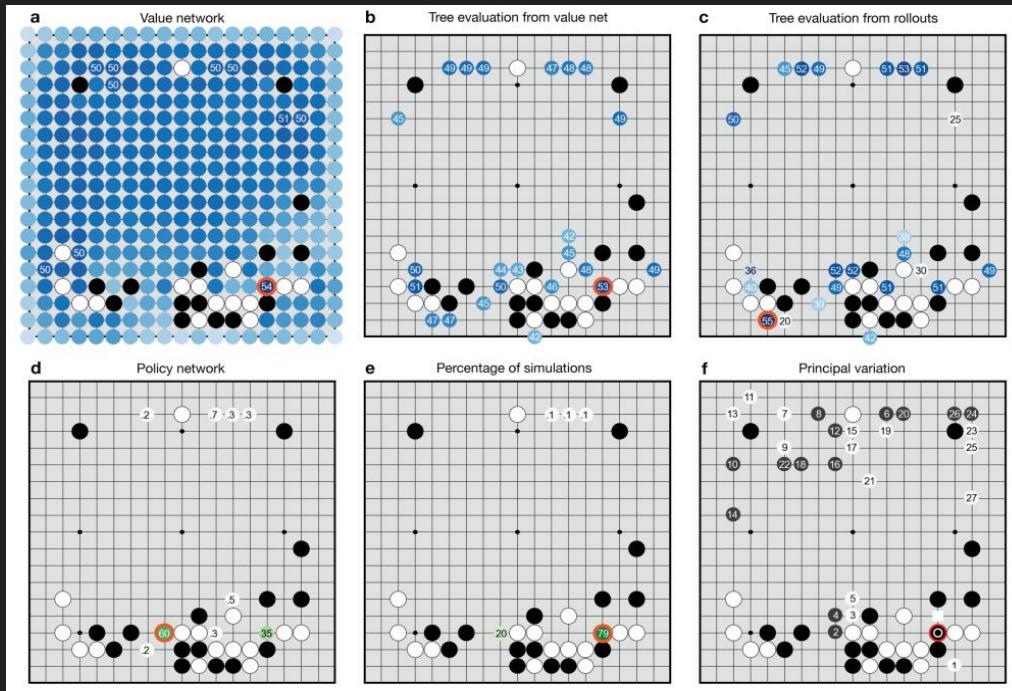
**이세돌이나 커제와 같은 기사의 경우 3,000~3,500점 사이

***이세돌 9단과 대전한 2016년 3월 시점, 이미 알파고는 4,000점 육박

결론

A L P H A G O

요약



알파고가 다음 수를 어떻게 선택하는지에 대한 요약

결론

- **인공지능 기술의 성능 향상**
 - 사람만이 할 수 있을 거라고 생각한 영역인 '바둑'에 인공지능을 적용함으로써 도전적인 연구를 했다.
 - 그동안 바둑은 경우의 수가 매우 많고 복잡하여 인공지능 분야에서 쉽사리 아직 정복하지 못했다. 그러나 알파고를 통해서 인공지능의 성능을 보여주는 실증 사례가 되었다.
- **몬테카를로 트리 탐색을 이용하여 바둑 문제 해결**
 - 가치네트워크처럼 바둑에서의 평가함수 정의
 - 딥러닝을 적용하지 않았던 방법들과 더불어 네트워크를 고려하여 기존의 문제를 더 나은 성능으로 해결한다는데 의의가 있다. 따라서, 어떤 도메인에 기존의 문제 풀이 방법과 딥러닝을 적용하여 새로운 방법론을 적용도 고려해보자.
- **엄청난 양의 데이터와 분산된 CPU / GPU 사용**
 - KGS 기보 사이트에서 16만 기보를 활용하여 약 2억 4,000만개 이상의 데이터 활용했다.
 - 1,202개의 CPU와 176개의 GPU 사용을 통해 학습을 극대화했다.

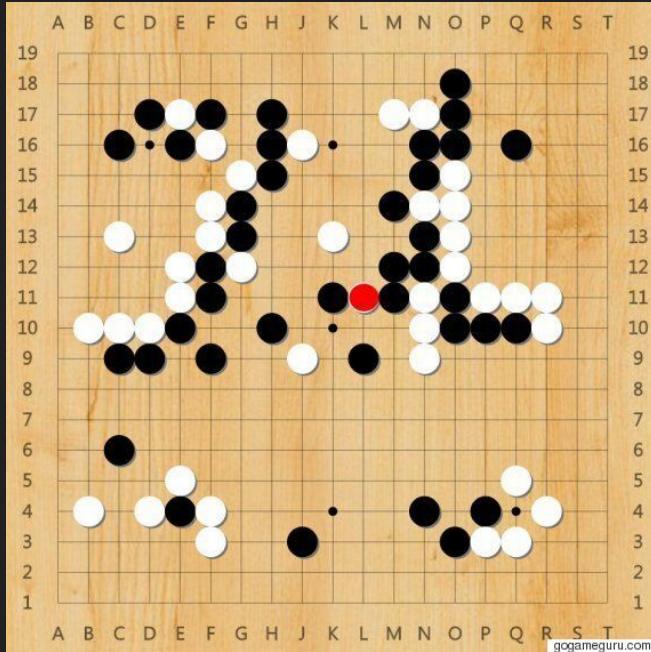
결론

- **인공지능 기술의 성능 향상**
 - 사람만이 할 수 있을 거라고 생각한 영역인 '바둑'에 인공지능을 적용함으로써 도전적인 연구를 했다.
 - 그동안 바둑은 경우의 수가 매우 많고 복잡하여 인공지능 분야에서 쉽사리 아직 정복하지 못했다. 그러나 알파고를 통해서 인공지능의 성능을 보여주는 실증 사례가 되었다.
- **몬테카를로 트리 탐색을 이용하여 바둑 문제 해결**
 - 가치네트워크처럼 바둑에서의 평가함수 정의
 - 딥러닝을 적용하지 않았던 방법들과 더불어 네트워크를 고려하여 기존의 문제를 더 나은 성능으로 해결한다는데 의의가 있다. 따라서, 어떤 도메인에 기존의 문제 풀이 방법과 딥러닝을 적용하여 새로운 방법론을 적용도 고려해보자.
- **엄청난 양의 데이터와 분산된 CPU / GPU 사용**
 - KGS 기보 사이트에서 16만 기보를 활용하여 약 2억 4,000만개 이상의 데이터 활용했다.
 - 1,202개의 CPU와 176개의 GPU 사용을 통해 학습을 극대화했다.

결론

- **인공지능 기술의 성능 향상**
 - 사람만이 할 수 있을 거라고 생각한 영역인 '바둑'에 인공지능을 적용함으로써 도전적인 연구를 했다.
 - 그동안 바둑은 경우의 수가 매우 많고 복잡하여 인공지능 분야에서 쉽사리 아직 정복하지 못했다. 그러나 알파고를 통해서 인공지능의 성능을 보여주는 실증 사례가 되었다.
- **몬테카를로 트리 탐색을 이용하여 바둑 문제 해결**
 - 가치네트워크처럼 바둑에서의 평가함수 정의
 - 딥러닝을 적용하지 않았던 방법들과 더불어 네트워크를 고려하여 기존의 문제를 더 나은 성능으로 해결한다는데 의의가 있다. 따라서, 어떤 도메인에 기존의 문제 풀이 방법과 딥러닝을 적용하여 새로운 방법론을 적용도 고려해보자.
- **엄청난 양의 데이터와 분산된 CPU / GPU 사용**
 - KGS 기보 사이트에서 16만 기보를 활용하여 약 2억 4,000만개 이상의 데이터 활용했다.
 - 1,202개의 CPU와 176개의 GPU 사용을 통해 학습을 극대화했다.

신의 한수



알파고 vs 이세돌, 제4국 78수



AlphaGo: The Movie 중
제4국 78수에 대한 확률을 계산하는 장면

알파고 그 이후: 알파고 제로(1/2)

ARTICLE

Mastering the game of Go without human knowledge

David Silver^{1*}, Julian Schrittwieser^{1*}, Karen Simonyan^{1*}, Ioannis Antonoglou¹, Aja Huang¹, Arthur Guez¹, Thomas Hubert¹, Lucas Baker¹, Matthew Lai¹, Adrian Bolton¹, Yutian Chen¹, Timothy Lillicrap¹, Fan Hui¹, Laurent Sifre¹, George van den Driessche¹, Thore Graepel¹ & Demis Hassabis¹

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

Much progress towards artificial intelligence has been made using supervised learning systems that are trained to replicate the decisions of human experts^{1–4}. However, expert data sets are often expensive, unreliable or simply unavailable. Even when reliable data sets are available, they may impose a ceiling on the performance of systems trained in this manner⁵. By contrast, reinforcement learning systems are trained from their own experience, in principle allowing them to exceed human capabilities, and to operate in domains where human expertise is lacking. Recently, there has been rapid progress towards this goal, using deep neural networks trained by reinforcement learning. These systems have outperformed humans in computer games, such as Atari^{6,7} and 3D virtual environments^{8–10}. However, the most chal-

lenged solely by self-play reinforcement learning, starting from random play, without any supervision or use of human data. Second, it uses only the black and white stones from the board as input features. Third, it uses a single neural network, rather than separate policy and value networks. Finally, it uses a simpler tree search that relies upon this single neural network to evaluate positions and sample moves, without performing any Monte Carlo rollouts. To achieve these results, we introduce a new reinforcement learning algorithm that incorporates lookahead search inside the training loop, resulting in rapid improvement and precise and stable learning. Further technical differences in the search algorithm, training procedure and network architecture are described in Methods.

알파고 제로 논문

[RLPR] D.Silver et al., “Mastering the game of Go with Deep Neural Network and Tree Search”, *Nature*, 2016

● 심층학습의 개선

- 듀얼 네트워크: 정책 네트워크와 가치 네트워크의 통합하여 학습
- ResNet과 같은 40층 이상의 잔차 네트워크 사용하여 멀티태스크 학습

● 강화학습의 개선

- KGS 바둑 데이터를 지도학습을 이용하지 않고 오로지 자기 대국(Self-play)만으로 프로를 뛰어넘는 수준까지 학습하는데 성공

● 몬테카를로 트리 탐색의 개선

- 기존 버전에서 실시하던 플레이 아웃을 사용하지 않음
- 이제 승률을 가치 네트워크만으로 판단하여 플레이 아웃의 승률을 고려하지 않음

알파고 그 이후: 알파고 제로(1/2)

ARTICLE

doi:10.1038/nature24270

Mastering the game of Go without human knowledge

David Silver^{1*}, Julian Schrittwieser^{1*}, Karen Simonyan^{1*}, Ioannis Antonoglou¹, Aja Huang¹, Arthur Guez¹, Thomas Hubert¹, Lucas Baker¹, Matthew Lai¹, Adrian Bolton¹, Yutian Chen¹, Timothy Lillicrap¹, Fan Hui¹, Laurent Sifre¹, George van den Driessche¹, Thore Graepel¹ & Demis Hassabis¹

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

Much progress towards artificial intelligence has been made using supervised learning systems that are trained to replicate the decisions of human experts^{1–4}. However, expert data sets are often expensive, unreliable or simply unavailable. Even when reliable data sets are available, they may impose a ceiling on the performance of systems trained in this manner⁵. By contrast, reinforcement learning systems are trained from their own experience, in principle allowing them to exceed human capabilities, and to operate in domains where human expertise is lacking. Recently, there has been rapid progress towards this goal, using deep neural networks trained by reinforcement learning. These systems have outperformed humans in computer games, such as Atari^{6,7} and 3D virtual environments^{8–10}. However, the most chal-

lenged solely by self-play reinforcement learning, starting from random play, without any supervision or use of human data. Second, it uses only the black and white stones from the board as input features. Third, it uses a single neural network, rather than separate policy and value networks. Finally, it uses a simpler tree search that relies upon this single neural network to evaluate positions and sample moves, without performing any Monte Carlo rollouts. To achieve these results, we introduce a new reinforcement learning algorithm that incorporates lookahead search inside the training loop, resulting in rapid improvement and precise and stable learning. Further technical differences in the search algorithm, training procedure and network architecture are described in Methods.

알파고 제로 논문

심층학습의 개선

- 듀얼 네트워크: 정책 네트워크와 가치 네트워크의 통합하여 학습
- ResNet과 같은 40층 이상의 잔차 네트워크 사용하여 멀티태스크 학습

강화학습의 개선

- KGS 바둑 데이터를 지도학습을 이용하지 않고 오로지 자기 대국(Self-play)만으로 프로를 뛰어넘는 수준까지 학습하는데 성공

몬테카를로 트리 탐색의 개선

- 기존 버전에서 실시하던 플레이 아웃을 사용하지 않음
- 이제 승률을 가치 네트워크만으로 판단하여 플레이 아웃의 승률을 고려하지 않음

알파고 그 이후: 알파고 제로(1/2)

ARTICLE

doi:10.1038/nature24270

Mastering the game of Go without human knowledge

David Silver^{1*}, Julian Schrittwieser^{1*}, Karen Simonyan^{1*}, Ioannis Antonoglou¹, Aja Huang¹, Arthur Guez¹, Thomas Hubert¹, Lucas Baker¹, Matthew Lai¹, Adrian Bolton¹, Yutian Chen¹, Timothy Lillicrap¹, Fan Hui¹, Laurent Sifre¹, George van den Driessche¹, Thore Graepel¹ & Demis Hassabis¹

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

Much progress towards artificial intelligence has been made using supervised learning systems that are trained to replicate the decisions of human experts^{1–4}. However, expert data sets are often expensive, unreliable or simply unavailable. Even when reliable data sets are available, they may impose a ceiling on the performance of systems trained in this manner⁵. By contrast, reinforcement learning systems are trained from their own experience, in principle allowing them to exceed human capabilities, and to operate in domains where human expertise is lacking. Recently, there has been rapid progress towards this goal, using deep neural networks trained by reinforcement learning. These systems have outperformed humans in computer games, such as Atari^{6,7} and 3D virtual environments^{8–10}. However, the most chal-

lenged solely by self-play reinforcement learning, starting from random play, without any supervision or use of human data. Second, it uses only the black and white stones from the board as input features. Third, it uses a single neural network, rather than separate policy and value networks. Finally, it uses a simpler tree search that relies upon this single neural network to evaluate positions and sample moves, without performing any Monte Carlo rollouts. To achieve these results, we introduce a new reinforcement learning algorithm that incorporates lookahead search inside the training loop, resulting in rapid improvement and precise and stable learning. Further technical differences in the search algorithm, training procedure and network architecture are described in Methods.

알파고 제로 논문

심층학습의 개선

- 듀얼 네트워크: 정책 네트워크와 가치 네트워크의 통합하여 학습
- ResNet과 같은 40층 이상의 잔차 네트워크 사용하여 멀티태스크 학습

강화학습의 개선

- KGS 바둑 데이터를 지도학습을 이용하지 않고 오로지 자기 대국(Self-play)만으로 프로를 뛰어넘는 수준까지 학습하는데 성공

몬테카를로 트리 탐색의 개선

- 기존 버전에서 실시하던 플레이 아웃을 사용하지 않음
- 이제 승률을 가치 네트워크만으로 판단하여 플레이 아웃의 승률을 고려하지 않음

알파고 그 이후: 알파고 제로(2/2)

The Atlantic

TECHNOLOGY

The AI That Has Nothing to Learn From Humans

DeepMind's new self-taught Go-playing program is making moves that other players describe as "alien" and "from an alternate dimension."

DAWN CHAN OCTOBER 20, 2017



MORE STORIES

- How Checkers Was Solved by ALEXIS C. MADRIGAL
- When Computers Started Beating Chess Champions by MARINA KOREN
- How Karen Became a Coronavirus Villain by KAITLYN TIFFANY

It was a tense, grueling day in 1835 Japan. The country's reigning Go player, Honinbo Jowa, took his seat across a board from a 25-year-old nobody by the name of Akaboshi Hidetaka. Both men had spent their lives mastering the two-player strategy game that's long been popular in East Asia. Their face-off, that day, was high-stakes: Honinbo and Akaboshi represented two Go houses fighting for power, and the rivalry between the two camps had lately exploded into accusations of foul play.

Little did they know that the match—now remembered by Go historians as the “blood-vomiting game”—would last for several grueling days. Or that it would lead to a grisly end.

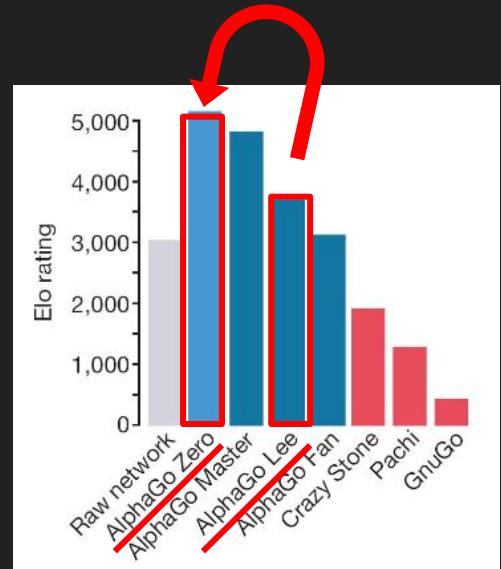
Early on, the young Akaboshi took a lead. But then, according to lore, “phobs” appeared and showed Honinbo three crucial moves. He comeback was so overwhelming that, as the story goes, his junior opponent keeled over and began coughing up blood. Weeks later, Akaboshi was found dead. Historians have speculated that he might have had an undiagnosed respiratory disease.

It makes a certain kind of sense that the game’s connoisseurs might have wondered if they’d seen glimpses of the occult in those three so-called ghost moves. Unlike something like tic-tac-toe, which is straightforward enough that the optimal strategy is always clear-cut, Go is so complex that new, unfamiliar strategies can feel astonishing, revolutionary, or even intensely

Enjoy unlimited access to The Atlantic for less than \$1 per week. [Sign in](#)

[Subscribe Now](#)

알파고 제로에게 전패 후 좌절한 캐제 9단의 모습



알파고 제로 엘로 평점: 5,000점

구현 / 실습

A L P H A G O



Tensorflow MiniGo

The screenshot shows the GitHub repository for "tensorflow/minigo". The repository has 1,421 commits, 6 branches, 0 packages, 1 release, 20 contributors, and follows the Apache-2.0 license. The pull requests section lists several recent changes, including fixes for MiniGUI-related crashes and updates to Docker images. The repository has 150 watchers, 3.1k stars, and 497 forks.

Branch	Commit Message	Time Ago
master	Fixed MiniGUI-related crashes (#990)	22 days ago
cluster	update docker images to TF 1.15.0	6 months ago
minigui	Fixed MiniGUI-related crashes (#990)	22 days ago
mt_perf	sample fixed fraction of examples from fixed number of games	4 months ago
oneoffs	update dependencies (#987)	last month
ratings	"Rate_subdir" utility (#918)	7 months ago
rl_loop	clean up	8 months ago
sgf	Added comments that games are public domain	2 years ago
testing	update docker images to TF 1.15.0	6 months ago
tests	Add channel order param to preprocessing tests. (#986)	22 days ago
.bazelrc	add generated .bazelrc files to .gitignore	10 months ago
.githignore	Adds a "Joseki Explorer" React App (#875)	8 months ago
.pylintrc	use explicit imports for tf.contrib modules	6 months ago
CODE_OF_CONDUCT.md	Add code of conduct.	2 years ago
CONTRIBUTING.md	Several python cleanups.	2 years ago
LICENSE	initial commit	3 years ago
README.md	update to TF 1.15.0	6 months ago
RESULTS.md	Minor formatting updates to RESULTS.md	last month
WORKSPACE	update to TF 1.15.0	6 months ago
pull_mu	Several author cleannings	2 years ago

Minigo: A minimalist Go engine modeled after AlphaGo Zero, built on MuGo

This is an implementation of a neural-network based Go AI, using TensorFlow. While inspired by DeepMind's AlphaGo algorithm, this project is not a DeepMind project nor is it affiliated with the official AlphaGo project.

This is NOT an official version of AlphaGo

Repeat, this is *not* the official AlphaGo program by DeepMind. This is an independent effort by Go enthusiasts to replicate the results of the AlphaGo Zero paper ("Mastering the Game of Go without Human Knowledge," *Nature*), with some resources generously made available by Google.

Minigo is based off of Brian Lee's "MuGo" -- a pure Python implementation of the first AlphaGo paper "Mastering the Game of Go with Deep Neural Networks and Tree Search" published in *Nature*. This implementation adds features and architecture changes present in the more recent AlphaGo Zero paper, "Mastering the Game of Go without Human Knowledge". More recently, this architecture was extended for Chess and Shogi in "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". These papers will often be abridged in Minigo documentation as AG (for AlphaGo), AGZ (for AlphaGo Zero), and AZ (for AlphaZero) respectively.

Goals of the Project

- Provide a clear set of learning examples using Tensorflow, Kubernetes, and Google Cloud Platform for establishing Reinforcement Learning pipelines on various hardware accelerators.
- Reproduce the methods of the original DeepMind AlphaGo papers as faithfully as possible, through an open-source implementation and open-source pipeline tools.
- Provide our data, results, and discoveries in the open to benefit the Go, machine learning, and Kubernetes communities.

An explicit non-goal of the project is to produce a competitive Go program that establishes itself as the top Go AI. Instead, we strive for a readable, understandable implementation that can benefit the community, even if that means our implementation is not as fast or efficient as possible.

While this product might produce such a strong model, we hope to focus on the process. Remember, getting there is half the fun. :)

We hope this project is an accessible way for interested developers to have access to a strong Go model with an easy-to-understand platform of python code available for extension, adaptation, etc.

If you'd like to read about our experiences training models, see [RESULTS.md](#).

To see our guidelines for contributing, see [CONTRIBUTING.md](#).

알파고 관련 사이트

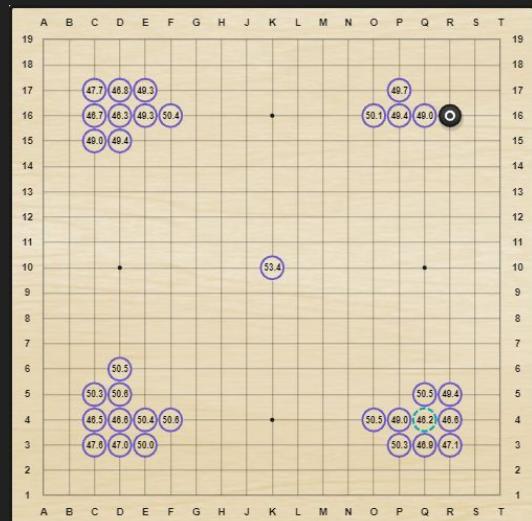


알파고 티칭 툴은 바둑에서 새롭고 창의적인 수를 둘 수 있도록 도와줍니다.

[▲ 줄이기](#)

이 툴은 사람의 대국 231,000판과 알파고와 프로기사 간의 대국 75판을 분석해 선별한 6,000가지 포석에 대한 연구 결과를 제공합니다.

알파고의 수들이 프로 및 아마 기사들의 수들과 어떻게 다른지 비교해 보세요.



지금까지 들어주셔서
정말 감사합니다

A L P H A G O

질문 및 피드백은 언제나 환영입니다.
오른쪽 상단의 메일로 질문이나 피드백 주세요.



여기에서 참조한 내용은

논문으로 짚어보는 딥러닝의 맥

최성준 | edwith 좋아요 247 | 수강생 7892

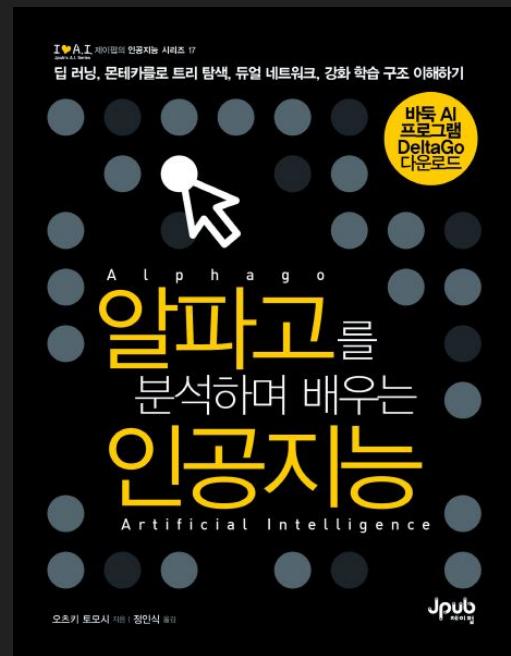
감의목록
공지게시판
성적조회

Nature 논문으로 살펴보는 AlphaGo 알고리즘

학습목표

전 세계의 이목을 집중시킨 AlphaGo에 대한 내용이 실린 Nature 논문을 참고하여 AlphaGo에 대해 알아보겠습니다. 알파고를 만들어낸 '딥마인드'라는 기업은 강화학습(Reinforcement Learning)을 활용하여 게임을 하는 인공지능을 만들었던 기업이었지만 이후 구글이 인수하여 AlphaGo라는 인공지능을 만들어 냈습니다. 이 강의에서는 AlphaGo가 하는 바둑(Go)이 얼마나 복잡하고 어려운지 알아보고 Monte-Carlo Tree Search, Networks, Experiments 등을 알아보면서 기존의 방법론들과 딥러닝과의 결합으로 나타난 AlphaGo가 AI분야에서 얼마나 유의미한 일인지 알아보겠습니다.

Nature 논문으로 살펴보는 AlphaGo 알고리즘, 최성준



알파고를 분석하며 배우는 인공지능,
제이펍 출판

여기에서 참조한 내용은



AlphaGo - The Movie, Deepmind (2017)

QnA

A L P H A G O

참고자료

- Mastering the game of Go with deep neural network and tree search: <https://www.nature.com/articles/nature24270?sf123103138=1>
- Mastering the game of Go without human knowledge: <https://www.nature.com/articles/nature24270.pdf>
- DeepMind AlphaGo Full Documentary: <https://youtu.be/WXuK6gekU1Y>
- [논문리뷰] DeepMind의 AlphaGO 논문 둡아보기: <https://wegoonnamakeit.tistory.com/17>
- Nature 논문으로 살펴보는 AlphaGo 알고리즘: <https://www.edwith.org/deeplearningchoi/lecture/15300>
- 이주열님의 AlphaGo 알고리즘 요약: <https://www.slideshare.net/zenithon/alphago>
- 파이썬과 캐라스를 이용한 알파제로 만들기:
https://github.com/KerasKorea/KEKOxTutorial/blob/master/04_%ED%8C%8C%EC%9D%B4%EC%8D%AC%EA%B3%BC_%EC%BC%80%EB%9D%BC%EC%8A%A4%EB%A5%BC_%EC%9D%B4%EC%9A%A9%ED%95%9C_%EC%95%8C%ED%8C%8C_%EC%A0%9C%EB%A1%9C_%EB%A7%8C%EB%93%A4%EA%B8%B0.md
- Tensorflow minigo GitHub: <https://github.com/tensorflow/minigo>
- David Silver RL Lecture: <https://www.davidsilver.uk/teaching/>
- AlphaGo Teach: <https://alphagoteach.deepmind.com/ko>
- AlphaGo의 알고리즘과 모델: <http://sanghyukchun.github.io/97/>

참고자료

- AlphaGo의 알고리즘과 모델: <http://sanghyukchun.github.io/97/>
- AlphaStar: <https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii>
- <https://shuuki4.wordpress.com/2016/03/11/alphago-alphago-pipeline-%ED%97%A4%EC%A7%91%EA%B8%B0/>
- <https://migom.tistory.com/23>
- AlphaGo의 인공지능 알고리즘 분석 - SPRi ISSUE REPORT 2016년 3월호
- 알파고가 바둑 두는 방법: <http://tmmse.xyz/2016/03/04/alphago/>
- <http://m.newspim.com/news/view/20160318000429>